glucat

0.12.0

Generated by Doxygen 1.12.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 cga3 Namespace Reference

Definitions for 3D Conformal Geometric Algebra [DL].

Functions

- template<typename Multivector_T >
  Multivector_T cga3 (const Multivector_T &x)

  Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].
- template<typename Multivector_T >
  Multivector_T cga3std (const Multivector_T &X)

  Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].
- template<typename Multivector_T >
  Multivector_T agc3 (const Multivector_T &X)

  Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].

5.1.1 Detailed Description

Definitions for 3D Conformal Geometric Algebra [DL].

5.1.2 Function Documentation

5.1.2.1 agc3()

```
template<typename Multivector_T >
Multivector_T cga3::agc3 (
            const Multivector_T & X) [inline]
```

Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].

Definition at line 126 of file PyClical.h.

References cga3std().

5.1.2.2 cga3()

```
template<typename Multivector_T >
Multivector_T cga3::cga3 (
            const Multivector_T & x) [inline]
```

Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].

Definition at line 103 of file PyClical.h.

5.1.2.3 cga3std()

```
template<typename Multivector_T >
Multivector_T cga3::cga3std (
            const Multivector_T & X) [inline]
```

Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].

Definition at line 114 of file PyClical.h.

Referenced by agc3().

5.2 glucat Namespace Reference

Namespaces

- namespace gen
- namespace matrix
- namespace timing

Classes

- class basis_table

  Table of basis elements used as a cache by basis_element()
- class bool_to_type

  Bool to type.
- class clifford_algebra

  clifford_algebra<> declares the operations of a Clifford algebra
- class compare_types

  Type comparison.
- class compare_types< T, T >
- class control_t

  Parameters to control tests.
- struct CTAssertion

  Compile time assertion.
- struct CTAssertion< true >
- class error

  Specific exception class.
- class framed_multi

  A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.

- class glucat_error

  Abstract exception class.
- class index_set

  Index set class based on std::bitset<> in Gnu standard C++ library.
- class index_set_hash
- class matrix_multi

  A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector.
- class numeric_traits

  Extra traits which extend numeric limits.
- class random_generator

  Random number generator with single instance per Scalar_T.
- class sorted_range

  Sorted range for use with output.
- class sorted_range< Sorted_Map_T, Sorted_Map_T >

Typedefs

- using index_t = int

  Size of index_t should be enough to represent LO, HI.
- using set_value_t = unsigned long

  Size of set_value_t should be enough to contain index_set<LO,HI>
- typedef int(∗ intfn) ()

  For exception catching: pointer to function returning int.
- typedef int(∗ intintfn) (int)

  For exception catching: pointer to function of int returning int.
- using tuning_slow
- using tuning_naive
- using tuning_fast

Functions

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

  Test for inequality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool

  Test for inequality of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator!= (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

  Test for inequality of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto error_squared_tol (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Quadratic norm error tolerance relative to a specific multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto error_squared (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T

Relative or absolute error using the quadratic norm.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool

  Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

  Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator- (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator* (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Geometric product.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator^ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator& (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inner product.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator% (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Left contraction.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto star (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

  Hestenes scalar product.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of multivector and scalar.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of scalar and multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator| (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Transformation via twisted adjoint action.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric multiplicative inverse.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Integer power of multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Multivector power of multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto outer_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product power of multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Real part: synonym for scalar part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Imaginary part: deprecated (always 0)

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Pure part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Even part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Odd part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto vector_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar_T >

  Vector part of multivector, as a vector_t with respect to frame()

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Main involution, each {i} is replaced by -{i} in each term, eg. {1}{2} -> (-{2})*(-{1})*

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Reversion, eg. {1}{2} -> {2}*{1}.*

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Conjugation, rev o invo == invo o rev.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

*Scalar_T quadratic form == (rev(x)*x)(0)*

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto norm (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar_T norm == sum of norm of coordinates.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Absolute value == sqrt(norm)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto max_abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Maximum of absolute values of components of multivector: multivector infinity norm.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto complexifier (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of -1 which commutes with all members of the frame of the given multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto elliptic (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto clifford_exp (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto log (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto log (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Cosine of multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto cosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto sinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Inverse hyperbolic sine of multivector with specified complexifier.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto tanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static void check_complex (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false)

  Check that i is a valid complexifier for val.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator* (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator^ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Outer product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator& (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Inner product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator% (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Left contraction.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto star (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

    Hestenes scalar product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Geometric quotient.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator| (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Transformation via twisted adjoint action.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator>> (std::istream &s, framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

    Read multivector from input.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (std::ostream &os, const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

    Write multivector to output.

- template<typename Scalar_T , const index_t LO, const index_t HI>
auto operator<< (std::ostream &os, const std::pair< const index_set< LO, HI >, Scalar_T > &term) -> std::ostream &

    Write term to output.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto exp (const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Exponential of multivector.

- template<typename Scalar_T , const index_t LO, const index_t HI>
static auto crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T

    Coordinate of product of terms.

- template<typename Scalar_T , const index_t LO, const index_t HI>
auto operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >

    Product of terms.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto sqrt (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Square root of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto log (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Natural logarithm of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI>
static auto crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T

Coordinate of product of terms.

- _GLUCAT_CTAssert (std::numeric_limits< unsigned char >::radix==2, CannotDetermineBitsPerChar) const index_t BITS_PER_CHAR

  If radix of unsigned char is not 2, we can't easily determine number of bits from sizeof.
- _GLUCAT_CTAssert (_GLUCAT_BITS_PER_ULONG==BITS_PER_SET_VALUE, BitsPerULongDoesNot↩ MatchSetValueT) const index_t DEFAULT_LO

  Default lowest index in an index set.
- template<typename LHS_T , typename RHS_T >
  auto pos_mod (LHS_T lhs, RHS_T rhs) -> LHS_T

  Modulo function which works reliably for lhs < 0.
- template<const index_t LO, const index_t HI>
  auto operator^ (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Symmetric set difference: exclusive or.
- template<const index_t LO, const index_t HI>
  auto operator& (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Set intersection: and.
- template<const index_t LO, const index_t HI>
  auto operator| (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Set union: or.
- template<const index_t LO, const index_t HI>
  auto compare (const index_set< LO, HI > &a, const index_set< LO, HI > &b) -> int

  "lexicographic compare" eg. {3,4,5} is less than {3,7,8}
- _GLUCAT_CTAssert (sizeof(set_value_t) >=sizeof(std::bitset< DEFAULT_HI-DEFAULT_LO >), Default_↩ index_set_too_big_for_value) template< const index_t LO

  Size of set_value_t should be enough to contain bitset<DEFAULT_HI-DEFAULT_LO>
- const index_t HI auto operator<< (std::ostream &os, const index_set< LO, HI > &ist) -> std::ostream &
- template<const index_t LO, const index_t HI>
  auto operator>> (std::istream &s, index_set< LO, HI > &ist) -> std::istream &

  Read in index set.
- auto sign_of_square (index_t j) -> int

  Square of generator {j}.
- template<const index_t LO, const index_t HI>
  auto min_neg (const index_set< LO, HI > &ist) -> index_t

  Minimum negative index, or 0 if none.
- template<const index_t LO, const index_t HI>
  auto max_pos (const index_set< LO, HI > &ist) -> index_t

  Maximum positive index, or 0 if none.
- template<const index_t LO, const index_t HI>
  auto operator<< (std::ostream &os, const index_set< LO, HI > &ist) -> std::ostream &

  Write out index set.
- static auto inverse_reversed_gray (unsigned long x) -> unsigned long

  Inverse reversed Gray code.
- static auto inverse_gray (unsigned long x) -> unsigned long

  Inverse Gray code.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator* (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto operator^ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

Outer product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator& (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Inner product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator% (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Left contraction.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto star (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

  Hestenes scalar product.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator| (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Transformation via twisted adjoint action.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator>> (std::istream &s, matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

  Read multivector from input.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (std::ostream &os, const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

  Write multivector to output.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto reframe (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs, matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs_reframed, matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs_reframed) -> const index_set< LO, HI >

  Find a common frame for operands of a binary operator.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto matrix_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto matrix_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto exp (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.

- auto offset_level (const index_t p, const index_t q) -> index_t

Determine the log2 dim corresponding to signature p, q.

- template<typename Matrix_Index_T , const index_t LO, const index_t HI>
  static auto folded_dim (const index_set< LO, HI > &sub) -> Matrix_Index_T

  Determine the matrix dimension of the fold of a subalegbra.
- template<typename Multivector_T , typename Matrix_T , typename Basis_Matrix_T >
  static auto fast (const Matrix_T &X, index_t level) -> Multivector_T

  Inverse generalized Fast Fourier Transform.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P , const size_t Size>
  static auto pade_approx (const std::array< Scalar_T, Size > &numer, const std::array< Scalar_T, Size > &denom, const matrix_multi< Scalar_T, LO, HI, Tune_P > &X) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Pade' approximation.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static void db_step (matrix_multi< Scalar_T, LO, HI, Tune_P > &M, matrix_multi< Scalar_T, LO, HI, Tune_P > &Y)

  Single step of product form of Denman-Beavers square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto db_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 4)) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Product form of Denman-Beavers square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto cr_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_Y_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 1)) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Cyclic reduction square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto pade_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Pade' approximation of log.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto cascade_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Incomplete square root cascade and Pade' approximation of log.
- template<typename Scalar_T >
  auto log2 (const Scalar_T &x) -> Scalar_T

  Log base 2 of scalar.
- template<typename Scalar_T >
  auto to_promote (const Scalar_T &val) -> typename numeric_traits< Scalar_T >::promoted::type

  Cast to promote.
- template<typename Scalar_T >
  auto to_demote (const Scalar_T &val) -> typename numeric_traits< Scalar_T >::demoted::type

  Cast to demote.
- int try_catch (intfn f)

  Exception catching for functions returning int.
- int try_catch (intintfn f, int arg)

  Exception catching for functions of int returning int.

Variables

- const double MS_PER_S = 1000.0

  Timing constant: deprecated here - moved to test/timing.h.
- const index_t BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits

  Number of bits in set_value_t.

- const index_t DEFAULT_HI = index_t(BITS_PER_SET_VALUE / 2)

    Default highest index in an index set.
- static const long double l_pi = 3.1415926535897932384626433832795029L
- static const long double l_ln2 = 0.6931471805599453094172321214581766L
- const unsigned int Tuning_Int_Digits = std::numeric_limits<int>::digits
- const unsigned int Tuning_Max_Threshold = 1 << Tuning_Int_Digits
- const unsigned int Tuning_Slow_Mult_Matrix_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Slow_Basis_Max_Count = 0
- const unsigned int Tuning_Slow_Fast_Size_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Slow_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Naive_Mult_Matrix_Threshold = 0
- const unsigned int Tuning_Naive_Basis_Max_Count = Tuning_Max_Threshold
- const unsigned int Tuning_Naive_Fast_Size_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Naive_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
- const unsigned int Tuning_Fast_Mult_Matrix_Threshold = 0
- const unsigned int Tuning_Fast_Div_Max_Steps = 0
- const unsigned int Tuning_Fast_CR_Sqrt_Max_Steps = 256
- const unsigned int Tuning_Fast_DB_Sqrt_Max_Steps = 256
- const unsigned int Tuning_Fast_Log_Max_Outer_Steps = 16
- const unsigned int Tuning_Fast_Log_Max_Inner_Steps = 8
- const unsigned int Tuning_Fast_Basis_Max_Count = 1
- const unsigned int Tuning_Fast_Fast_Size_Threshold = 0
- const unsigned int Tuning_Fast_Inv_Fast_Dim_Threshold = 0
- const unsigned int Tuning_Fast_Products_Size_Threshold = 0

5.2.1 Typedef Documentation

5.2.1.1 index_t

```
using glucat::index_t = int
```

Size of index_t should be enough to represent LO, HI.

Definition at line 77 of file global.h.

5.2.1.2 intfn

```
typedef int(* glucat::intfn) ()
```

For exception catching: pointer to function returning int.

Definition at line 37 of file try_catch.h.

5.2.1.3 intintfn

```
typedef int(* glucat::intintfn) (int)
```

For exception catching: pointer to function of int returning int.

Definition at line 40 of file try_catch.h.

5.2.1.4 set_value_t

```
using glucat::set_value_t = unsigned long
```

Size of set_value_t should be enough to contain index_set<LO,HI>

Definition at line 79 of file global.h.

5.2.1.5 tuning_fast

```
using glucat::tuning_fast
```

Initial value:

```
tuning
    <
      Tuning_Fast_Mult_Matrix_Threshold,
      Tuning_Fast_Div_Max_Steps,
      Tuning_Fast_CR_Sqrt_Max_Steps,
      Tuning_Fast_DB_Sqrt_Max_Steps,
      Tuning_Fast_Log_Max_Outer_Steps,
      Tuning_Fast_Log_Max_Inner_Steps,
      Tuning_Fast_Basis_Max_Count,
      Tuning_Fast_Fast_Size_Threshold,
      Tuning_Fast_Inv_Fast_Dim_Threshold,
      Tuning_Fast_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      Tuning_Default_Function_Precision
    >
```

Definition at line 97 of file tuning.h.

5.2.1.6 tuning_naive

```
using glucat::tuning_naive
```

Initial value:

```
tuning
    <
      Tuning_Naive_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Naive_Basis_Max_Count,
      Tuning_Naive_Fast_Size_Threshold,
      Tuning_Naive_Inv_Fast_Dim_Threshold,
      Tuning_Default_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      Tuning_Default_Function_Precision
    >
```

Definition at line 69 of file tuning.h.

5.2.1.7 tuning_slow

```
using glucat::tuning_slow
```

Initial value:

```
tuning
    <
      Tuning_Slow_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Slow_Basis_Max_Count,
      Tuning_Slow_Fast_Size_Threshold,
      Tuning_Slow_Inv_Fast_Dim_Threshold,
      Tuning_Slow_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      Tuning_Default_Function_Precision
    >
```

Definition at line 47 of file tuning.h.

5.2.2 Function Documentation

5.2.2.1 _GLUCAT_CTAssert() [1/3]

```
glucat::_GLUCAT_CTAssert (
            _GLUCAT_BITS_PER_ULONG  = =BITS_PER_SET_VALUE,
            BitsPerULongDoesNotMatchSetValueT ) const
```

Default lowest index in an index set.

5.2.2.2 _GLUCAT_CTAssert() [2/3]

```
glucat::_GLUCAT_CTAssert (
            sizeof(set_value_t) >=sizeof(std::bitset< DEFAULT_HI-DEFAULT_LO >) ,
            Default_index_set_too_big_for_value ) const
```

Size of set_value_t should be enough to contain bitset<DEFAULT_HI-DEFAULT_LO>

Write out index set

5.2.2.3 _GLUCAT_CTAssert() [3/3]

```
glucat::_GLUCAT_CTAssert (
            std::numeric_limits< unsigned char >::radix  = =2,
            CannotDetermineBitsPerChar ) const
```

If radix of unsigned char is not 2, we can't easily determine number of bits from sizeof.

Number of bits per char is used to determine number of bits in set_value_t

5.2.2.4 abs()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::abs (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Absolute value == sqrt(norm)

Definition at line 577 of file clifford_algebra_imp.h.

References glucat::numeric_traits< Scalar_T >::sqrt().

Referenced by PyClical.clifford::abs(), clifford_to_str(), matrix_log(), and matrix_sqrt().

5.2.2.5 acos() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acos (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse cosine of multivector.

Definition at line 903 of file clifford_algebra_imp.h.

References acos(), and complexifier().

5.2.2.6 acos() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acos (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse cosine of multivector with specified complexifier.

Definition at line 883 of file clifford_algebra_imp.h.

References acosh(), and check_complex().

Referenced by acos().

5.2.2.7 acosh() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acosh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic cosine of multivector.

Definition at line 844 of file clifford_algebra_imp.h.

References acosh(), and complexifier().

5.2.2.8 acosh() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acosh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic cosine of multivector with specified complexifier.

Definition at line 825 of file clifford_algebra_imp.h.

References check_complex(), log(), norm(), and sqrt().

Referenced by acos(), and acosh().

5.2.2.9 approx_equal() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> bool  [inline]
```

Test for approximate equality of multivectors.

Definition at line 169 of file clifford_algebra_imp.h.

References approx_equal(), and error_squared_tol().

5.2.2.10 approx_equal() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs,
            const Scalar_T threshold,
            const Scalar_T tolerance) -> bool  [inline]
```

Test for approximate equality of multivectors.

Definition at line 154 of file clifford_algebra_imp.h.

References error_squared().

Referenced by approx_equal(), and matrix_sqrt().

5.2.2.11 asin() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asin (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse sine of multivector.

Definition at line 1008 of file clifford_algebra_imp.h.

References asin(), and complexifier().

5.2.2.12 asin() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asin (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val,
             const Multivector< Scalar_T, LO, HI, Tune_P > & i,
             const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse sine of multivector with specified complexifier.

Definition at line 988 of file clifford_algebra_imp.h.

References asinh(), and check_complex().

Referenced by asin().

5.2.2.13 asinh() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asinh (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic sine of multivector.

Definition at line 949 of file clifford_algebra_imp.h.

References asinh(), and complexifier().

5.2.2.14 asinh() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asinh (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val,
             const Multivector< Scalar_T, LO, HI, Tune_P > & i,
             const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic sine of multivector with specified complexifier.

Definition at line 930 of file clifford_algebra_imp.h.

References check_complex(), log(), norm(), and sqrt().

Referenced by asin(), and asinh().

5.2.2.15 atan() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atan (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse tangent of multivector.

Definition at line 1108 of file clifford_algebra_imp.h.

References atan(), and complexifier().

5.2.2.16 atan() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atan (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse tangent of multivector with specified complexifier.

Definition at line 1088 of file clifford_algebra_imp.h.

References atanh(), and check_complex().

Referenced by atan().

5.2.2.17 atanh() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atanh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic tangent of multivector.

Definition at line 1052 of file clifford_algebra_imp.h.

References atanh(), and complexifier().

5.2.2.18 atanh() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atanh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Inverse hyperbolic tangent of multivector with specified complexifier.

Definition at line 1035 of file clifford_algebra_imp.h.

References check_complex(), log(), and norm().

Referenced by atan(), and atanh().

5.2.2.19 cascade_log()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static auto glucat::cascade_log (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix_multi<Scalar↩
_T,LO,HI,Tune_P>   [static]
```

Incomplete square root cascade and Pade' approximation of log.

Definition at line 1920 of file matrix_multi_imp.h.

References db_step(), epsilon, glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), norm(), and pade_log().

Referenced by matrix_log().

5.2.2.20 check_complex()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static void glucat::check_complex (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false)  [inline], [static]
```

Check that i is a valid complexifier for val.

Definition at line 652 of file clifford_algebra_imp.h.

References complexifier().

Referenced by acos(), acosh(), asin(), asinh(), atan(), atanh(), cos(), log(), log(), sin(), sqrt(), sqrt(), and tan().

5.2.2.21 clifford_exp()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::clifford_exp (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line 690 of file clifford_algebra_imp.h.

References log2().

Referenced by exp(), and exp().

5.2.2.22 compare()

```
template<const index_t LO, const index_t HI>
auto glucat::compare (
            const index_set< LO, HI > & a,
            const index_set< LO, HI > & b) -> int  [inline]
```

"lexicographic compare" eg. {3,4,5} is less than {3,7,8}

Lexicographic ordering of two sets: -1 if a<b, +1 if a>b, 0 if a==b.

Definition at line 574 of file index_set_imp.h.

5.2.2.23 complexifier()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::complexifier (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>
```

Square root of -1 which commutes with all members of the frame of the given multivector.

Definition at line 592 of file clifford_algebra_imp.h.

References pos_mod().

Referenced by acos(), acosh(), asin(), asinh(), atan(), atanh(), check_complex(), cos(), elliptic(), log(), sin(), sqrt(), and tan().

5.2.2.24 conj()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::conj (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Conjugation, rev o invo == invo o rev.

Definition at line 553 of file clifford_algebra_imp.h.

5.2.2.25 cos() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cos (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Cosine of multivector.

Definition at line 874 of file clifford_algebra_imp.h.

References complexifier(), and cos().

5.2.2.26 cos() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cos (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Cosine of multivector with specified complexifier.

Definition at line 851 of file clifford_algebra_imp.h.

References check_complex(), and exp().

Referenced by cos(), and tan().

5.2.2.27 cosh()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cosh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>  [inline]
```

Hyperbolic cosine of multivector.

Definition at line 807 of file clifford_algebra_imp.h.

References exp().

Referenced by tanh().

5.2.2.28 cr_sqrt()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static auto glucat::cr_sqrt (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            Scalar_T norm_Y_tol = std::pow(std::numeric_limits<Scalar_T>::epsilon(), 1)) ->
const matrix_multi<Scalar_T,LO,HI,Tune_P>   [static]
```

Cyclic reduction square root iteration.

Definition at line 1349 of file matrix_multi_imp.h.

References glucat::numeric_traits< Scalar_T >::NaN(), and norm().

Referenced by matrix_sqrt().

5.2.2.29 crd_of_mult() [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI>
static auto glucat::crd_of_mult (
            const std::pair< const index_set< LO, HI >, Scalar_T > & lhs,
            const std::pair< const index_set< LO, HI >, Scalar_T > & rhs) -> Scalar_T  [inline],
[static]
```

Coordinate of product of terms.

Referenced by operator%(), operator&(), operator*(), and operator^().

5.2.2.30 crd_of_mult() [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI>
static auto glucat::crd_of_mult (
            const std::pair< const index_set< LO, HI >, Scalar_T > & lhs,
            const std::pair< const index_set< LO, HI >, Scalar_T > & rhs) -> Scalar_T  [inline],
[static]
```

Coordinate of product of terms.

Definition at line 1709 of file framed_multi_imp.h.

5.2.2.31 db_sqrt()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static auto glucat::db_sqrt (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            Scalar_T norm_tol = std::pow(std::numeric_limits<Scalar_T>::epsilon(), 4)) ->
const matrix_multi<Scalar_T,LO,HI,Tune_P>  [static]
```

Product form of Denman-Beavers square root iteration.

Definition at line 1320 of file matrix_multi_imp.h.

References db_step(), glucat::numeric_traits< Scalar_T >::NaN(), and norm().

Referenced by matrix_sqrt().

5.2.2.32 db_step()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static void glucat::db_step (
            matrix_multi< Scalar_T, LO, HI, Tune_P > & M,
            matrix_multi< Scalar_T, LO, HI, Tune_P > & Y)  [inline], [static]
```

Single step of product form of Denman-Beavers square root iteration.

Definition at line 1308 of file matrix_multi_imp.h.

References inv().

Referenced by cascade_log(), and db_sqrt().

5.2.2.33 elliptic()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::elliptic (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Square root of -1 which commutes with all members of the frame of the given multivector The name "elliptic" is now deprecated: use "complexifier" instead.

Definition at line 643 of file clifford_algebra_imp.h.

References complexifier().

5.2.2.34 error_squared()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs,
            const Scalar_T threshold) -> Scalar_T  [inline]
```

Relative or absolute error using the quadratic norm.

Definition at line 134 of file clifford_algebra_imp.h.

References norm().

Referenced by approx_equal().

5.2.2.35 error_squared_tol()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared_tol (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T
```

Quadratic norm error tolerance relative to a specific multivector.

Definition at line 112 of file clifford_algebra_imp.h.

References glucat::numeric_traits< Scalar_T >::pow().

Referenced by approx_equal().

5.2.2.36 even()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::even (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Even part.

Definition at line 513 of file clifford_algebra_imp.h.

5.2.2.37 exp() [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::exp (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & val) -> const framed_multi<Scalar←
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line 1750 of file framed_multi_imp.h.

References clifford_exp(), exp(), and scalar().

Referenced by cos(), cosh(), exp(), matrix_log(), matrix_sqrt(), pow(), sin(), and sinh().

5.2.2.38 exp() [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::exp (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix_multi<Scalar←
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line 2086 of file matrix_multi_imp.h.

References clifford_exp(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar().

5.2.2.39 fast()

```
template<typename Multivector_T , typename Matrix_T , typename Basis_Matrix_T >
static auto glucat::fast (
            const Matrix_T & X,
            index_t level) -> Multivector_T [static]
```

Inverse generalized Fast Fourier Transform.

Definition at line 1027 of file matrix_multi_imp.h.

References fast(), glucat::matrix::signed_perm_nork(), and glucat::matrix::unit().

Referenced by fast(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi().

5.2.2.40 folded_dim()

```
template<typename Matrix_Index_T , const index_t LO, const index_t HI>
static auto glucat::folded_dim (
            const index_set< LO, HI > & sub) -> Matrix_Index_T   [inline], [static]
```

Determine the matrix dimension of the fold of a subalegbra.

Definition at line 101 of file matrix_multi_imp.h.

References offset_level().

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi().

5.2.2.41 imag()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::imag (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Imaginary part: deprecated (always 0)

Definition at line 497 of file clifford_algebra_imp.h.

5.2.2.42 inv()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::inv (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Geometric multiplicative inverse.

Definition at line 400 of file clifford_algebra_imp.h.

Referenced by db_step(), and matrix_log().

5.2.2.43 inverse_gray()

```
static auto glucat::inverse_gray (
            unsigned long x) -> unsigned long   [inline], [static]
```

Inverse Gray code.

Definition at line 863 of file index_set_imp.h.

Referenced by glucat::index_set< LO, HI >::sign_of_mult().

5.2.2.44 inverse_reversed_gray()

```
static auto glucat::inverse_reversed_gray (
            unsigned long x) -> unsigned long   [inline], [static]
```

Inverse reversed Gray code.

Definition at line 846 of file index_set_imp.h.

Referenced by glucat::index_set< LO, HI >::sign_of_mult().

5.2.2.45 involute()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::involute (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Main involution, each {i} is replaced by -{i} in each term, eg. {1}∗{2} -> (-{2})∗(-{1})

Main involution, each {i} is replaced by -{i} in each term, eg. {1} ->-{1}.

Definition at line 537 of file clifford_algebra_imp.h.

5.2.2.46 log() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & val,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & i,
            bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 1800 of file framed_multi_imp.h.

References check_complex(), and log().

5.2.2.47 log() [2/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & i,
            bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 2045 of file matrix_multi_imp.h.

References check_complex(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), and matrix_log().

5.2.2.48 log() [3/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P> [inline]
```

Natural logarithm of multivector.

Definition at line 799 of file clifford_algebra_imp.h.

References complexifier(), and log().

5.2.2.49 log() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Natural logarithm of multivector with specified complexifier.

Definition at line 791 of file clifford_algebra_imp.h.

References log().

Referenced by acosh(), asinh(), atanh(), log(), log(), log(), and pow().

5.2.2.50 log2()

```
template<typename Scalar_T >
auto glucat::log2 (
            const Scalar_T & x) -> Scalar_T [inline]
```

Log base 2 of scalar.

Definition at line 303 of file scalar.h.

References glucat::numeric_traits< Scalar_T >::log2().

Referenced by clifford_exp().

5.2.2.51 matrix_log()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_log (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & i,
            const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 1967 of file matrix_multi_imp.h.

References abs(), cascade_log(), glucat::matrix::classify_eigenvalues(), exp(), inv(), glucat::clifford_algebra< Scalar_T, Index_Set_T, matrix_log(), norm(), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar().

Referenced by log(), and matrix_log().

5.2.2.52 matrix_sqrt()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_sqrt (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & i,
            const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line 1571 of file matrix_multi_imp.h.

References abs(), approx_equal(), glucat::matrix::classify_eigenvalues(), cr_sqrt(), db_sqrt(), exp(), glucat::clifford_algebra< Scalar_T
matrix_sqrt(), norm(), pade_approx(), pow(), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar().

Referenced by matrix_sqrt(), and sqrt().

5.2.2.53 max_abs()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::max_abs (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Maximum of absolute values of components of multivector: multivector infinity norm.

Definition at line 585 of file clifford_algebra_imp.h.

5.2.2.54 max_pos()

```
template<const index_t LO, const index_t HI>
auto glucat::max_pos (
            const index_set< LO, HI > & ist) -> index_t  [inline]
```

Maximum positive index, or 0 if none.

Definition at line 977 of file index_set_imp.h.

5.2.2.55 min_neg()

```
template<const index_t LO, const index_t HI>
auto glucat::min_neg (
            const index_set< LO, HI > & ist) -> index_t  [inline]
```

Minimum negative index, or 0 if none.

Definition at line 970 of file index_set_imp.h.

5.2.2.56 norm()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::norm (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Scalar_T norm == sum of norm of coordinates.

Definition at line 569 of file clifford_algebra_imp.h.

Referenced by acosh(), asinh(), atanh(), cascade_log(), cr_sqrt(), db_sqrt(), error_squared(), matrix_log(), and matrix_sqrt().

5.2.2.57 odd()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::odd (
             const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Odd part.

Definition at line 521 of file clifford_algebra_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast().

5.2.2.58 offset_level()

```
auto glucat::offset_level (
             const index_t p,
             const index_t q) -> index_t   [inline]
```

Determine the log2 dim corresponding to signature p, q.

Definition at line 86 of file matrix_multi_imp.h.

References pos_mod().

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element(), and folded_dim().

5.2.2.59 operator"!=() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (
             const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
             const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> bool  [inline]
```

Test for inequality of multivectors.

Definition at line 86 of file clifford_algebra_imp.h.

5.2.2.60 operator"!=() [2/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const Scalar_T & scr) -> bool  [inline]
```

Test for inequality of multivector and scalar.

Definition at line 94 of file clifford_algebra_imp.h.

5.2.2.61 operator"!=() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (
            const Scalar_T & scr,
            const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> bool  [inline]
```

Test for inequality of scalar and multivector.

Definition at line 102 of file clifford_algebra_imp.h.

5.2.2.62 operator%() [1/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar↩
_T,LO,HI,Tune_P>
```

Left contraction.

Definition at line 597 of file framed_multi_imp.h.

References _GLUCAT_HASH_SIZE_T, and crd_of_mult().

5.2.2.63 operator%() [2/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Left contraction.

Definition at line 581 of file matrix_multi_imp.h.

5.2.2.64 operator%() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (
             const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
             const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P>  [inline]
```

Left contraction.

Definition at line 322 of file clifford_algebra_imp.h.

5.2.2.65 operator&() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
             const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
             const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar↩
_T,LO,HI,Tune_P>
```

Inner product.

Definition at line 495 of file framed_multi_imp.h.

References _GLUCAT_HASH_SIZE_T, and crd_of_mult().

5.2.2.66 operator&() [2/4]

```
template<const index_t LO, const index_t HI>
auto glucat::operator& (
             const index_set< LO, HI > & lhs,
             const index_set< LO, HI > & rhs) -> const index_set<LO,HI>  [inline]
```

Set intersection: and.

Definition at line 186 of file index_set_imp.h.

5.2.2.67 operator&() [3/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
             const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
             const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Inner product.

Definition at line 562 of file matrix_multi_imp.h.

5.2.2.68 operator&() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P> [inline]
```

Inner product.

Definition at line 307 of file clifford_algebra_imp.h.

5.2.2.69 operator*() [1/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar↩
_T,LO,HI,Tune_P>
```

Geometric product.

Definition at line 374 of file framed_multi_imp.h.

References _GLUCAT_HASH_SIZE_T.

5.2.2.70 operator*() [2/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar↩
_T,LO,HI,Tune_P> [inline]
```

Geometric product.

Definition at line 502 of file matrix_multi_imp.h.

References glucat::numeric_traits< Scalar_T >::NaN(), and reframe().

5.2.2.71 operator*() [3/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P> [inline]
```

Geometric product.

Definition at line 277 of file clifford_algebra_imp.h.

5.2.2.72 operator*() [4/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Product of multivector and scalar.

Definition at line 251 of file clifford_algebra_imp.h.

5.2.2.73 operator*() [5/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
            const Scalar_T & scr,
            const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Product of scalar and multivector.

Definition at line 262 of file clifford_algebra_imp.h.

5.2.2.74 operator*() [6/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI>
auto glucat::operator* (
            const std::pair< const index_set< LO, HI >, Scalar_T > & lhs,
            const std::pair< const index_set< LO, HI >, Scalar_T > & rhs) -> const std↩
::pair<const index_set<LO,HI>, Scalar_T>  [inline]
```

Product of terms.

Definition at line 1717 of file framed_multi_imp.h.

References crd_of_mult().

5.2.2.75 operator+() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P>  [inline]
```

Geometric sum.

Definition at line 206 of file clifford_algebra_imp.h.

5.2.2.76 operator+() [2/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Geometric sum of multivector and scalar.

Definition at line 181 of file clifford_algebra_imp.h.

5.2.2.77 operator+() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
            const Scalar_T & scr,
            const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Geometric sum of scalar and multivector.

Definition at line 192 of file clifford_algebra_imp.h.

5.2.2.78 operator-() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P>  [inline]
```

Geometric difference.

Definition at line 240 of file clifford_algebra_imp.h.

5.2.2.79 operator-() [2/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Geometric difference of multivector and scalar.

Definition at line 217 of file clifford_algebra_imp.h.

5.2.2.80 operator-() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (
            const Scalar_T & scr,
            const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Geometric difference of scalar and multivector.

Definition at line 228 of file clifford_algebra_imp.h.

5.2.2.81 operator/() [1/5]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Geometric quotient.

Definition at line 734 of file framed_multi_imp.h.

5.2.2.82 operator/() [2/5]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar↩
_T,LO,HI,Tune_P>
```

Geometric quotient.

Definition at line 614 of file matrix_multi_imp.h.

References glucat::matrix::isnan(), and reframe().

5.2.2.83 operator/() [3/5]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P>  [inline]
```

Geometric quotient.

Definition at line 374 of file clifford_algebra_imp.h.

5.2.2.84 operator/() [4/5]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Quotient of multivector and scalar.

Definition at line 348 of file clifford_algebra_imp.h.

5.2.2.85 operator/() [5/5]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (
            const Scalar_T & scr,
            const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Quotient of scalar and multivector.

Definition at line 359 of file clifford_algebra_imp.h.

5.2.2.86 operator<<() [1/5]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator<< (
            std::ostream & os,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::ostream&
```

Write multivector to output.

Definition at line 1148 of file framed_multi_imp.h.

References scalar(), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated().

5.2.2.87 operator<<() [2/5]

```
const index_t HI auto glucat::operator<< (
            std::ostream & os,
            const index_set< LO, HI > & ist) -> std::ostream &
```

5.2.2.88 operator<<() [3/5]

```
template<const index_t LO, const index_t HI>
auto glucat::operator<< (
            std::ostream & os,
            const index_set< LO, HI > & ist) -> std::ostream&
```

Write out index set.

Definition at line 611 of file index_set_imp.h.

5.2.2.89 operator<<() [4/5]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator<< (
            std::ostream & os,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::ostream&  [inline]
```

Write multivector to output.

Definition at line 955 of file matrix_multi_imp.h.

5.2.2.90 operator<<() [5/5]

```
template<typename Scalar_T , const index_t LO, const index_t HI>
auto glucat::operator<< (
            std::ostream & os,
            const std::pair< const index_set< LO, HI >, Scalar_T > & term) -> std::ostream&
```

Write term to output.

Definition at line 1209 of file framed_multi_imp.h.

References glucat::numeric_traits< Scalar_T >::to_double().

5.2.2.91 operator>>() [1/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator>> (
            std::istream & s,
            framed_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::istream&
```

Read multivector from input.

Definition at line 1248 of file framed_multi_imp.h.

5.2.2.92 operator>>() [2/3]

```
template<const index_t LO, const index_t HI>
auto glucat::operator>> (
            std::istream & s,
            index_set< LO, HI > & ist) -> std::istream&
```

Read in index set.

Definition at line 634 of file index_set_imp.h.

5.2.2.93 operator>>() [3/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator>> (
            std::istream & s,
            matrix_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::istream&  [inline]
```

Read multivector from input.

Definition at line 966 of file matrix_multi_imp.h.

5.2.2.94 operator^() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar←
_T,LO,HI,Tune_P>
```

Outer product.

Definition at line 416 of file framed_multi_imp.h.

References _GLUCAT_HASH_SIZE_T, and crd_of_mult().

5.2.2.95 operator^() [2/4]

```
template<const index_t LO, const index_t HI>
auto glucat::operator^ (
            const index_set< LO, HI > & lhs,
            const index_set< LO, HI > & rhs) -> const index_set<LO,HI>  [inline]
```

Symmetric set difference: exclusive or.

Definition at line 161 of file index_set_imp.h.

5.2.2.96 operator^() [3/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar←
_T,LO,HI,Tune_P>  [inline]
```

Outer product.

Definition at line 543 of file matrix_multi_imp.h.

5.2.2.97 operator^() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_←
T,LO,HI,Tune_P>  [inline]
```

Outer product.

Definition at line 292 of file clifford_algebra_imp.h.

5.2.2.98 operator"|() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator| (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar←
_T,LO,HI,Tune_P>  [inline]
```

Transformation via twisted adjoint action.

Definition at line 760 of file framed_multi_imp.h.

5.2.2.99 operator"|() [2/4]

```
template<const index_t LO, const index_t HI>
auto glucat::operator| (
            const index_set< LO, HI > & lhs,
            const index_set< LO, HI > & rhs) -> const index_set<LO,HI>  [inline]
```

Set union: or.

Definition at line 211 of file index_set_imp.h.

5.2.2.100 operator"|() [3/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator| (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar←
_T,LO,HI,Tune_P>  [inline]
```

Transformation via twisted adjoint action.

Definition at line 717 of file matrix_multi_imp.h.

5.2.2.101 operator"|() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator| (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_←
T,LO,HI,Tune_P>  [inline]
```

Transformation via twisted adjoint action.

Definition at line 389 of file clifford_algebra_imp.h.

5.2.2.102 outer_pow()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::outer_pow (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Outer product power of multivector.

Definition at line 470 of file clifford_algebra_imp.h.

5.2.2.103 pade_approx()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P , const size←
_t Size>
static auto glucat::pade_approx (
            const std::array< Scalar_T, Size > & numer,
            const std::array< Scalar_T, Size > & denom,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & X) -> const matrix_multi<Scalar←
_T,LO,HI,Tune_P>   [inline], [static]
```

Pade' approximation.

Definition at line 1245 of file matrix_multi_imp.h.

Referenced by matrix_sqrt(), and pade_log().

5.2.2.104 pade_log()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static auto glucat::pade_log (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix_multi<Scalar←
_T,LO,HI,Tune_P>   [static]
```

Pade' approximation of log.

Definition at line 1900 of file matrix_multi_imp.h.

References glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), and pade_approx().

Referenced by cascade_log().

5.2.2.105 pos_mod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::pos_mod (
            LHS_T lhs,
            RHS_T rhs) -> LHS_T   [inline]
```

Modulo function which works reliably for lhs < 0.

Definition at line 117 of file global.h.

Referenced by complexifier(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi(), glucat::framed_multi< Scalar_
glucat::gen::generator_table< Matrix_T >::gen_vector(), offset_level(), and glucat::gen::generator_table< Matrix_T >::operator()().

5.2.2.106 pow() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pow (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↩
T,LO,HI,Tune_P>  [inline]
```

Multivector power of multivector.

Definition at line 446 of file clifford_algebra_imp.h.

References exp(), and log().

5.2.2.107 pow() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pow (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Integer power of multivector.

Definition at line 407 of file clifford_algebra_imp.h.

Referenced by matrix_sqrt().

5.2.2.108 pure()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pure (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Pure part.

Definition at line 505 of file clifford_algebra_imp.h.

5.2.2.109 quad()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::quad (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Scalar_T quadratic form == (rev(x)*x)(0)

Definition at line 561 of file clifford_algebra_imp.h.

5.2.2.110 real()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::real (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Real part: synonym for scalar part.

Definition at line 486 of file clifford_algebra_imp.h.

5.2.2.111 reframe()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::reframe (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs,
            matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs_reframed,
            matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs_reframed) -> const index_set<LO,HI>
[inline]
```

Find a common frame for operands of a binary operator.

Definition at line 345 of file matrix_multi_imp.h.

Referenced by operator*(), and operator/().

5.2.2.112 reverse()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::reverse (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Reversion, eg. {1}*{2} -> {2}*{1}.

Definition at line 545 of file clifford_algebra_imp.h.

5.2.2.113 scalar()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::scalar (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T  [inline]
```

Scalar part.

Definition at line 478 of file clifford_algebra_imp.h.

Referenced by exp(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast(), and operator<<().

5.2.2.114 sign_of_square()

```
auto glucat::sign_of_square (
            index_t j) -> int  [inline]
```

Square of generator {j}.

Square of generator index j.

Definition at line 963 of file index_set_imp.h.

5.2.2.115 sin() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Sine of multivector.

Definition at line 979 of file clifford_algebra_imp.h.

References complexifier(), and sin().

5.2.2.116 sin() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Sine of multivector with specified complexifier.

Definition at line 956 of file clifford_algebra_imp.h.

References check_complex(), and exp().

Referenced by sin(), and tan().

5.2.2.117 sinh()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sinh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Hyperbolic sine of multivector.

Definition at line 911 of file clifford_algebra_imp.h.

References exp().

Referenced by tanh().

5.2.2.118 sqrt() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & val,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & i,
            bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line 1727 of file framed_multi_imp.h.

References check_complex(), and sqrt().

5.2.2.119 sqrt() [2/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & i,
            bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line 1667 of file matrix_multi_imp.h.

References check_complex(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), and matrix_sqrt().

5.2.2.120 sqrt() [3/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Square root of multivector.

Definition at line 683 of file clifford_algebra_imp.h.

References complexifier(), and sqrt().

5.2.2.121 sqrt() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Square root of multivector with specified complexifier.

Definition at line 675 of file clifford_algebra_imp.h.

References sqrt().

Referenced by acosh(), asinh(), sqrt(), sqrt(), and sqrt().

5.2.2.122 star() [1/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (
            const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T
```

Hestenes scalar product.

Definition at line 684 of file framed_multi_imp.h.

5.2.2.123 star() [2/3]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
            const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T  [inline]
```

Hestenes scalar product.

Definition at line 600 of file matrix_multi_imp.h.

5.2.2.124 star() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (
            const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
            const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T  [inline]
```

Hestenes scalar product.

Definition at line 337 of file clifford_algebra_imp.h.

References star().

Referenced by star().

5.2.2.125 tan() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::tan (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↩
_T,LO,HI,Tune_P>  [inline]
```

Tangent of multivector.

Definition at line 1079 of file clifford_algebra_imp.h.

References complexifier(), and tan().

5.2.2.126 tan() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::tan (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val,
            const Multivector< Scalar_T, LO, HI, Tune_P > & i,
            const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>  [inline]
```

Tangent of multivector with specified complexifier.

Definition at line 1060 of file clifford_algebra_imp.h.

References check_complex(), cos(), and sin().

Referenced by tan().

5.2.2.127 tanh()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::tanh (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>  [inline]
```

Hyperbolic tangent of multivector.

Definition at line 1016 of file clifford_algebra_imp.h.

References cosh(), and sinh().

5.2.2.128 to_demote()

```
template<typename Scalar_T >
auto glucat::to_demote (
            const Scalar_T & val) -> typename numeric_traits<Scalar_T>::demoted::type   [inline]
```

Cast to demote.

Definition at line 135 of file scalar_imp.h.

References glucat::numeric_traits< Scalar_T >::to_scalar_t().

5.2.2.129 to_promote()

```
template<typename Scalar_T >
auto glucat::to_promote (
            const Scalar_T & val) -> typename numeric_traits<Scalar_T>::promoted::type   [inline]
```

Cast to promote.

Definition at line 125 of file scalar_imp.h.

References glucat::numeric_traits< Scalar_T >::to_scalar_t().

5.2.2.130 try_catch() [1/2]

```
int glucat::try_catch (
            intfn f)
```

Exception catching for functions returning int.

Definition at line 49 of file try_catch.h.

Referenced by glucat::control_t::call(), and glucat::control_t::call().

5.2.2.131 try_catch() [2/2]

```
int glucat::try_catch (
            intintfn f,
            int arg)
```

Exception catching for functions of int returning int.

Definition at line 64 of file try_catch.h.

5.2.2.132 vector_part()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::vector_part (
            const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const std::vector<Scalar↩
_T>  [inline]
```

Vector part of multivector, as a vector_t with respect to frame()

Definition at line 529 of file clifford_algebra_imp.h.

5.2.3 Variable Documentation

5.2.3.1 BITS_PER_SET_VALUE

```
const index_t glucat::BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits
```

Number of bits in set_value_t.

Definition at line 103 of file global.h.

5.2.3.2 DEFAULT_HI

```
const index_t glucat::DEFAULT_HI = index_t(BITS_PER_SET_VALUE / 2)
```

Default highest index in an index set.

Definition at line 111 of file global.h.

5.2.3.3 l_ln2

```
const long double glucat::l_ln2 = 0.693147180559945309417232121458176 6L  [static]
```

Definition at line 44 of file long_double.h.

Referenced by glucat::numeric_traits< Scalar_T >::ln_2().

5.2.3.4 l_pi

```
const long double glucat::l_pi = 3.14159265358979323846264338327950 29L  [static]
```

Definition at line 43 of file long_double.h.

Referenced by glucat::numeric_traits< Scalar_T >::pi().

5.2.3.5 MS_PER_S

```
const double glucat::MS_PER_S = 1000.0
```

Timing constant: deprecated here - moved to test/timing.h.

Definition at line 83 of file global.h.

5.2.3.6 Tuning_Fast_Basis_Max_Count

```
const unsigned int glucat::Tuning_Fast_Basis_Max_Count = 1
```

Definition at line 92 of file tuning.h.

5.2.3.7 Tuning_Fast_CR_Sqrt_Max_Steps

```
const unsigned int glucat::Tuning_Fast_CR_Sqrt_Max_Steps = 256
```

Definition at line 88 of file tuning.h.

5.2.3.8 Tuning_Fast_DB_Sqrt_Max_Steps

```
const unsigned int glucat::Tuning_Fast_DB_Sqrt_Max_Steps = 256
```

Definition at line 89 of file tuning.h.

5.2.3.9 Tuning_Fast_Div_Max_Steps

```
const unsigned int glucat::Tuning_Fast_Div_Max_Steps = 0
```

Definition at line 87 of file tuning.h.

5.2.3.10 Tuning_Fast_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Fast_Fast_Size_Threshold = 0
```

Definition at line 93 of file tuning.h.

5.2.3.11 Tuning_Fast_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Fast_Inv_Fast_Dim_Threshold = 0
```

Definition at line 94 of file tuning.h.

5.2.3.12 Tuning_Fast_Log_Max_Inner_Steps

```
const unsigned int glucat::Tuning_Fast_Log_Max_Inner_Steps = 8
```

Definition at line 91 of file tuning.h.

5.2.3.13 Tuning_Fast_Log_Max_Outer_Steps

```
const unsigned int glucat::Tuning_Fast_Log_Max_Outer_Steps = 16
```

Definition at line 90 of file tuning.h.

5.2.3.14 Tuning_Fast_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Fast_Mult_Matrix_Threshold = 0
```

Definition at line 86 of file tuning.h.

5.2.3.15 Tuning_Fast_Products_Size_Threshold

```
const unsigned int glucat::Tuning_Fast_Products_Size_Threshold = 0
```

Definition at line 95 of file tuning.h.

5.2.3.16 Tuning_Int_Digits

```
const unsigned int glucat::Tuning_Int_Digits = std::numeric_limits<int>::digits
```

Definition at line 36 of file tuning.h.

5.2.3.17 Tuning_Max_Threshold

```
const unsigned int glucat::Tuning_Max_Threshold = 1 << Tuning_Int_Digits
```

Definition at line 37 of file tuning.h.

5.2.3.18 Tuning_Naive_Basis_Max_Count

```
const unsigned int glucat::Tuning_Naive_Basis_Max_Count = Tuning_Max_Threshold
```

Definition at line 65 of file tuning.h.

5.2.3.19 Tuning_Naive_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Naive_Fast_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 66 of file tuning.h.

5.2.3.20 Tuning_Naive_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Naive_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
```

Definition at line 67 of file tuning.h.

5.2.3.21 Tuning_Naive_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Naive_Mult_Matrix_Threshold = 0
```

Definition at line 64 of file tuning.h.

5.2.3.22 Tuning_Slow_Basis_Max_Count

```
const unsigned int glucat::Tuning_Slow_Basis_Max_Count = 0
```

Definition at line 42 of file tuning.h.

5.2.3.23 Tuning_Slow_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Slow_Fast_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 43 of file tuning.h.

5.2.3.24 Tuning_Slow_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Slow_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
```

Definition at line 44 of file tuning.h.

5.2.3.25 Tuning_Slow_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Slow_Mult_Matrix_Threshold = Tuning_Max_Threshold
```

Definition at line 41 of file tuning.h.

5.2.3.26 Tuning_Slow_Products_Size_Threshold

```
const unsigned int glucat::Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 45 of file tuning.h.

5.3 glucat::gen Namespace Reference

Classes

- class generator_table

  Table of generators for specific signatures.

Typedefs

- using signature_t = std::pair<index_t, index_t>

  A signature is a pair of indices, p, q, with p == frame.max(), q == -frame.min()

Variables

- static const std::array< index_t, 8 > offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1}

  Offsets between the current signature and that of the real superalgebra.

5.3.1 Typedef Documentation

5.3.1.1 signature_t

```
using glucat::gen::signature_t = std::pair<index_t, index_t>
```

A signature is a pair of indices, p, q, with p == frame.max(), q == -frame.min()

Definition at line 48 of file generation.h.

5.3.2 Variable Documentation

5.3.2.1 offset_to_super

```
const std::array<index_t, 8> glucat::gen::offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1}  [static]
```

Offsets between the current signature and that of the real superalgebra.

Definition at line 86 of file generation.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi(), glucat::framed_multi< Scalar_T, LO, HI, Tune
and glucat::gen::generator_table< Matrix_T >::operator()().

5.4 glucat::matrix Namespace Reference

Classes

- struct eig_genus

  Structure containing classification of eigenvalues.

Typedefs

- using eig_case_t

  Classification of eigenvalues of a matrix.

Functions

- template<typename LHS_T , typename RHS_T >
  auto kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Kronecker tensor product of matrices - as per Matlab kron.
- template<typename LHS_T , typename RHS_T >
  auto mono_kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Sparse Kronecker tensor product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
  auto nork (const LHS_T &lhs, const RHS_T &rhs, const bool mono=true) -> const RHS_T

  Left inverse of Kronecker product.
- template<typename LHS_T , typename RHS_T >
  auto signed_perm_nork (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Left inverse of Kronecker product where lhs is a signed permutation matrix.
- template<typename Matrix_T >
  auto nnz (const Matrix_T &m) -> typename Matrix_T::size_type

  Number of non-zeros.
- template<typename Matrix_T >
  auto isinf (const Matrix_T &m) -> bool

  Infinite.
- template<typename Matrix_T >
  auto isnan (const Matrix_T &m) -> bool

  Not a Number.
- template<typename Matrix_T >
  auto unit (const typename Matrix_T::size_type n) -> const Matrix_T

  Unit matrix - as per Matlab eye.
- template<typename LHS_T , typename RHS_T >
  auto mono_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
  auto sparse_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of sparse matrices.
- template<typename LHS_T , typename RHS_T >
  auto prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of matrices.

- template<typename Scalar_T , typename LHS_T , typename RHS_T >
  auto inner (const LHS_T &lhs, const RHS_T &rhs) -> Scalar_T

  *Inner product: sum(x(i,j)*y(i,j))/x.nrows()*
- template<typename Matrix_T >
  auto norm_frob2 (const Matrix_T &val) -> typename Matrix_T::value_type

  Square of Frobenius norm.
- template<typename Matrix_T >
  auto trace (const Matrix_T &val) -> typename Matrix_T::value_type

  Matrix trace.
- template<typename Matrix_T >
  auto eigenvalues (const Matrix_T &val) -> std::vector< std::complex< double > >

  Eigenvalues of a matrix.
- template<typename Matrix_T >
  auto classify_eigenvalues (const Matrix_T &val) -> eig_genus< Matrix_T >

  Classify the eigenvalues of a matrix.
- template<typename LHS_T , typename RHS_T >
  void nork_range (RHS_T &result, const typename LHS_T::const_iterator2 lhs_it2, const RHS_T &rhs, const typename RHS_T::size_type res_s1, const typename RHS_T::size_type res_s2)

  Utility routine for nork: calculate result for a range of indices.
- template<typename Matrix_T >
  static auto to_lapack (const Matrix_T &val) -> ublas::matrix< double, ublas::column_major >

  Convert matrix to LAPACK format.

5.4.1 Typedef Documentation

5.4.1.1 eig_case_t

```
using glucat::matrix::eig_case_t
```

Initial value:
```
enum {
      safe_eigs,
      neg_real_eigs,
      both_eigs}
```

Classification of eigenvalues of a matrix.

Definition at line 133 of file matrix.h.

5.4.2 Function Documentation

5.4.2.1 classify_eigenvalues()

```
template<typename Matrix_T >
auto glucat::matrix::classify_eigenvalues (
              const Matrix_T & val) -> eig_genus<Matrix_T>
```

Classify the eigenvalues of a matrix.

Definition at line 548 of file matrix_imp.h.

References eigenvalues(), epsilon, glucat::matrix::eig_genus< Matrix_T >::m_eig_case, glucat::matrix::eig_genus< Matrix_T >::m_
glucat::matrix::eig_genus< Matrix_T >::m_safe_arg, and glucat::numeric_traits< Scalar_T >::pi().

Referenced by glucat::matrix_log(), and glucat::matrix_sqrt().

5.4.2.2 eigenvalues()

```
template<typename Matrix_T >
auto glucat::matrix::eigenvalues (
            const Matrix_T & val) -> std::vector< std::complex<double> >
```

Eigenvalues of a matrix.

Definition at line 500 of file matrix_imp.h.

References to_lapack().

Referenced by classify_eigenvalues().

5.4.2.3 inner()

```
template<typename Scalar_T , typename LHS_T , typename RHS_T >
auto glucat::matrix::inner (
            const LHS_T & lhs,
            const RHS_T & rhs) -> Scalar_T
```

Inner product: sum(x(i,j)∗y(i,j))/x.nrows()

Inner product: sum(lhs(i,j)∗rhs(i,j))/lhs.nrows()

Definition at line 373 of file matrix_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

5.4.2.4 isinf()

```
template<typename Matrix_T >
auto glucat::matrix::isinf (
            const Matrix_T & m) -> bool
```

Infinite.

Definition at line 275 of file matrix_imp.h.

5.4.2.5 isnan()

```
template<typename Matrix_T >
auto glucat::matrix::isnan (
            const Matrix_T & m) -> bool
```

Not a Number.

Definition at line 292 of file matrix_imp.h.

Referenced by glucat::operator/().

5.4.2.6 kron()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::kron (
            const LHS_T & lhs,
            const RHS_T & rhs) -> const RHS_T
```

Kronecker tensor product of matrices - as per Matlab kron.

Definition at line 83 of file matrix_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast().

5.4.2.7 mono_kron()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::mono_kron (
            const LHS_T & lhs,
            const RHS_T & rhs) -> const RHS_T
```

Sparse Kronecker tensor product of monomial matrices.

Definition at line 119 of file matrix_imp.h.

Referenced by glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1().

5.4.2.8 mono_prod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::mono_prod (
            const ublas::matrix_expression< LHS_T > & lhs,
            const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↩
_type
```

Product of monomial matrices.

Definition at line 320 of file matrix_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element(), glucat::gen::generator_table< Matrix_T >::gen_f
glucat::gen::generator_table< Matrix_T >::gen_from_pp4_qm4(), and glucat::gen::generator_table< Matrix_T >::gen_from_qp1_pm

5.4.2.9 nnz()

```
template<typename Matrix_T >
auto glucat::matrix::nnz (
            const Matrix_T & m) -> typename Matrix_T::size_type
```

Number of non-zeros.

Definition at line 258 of file matrix_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

5.4.2.10 nork()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::nork (
            const LHS_T & lhs,
            const RHS_T & rhs,
            const bool mono = true) -> const RHS_T
```

Left inverse of Kronecker product.

Definition at line 182 of file matrix_imp.h.

References nork_range(), and norm_frob2().

5.4.2.11 nork_range()

```
template<typename LHS_T , typename RHS_T >
void glucat::matrix::nork_range (
            RHS_T & result,
            const typename LHS_T::const_iterator2 lhs_it2,
            const RHS_T & rhs,
            const typename RHS_T::size_type res_s1,
            const typename RHS_T::size_type res_s2)
```

Utility routine for nork: calculate result for a range of indices.

Definition at line 152 of file matrix_imp.h.

References glucat::numeric_traits< Scalar_T >::to_scalar_t().

Referenced by nork(), and signed_perm_nork().

5.4.2.12 norm_frob2()

```
template<typename Matrix_T >
auto glucat::matrix::norm_frob2 (
            const Matrix_T & val) -> typename Matrix_T::value_type
```

Square of Frobenius norm.

Definition at line 395 of file matrix_imp.h.

References glucat::numeric_traits< Scalar_T >::NaN().

Referenced by nork().

5.4.2.13 prod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::prod (
            const ublas::matrix_expression< LHS_T > & lhs,
            const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↵
_type [inline]
```

Product of matrices.

Definition at line 361 of file matrix_imp.h.

5.4.2.14 signed_perm_nork()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::signed_perm_nork (
            const LHS_T & lhs,
            const RHS_T & rhs) -> const RHS_T
```

Left inverse of Kronecker product where lhs is a signed permutation matrix.

Definition at line 228 of file matrix_imp.h.

References nork_range().

Referenced by glucat::fast().

5.4.2.15 sparse_prod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::sparse_prod (
            const ublas::matrix_expression< LHS_T > & lhs,
            const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↩
_type  [inline]
```

Product of sparse matrices.

Definition at line 350 of file matrix_imp.h.

5.4.2.16 to_lapack()

```
template<typename Matrix_T >
static auto glucat::matrix::to_lapack (
            const Matrix_T & val) -> ublas::matrix<double, ublas::column_major>   [static]
```

Convert matrix to LAPACK format.

Definition at line 440 of file matrix_imp.h.

Referenced by eigenvalues().

5.4.2.17 trace()

```
template<typename Matrix_T >
auto glucat::matrix::trace (
            const Matrix_T & val) -> typename Matrix_T::value_type
```

Matrix trace.

Definition at line 416 of file matrix_imp.h.

References glucat::numeric_traits< Scalar_T >::NaN().

5.4.2.18 unit()

```
template<typename Matrix_T >
auto glucat::matrix::unit (
            const typename Matrix_T::size_type n) -> const Matrix_T  [inline]
```

Unit matrix - as per Matlab eye.

Definition at line 310 of file matrix_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element(), glucat::fast(), glucat::framed_multi< Scalar_T, LO glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1(), and glucat::gen::generator_table< Matrix_T >::gen_vector().

5.5 glucat::timing Namespace Reference

Functions

- static double elapsed (clock_t cpu_time)

  Elapsed time in milliseconds.

Variables

- const double MS_PER_SEC = 1000.0

  Timing constant: milliseconds per second.
- const double MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC)

  Timing constant: milliseconds per clock.
- const int EXTRA_TRIALS = 2

  Timing constant: trial expansion factor.

5.5.1 Function Documentation

5.5.1.1 elapsed()

```
static double glucat::timing::elapsed (
            clock_t cpu_time)  [inline], [static]
```

Elapsed time in milliseconds.

Definition at line 51 of file timing.h.

References MS_PER_CLOCK.

5.5.2 Variable Documentation

5.5.2.1 EXTRA_TRIALS

```
const int glucat::timing::EXTRA_TRIALS = 2
```

Timing constant: trial expansion factor.

Definition at line 45 of file timing.h.

5.5.2.2 MS_PER_CLOCK

```
const double glucat::timing::MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC)
```

Timing constant: milliseconds per clock.

Definition at line 42 of file timing.h.

Referenced by elapsed().

5.5.2.3 MS_PER_SEC

```
const double glucat::timing::MS_PER_SEC = 1000.0
```

Timing constant: milliseconds per second.

Definition at line 39 of file timing.h.

5.6 pade Namespace Reference

Classes

- struct pade_log_denom

  Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)
- struct pade_log_denom< dd_real >
- struct pade_log_denom< float >
- struct pade_log_denom< long double >
- struct pade_log_denom< qd_real >
- struct pade_log_numer

  Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)
- struct pade_log_numer< dd_real >
- struct pade_log_numer< float >
- struct pade_log_numer< long double >
- struct pade_log_numer< qd_real >
- struct pade_sqrt_denom

  Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)
- struct pade_sqrt_denom< dd_real >
- struct pade_sqrt_denom< float >
- struct pade_sqrt_denom< long double >
- struct pade_sqrt_denom< qd_real >
- struct pade_sqrt_numer

  Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)
- struct pade_sqrt_numer< dd_real >
- struct pade_sqrt_numer< float >
- struct pade_sqrt_numer< long double >
- struct pade_sqrt_numer< qd_real >

5.7 PyClical Namespace Reference

Classes

- class clifford
- class index_set

Functions

- index_set_hidden_doctests ()
- clifford_hidden_doctests ()
- e (obj)
- istpq (p, q)
- _test ()

Variables

- __version__ = str(glucat_package_version,'utf-8')
- lhs
- rhs
- threshold = error_squared_tol(rhs) if threshold is None else threshold
- None
- tol = error_squared_tol(rhs) if tol is None else tol
- obj
- i
- ixt
- fill
- scalar_epsilon = epsilon
- float pi = atan(clifford(1.0)) ∗ 4.0
- float tau = atan(clifford(1.0)) ∗ 8.0
- cl = clifford
- ist = index_set
- ninf3 = e(4) + e(-1)
- nbar3 = e(4) - e(-1)

5.7.1 Function Documentation

5.7.1.1 _test()

```
PyClical._test ()  [protected]
```

Definition at line 1962 of file PyClical.pyx.

References _test().

Referenced by _test().

5.7.1.2 clifford_hidden_doctests()

```
PyClical.clifford_hidden_doctests ()
```

```
Tests for functions that Doctest cannot see.

For clifford.__cinit__: Construct an object of type clifford.

>>> print(clifford(2))
2
>>> print(clifford(2.0))
2
>>> print(clifford(1.0e-1))
0.1
>>> print(clifford("2"))
2
>>> print(clifford("2{1,2,3}"))
2{1,2,3}
>>> print(clifford(clifford("2{1,2,3}")))
2{1,2,3}
>>> print(clifford("-{1}"))
-{1}
>>> print(clifford(2,index_set({1,2})))
2{1,2}
>>> print(clifford([2,3],index_set({1,2})))
2{1}+3{2}
>>> print(clifford([1,2]))
Traceback (most recent call last):
  ...
TypeError: Cannot initialize clifford object from <class 'list'>.
>>> print(clifford(None))
Traceback (most recent call last):
  ...
TypeError: Cannot initialize clifford object from <class 'NoneType'>.
>>> print(clifford(None,[1,2]))
Traceback (most recent call last):
  ...
TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).
>>> print(clifford([1,2],[1,2]))
Traceback (most recent call last):
  ...
TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).
>>> print(clifford(""))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string ''.
>>> print(clifford("{"))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string '{'.
>>> print(clifford("{1"))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string '{1'.
>>> print(clifford("+"))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string '+'.
>>> print(clifford("-"))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string '-'.
>>> print(clifford("{1}+"))
Traceback (most recent call last):
  ...
ValueError: Cannot initialize clifford object from invalid string '{1}+'.

For clifford.__richcmp__: Compare objects of type clifford.

>>> clifford("{1}") == clifford("1{1}")
True
```

```
>>> clifford("{1}") != clifford("1.0{1}")
False
>>> clifford("{1}") != clifford("1.0")
True
>>> clifford("{1,2}") == None
False
>>> clifford("{1,2}") != None
True
>>> None == clifford("{1,2}")
False
>>> None != clifford("{1,2}")
True
```

Definition at line 1253 of file PyClical.pyx.

5.7.1.3 e()

```
PyClical.e (
              obj)
```

```
Abbreviation for clifford(index_set(obj)).

>>> print(e(1))
{1}
>>> print(e(-1))
{-1}
>>> print(e(0))
1
```

Definition at line 1936 of file PyClical.pyx.

5.7.1.4 index_set_hidden_doctests()

```
PyClical.index_set_hidden_doctests ()
```

```
Tests for functions that Doctest cannot see.

For index_set.__cinit__: Construct index_set.

>>> print(index_set(1))
{1}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set(index_set({1,2})))
{1,2}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set(""))
{}
>>> print(index_set("{"))
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{'.
>>> print(index_set("{1"))
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{1'.
```

```
>>> print(index_set("{1,2,100}"))
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{1,2,100}'.
>>> print(index_set({1,2,100}))
Traceback (most recent call last):
...
IndexError: Cannot initialize index_set object from invalid {1, 2, 100}.
>>> print(index_set([1,2]))
Traceback (most recent call last):
...
TypeError: Cannot initialize index_set object from <class 'list'>.

For index_set.__richcmp__: Compare two objects of class index_set.

>>> index_set(1) == index_set({1})
True
>>> index_set({1}) != index_set({1})
False
>>> index_set({1}) != index_set({2})
True
>>> index_set({1}) == index_set({2})
False
>>> index_set({1}) < index_set({2})
True
>>> index_set({1}) <= index_set({2})
True
>>> index_set({1}) > index_set({2})
False
>>> index_set({1}) >= index_set({2})
False
>>> None == index_set({1,2})
False
>>> None != index_set({1,2})
True
>>> None < index_set({1,2})
False
>>> None <= index_set({1,2})
False
>>> None > index_set({1,2})
False
>>> None >= index_set({1,2})
False
>>> index_set({1,2}) == None
False
>>> index_set({1,2}) != None
True
>>> index_set({1,2}) < None
False
>>> index_set({1,2}) <= None
False
>>> index_set({1,2}) > None
False
>>> index_set({1,2}) >= None
False
```

Definition at line 406 of file PyClical.pyx.

5.7.1.5 istpq()

```
PyClical.istpq (
              p,
              q)
```

```
Abbreviation for index_set({-q,...p}).

>>> print(istpq(2,3))
{-3,-2,-1,1,2}
```

Definition at line 1949 of file PyClical.pyx.

5.7.2 Variable Documentation

5.7.2.1 __version__

```
PyClical.__version__ = str(glucat_package_version,'utf-8')  [private]
```

Definition at line 35 of file PyClical.pyx.

5.7.2.2 cl

```
PyClical.cl = clifford
```

Definition at line 1910 of file PyClical.pyx.

5.7.2.3 fill

```
PyClical.fill
```

Definition at line 1864 of file PyClical.pyx.

5.7.2.4 i

```
PyClical.i
```

Definition at line 1591 of file PyClical.pyx.

5.7.2.5 ist

```
PyClical.ist = index_set
```

Definition at line 1928 of file PyClical.pyx.

5.7.2.6 ixt

```
PyClical.ixt
```

Definition at line 1864 of file PyClical.pyx.

5.7.2.7 lhs

```
PyClical.lhs
```

Definition at line 1359 of file PyClical.pyx.

5.7.2.8 nbar3

```
PyClical.nbar3 = e(4) - e(-1)
```

Definition at line 1959 of file PyClical.pyx.

5.7.2.9 ninf3

```
PyClical.ninf3 = e(4) + e(-1)
```

Definition at line 1958 of file PyClical.pyx.

5.7.2.10 None

```
PyClical.None
```

Definition at line 1359 of file PyClical.pyx.

5.7.2.11 obj

```
PyClical.obj
```

Definition at line 1591 of file PyClical.pyx.

5.7.2.12 pi

```
float PyClical.pi = atan(clifford(1.0)) * 4.0
```

Definition at line 1907 of file PyClical.pyx.

5.7.2.13 rhs

```
PyClical.rhs
```

Definition at line 1359 of file PyClical.pyx.

5.7.2.14 scalar_epsilon

```
PyClical.scalar_epsilon = epsilon
```

Definition at line 1905 of file PyClical.pyx.

5.7.2.15 tau

```
float PyClical.tau = atan(clifford(1.0)) * 8.0
```

Definition at line 1908 of file PyClical.pyx.

5.7.2.16 threshold

```
PyClical.threshold = error_squared_tol(rhs) if threshold is None else threshold
```

Definition at line 1359 of file PyClical.pyx.

5.7.2.17 tol

```
PyClical.tol = error_squared_tol(rhs) if tol is None else tol
```

Definition at line 1359 of file PyClical.pyx.

5.8 std Namespace Reference

Classes

- struct numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >

    Numeric limits for framed_multi inherit limits for the corresponding scalar type.
- struct numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >

    Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

Chapter 6

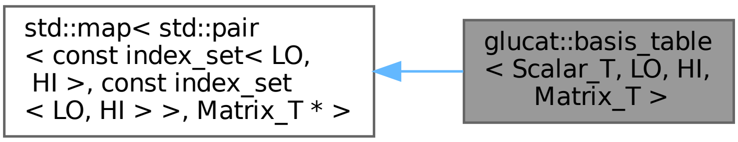
Class Documentation

6.1 glucat::basis_table< Scalar_T, LO, HI, Matrix_T > Class Template Reference

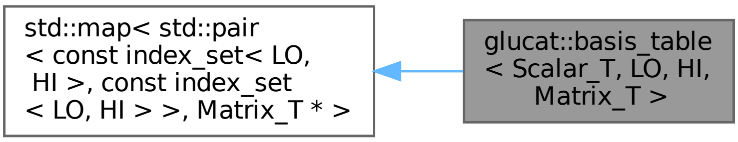
Table of basis elements used as a cache by basis_element()

`#include <matrix_multi_imp.h>`

Inheritance diagram for glucat::basis_table< Scalar_T, LO, HI, Matrix_T >:



Collaboration diagram for glucat::basis_table< Scalar_T, LO, HI, Matrix_T >:



Public Member Functions

- basis_table (const basis_table &)=delete
- auto operator= (const basis_table &) -> basis_table &=delete

Static Public Member Functions

- static auto basis () -> basis_table &

  Single instance of basis table.

Private Member Functions

- basis_table ()=default
- ∼basis_table ()=default

Friends

- class friend_for_private_destructor

6.1.1 Detailed Description

template<typename Scalar_T, const index_t LO, const index_t HI, typename Matrix_T>
class glucat::basis_table< Scalar_T, LO, HI, Matrix_T >

Table of basis elements used as a cache by basis_element()

Definition at line 1162 of file matrix_multi_imp.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 basis_table() [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis_table ()  [private], [default]
```

6.1.2.2 ∼basis_table()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::∼basis_table ()  [private], [default]
```

6.1.2.3 basis_table() [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis_table (
            const basis_table< Scalar_T, LO, HI, Matrix_T > & )  [delete]
```

6.1.3 Member Function Documentation

6.1.3.1 basis()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
static auto glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis () -> basis_table&   [inline],
[static]
```

Single instance of basis table.

Definition at line 1168 of file matrix_multi_imp.h.

6.1.3.2 operator=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
auto glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::operator= (
            const basis_table< Scalar_T, LO, HI, Matrix_T > & ) -> basis_table &=delete
[delete]
```

6.1.4 Friends And Related Symbol Documentation

6.1.4.1 friend_for_private_destructor

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
friend class friend_for_private_destructor  [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE http://doc.ece.uci.edu

Definition at line 1173 of file matrix_multi_imp.h.

The documentation for this class was generated from the following file:

- glucat/matrix_multi_imp.h

6.2 glucat::bool_to_type< truth_value > Class Template Reference

Bool to type.

```
#include <global.h>
```

Private Types

- enum { value = truth_value }

6.2.1 Detailed Description

template<bool truth_value>
class glucat::bool_to_type< truth_value >

Bool to type.

Definition at line 69 of file global.h.

6.2.2 Member Enumeration Documentation

6.2.2.1 anonymous enum

```
template<bool truth_value>
anonymous enum  [private]
```

Enumerator

value	

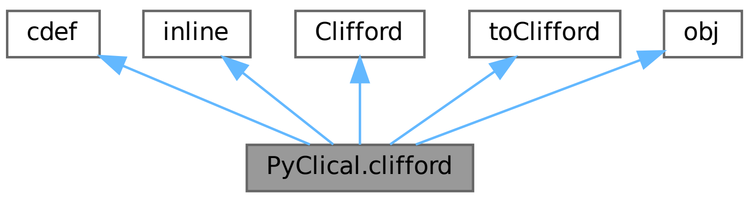
Definition at line 72 of file global.h.

The documentation for this class was generated from the following file:

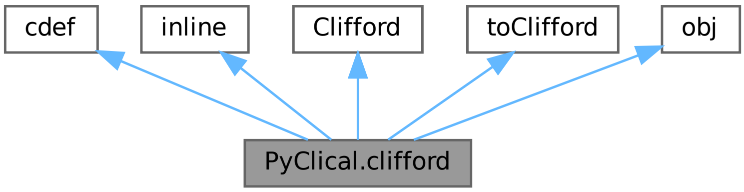
- glucat/global.h

6.3 PyClical.clifford Class Reference

Inheritance diagram for PyClical.clifford:



Collaboration diagram for PyClical.clifford:



Public Member Functions

- __cinit__ (self, other=0, ixt=None)
- __dealloc__ (self)
- __contains__ (self, x)
- __iter__ (self)
- reframe (self, ixt)
- __richcmp__ (lhs, rhs, int, op)
- __getitem__ (self, ixt)
- __neg__ (self)
- __pos__ (self)
- __add__ (lhs, rhs)
- __iadd__ (self, rhs)
- __sub__ (lhs, rhs)
- __isub__ (self, rhs)
- __mul__ (lhs, rhs)
- __imul__ (self, rhs)
- __mod__ (lhs, rhs)
- __imod__ (self, rhs)
- __and__ (lhs, rhs)
- __iand__ (self, rhs)
- __xor__ (lhs, rhs)
- __ixor__ (self, rhs)
- __truediv__ (lhs, rhs)
- __idiv__ (self, rhs)
- inv (self)
- __or__ (lhs, rhs)
- __ior__ (self, rhs)
- __pow__ (self, m, dummy)
- pow (self, m)
- outer_pow (self, m)
- __call__ (self, grade)
- scalar (self)
- pure (self)
- even (self)
- odd (self)
- vector_part (self, frm=None)
- involute (self)
- reverse (self)

- conj (self)
- quad (self)
- norm (self)
- abs (self)
- max_abs (self)
- truncated (self, limit)
- isinf (self)
- isnan (self)
- frame (self)
- __repr__ (self)
- __str__ (self)

Public Attributes

- instance = new Clifford((<clifford>other).unwrap())

6.3.1 Detailed Description

```
Python class clifford wraps C++ class Clifford.
```

Definition at line 537 of file PyClical.pyx.

6.3.2 Member Function Documentation

6.3.2.1 __add__()

```
PyClical.clifford.__add__ (
              lhs,
              rhs)
```

```
Geometric sum.

>>> print(clifford(1) + clifford("{2}"))
1+{2}
>>> print(clifford("{1}") + clifford("{2}"))
{1}+{2}
```

Definition at line 740 of file PyClical.pyx.

6.3.2.2 __and__()

```
PyClical.clifford.__and__ (
              lhs,
              rhs)
```

```
Inner product.

>>> print(clifford("{1}") & clifford("{2}"))
0
>>> print(clifford(2) & clifford("{2}"))
0
>>> print(clifford("{1}") & clifford("{1}"))
1
>>> print(clifford("{1}") & clifford("{1,2}"))
{2}
```

Definition at line 836 of file PyClical.pyx.

6.3.2.3 __call__()

```
PyClical.clifford.__call__ (
              self,
              grade)
```

```
Pure grade-vector part.

>>> print(clifford("{1}")(1))
{1}
>>> print(clifford("{1}")(0))
0
>>> print(clifford("1+{1}+{1,2}")(0))
1
>>> print(clifford("1+{1}+{1,2}")(1))
{1}
>>> print(clifford("1+{1}+{1,2}")(2))
{1,2}
>>> print(clifford("1+{1}+{1,2}")(3))
0
```

Definition at line 1020 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.3.2.4 __cinit__()

```
PyClical.clifford.__cinit__ (
              self,
              other = 0,
              ixt = None)
```

```
Construct an object of type clifford.

>>> print(clifford(2))
2
>>> print(clifford(2.0))
2
>>> print(clifford(1.0e-1))
0.1
>>> print(clifford("2"))
2
>>> print(clifford("2{1,2,3}"))
2{1,2,3}
>>> print(clifford(clifford("2{1,2,3}")))
2{1,2,3}
>>> print(clifford("-{1}"))
-{1}
>>> print(clifford(2,index_set({1,2})))
2{1,2}
>>> print(clifford([2,3],index_set({1,2})))
2{1}+3{2}
```

Definition at line 565 of file PyClical.pyx.

6.3.2.5 __contains__()

```
PyClical.clifford.__contains__ (
              self,
              x)
```

```
Not applicable.

>>> x=clifford(index_set({-3,4,7})); -3 in x
Traceback (most recent call last):
  ...
TypeError: Not applicable.
```

Definition at line 627 of file PyClical.pyx.

6.3.2.6 __dealloc__()

```
PyClical.clifford.__dealloc__ (
              self)
```

```
Clean up by deallocating the instance of C++ class Clifford.
```

Definition at line 621 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.3.2.7 __getitem__()

```
PyClical.clifford.__getitem__ (
              self,
              ixt)
```

```
Subscripting: map from index set to scalar coordinate.

>>> clifford("{1}")[index_set(1)]
1.0
>>> clifford("{1}")[index_set({1})]
1.0
>>> clifford("{1}")[index_set({1,2})]
0.0
>>> clifford("2{1,2}")[index_set({1,2})]
2.0
```

Definition at line 707 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.3.2.8 __iadd__()

```
PyClical.clifford.__iadd__ (
              self,
              rhs)
```

```
Geometric sum.

>>> x = clifford(1); x += clifford("{2}"); print(x)
1+{2}
```

Definition at line 751 of file PyClical.pyx.

6.3.2.9 __iand__()

```
PyClical.clifford.__iand__ (
              self,
              rhs)
```

```
Inner product.

>>> x = clifford("{1}"); x &= clifford("{2}"); print(x)
0
>>> x = clifford(2); x &= clifford("{2}"); print(x)
0
>>> x = clifford("{1}"); x &= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x &= clifford("{1,2}"); print(x)
{2}
```

Definition at line 851 of file PyClical.pyx.

6.3.2.10 __idiv__()

```
PyClical.clifford.__idiv__ (
              self,
              rhs)
```

```
Geometric quotient.

>>> x = clifford("{1}"); x /= clifford("{2}"); print(x)
{1,2}
>>> x = clifford(2); x /= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x /= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x /= clifford("{1,2}"); print(x)
-{2}
```

Definition at line 911 of file PyClical.pyx.

6.3.2.11 __imod__()

```
PyClical.clifford.__imod__ (
              self,
              rhs)
```

```
Contraction.

>>> x = clifford("{1}"); x %= clifford("{2}"); print(x)
0
>>> x = clifford(2); x %= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x %= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x %= clifford("{1,2}"); print(x)
{2}
```

Definition at line 821 of file PyClical.pyx.

6.3.2.12 __imul__()

```
PyClical.clifford.__imul__ (
              self,
              rhs)
```

```
Geometric product.

>>> x = clifford(2); x *= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x *= clifford("{2}"); print(x)
{1,2}
>>> x = clifford("{1}"); x *= clifford("{1,2}"); print(x)
{2}
```

Definition at line 793 of file PyClical.pyx.

6.3.2.13 __ior__()

```
PyClical.clifford.__ior__ (
              self,
              rhs)
```

```
Transform left hand side, using right hand side as a transformation.

>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=x; print(y)
-{1}
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=exp(x); print(y)
-{1}
```

Definition at line 950 of file PyClical.pyx.

6.3.2.14 __isub__()

```
PyClical.clifford.__isub__ (
              self,
              rhs)
```

```
Geometric difference.

>>> x = clifford(1); x -= clifford("{2}"); print(x)
1-{2}
```

Definition at line 771 of file PyClical.pyx.

6.3.2.15 __iter__()

```
PyClical.clifford.__iter__ (
              self)
```

```
Not applicable.

>>> for a in clifford(index_set({-3,4,7})):print(a, end=",")
Traceback (most recent call last):
  ...
TypeError: Not applicable.
```

Definition at line 638 of file PyClical.pyx.

6.3.2.16 __ixor__()

```
PyClical.clifford.__ixor__ (
              self,
              rhs)
```

```
Outer product.

>>> x = clifford("{1}"); x ^= clifford("{2}"); print(x)
{1,2}
>>> x = clifford(2); x ^= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x ^= clifford("{1}"); print(x)
0
>>> x = clifford("{1}"); x ^= clifford("{1,2}"); print(x)
0
```

Definition at line 881 of file PyClical.pyx.

6.3.2.17 __mod__()

```
PyClical.clifford.__mod__ (
              lhs,
              rhs)
```

```
Contraction.

>>> print(clifford("{1}") % clifford("{2}"))
0
>>> print(clifford(2) % clifford("{2}"))
2{2}
>>> print(clifford("{1}") % clifford("{1}"))
1
>>> print(clifford("{1}") % clifford("{1,2}"))
{2}
```

Definition at line 806 of file PyClical.pyx.

6.3.2.18 __mul__()

```
PyClical.clifford.__mul__ (
              lhs,
              rhs)
```

```
Geometric product.

>>> print(clifford("{1}") * clifford("{2}"))
{1,2}
>>> print(clifford(2) * clifford("{2}"))
2{2}
>>> print(clifford("{1}") * clifford("{1,2}"))
{2}
```

Definition at line 780 of file PyClical.pyx.

6.3.2.19 __neg__()

```
PyClical.clifford.__neg__ (
              self)
```

```
Unary -.

>>> print(-clifford("{1}"))
-{1}
```

Definition at line 722 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.3.2.20 __or__()

```
PyClical.clifford.__or__ (
              lhs,
              rhs)
```

```
Transform left hand side, using right hand side as a transformation.

>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|x)
-{1}
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|exp(x))
-{1}
```

Definition at line 939 of file PyClical.pyx.

6.3.2.21 __pos__()

```
PyClical.clifford.__pos__ (
              self)
```

```
Unary +.

>>> print(+clifford("{1}"))
{1}
```

Definition at line 731 of file PyClical.pyx.

6.3.2.22 __pow__()

```
PyClical.clifford.__pow__ (
              self,
              m,
              dummy)
```

```
Power: self to the m.

>>> x=clifford("{1}"); print(x ** 2)
1
>>> x=clifford("2"); print(x ** 2)
4
>>> x=clifford("2+{1}"); print(x ** 0)
1
>>> x=clifford("2+{1}"); print(x ** 1)
2+{1}
>>> x=clifford("2+{1}"); print(x ** 2)
5+4{1}
>>> i=clifford("{1,2}"); print(exp(pi/2) * (i ** i))
1
```

Definition at line 961 of file PyClical.pyx.

References PyClical.clifford.pow().

6.3.2.23 __repr__()

```
PyClical.clifford.__repr__ (
              self)
```

```
The "official" string representation of self.

>>> clifford("1+3{-1}+2{1,2}+4{-2,7}").__repr__()
'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
```

Definition at line 1235 of file PyClical.pyx.

References clifford_to_repr().

6.3.2.24 __richcmp__()

```
PyClical.clifford.__richcmp__ (
              lhs,
              rhs,
              int,
              op)
```

```
Compare objects of type clifford.

>>> clifford("{1}") == clifford("1{1}")
True
>>> clifford("{1}") != clifford("1.0{1}")
False
>>> clifford("{1}") != clifford("1.0")
True
>>> clifford("{1,2}") == None
False
>>> clifford("{1,2}") != None
True
>>> None == clifford("{1,2}")
False
>>> None != clifford("{1,2}")
True
```

Definition at line 672 of file PyClical.pyx.

6.3.2.25 __str__()

```
PyClical.clifford.__str__ (
              self)
```

```
The "informal" string representation of self.

>>> clifford("1+3{-1}+2{1,2}+4{-2,7}").__str__()
'1+3{-1}+2{1,2}+4{-2,7}'
```

Definition at line 1244 of file PyClical.pyx.

References clifford_to_str().

6.3.2.26 __sub__()

```
PyClical.clifford.__sub__ (
              lhs,
              rhs)
```

```
Geometric difference.

>>> print(clifford(1) - clifford("{2}"))
1-{2}
>>> print(clifford("{1}") - clifford("{2}"))
{1}-{2}
```

Definition at line 760 of file PyClical.pyx.

6.3.2.27 __truediv__()

```
PyClical.clifford.__truediv__ (
              lhs,
              rhs)
```

```
Geometric quotient.

>>> print(clifford("{1}") / clifford("{2}"))
{1,2}
>>> print(clifford(2) / clifford("{2}"))
2{2}
>>> print(clifford("{1}") / clifford("{1}"))
1
>>> print(clifford("{1}") / clifford("{1,2}"))
-{2}
```

Definition at line 896 of file PyClical.pyx.

6.3.2.28 __xor__()

```
PyClical.clifford.__xor__ (
              lhs,
              rhs)
```

```
Outer product.

>>> print(clifford("{1}") ^ clifford("{2}"))
{1,2}
>>> print(clifford(2) ^ clifford("{2}"))
2{2}
>>> print(clifford("{1}") ^ clifford("{1}"))
0
>>> print(clifford("{1}") ^ clifford("{1,2}"))
0
```

Definition at line 866 of file PyClical.pyx.

6.3.2.29 abs()

```
PyClical.clifford.abs (
              self)
```

```
Absolute value: square root of norm.

>>> clifford("1+{-1}+{1,2}+{1,2,3}").abs()
2.0
```

Definition at line 1175 of file PyClical.pyx.

References glucat.abs().

6.3.2.30 conj()

```
PyClical.clifford.conj (
              self)
```

```
Conjugation, reverse o involute == involute o reverse.

>>> print((clifford("{1}")).conj())
-{1}
>>> print((clifford("{2}") * clifford("{1}")).conj())
{1,2}
>>> print((clifford("{1}") * clifford("{2}")).conj())
-{1,2}
>>> print(clifford("1+{1}+{1,2}").conj())
1-{1}-{1,2}
```

Definition at line 1138 of file PyClical.pyx.

References PyClical.clifford.conj(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.clifford.conj().

6.3.2.31 even()

```
PyClical.clifford.even (
              self)
```

```
Even part of multivector, sum of even grade terms.

>>> print(clifford("1+{1}+{1,2}").even())
1+{1,2}
```

Definition at line 1061 of file PyClical.pyx.

References PyClical.clifford.even(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.clifford.even().

6.3.2.32 frame()

```
PyClical.clifford.frame (
              self)
```

```
Subalgebra generated by all generators of terms of given multivector.

>>> print(clifford("1+3{-1}+2{1,2}+4{-2,7}").frame())
{-2,-1,1,2,7}
>>> s=clifford("1+3{-1}+2{1,2}+4{-2,7}").frame(); type(s)
<class 'PyClical.index_set'>
```

Definition at line 1224 of file PyClical.pyx.

References PyClical.clifford.frame(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.clifford.frame().

6.3.2.33 inv()

```
PyClical.clifford.inv (
              self)
```

```
Geometric multiplicative inverse.

>>> x = clifford("{1}"); print(x.inv())
{1}
>>> x = clifford(2); print(x.inv())
0.5
>>> x = clifford("{1,2}"); print(x.inv())
-{1,2}
```

Definition at line 926 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.inv().

Referenced by PyClical.clifford.inv().

6.3.2.34 involute()

```
PyClical.clifford.involute (
              self)
```

```
Main involution, each {i} is replaced by -{i} in each term,
eg. clifford("{1}") -> -clifford("{1}").

>>> print(clifford("{1}").involute())
-{1}
>>> print((clifford("{2}") * clifford("{1}")).involute())
-{1,2}
>>> print((clifford("{1}") * clifford("{2}")).involute())
{1,2}
>>> print(clifford("1+{1}+{1,2}").involute())
1-{1}+{1,2}
```

Definition at line 1107 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.involute().

Referenced by PyClical.clifford.involute().

6.3.2.35 isinf()

```
PyClical.clifford.isinf (
              self)
```

```
Check if a multivector contains any infinite values.

>>> clifford().isinf()
False
```

Definition at line 1206 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.isnan().

6.3.2.36 isnan()

```
PyClical.clifford.isnan (
              self)
```

```
Check if a multivector contains any IEEE NaN values.

>>> clifford().isnan()
False
```

Definition at line 1215 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.isnan().

Referenced by PyClical.clifford.isinf(), and PyClical.clifford.isnan().

6.3.2.37 max_abs()

```
PyClical.clifford.max_abs (
              self)
```

```
Maximum of absolute values of components of multivector: multivector infinity norm.

>>> clifford("1+{-1}+{1,2}+{1,2,3}").max_abs()
1.0
>>> clifford("3+2{1}+{1,2}").max_abs()
3.0
```

Definition at line 1184 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.max_abs().

Referenced by PyClical.clifford.max_abs().

6.3.2.38 norm()

```
PyClical.clifford.norm (
              self)
```

```
Norm == sum of squares of coordinates.

>>> clifford("1+{1}+{1,2}").norm()
3.0
>>> clifford("1+{-1}+{1,2}+{1,2,3}").norm()
4.0
```

Definition at line 1164 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.norm().

Referenced by PyClical.clifford.norm().

6.3.2.39 odd()

```
PyClical.clifford.odd (
              self)
```

```
Odd part of multivector, sum of odd grade terms.

>>> print(clifford("1+{1}+{1,2}").odd())
{1}
```

Definition at line 1070 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.odd().

Referenced by PyClical.clifford.odd().

6.3.2.40 outer_pow()

```
PyClical.clifford.outer_pow (
              self,
              m)
```

```
Outer product power.

>>> x=clifford("2+{1}"); print(x.outer_pow(0))
1
>>> x=clifford("2+{1}"); print(x.outer_pow(1))
2+{1}
>>> x=clifford("2+{1}"); print(x.outer_pow(2))
4+4{1}
>>> print(clifford("1+{1}+{1,2}").outer_pow(3))
1+3{1}+3{1,2}
```

Definition at line 1004 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.outer_pow().

Referenced by PyClical.clifford.outer_pow().

6.3.2.41 pow()

```
PyClical.clifford.pow (
              self,
              m)
```

```
Power: self to the m.

>>> x=clifford("{1}"); print(x.pow(2))
1
>>> x=clifford("2"); print(x.pow(2))
4
>>> x=clifford("2+{1}"); print(x.pow(0))
1
>>> x=clifford("2+{1}"); print(x.pow(1))
2+{1}
>>> x=clifford("2+{1}"); print(x.pow(2))
5+4{1}
>>> print(clifford("1+{1}+{1,2}").pow(3))
1+3{1}+3{1,2}
>>> i=clifford("{1,2}"); print(exp(pi/2) * i.pow(i))
1
```

Definition at line 980 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.pow().

Referenced by PyClical.clifford.__pow__(), and PyClical.clifford.pow().

6.3.2.42 pure()

```
PyClical.clifford.pure (
              self)
```

```
Pure part.

>>> print(clifford("1+{1}+{1,2}").pure())
{1}+{1,2}
>>> print(clifford("{1,2}").pure())
{1,2}
```

Definition at line 1050 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.pure().

Referenced by PyClical.clifford.pure().

6.3.2.43 quad()

```
PyClical.clifford.quad (
              self)
```

```
Quadratic form == (rev(x)*x)(0).

>>> print(clifford("1+{1}+{1,2}").quad())
3.0
>>> print(clifford("1+{-1}+{1,2}+{1,2,3}").quad())
2.0
```

Definition at line 1153 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.quad().

Referenced by PyClical.clifford.quad().

6.3.2.44 reframe()

```
PyClical.clifford.reframe (
              self,
              ixt)
```

```
Put self into a larger frame, containing the union of self.frame() and index set ixt.
This can be used to make multiplication faster, by multiplying within a common frame.

>>> clifford("2+3{1}").reframe(index_set({1,2,3}))
clifford("2+3{1}")
>>> s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() == (s|t);
True
```

Definition at line 649 of file PyClical.pyx.

6.3.2.45 reverse()

```
PyClical.clifford.reverse (
              self)
```

```
Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").

>>> print(clifford("{1}").reverse())
{1}
>>> print((clifford("{2}") * clifford("{1}")).reverse())
{1,2}
>>> print((clifford("{1}") * clifford("{2}")).reverse())
-{1,2}
>>> print(clifford("1+{1}+{1,2}").reverse())
1+{1}-{1,2}
```

Definition at line 1123 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.reverse().

Referenced by PyClical.clifford.reverse().

6.3.2.46 scalar()

```
PyClical.clifford.scalar (
              self)
```

```
Scalar part.

>>> clifford("1+{1}+{1,2}").scalar()
1.0
>>> clifford("{1,2}").scalar()
0.0
```

Definition at line 1039 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.scalar().

Referenced by PyClical.clifford.scalar().

6.3.2.47 truncated()

```
PyClical.clifford.truncated (
              self,
              limit)
```

```
Remove all terms of self with relative size smaller than limit.

>>> clifford("1e8+{1}+1e-8{1,2}").truncated(1.0e-6)
clifford("100000000")
>>> clifford("1e4+{1}+1e-4{1,2}").truncated(1.0e-6)
clifford("10000+{1}")
```

Definition at line 1195 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.truncated().

Referenced by PyClical.clifford.truncated().

6.3.2.48 vector_part()

```
PyClical.clifford.vector_part (
              self,
              frm = None)
```

```
Vector part of multivector, as a Python list, with respect to frm.

>>> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part())
[2.0, 3.0]
>>> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part(index_set({-1,1,2})))
[0.0, 2.0, 3.0]
```

Definition at line 1079 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.clifford.vector_part().

Referenced by PyClical.clifford.vector_part().

6.3.3 Member Data Documentation

6.3.3.1 instance

```
PyClical.clifford.instance = new Clifford((<clifford>other).unwrap())
```

Definition at line 592 of file PyClical.pyx.

Referenced by PyClical.clifford.__call__(), PyClical.index_set.__contains__(), PyClical.clifford.__dealloc__(), PyClical.index_set.__dealloc__(), PyClical.clifford.__getitem__(), PyClical.index_set.__getitem__(), PyClical.index_set.__invert__(), PyClical.clifford.__neg__(), PyClical.index_set.__setitem__(), PyClical.clifford.conj(), PyClical.index_set.count(), PyClical.index_set.count_neg(), PyClical.index_set.count_pos(), PyClical.clifford.even(), PyClical.clifford.frame(), PyClical.index_set.hash_fn(), PyClical.clifford.inv(), PyClical.clifford.involute(), PyClical.clifford.isinf(), PyClical.clifford.isnan(), PyClical.index_set.max(), PyClical.clifford.max_abs(), PyClical.index_set.min(), PyClical.clifford.norm(), PyClical.clifford.odd(), PyClical.clifford.outer_pow(), PyClical.clifford.pow(), PyClical.clifford.pure(), PyClical.clifford.quad(), PyClical.clifford.reverse(), PyClical.clifford.scalar(), PyClical.index_set.sign_of_mult(), PyClical.index_set.sign_of_square(), PyClical.clifford.truncated(), and PyClical.clifford.vector_part().

The documentation for this class was generated from the following file:

- pyclical/PyClical.pyx

6.4 glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T > Class Template Reference

clifford_algebra<> declares the operations of a Clifford algebra

```
#include <clifford_algebra.h>
```

Public Types

- using scalar_t = Scalar_T
- using index_set_t = Index_Set_T
- using multivector_t = Multivector_T
- using pair_t = std::pair<const index_set_t, Scalar_T>
- using vector_t = std::vector<Scalar_T>

Public Member Functions

- virtual ~clifford_algebra ()=default
- virtual auto operator== (const multivector_t &val) const -> bool=0

  Test for equality of multivectors.
- virtual auto operator== (const Scalar_T &scr) const -> bool=0

  Test for equality of multivector and scalar.
- virtual auto operator+= (const multivector_t &rhs) -> multivector_t &=0

  Geometric sum.
- virtual auto operator+= (const Scalar_T &scr) -> multivector_t &=0

  Geometric sum of multivector and scalar.
- virtual auto operator-= (const multivector_t &rhs) -> multivector_t &=0

  Geometric difference.
- virtual auto operator-= (const Scalar_T &scr) -> multivector_t &=0

  Geometric difference of multivector and scalar.
- virtual auto operator- () const -> const multivector_t=0

  Unary -.
- virtual auto operator*= (const Scalar_T &scr) -> multivector_t &=0

  Product of multivector and scalar.
- virtual auto operator*= (const multivector_t &rhs) -> multivector_t &=0

  Geometric product.
- virtual auto operator%= (const multivector_t &rhs) -> multivector_t &=0

  Contraction.
- virtual auto operator&= (const multivector_t &rhs) -> multivector_t &=0

  Inner product.
- virtual auto operator^= (const multivector_t &rhs) -> multivector_t &=0

  Outer product.
- virtual auto operator/= (const Scalar_T &scr) -> multivector_t &=0

  Quotient of multivector and scalar.
- virtual auto operator/= (const multivector_t &rhs) -> multivector_t &=0

  Geometric quotient.
- virtual auto operator|= (const multivector_t &rhs) -> multivector_t &=0

  Transformation via twisted adjoint action.
- virtual auto inv () const -> const multivector_t=0

Geometric multiplicative inverse.
- virtual auto pow (int m) const -> const multivector_t=0

  ∗this to the m
- virtual auto outer_pow (int m) const -> const multivector_t=0

  Outer product power.
- virtual auto frame () const -> const index_set_t=0

  Subalgebra generated by all generators of terms of given multivector.
- virtual auto grade () const -> index_t=0

  Maximum of the grades of each term.
- virtual auto operator[ ] (const index_set_t ist) const -> Scalar_T=0

  Subscripting: map from index set to scalar coordinate.
- virtual auto operator() (index_t grade) const -> const multivector_t=0

  Pure grade-vector part.
- virtual auto scalar () const -> Scalar_T=0

  Scalar part.
- virtual auto pure () const -> const multivector_t=0

  Pure part.
- virtual auto even () const -> const multivector_t=0

  Even part of multivector, sum of even grade terms.
- virtual auto odd () const -> const multivector_t=0

  Odd part of multivector, sum of odd grade terms.
- virtual auto vector_part () const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frame()
- virtual auto vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frm.
- virtual auto involute () const -> const multivector_t=0

  Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.
- virtual auto reverse () const -> const multivector_t=0

  Reversion, eg. {1}∗{2} -> {2}∗{1}.
- virtual auto conj () const -> const multivector_t=0

  Conjugation, reverse o involute == involute o reverse.
- virtual auto quad () const -> Scalar_T=0

  Scalar_T quadratic form == (rev(x)∗x)(0)
- virtual auto norm () const -> Scalar_T=0

  Scalar_T norm == sum of norm of coordinates.
- virtual auto max_abs () const -> Scalar_T=0

  Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto truncated (const Scalar_T &limit=default_truncation) const -> const multivector_t=0

  Remove all terms with relative size smaller than limit.
- virtual auto isinf () const -> bool=0

  Check if a multivector contains any infinite values.
- virtual auto isnan () const -> bool=0

  Check if a multivector contains any IEEE NaN values.
- virtual void write (const std::string &msg="") const =0

  Write formatted multivector to output.
- virtual void write (std::ofstream &ofile, const std::string &msg="") const =0

  Write formatted multivector to file.

Static Public Member Functions

- static auto classname () -> const std::string

Static Public Attributes

- static const index_t v_lo = index_set_t::v_lo
- static const index_t v_hi = index_set_t::v_hi
- static const Scalar_T default_truncation = std::numeric_limits<Scalar_T>::epsilon()

    Default for truncation.

6.4.1 Detailed Description

template<typename Scalar_T, typename Index_Set_T, typename Multivector_T>
class glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >

clifford_algebra<> declares the operations of a Clifford algebra

Definition at line 45 of file clifford_algebra.h.

6.4.2 Member Typedef Documentation

6.4.2.1 index_set_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::index_set_t = Index_↩
Set_T
```

Definition at line 49 of file clifford_algebra.h.

6.4.2.2 multivector_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::multivector_t = Multivector↩
_T
```

Definition at line 52 of file clifford_algebra.h.

6.4.2.3 pair_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pair_t = std::pair<const
index_set_t, Scalar_T>
```

Definition at line 53 of file clifford_algebra.h.

6.4.2.4 scalar_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar_t = Scalar_T
```

Definition at line 48 of file clifford_algebra.h.

6.4.2.5 vector_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_t = std::vector<Scalar↩
_T>
```

Definition at line 54 of file clifford_algebra.h.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 ∼clifford_algebra()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::∼clifford_algebra
() [virtual], [default]
```

6.4.4 Member Function Documentation

6.4.4.1 classname()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::classname () -> const
std::string [static]
```

Definition at line 66 of file clifford_algebra_imp.h.

6.4.4.2 conj()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::conj () const
-> const multivector_t [pure virtual]
```

Conjugation, reverse o involute == involute o reverse.

6.4.4.3 even()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::even () const
-> const multivector_t [pure virtual]
```

Even part of multivector, sum of even grade terms.

6.4.4.4 frame()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame () const
-> const index_set_t [pure virtual]
```

Subalgebra generated by all generators of terms of given multivector.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >
and glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

6.4.4.5 grade()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::grade () const
-> index_t  [pure virtual]
```

Maximum of the grades of each term.

6.4.4.6 inv()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::inv () const
-> const multivector_t  [pure virtual]
```

Geometric multiplicative inverse.

6.4.4.7 involute()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::involute ()
const -> const multivector_t  [pure virtual]
```

Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.

6.4.4.8 isinf()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isinf () const
-> bool  [pure virtual]
```

Check if a multivector contains any infinite values.

6.4.4.9 isnan()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan () const
-> bool  [pure virtual]
```

Check if a multivector contains any IEEE NaN values.

Referenced by glucat::cascade_log(), glucat::exp(), glucat::log(), glucat::matrix_log(), glucat::matrix_sqrt(), glucat::pade_log(), and glucat::sqrt().

6.4.4.10 max_abs()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::max_abs ()
const -> Scalar_T  [pure virtual]
```

Maximum of absolute values of components of multivector: multivector infinity norm.

6.4.4.11 norm()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::norm () const
-> Scalar_T [pure virtual]
```

Scalar_T norm == sum of norm of coordinates.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

6.4.4.12 odd()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::odd () const
-> const multivector_t [pure virtual]
```

Odd part of multivector, sum of odd grade terms.

6.4.4.13 operator%=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator%= (
            const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Contraction.

6.4.4.14 operator&=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator&= (
            const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Inner product.

6.4.4.15 operator()()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator() (
            index_t grade) const -> const multivector_t [pure virtual]
```

Pure grade-vector part.

6.4.4.16 operator*=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator*= (
            const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Geometric product.

6.4.4.17 operator*=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator*= (
            const Scalar_T & scr) -> multivector_t &  [pure virtual]
```

Product of multivector and scalar.

6.4.4.18 operator+=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator+= (
            const multivector_t & rhs) -> multivector_t &  [pure virtual]
```

Geometric sum.

6.4.4.19 operator+=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator+= (
            const Scalar_T & scr) -> multivector_t &  [pure virtual]
```

Geometric sum of multivector and scalar.

6.4.4.20 operator-()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator- ()
const -> const multivector_t  [pure virtual]
```

Unary -.

6.4.4.21 operator-=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator-= (
            const multivector_t & rhs) -> multivector_t &  [pure virtual]
```

Geometric difference.

6.4.4.22 operator-=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator-= (
            const Scalar_T & scr) -> multivector_t &  [pure virtual]
```

Geometric difference of multivector and scalar.

6.4.4.23 operator/=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator/= (
            const multivector_t & rhs) -> multivector_t &  [pure virtual]
```

Geometric quotient.

6.4.4.24 operator/=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator/= (
            const Scalar_T & scr) -> multivector_t &  [pure virtual]
```

Quotient of multivector and scalar.

6.4.4.25 operator==() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator== (
            const multivector_t & val) const -> bool  [pure virtual]
```

Test for equality of multivectors.

6.4.4.26 operator==() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator== (
            const Scalar_T & scr) const -> bool  [pure virtual]
```

Test for equality of multivector and scalar.

6.4.4.27 operator[]()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator[] (
            const index_set_t ist) const -> Scalar_T  [pure virtual]
```

Subscripting: map from index set to scalar coordinate.

6.4.4.28 operator^=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator^= (
            const multivector_t & rhs) -> multivector_t &  [pure virtual]
```

Outer product.

6.4.4.29 operator"|=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator|= (
            const multivector_t & rhs) -> multivector_t &  [pure virtual]
```

Transformation via twisted adjoint action.

6.4.4.30 outer_pow()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::outer_pow (
            int m) const -> const multivector_t  [pure virtual]
```

Outer product power.

6.4.4.31 pow()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pow (
            int m) const -> const multivector_t  [pure virtual]
```

∗this to the m

6.4.4.32 pure()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pure () const
-> const multivector_t  [pure virtual]
```

Pure part.

6.4.4.33 quad()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::quad () const
-> Scalar_T  [pure virtual]
```

Scalar_T quadratic form == (rev(x)∗x)(0)

6.4.4.34 reverse()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::reverse ()
const -> const multivector_t  [pure virtual]
```

Reversion, eg. {1}∗{2} -> {2}∗{1}.

6.4.4.35 scalar()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar () const
-> Scalar_T [pure virtual]
```

Scalar part.

Referenced by glucat::exp(), glucat::matrix_log(), and glucat::matrix_sqrt().

6.4.4.36 truncated()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated (
            const Scalar_T & limit = default_truncation) const -> const multivector_t [pure
virtual]
```

Remove all terms with relative size smaller than limit.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and glucat::operator<<().

6.4.4.37 vector_part() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_part ()
const -> const vector_t [pure virtual]
```

Vector part of multivector, as a vector_t with respect to frame()

6.4.4.38 vector_part() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_part (
            const index_set_t frm,
            const bool prechecked) const -> const vector_t [pure virtual]
```

Vector part of multivector, as a vector_t with respect to frm.

6.4.4.39 write() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual void glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::write (
            const std::string & msg = "") const [pure virtual]
```

Write formatted multivector to output.

6.4.4.40 write() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual void glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::write (
            std::ofstream & ofile,
            const std::string & msg = "") const  [pure virtual]
```

Write formatted multivector to file.

6.4.5 Member Data Documentation

6.4.5.1 default_truncation

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const Scalar_T glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::default_↩
truncation = std::numeric_limits<Scalar_T>::epsilon()  [static]
```

Default for truncation.

Definition at line 59 of file clifford_algebra.h.

6.4.5.2 v_hi

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const index_t glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::v_hi = index↩
_set_t::v_hi  [static]
```

Definition at line 51 of file clifford_algebra.h.

6.4.5.3 v_lo

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const index_t glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::v_lo = index↩
_set_t::v_lo  [static]
```

Definition at line 50 of file clifford_algebra.h.

The documentation for this class was generated from the following files:

- glucat/clifford_algebra.h
- glucat/clifford_algebra_imp.h

6.5 glucat::compare_types< LHS_T, RHS_T > Class Template Reference

Type comparison.

```
#include <global.h>
```

Public Types

- enum { are_same = false }

6.5.1 Detailed Description

template<typename LHS_T, typename RHS_T>
class glucat::compare_types< LHS_T, RHS_T >

Type comparison.

Definition at line 54 of file global.h.

6.5.2 Member Enumeration Documentation

6.5.2.1 anonymous enum

```
template<typename LHS_T , typename RHS_T >
anonymous enum
```

Enumerator

are_same	

Definition at line 57 of file global.h.

The documentation for this class was generated from the following file:

- glucat/global.h

6.6 glucat::compare_types< T, T > Class Template Reference

```
#include <global.h>
```

Public Types

- enum { are_same = true }

6.6.1 Detailed Description

template<typename T>
class glucat::compare_types< T, T >

Definition at line 60 of file global.h.

6.6.2 Member Enumeration Documentation

6.6.2.1 anonymous enum

```
template<typename T >
anonymous enum
```

Enumerator

are_same	

Definition at line 63 of file global.h.

The documentation for this class was generated from the following file:

- glucat/global.h

6.7 glucat::control_t Class Reference

Parameters to control tests.

```
#include <control.h>
```

Public Member Functions

- int call (intfn f) const

  Call a function that returns int.
- int call (intintfn f, int arg) const

  Call a function of int that returns int.

Static Public Member Functions

- static const control_t & control (int argc, char **argv)
- static bool verbose ()

  Produce more detailed output from tests.

Private Member Functions

- bool valid () const
- bool catch_exceptions () const
- control_t (int argc, char **argv)

  Constructor from program arguments.
- control_t ()=default
- ∼control_t ()=default
- control_t (const control_t &)=delete
- control_t & operator= (const control_t &)=delete

Private Attributes

- bool m_valid

  Test parameters are valid.
- bool m_catch_exceptions

  Catch exceptions.

Static Private Attributes

- static bool m_verbose_output = false

    Produce more detailed output from tests.

Friends

- class friend_for_private_destructor

6.7.1 Detailed Description

Parameters to control tests.

Definition at line 39 of file control.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 control_t() [1/3]

```
glucat::control_t::control_t (
            int argc,
            char ** argv)  [private]
```

Constructor from program arguments.

Test control constructor from program arguments.

Definition at line 88 of file control.h.

References GLUCAT_PACKAGE_NAME, GLUCAT_VERSION, m_catch_exceptions, m_valid, m_verbose_output, and valid().

6.7.2.2 control_t() [2/3]

```
glucat::control_t::control_t ()  [private], [default]
```

6.7.2.3 ∼control_t()

```
glucat::control_t::~control_t ()  [private], [default]
```

6.7.2.4 control_t() [3/3]

```
glucat::control_t::control_t (
            const control_t & )  [private], [delete]
```

6.7.3 Member Function Documentation

6.7.3.1 call() [1/2]

```
int glucat::control_t::call (
            intfn f) const  [inline]
```

Call a function that returns int.

Definition at line 136 of file control.h.

References catch_exceptions(), glucat::try_catch(), and valid().

6.7.3.2 call() [2/2]

```
int glucat::control_t::call (
            intintfn f,
            int arg) const  [inline]
```

Call a function of int that returns int.

Definition at line 150 of file control.h.

References catch_exceptions(), glucat::try_catch(), and valid().

6.7.3.3 catch_exceptions()

```
bool glucat::control_t::catch_exceptions () const  [inline], [private]
```

Definition at line 49 of file control.h.

References m_catch_exceptions.

Referenced by call(), and call().

6.7.3.4 control()

```
static const control_t & glucat::control_t::control (
            int argc,
            char ** argv)  [inline], [static]
```

Single instance Ref: Scott Meyers, "Effective C++" Second Edition, Addison-Wesley, 1998.

Definition at line 71 of file control.h.

6.7.3.5 operator=()

```
control_t & glucat::control_t::operator= (
            const control_t & )  [private], [delete]
```

6.7.3.6 valid()

```
bool glucat::control_t::valid () const [inline], [private]
```

Definition at line 44 of file control.h.

References m_valid.

Referenced by call(), call(), and control_t().

6.7.3.7 verbose()

```
static bool glucat::control_t::verbose () [inline], [static]
```

Produce more detailed output from tests.

Definition at line 80 of file control.h.

References m_verbose_output.

6.7.4 Friends And Related Symbol Documentation

6.7.4.1 friend_for_private_destructor

```
friend class friend_for_private_destructor [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE `http://doc.ece.uci.edu`

Definition at line 67 of file control.h.

6.7.5 Member Data Documentation

6.7.5.1 m_catch_exceptions

```
bool glucat::control_t::m_catch_exceptions [private]
```

Catch exceptions.

Definition at line 48 of file control.h.

Referenced by catch_exceptions(), and control_t().

6.7.5.2 m_valid

```
bool glucat::control_t::m_valid [private]
```

Test parameters are valid.

Definition at line 43 of file control.h.

Referenced by control_t(), and valid().

6.7.5.3 m_verbose_output

```
bool glucat::control_t::m_verbose_output = false [static], [private]
```

Produce more detailed output from tests.

Definition at line 53 of file control.h.

Referenced by control_t(), and verbose().

The documentation for this class was generated from the following file:

- test/control.h

6.8 glucat::CTAssertion< bool > Struct Template Reference

Compile time assertion.

6.8.1 Detailed Description

template<bool>
struct glucat::CTAssertion< bool >

Compile time assertion.

Definition at line 46 of file global.h.

The documentation for this struct was generated from the following file:

- glucat/global.h

6.9 glucat::CTAssertion< true > Struct Reference

```
#include <global.h>
```

6.9.1 Detailed Description

Definition at line 47 of file global.h.

The documentation for this struct was generated from the following file:

- glucat/global.h

6.10 glucat::numeric_traits< Scalar_T >::demoted Struct Reference

Demoted type for long double.

```
#include <promotion.h>
```

Public Types

- using type = float

6.10.1 Detailed Description

template<typename Scalar_T>
struct glucat::numeric_traits< Scalar_T >::demoted

Demoted type for long double.

Demoted type.

Definition at line 148 of file scalar.h.

6.10.2 Member Typedef Documentation

6.10.2.1 type

```
template<typename Scalar_T >
typedef float glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 78 of file promotion.h.

The documentation for this struct was generated from the following files:

- glucat/promotion.h
- glucat/scalar.h

6.11 glucat::matrix::eig_genus< Matrix_T > Struct Template Reference

Structure containing classification of eigenvalues.

```
#include <matrix.h>
```

Public Types

- using Scalar_T = typename Matrix_T::value_type

Public Attributes

- bool m_is_singular = false
  Is the matrix singular?
- eig_case_t m_eig_case = safe_eigs
  What kind of eigenvalues does the matrix contain?
- Scalar_T m_safe_arg = Scalar_T(0)
  Argument such that exp(pi-m_safe_arg) lies between arguments of eigenvalues.

6.11.1 Detailed Description

template<typename Matrix_T>
struct glucat::matrix::eig_genus< Matrix_T >

Structure containing classification of eigenvalues.

Definition at line 140 of file matrix.h.

6.11.2 Member Typedef Documentation

6.11.2.1 Scalar_T

```
template<typename Matrix_T >
using glucat::matrix::eig_genus< Matrix_T >::Scalar_T = typename Matrix_T::value_type
```

Definition at line 142 of file matrix.h.

6.11.3 Member Data Documentation

6.11.3.1 m_eig_case

```
template<typename Matrix_T >
eig_case_t glucat::matrix::eig_genus< Matrix_T >::m_eig_case = safe_eigs
```

What kind of eigenvalues does the matrix contain?

Definition at line 146 of file matrix.h.

Referenced by glucat::matrix::classify_eigenvalues().

6.11.3.2 m_is_singular

```
template<typename Matrix_T >
bool glucat::matrix::eig_genus< Matrix_T >::m_is_singular = false
```

Is the matrix singular?

Definition at line 144 of file matrix.h.

Referenced by glucat::matrix::classify_eigenvalues().

6.11.3.3 m_safe_arg

```
template<typename Matrix_T >
Scalar_T glucat::matrix::eig_genus< Matrix_T >::m_safe_arg = Scalar_T(0)
```

Argument such that exp(pi-m_safe_arg) lies between arguments of eigenvalues.

Definition at line 148 of file matrix.h.

Referenced by glucat::matrix::classify_eigenvalues().

The documentation for this struct was generated from the following file:

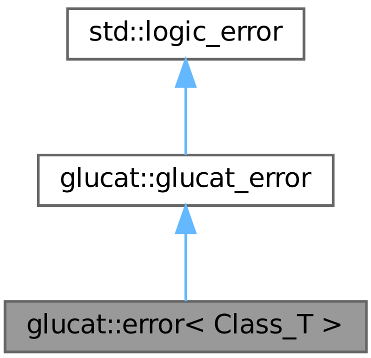
- glucat/matrix.h

6.12 glucat::error< Class_T > Class Template Reference

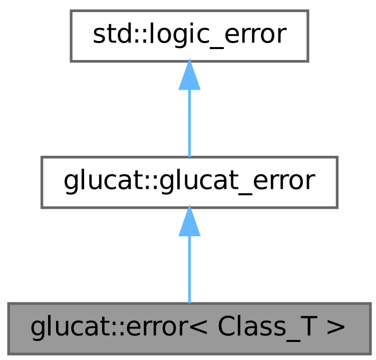
Specific exception class.

```
#include <errors.h>
```

Inheritance diagram for glucat::error< Class_T >:



Collaboration diagram for glucat::error< Class_T >:



Public Member Functions

- error (const std::string &msg)

  Specific exception class.
- error (const std::string &context, const std::string &msg)
- auto heading () const noexcept -> const std::string override
- auto classname () const noexcept -> const std::string override
- void print_error_msg () const override

Public Member Functions inherited from glucat::glucat_error

- glucat_error (const std::string &context, const std::string &msg)
- ∼glucat_error () noexcept override=default

Additional Inherited Members

Public Attributes inherited from glucat::glucat_error

- std::string name

6.12.1 Detailed Description

template<class Class_T>
class glucat::error< Class_T >

Specific exception class.

Definition at line 56 of file errors.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 error() [1/2]

```
template<class Class_T >
glucat::error< Class_T >::error (
            const std::string & msg)
```

Specific exception class.

Definition at line 44 of file errors_imp.h.

6.12.2.2 error() [2/2]

```
template<class Class_T >
glucat::error< Class_T >::error (
            const std::string & context,
            const std::string & msg)
```

Definition at line 50 of file errors_imp.h.

6.12.3 Member Function Documentation

6.12.3.1 classname()

```
template<class Class_T >
auto glucat::error< Class_T >::classname () const -> const std::string  [override], [virtual],
[noexcept]
```

Implements glucat::glucat_error.

Definition at line 63 of file errors_imp.h.

6.12.3.2 heading()

```
template<class Class_T >
auto glucat::error< Class_T >::heading () const -> const std::string  [override], [virtual],
[noexcept]
```

Implements glucat::glucat_error.

Definition at line 57 of file errors_imp.h.

6.12.3.3 print_error_msg()

```
template<class Class_T >
void glucat::error< Class_T >::print_error_msg () const  [override], [virtual]
```

Implements glucat::glucat_error.

Definition at line 69 of file errors_imp.h.

The documentation for this class was generated from the following files:

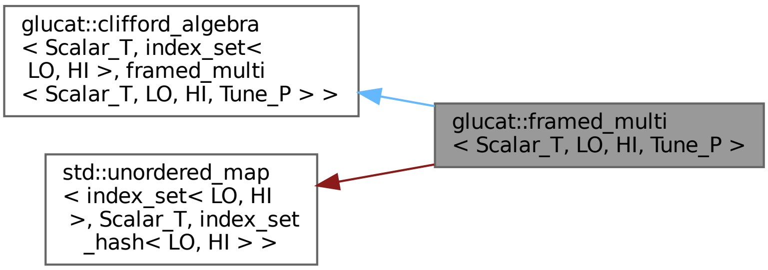
- glucat/errors.h
- glucat/errors_imp.h

6.13 glucat::framed_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference

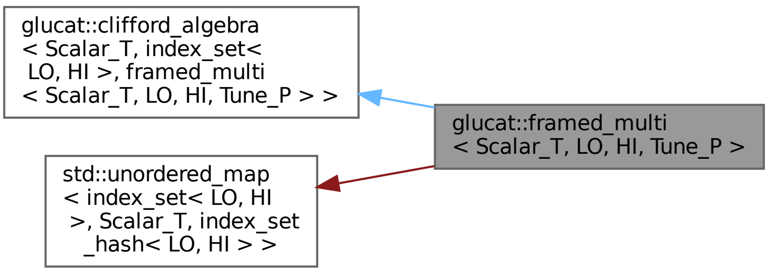
A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.

`#include <framed_multi.h>`

Inheritance diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >:



Collaboration diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >:



Classes

- class hash_size_t
- class var_term

    Variable term.

Public Types

- using multivector_t = framed_multi
- using framed_multi_t = multivector_t
- using scalar_t = Scalar_T
- using tune_p = Tune_P
- using index_set_t = index_set<LO, HI>
- using term_t = std::pair<const index_set_t, Scalar_T>
- using vector_t = std::vector<Scalar_T>
- using error_t = error<multivector_t>
- using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P >

Public Types inherited from glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >

- using scalar_t
- using index_set_t
- using multivector_t
- using pair_t
- using vector_t

Public Member Functions

- ~framed_multi () override=default

  Destructor.
- framed_multi ()

  Default constructor.
- template<typename Other_Scalar_T >
  framed_multi (const framed_multi< Other_Scalar_T, LO, HI, Tune_P > &val)

  Construct a multivector from a multivector with a different scalar type.
- template<typename Other_Scalar_T >
  framed_multi (const framed_multi< Other_Scalar_T, LO, HI, Tune_P > &val, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given multivector.
- framed_multi (const framed_multi_t &val, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given multivector.
- framed_multi (const index_set_t ist, const Scalar_T &crd=Scalar_T(1))

  Construct a multivector from an index set and a scalar coordinate.
- framed_multi (const index_set_t ist, const Scalar_T &crd, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from an index set and a scalar coordinate.
- framed_multi (const Scalar_T &scr, const index_set_t frm=index_set_t())

  Construct a multivector from a scalar (within a frame, if given)
- framed_multi (const int scr, const index_set_t frm=index_set_t())

  Construct a multivector from an int (within a frame, if given)
- framed_multi (const vector_t &vec, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given vector.
- framed_multi (const std::string &str)

  Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- framed_multi (const std::string &str, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- framed_multi (const char ∗str)

  Construct a multivector from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".
- framed_multi (const char ∗str, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".
- template<typename Other_Scalar_T >
  framed_multi (const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > &val)

  Construct a multivector from a matrix_multi_t.
- template<typename Other_Scalar_T >
  auto fast_matrix_multi (const index_set_t frm) const -> const matrix_multi< Other_Scalar_T, LO, HI, Tune_P >

  Use generalized FFT to construct a matrix_multi_t.
- auto fast_framed_multi () const -> const framed_multi_t

  Use inverse generalized FFT to construct a framed_multi_t.
- _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS auto nbr_terms () const -> unsigned long

  Number of terms.
- auto operator+= (const term_t &term) -> multivector_t &

  Add a term, if non-zero.

Public Member Functions inherited from glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >

- virtual ~clifford_algebra ()=default
- virtual auto operator== (const multivector_t &val) const -> bool=0

  Test for equality of multivectors.
- virtual auto operator== (const Scalar_T &scr) const -> bool=0

  Test for equality of multivector and scalar.
- virtual auto operator+= (const multivector_t &rhs) -> multivector_t &=0

  Geometric sum.
- virtual auto operator+= (const Scalar_T &scr) -> multivector_t &=0

  Geometric sum of multivector and scalar.
- virtual auto operator-= (const multivector_t &rhs) -> multivector_t &=0

  Geometric difference.
- virtual auto operator-= (const Scalar_T &scr) -> multivector_t &=0

  Geometric difference of multivector and scalar.
- virtual auto operator- () const -> const multivector_t=0

  Unary -.
- virtual auto operator*= (const Scalar_T &scr) -> multivector_t &=0

  Product of multivector and scalar.
- virtual auto operator*= (const multivector_t &rhs) -> multivector_t &=0

  Geometric product.
- virtual auto operator%= (const multivector_t &rhs) -> multivector_t &=0

  Contraction.
- virtual auto operator&= (const multivector_t &rhs) -> multivector_t &=0

  Inner product.
- virtual auto operator^= (const multivector_t &rhs) -> multivector_t &=0

  Outer product.
- virtual auto operator/= (const Scalar_T &scr) -> multivector_t &=0

  Quotient of multivector and scalar.
- virtual auto operator/= (const multivector_t &rhs) -> multivector_t &=0

  Geometric quotient.
- virtual auto operator|= (const multivector_t &rhs) -> multivector_t &=0

  Transformation via twisted adjoint action.
- virtual auto inv () const -> const multivector_t=0

  Geometric multiplicative inverse.
- virtual auto pow (int m) const -> const multivector_t=0

  **this to the m*
- virtual auto outer_pow (int m) const -> const multivector_t=0

  Outer product power.
- virtual auto frame () const -> const index_set_t=0

  Subalgebra generated by all generators of terms of given multivector.
- virtual auto grade () const -> index_t=0

  Maximum of the grades of each term.
- virtual auto operator[ ] (const index_set_t ist) const -> Scalar_T=0

  Subscripting: map from index set to scalar coordinate.
- virtual auto operator() (index_t grade) const -> const multivector_t=0

  Pure grade-vector part.
- virtual auto scalar () const -> Scalar_T=0

  Scalar part.
- virtual auto pure () const -> const multivector_t=0

Pure part.

- virtual auto even () const -> const multivector_t=0

  Even part of multivector, sum of even grade terms.
- virtual auto odd () const -> const multivector_t=0

  Odd part of multivector, sum of odd grade terms.
- virtual auto vector_part () const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frame()
- virtual auto vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frm.
- virtual auto involute () const -> const multivector_t=0

  Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.
- virtual auto reverse () const -> const multivector_t=0

  Reversion, eg. {1}{2} -> {2}*{1}.*
- virtual auto conj () const -> const multivector_t=0

  Conjugation, reverse o involute == involute o reverse.
- virtual auto quad () const -> Scalar_T=0

  *Scalar_T quadratic form == (rev(x)*x)(0)*
- virtual auto norm () const -> Scalar_T=0

  Scalar_T norm == sum of norm of coordinates.
- virtual auto max_abs () const -> Scalar_T=0

  Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto truncated (const Scalar_T &limit=default_truncation) const -> const multivector_t=0

  Remove all terms with relative size smaller than limit.
- virtual auto isinf () const -> bool=0

  Check if a multivector contains any infinite values.
- virtual auto isnan () const -> bool=0

  Check if a multivector contains any IEEE NaN values.
- virtual void write (const std::string &msg="") const=0

  Write formatted multivector to output.
- virtual void write (std::ofstream &ofile, const std::string &msg="") const=0

  Write formatted multivector to file.

Static Public Member Functions

- static auto classname () -> const std::string

  Class name used in messages.
- static auto random (const index_set_t frm, Scalar_T fill=Scalar_T(1)) -> const multivector_t

  Random multivector within a frame.

Static Public Member Functions inherited from glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >

- static auto classname () -> const std::string

Private Types

- using var_term_t = class var_term
- using matrix_t = typename matrix_multi_t::matrix_t
- using sorted_map_t = std::map< index_set_t, Scalar_T, std::less<const index_set_t> >
- using map_t = std::unordered_map<index_set_t, Scalar_T, index_set_hash<LO, HI>>
- using framed_pair_t = std::pair<const multivector_t, const multivector_t>
- using size_type = typename map_t::size_type
- using iterator = typename map_t::iterator
- using const_iterator = typename map_t::const_iterator

Private Member Functions

- framed_multi (const hash_size_t &hash_size)

  Private constructor using hash_size.
- auto fold (const index_set_t frm) const -> multivector_t

  Subalgebra isomorphism: fold each term within the given frame.
- auto unfold (const index_set_t frm) const -> multivector_t

  Subalgebra isomorphism: unfold each term within the given frame.
- auto centre_pm4_qp4 (index_t &p, index_t &q) -> multivector_t &

  Subalgebra isomorphism: R_{p,q} to R_{p-4,q+4}.
- auto centre_pp4_qm4 (index_t &p, index_t &q) -> multivector_t &

  Subalgebra isomorphism: R_{p,q} to R_{p+4,q-4}.
- auto centre_qp1_pm1 (index_t &p, index_t &q) -> multivector_t &

  Subalgebra isomorphism: R_{p,q} to R_{q+1,p-1}.
- auto divide (const index_set_t ist) const -> const framed_pair_t

  Divide multivector into part divisible by index_set and remainder.
- auto fast (const index_t level, const bool odd) const -> const matrix_t

  Generalized FFT from multivector_t to matrix_t.

Friends

- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  class matrix_multi
- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  class framed_multi
- auto operator∗ (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto operator^ (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto operator& (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto operator% (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto star (const multivector_t &lhs, const multivector_t &rhs) -> Scalar_T
- auto operator/ (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto operator| (const multivector_t &lhs, const multivector_t &rhs) -> const multivector_t
- auto operator>> (std::istream &s, multivector_t &val) -> std::istream &
- auto operator<< (std::ostream &os, const multivector_t &val) -> std::ostream &
- auto operator<< (std::ostream &os, const term_t &term) -> std::ostream &
- auto exp (const multivector_t &val) -> const multivector_t

Additional Inherited Members

Static Public Attributes inherited from glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >

- static const index_t v_lo
- static const index_t v_hi
- static const Scalar_T default_truncation

    Default for truncation.

6.13.1 Detailed Description

template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P>
class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >

A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.

Definition at line 55 of file matrix_multi.h.

6.13.2 Member Typedef Documentation

6.13.2.1 const_iterator

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::const_iterator = typename map_t↩
::const_iterator  [private]
```

Definition at line 167 of file framed_multi.h.

6.13.2.2 error_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::error_t = error<multivector_t>
```

Definition at line 138 of file framed_multi.h.

6.13.2.3 framed_multi_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi_t = multivector_t
```

Definition at line 132 of file framed_multi.h.

6.13.2.4 framed_pair_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_pair_t = std::pair<const multivector_t,
const multivector_t>  [private]
```

Definition at line 164 of file framed_multi.h.

6.13.2.5 index_set_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::index_set_t = index_set<LO, HI>
```

Definition at line 135 of file framed_multi.h.

6.13.2.6 iterator

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::iterator = typename map_t::iterator
[private]
```

Definition at line 166 of file framed_multi.h.

6.13.2.7 map_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::map_t = std::unordered_map<index_set_t,
Scalar_T, index_set_hash<LO, HI>>  [private]
```

Definition at line 150 of file framed_multi.h.

6.13.2.8 matrix_multi_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi_t = matrix_multi<Scalar↩
_T,LO,HI,Tune_P >
```

Definition at line 139 of file framed_multi.h.

6.13.2.9 matrix_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::matrix_t = typename matrix_multi_t::matrix_t
[private]
```

Definition at line 148 of file framed_multi.h.

6.13.2.10 multivector_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::multivector_t = framed_multi
```

Definition at line 131 of file framed_multi.h.

6.13.2.11 scalar_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::scalar_t = Scalar_T
```

Definition at line 133 of file framed_multi.h.

6.13.2.12 size_type

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::size_type = typename map_t::size_type
[private]
```

Definition at line 165 of file framed_multi.h.

6.13.2.13 sorted_map_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::sorted_map_t = std::map< index_set_t,
Scalar_T, std::less<const index_set_t> >  [private]
```

Definition at line 149 of file framed_multi.h.

6.13.2.14 term_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::term_t = std::pair<const index_set_t,
Scalar_T>
```

Definition at line 136 of file framed_multi.h.

6.13.2.15 tune_p

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::tune_p = Tune_P
```

Definition at line 134 of file framed_multi.h.

6.13.2.16 var_term_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term_t = class var_term  [private]
```

Definition at line 147 of file framed_multi.h.

6.13.2.17 vector_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::vector_t = std::vector<Scalar_T>
```

Definition at line 137 of file framed_multi.h.

6.13.3 Constructor & Destructor Documentation

6.13.3.1 ~framed_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::~framed_multi () [override], [default]
```

Destructor.

6.13.3.2 framed_multi() [1/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi ()
```

Default constructor.

Definition at line 59 of file framed_multi_imp.h.

6.13.3.3 framed_multi() [2/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const hash_size_t & hash_size) [private]
```

Private constructor using hash_size.

Definition at line 66 of file framed_multi_imp.h.

6.13.3.4 framed_multi() [3/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a multivector with a different scalar type.

Definition at line 74 of file framed_multi_imp.h.

6.13.3.5 framed_multi() [4/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 85 of file framed_multi_imp.h.

References glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame().

6.13.3.6 framed_multi() [5/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const framed_multi_t & val,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 98 of file framed_multi_imp.h.

References glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame().

6.13.3.7 framed_multi() [6/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const index_set_t ist,
            const Scalar_T & crd = Scalar_T(1))
```

Construct a multivector from an index set and a scalar coordinate.

Definition at line 111 of file framed_multi_imp.h.

6.13.3.8 framed_multi() [7/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const index_set_t ist,
            const Scalar_T & crd,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from an index set and a scalar coordinate.

Definition at line 121 of file framed_multi_imp.h.

6.13.3.9 framed_multi() [8/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const Scalar_T & scr,
            const index_set_t frm = index_set_t())
```

Construct a multivector from a scalar (within a frame, if given)

Definition at line 134 of file framed_multi_imp.h.

6.13.3.10 framed_multi() [9/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const int scr,
            const index_set_t frm = index_set_t())
```

Construct a multivector from an int (within a frame, if given)

Definition at line 144 of file framed_multi_imp.h.

6.13.3.11 framed_multi() [10/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const vector_t & vec,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given vector.

Definition at line 154 of file framed_multi_imp.h.

References glucat::index_set< LO, HI >::count(), glucat::index_set< LO, HI >::max(), and glucat::index_set< LO, HI >::min().

6.13.3.12 framed_multi() [11/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const std::string & str)
```

Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 176 of file framed_multi_imp.h.

6.13.3.13 framed_multi() [12/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const std::string & str,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 192 of file framed_multi_imp.h.

6.13.3.14 framed_multi() [13/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const char * str) [inline]
```

Construct a multivector from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 209 of file framed_multi.h.

References glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi.

6.13.3.15 framed_multi() [14/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const char * str,
            const index_set_t frm,
            const bool prechecked = false) [inline]
```

Construct a multivector, within a given frame, from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 212 of file framed_multi.h.

References glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi.

6.13.3.16 framed_multi() [15/15]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
            const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a matrix_multi_t.

Definition at line 205 of file framed_multi_imp.h.

References _GLUCAT_HASH_SIZE_T, glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector glucat::matrix::inner(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix, glucat::matrix::nnz(), glucat::clifford_algebra< Sc and glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >::truncated().

6.13.4 Member Function Documentation

6.13.4.1 centre_pm4_qp4()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pm4_qp4 (
            index_t & p,
            index_t & q) -> multivector_t&  [private]
```

Subalgebra isomorphism: R_{p,q} to R_{p-4,q+4}.

Definition at line 1469 of file framed_multi_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi().

6.13.4.2 centre_pp4_qm4()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pp4_qm4 (
            index_t & p,
            index_t & q) -> multivector_t&  [private]
```

Subalgebra isomorphism: R_{p,q} to R_{p+4,q-4}.

Definition at line 1511 of file framed_multi_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi().

6.13.4.3 centre_qp1_pm1()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_qp1_pm1 (
            index_t & p,
            index_t & q) -> multivector_t&  [private]
```

Subalgebra isomorphism: R_{p,q} to R_{q+1,p-1}.

Definition at line 1553 of file framed_multi_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi().

6.13.4.4 classname()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::classname () -> const std::string
[static]
```

Class name used in messages.

Definition at line 50 of file framed_multi_imp.h.

6.13.4.5 divide()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::divide (
            const index_set_t ist) const -> const framed_pair_t  [private]
```

Divide multivector into part divisible by index_set and remainder.

Divide multivector into quotient with terms divisible by index set, and remainder.

Definition at line 1586 of file framed_multi_imp.h.

6.13.4.6 fast()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast (
            const index_t level,
            const bool odd) const -> const matrix_t  [private]
```

Generalized FFT from multivector_t to matrix_t.

Definition at line 1602 of file framed_multi_imp.h.

References glucat::matrix::kron(), glucat::odd(), glucat::scalar(), and glucat::matrix::unit().

6.13.4.7 fast_framed_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi () const -> const
framed_multi_t  [inline]
```

Use inverse generalized FFT to construct a framed_multi_t.

Definition at line 1700 of file framed_multi_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

6.13.4.8 fast_matrix_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_matrix_multi (
            const index_set_t frm) const -> const matrix_multi<Other_Scalar_T,LO,HI,Tune_P >
```

Use generalized FFT to construct a matrix_multi_t.

Definition at line 1668 of file framed_multi_imp.h.

References glucat::gen::offset_to_super, and glucat::pos_mod().

6.13.4.9 fold()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fold (
            const index_set_t frm) const -> multivector_t  [private]
```

Subalgebra isomorphism: fold each term within the given frame.

Definition at line 1434 of file framed_multi_imp.h.

6.13.4.10 nbr_terms()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::nbr_terms () const -> unsigned long
```

Number of terms.

Definition at line 1356 of file framed_multi_imp.h.

6.13.4.11 operator+=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::operator+= (
            const term_t & term) -> multivector_t&  [inline]
```

Add a term, if non-zero.

Insert a term into a multivector, add terms with same index set.

Geometric sum.

Geometric sum of multivector and scalar.

Definition at line 295 of file framed_multi_imp.h.

6.13.4.12 random()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::random (
            const index_set_t frm,
            Scalar_T fill = Scalar_T(1)) -> const multivector_t  [static]
```

Random multivector within a frame.

Definition at line 1058 of file framed_multi_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::random().

6.13.4.13 unfold()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::unfold (
            const index_set_t frm) const -> multivector_t  [private]
```

Subalgebra isomorphism: unfold each term within the given frame.

Definition at line 1451 of file framed_multi_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi().

6.13.5 Friends And Related Symbol Documentation

6.13.5.1 exp

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto exp (
            const multivector_t & val) -> const multivector_t  [friend]
```

6.13.5.2 framed_multi

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
friend class framed_multi  [friend]
```

Definition at line 143 of file framed_multi.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), and glucat::framed_multi< Scalar_T, LO, HI, Tune

6.13.5.3 matrix_multi

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
friend class matrix_multi  [friend]
```

Definition at line 141 of file framed_multi.h.

6.13.5.4 operator%

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator% (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.5 operator&

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator& (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.6 operator*

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator* (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.7 operator/

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.8 operator<< [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (
            std::ostream & os,
            const multivector_t & val) -> std::ostream &  [friend]
```

6.13.5.9 operator<< [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (
            std::ostream & os,
            const term_t & term) -> std::ostream &  [friend]
```

6.13.5.10 operator>>

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator>> (
            std::istream & s,
            multivector_t & val) -> std::istream &  [friend]
```

6.13.5.11 operator^

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator^ (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.12 operator" |

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator| (
            const multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.13 star

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto star (
            const multivector_t & lhs,
            const multivector_t & rhs) -> Scalar_T  [friend]
```

The documentation for this class was generated from the following files:

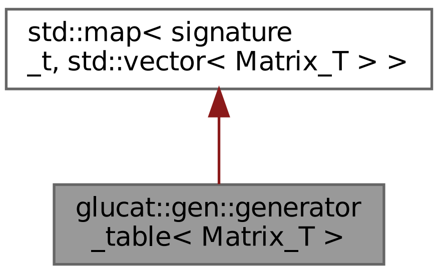
- glucat/framed_multi.h
- glucat/matrix_multi.h
- glucat/framed_multi_imp.h

6.14 glucat::gen::generator_table< Matrix_T > Class Template Reference

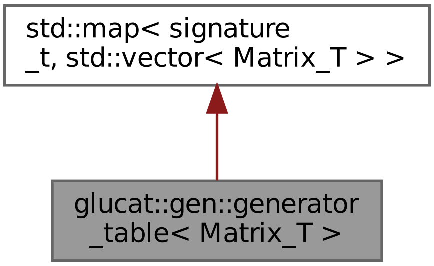
Table of generators for specific signatures.

```
#include <generation.h>
```

Inheritance diagram for glucat::gen::generator_table< Matrix_T >:



Collaboration diagram for glucat::gen::generator_table< Matrix_T >:



Public Member Functions

- auto operator() (const index_t p, const index_t q) -> const Matrix_T ∗
  Pointer to generators for a specific signature.
- generator_table (const generator_table &)=delete
- auto operator= (const generator_table &) -> generator_table &=delete

Static Public Member Functions

- static auto generator () -> generator_table< Matrix_T > &
  Single instance of generator table.

Private Member Functions

- auto gen_vector (const index_t p, const index_t q) -> const std::vector< Matrix_T > &
  Construct a vector of generators for a specific signature.
- void gen_from_pm1_qm1 (const std::vector< Matrix_T > &old, const signature_t sig)
  Construct generators for p,q given generators for p-1,q-1.
- void gen_from_pm4_qp4 (const std::vector< Matrix_T > &old, const signature_t sig)
  Construct generators for p,q given generators for p-4,q+4.
- void gen_from_pp4_qm4 (const std::vector< Matrix_T > &old, const signature_t sig)
  Construct generators for p,q given generators for p+4,q-4.
- void gen_from_qp1_pm1 (const std::vector< Matrix_T > &old, const signature_t sig)
  Construct generators for p,q given generators for q+1,p-1.
- generator_table ()=default
- ∼generator_table ()=default

Friends

- class friend_for_private_destructor

6.14.1 Detailed Description

template<class Matrix_T>
class glucat::gen::generator_table< Matrix_T >

Table of generators for specific signatures.

Definition at line 52 of file generation.h.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 generator_table() [1/2]

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::generator_table ()  [private], [default]
```

6.14.2.2 ∼generator_table()

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::~generator_table ()  [private], [default]
```

6.14.2.3 generator_table() [2/2]

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::generator_table (
            const generator_table< Matrix_T > & )  [delete]
```

6.14.3 Member Function Documentation

6.14.3.1 gen_from_pm1_qm1()

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1 (
            const std::vector< Matrix_T > & old,
            const signature_t sig)  [private]
```

Construct generators for p,q given generators for p-1,q-1.

Definition at line 127 of file generation_imp.h.

References glucat::matrix::mono_kron(), and glucat::matrix::unit().

6.14.3.2 gen_from_pm4_qp4()

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pm4_qp4 (
            const std::vector< Matrix_T > & old,
            const signature_t sig)  [private]
```

Construct generators for p,q given generators for p-4,q+4.

Definition at line 165 of file generation_imp.h.

References glucat::matrix::mono_prod().

6.14.3.3 gen_from_pp4_qm4()

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pp4_qm4 (
            const std::vector< Matrix_T > & old,
            const signature_t sig)  [private]
```

Construct generators for p,q given generators for p+4,q-4.

Definition at line 198 of file generation_imp.h.

References glucat::matrix::mono_prod().

6.14.3.4 gen_from_qp1_pm1()

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_qp1_pm1 (
            const std::vector< Matrix_T > & old,
            const signature_t sig)  [private]
```

Construct generators for p,q given generators for q+1,p-1.

Definition at line 231 of file generation_imp.h.

References glucat::matrix::mono_prod().

6.14.3.5 gen_vector()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::gen_vector (
            const index_t p,
            const index_t q) -> const std::vector<Matrix_T>&  [private]
```

Construct a vector of generators for a specific signature.

Definition at line 79 of file generation_imp.h.

References glucat::pos_mod(), and glucat::matrix::unit().

6.14.3.6 generator()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::generator () -> generator_table<Matrix_T>&
[static]
```

Single instance of generator table.

Definition at line 49 of file generation_imp.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element().

6.14.3.7 operator()()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::operator() (
            const index_t p,
            const index_t q) -> const Matrix_T*  [inline]
```

Pointer to generators for a specific signature.

Definition at line 58 of file generation_imp.h.

References glucat::gen::offset_to_super, and glucat::pos_mod().

6.14.3.8 operator=()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::operator= (
            const generator_table< Matrix_T > & ) -> generator_table &=delete  [delete]
```

6.14.4 Friends And Related Symbol Documentation

6.14.4.1 friend_for_private_destructor

```
template<class Matrix_T >
friend class friend_for_private_destructor  [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE `http://doc.ece.uci.edu`

Definition at line 75 of file generation.h.

The documentation for this class was generated from the following files:

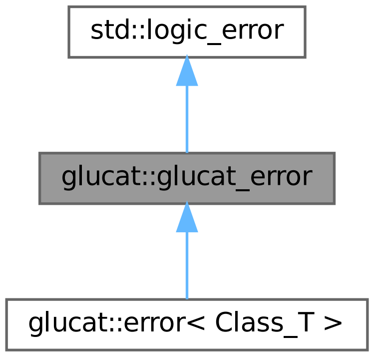
- glucat/generation.h
- glucat/generation_imp.h

6.15 glucat::glucat_error Class Reference

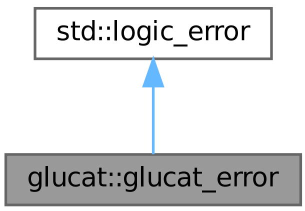
Abstract exception class.

```
#include <errors.h>
```

Inheritance diagram for glucat::glucat_error:



Collaboration diagram for glucat::glucat_error:



Public Member Functions

- glucat_error (const std::string &context, const std::string &msg)
- ∼glucat_error () noexcept override=default
- virtual auto heading () const noexcept -> const std::string=0
- virtual auto classname () const noexcept -> const std::string=0
- virtual void print_error_msg () const =0

Public Attributes

- std::string name

6.15.1 Detailed Description

Abstract exception class.

Definition at line 41 of file errors.h.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 glucat_error()

```
glucat::glucat_error::glucat_error (
            const std::string & context,
            const std::string & msg)  [inline]
```

Definition at line 44 of file errors.h.

6.15.2.2 ∼glucat_error()

```
glucat::glucat_error::~glucat_error ()  [override], [default], [noexcept]
```

6.15.3 Member Function Documentation

6.15.3.1 classname()

```
virtual auto glucat::glucat_error::classname () const -> const std::string  [pure virtual],
[noexcept]
```

Implemented in glucat::error< Class_T >.

6.15.3.2 heading()

```
virtual auto glucat::glucat_error::heading () const -> const std::string  [pure virtual],
[noexcept]
```

Implemented in glucat::error< Class_T >.

6.15.3.3 print_error_msg()

```
virtual void glucat::glucat_error::print_error_msg () const  [pure virtual]
```

Implemented in glucat::error< Class_T >.

6.15.4 Member Data Documentation

6.15.4.1 name

```
std::string glucat::glucat_error::name
```

Definition at line 51 of file errors.h.

The documentation for this class was generated from the following file:

- glucat/errors.h

6.16 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t Class Reference

Public Member Functions

- hash_size_t (size_t hash_size)
- auto operator() () const -> size_t

Private Attributes

- size_t n

6.16.1 Detailed Description

template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P>
class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t

Definition at line 152 of file framed_multi.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 hash_size_t()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::hash_size_t (
            size_t hash_size) [inline]
```

Definition at line 155 of file framed_multi.h.

6.16.3 Member Function Documentation

6.16.3.1 operator()()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::operator() () const ->
size_t  [inline]
```

Definition at line 158 of file framed_multi.h.

References glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::n.

6.16.4 Member Data Documentation

6.16.4.1 n

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
size_t glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::n  [private]
```

Definition at line 161 of file framed_multi.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::operator()().

The documentation for this class was generated from the following file:

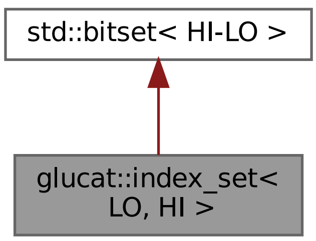
- glucat/framed_multi.h

6.17 glucat::index_set< LO, HI > Class Template Reference

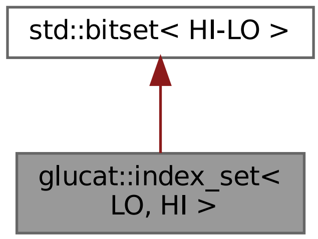
Index set class based on std::bitset<> in Gnu standard C++ library.

```
#include <index_set.h>
```

Inheritance diagram for glucat::index_set< LO, HI >:



Collaboration diagram for glucat::index_set< LO, HI >:



Classes

- class reference

    Index set member reference.

Public Types

- using index_set_t = index_set
- using index_pair_t = std::pair< index_t, index_t >

Public Member Functions

- index_set ()=default

    Default constructor creates an empty set.
- index_set (const bitset_t bst)

    Constructor from bitset_t.
- index_set (const index_t idx)

    Constructor from index.
- index_set (const set_value_t folded_val, const index_set_t frm, const bool prechecked=false)

    Constructor from set value of an index set folded within the given frame.
- index_set (const index_pair_t &range, const bool prechecked=false)

    Constructor from range of indices from range.first to range.second.
- index_set (const std::string &str)

    Constructor from string.
- auto operator== (const index_set_t rhs) const -> bool

    Equality.
- auto operator!= (const index_set_t rhs) const -> bool

    Inequality.
- auto operator∼ () const -> index_set_t

    Set complement: not.
- auto operator^= (const index_set_t rhs) -> index_set_t &

    Symmetric set difference: exclusive or.
- auto operator&= (const index_set_t rhs) -> index_set_t &

Set intersection: and.
- auto operator\|= (const index_set_t rhs) -> index_set_t &

  Set union: or.
- auto operator[ ] (const index_t idx) const -> bool

  Subscripting: Test idx for membership: test value of bit idx.
- auto test (const index_t idx) const -> bool

  Test idx for membership: test value of bit idx.
- auto set () -> index_set_t &

  Include all indices except 0: set all bits except 0.
- auto set (const index_t idx) -> index_set_t &

  Include idx: Set bit at idx if idx != 0.
- auto set (const index_t idx, const int val) -> index_set_t &

  Set membership of idx to val if idx != 0: Set bit at idx to val if idx != 0.
- auto reset () -> index_set_t &

  Make set empty: Set all bits to 0.
- auto reset (const index_t idx) -> index_set_t &

  Exclude idx: Set bit at idx to 0.
- auto flip () -> index_set_t &

  Set complement, except 0: flip all bits, except 0.
- auto flip (const index_t idx) -> index_set_t &

  Complement membership of idx if idx != 0: flip bit at idx if idx != 0.
- auto count () const -> index_t

  Cardinality: Number of indices included in set.
- auto count_neg () const -> index_t

  Number of negative indices included in set.
- auto count_pos () const -> index_t

  Number of positive indices included in set.
- auto min () const -> index_t

  Minimum member.
- auto max () const -> index_t

  Maximum member.
- auto operator< (const index_set_t rhs) const -> bool

  Less than operator used for comparisons, map, etc.
- auto is_contiguous () const -> bool

  Determine if the index set is contiguous, ie. has no gaps.
- auto fold () const -> const index_set_t

  Fold this index set within itself as a frame.
- auto fold (const index_set_t frm, const bool prechecked=false) const -> const index_set_t

  Fold this index set within the given frame.
- auto unfold (const index_set_t frm, const bool prechecked=false) const -> const index_set_t

  Unfold this index set within the given frame.
- auto value_of_fold (const index_set_t frm) const -> set_value_t

  The set value of the fold of this index set within the given frame.
- auto sign_of_mult (const index_set_t ist) const -> int

  Sign of geometric product of two Clifford basis elements.
- auto sign_of_square () const -> int

  Sign of geometric square of a Clifford basis element.
- auto hash_fn () const -> size_t

  Hash function.
- auto operator[ ] (index_t idx) -> reference

  Subscripting: Element access.

Static Public Member Functions

- static auto classname () -> const std::string

Static Public Attributes

- static const index_t v_lo = LO
- static const index_t v_hi = HI

Private Types

- using bitset_t = std::bitset<HI - LO>
- using error_t = error<index_set>

Private Member Functions

- BOOST_STATIC_ASSERT ((LO<=0) &&(0<=HI) &&(LO< HI) &&(-LO< _GLUCAT_BITS_PER_ULONG) &&(HI< _GLUCAT_BITS_PER_ULONG) &&(HI-LO<=_GLUCAT_BITS_PER_ULONG))
- auto lex_less_than (const index_set_t rhs) const -> bool

    Lexicographic ordering of two sets: ∗this < rhs.

Friends

- class reference
- auto operator^ (const index_set_t &lhs, const index_set_t &rhs) -> const index_set_t
- auto operator& (const index_set_t &lhs, const index_set_t &rhs) -> const index_set_t
- auto operator| (const index_set_t &lhs, const index_set_t &rhs) -> const index_set_t
- auto compare (const index_set_t &lhs, const index_set_t &rhs) -> int

6.17.1 Detailed Description

template<const index_t LO, const index_t HI>
class glucat::index_set< LO, HI >

Index set class based on std::bitset<> in Gnu standard C++ library.

Definition at line 73 of file index_set.h.

6.17.2 Member Typedef Documentation

6.17.2.1 bitset_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::bitset_t = std::bitset<HI - LO>  [private]
```

Definition at line 81 of file index_set.h.

6.17.2.2 error_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::error_t = error<index_set>  [private]
```

Definition at line 82 of file index_set.h.

6.17.2.3 index_pair_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::index_pair_t = std::pair<index_t, index_t>
```

Definition at line 85 of file index_set.h.

6.17.2.4 index_set_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::index_set_t = index_set
```

Definition at line 84 of file index_set.h.

6.17.3 Constructor & Destructor Documentation

6.17.3.1 index_set() [1/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set ()  [default]
```

Default constructor creates an empty set.

6.17.3.2 index_set() [2/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
             const bitset_t bst)
```

Constructor from bitset_t.

Definition at line 61 of file index_set_imp.h.

6.17.3.3 index_set() [3/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
             const index_t idx)
```

Constructor from index.

Constructor from index value.

Definition at line 55 of file index_set_imp.h.

6.17.3.4 index_set() [4/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
            const set_value_t folded_val,
            const index_set_t frm,
            const bool prechecked = false)
```

Constructor from set value of an index set folded within the given frame.

Definition at line 68 of file index_set_imp.h.

References glucat::index_set< LO, HI >::count(), glucat::index_set< LO, HI >::fold(), glucat::index_set< LO, HI >::min(), and glucat::index_set< LO, HI >::unfold().

6.17.3.5 index_set() [5/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
            const index_pair_t & range,
            const bool prechecked = false)
```

Constructor from range of indices from range.first to range.second.

Definition at line 82 of file index_set_imp.h.

6.17.3.6 index_set() [6/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
            const std::string & str)
```

Constructor from string.

Definition at line 102 of file index_set_imp.h.

6.17.4 Member Function Documentation

6.17.4.1 BOOST_STATIC_ASSERT()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::BOOST_STATIC_ASSERT (
            (LO<=0) &&(0<=HI) &&(LO< HI) &&(-LO< _GLUCAT_BITS_PER_ULONG) &&(HI< _GLUCAT_↩
BITS_PER_ULONG) &&(HI-LO<=_GLUCAT_BITS_PER_ULONG) )  [private]
```

6.17.4.2 classname()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::classname () -> const std::string  [inline], [static]
```

Definition at line 49 of file index_set_imp.h.

6.17.4.3 count()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count () const -> index_t  [inline]
```

Cardinality: Number of indices included in set.

Definition at line 344 of file index_set_imp.h.

Referenced by glucat::index_set< LO, HI >::count_neg(), glucat::index_set< LO, HI >::count_pos(), glucat::framed_multi< Scalar_T
glucat::index_set< LO, HI >::index_set(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi().

6.17.4.4 count_neg()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count_neg () const -> index_t  [inline]
```

Number of negative indices included in set.

Definition at line 364 of file index_set_imp.h.

References glucat::index_set< LO, HI >::count().

6.17.4.5 count_pos()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count_pos () const -> index_t  [inline]
```

Number of positive indices included in set.

Definition at line 376 of file index_set_imp.h.

References glucat::index_set< LO, HI >::count().

6.17.4.6 flip() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::flip () -> index_set_t&  [inline]
```

Set complement, except 0: flip all bits, except 0.

Definition at line 319 of file index_set_imp.h.

6.17.4.7 flip() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::flip (
             const index_t idx) -> index_set_t&  [inline]
```

Complement membership of idx if idx != 0: flip bit at idx if idx != 0.

Definition at line 330 of file index_set_imp.h.

6.17.4.8 fold() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::fold () const -> const index_set_t  [inline]
```

Fold this index set within itself as a frame.

Definition at line 747 of file index_set_imp.h.

Referenced by glucat::index_set< LO, HI >::index_set().

6.17.4.9 fold() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::fold (
            const index_set_t frm,
            const bool prechecked = false) const -> const index_set_t
```

Fold this index set within the given frame.

Definition at line 755 of file index_set_imp.h.

6.17.4.10 hash_fn()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::hash_fn () const -> size_t  [inline]
```

Hash function.

Definition at line 950 of file index_set_imp.h.

6.17.4.11 is_contiguous()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::is_contiguous () const -> bool  [inline]
```

Determine if the index set is contiguous, ie. has no gaps.

Determine if the index set is contiguous, ie. has no gaps when 0 is included.

Definition at line 732 of file index_set_imp.h.

6.17.4.12 lex_less_than()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::lex_less_than (
            const index_set_t rhs) const -> bool  [inline], [private]
```

Lexicographic ordering of two sets: $*$this $<$ rhs.

Definition at line 588 of file index_set_imp.h.

6.17.4.13 max()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::max () const -> index_t
```

Maximum member.

Maximum member, or 0 if none.

Definition at line 550 of file index_set_imp.h.

Referenced by PyClical.index_set::__iter__(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi().

6.17.4.14 min()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::min () const -> index_t
```

Minimum member.

Minimum member, or 0 if none.

Definition at line 461 of file index_set_imp.h.

Referenced by PyClical.index_set::__iter__(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), glucat::index_set< LO, HI >::index_set(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi().

6.17.4.15 operator"!=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator!= (
            const index_set_t rhs) const -> bool  [inline]
```

Inequality.

Definition at line 130 of file index_set_imp.h.

6.17.4.16 operator&=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator&= (
            const index_set_t rhs) -> index_set_t&  [inline]
```

Set intersection: and.

Definition at line 174 of file index_set_imp.h.

6.17.4.17 operator<()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator< (
            const index_set_t rhs) const -> bool  [inline]
```

Less than operator used for comparisons, map, etc.

Definition at line 596 of file index_set_imp.h.

6.17.4.18 operator==()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator== (
            const index_set_t rhs) const -> bool  [inline]
```

Equality.

Definition at line 119 of file index_set_imp.h.

6.17.4.19 operator[]() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator[] (
            const index_t idx) const -> bool  [inline]
```

Subscripting: Test idx for membership: test value of bit idx.

Definition at line 232 of file index_set_imp.h.

6.17.4.20 operator[]() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator[] (
            index_t idx) -> reference  [inline]
```

Subscripting: Element access.

Definition at line 224 of file index_set_imp.h.

6.17.4.21 operator^=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator^= (
            const index_set_t rhs) -> index_set_t&  [inline]
```

Symmetric set difference: exclusive or.

Definition at line 149 of file index_set_imp.h.

6.17.4.22 operator"|=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator|= (
            const index_set_t rhs) -> index_set_t&  [inline]
```

Set union: or.

Definition at line 199 of file index_set_imp.h.

6.17.4.23 operator∼()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator~ () const -> index_set_t  [inline]
```

Set complement: not.

Definition at line 141 of file index_set_imp.h.

6.17.4.24 reset() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reset () -> index_set_t&  [inline]
```

Make set empty: Set all bits to 0.

Definition at line 294 of file index_set_imp.h.

6.17.4.25 reset() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reset (
            const index_t idx) -> index_set_t&  [inline]
```

Exclude idx: Set bit at idx to 0.

Definition at line 305 of file index_set_imp.h.

6.17.4.26 set() [1/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set () -> index_set_t&  [inline]
```

Include all indices except 0: set all bits except 0.

Definition at line 255 of file index_set_imp.h.

6.17.4.27 set() [2/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set (
            const index_t idx) -> index_set_t&  [inline]
```

Include idx: Set bit at idx if idx != 0.

Definition at line 266 of file index_set_imp.h.

6.17.4.28 set() [3/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set (
            const index_t idx,
            const int val) -> index_set_t&  [inline]
```

Set membership of idx to val if idx != 0: Set bit at idx to val if idx != 0.

Definition at line 280 of file index_set_imp.h.

6.17.4.29 sign_of_mult()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::sign_of_mult (
            const index_set_t ist) const -> int
```

Sign of geometric product of two Clifford basis elements.

Definition at line 880 of file index_set_imp.h.

References glucat::inverse_gray(), and glucat::inverse_reversed_gray().

6.17.4.30 sign_of_square()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::sign_of_square () const -> int  [inline]
```

Sign of geometric square of a Clifford basis element.

Definition at line 930 of file index_set_imp.h.

6.17.4.31 test()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::test (
            const index_t idx) const -> bool  [inline]
```

Test idx for membership: test value of bit idx.

Definition at line 240 of file index_set_imp.h.

6.17.4.32 unfold()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::unfold (
            const index_set_t frm,
            const bool prechecked = false) const -> const index_set_t
```

Unfold this index set within the given frame.

Definition at line 794 of file index_set_imp.h.

Referenced by glucat::index_set< LO, HI >::index_set().

6.17.4.33 value_of_fold()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::value_of_fold (
            const index_set_t frm) const -> set_value_t  [inline]
```

The set value of the fold of this index set within the given frame.

Definition at line 829 of file index_set_imp.h.

6.17.5 Friends And Related Symbol Documentation

6.17.5.1 compare

```
template<const index_t LO, const index_t HI>
auto compare (
            const index_set_t & lhs,
            const index_set_t & rhs) -> int  [friend]
```

6.17.5.2 operator&

```
template<const index_t LO, const index_t HI>
auto operator& (
            const index_set_t & lhs,
            const index_set_t & rhs) -> const index_set_t  [friend]
```

6.17.5.3 operator^

```
template<const index_t LO, const index_t HI>
auto operator^ (
            const index_set_t & lhs,
            const index_set_t & rhs) -> const index_set_t  [friend]
```

6.17.5.4 operator" |

```
template<const index_t LO, const index_t HI>
auto operator| (
            const index_set_t & lhs,
            const index_set_t & rhs) -> const index_set_t  [friend]
```

6.17.5.5 reference

```
template<const index_t LO, const index_t HI>
friend class reference  [friend]
```

Definition at line 174 of file index_set.h.

6.17.6 Member Data Documentation

6.17.6.1 v_hi

```
template<const index_t LO, const index_t HI>
const index_t glucat::index_set< LO, HI >::v_hi = HI  [static]
```

Definition at line 88 of file index_set.h.

6.17.6.2 v_lo

```
template<const index_t LO, const index_t HI>
const index_t glucat::index_set< LO, HI >::v_lo = LO  [static]
```

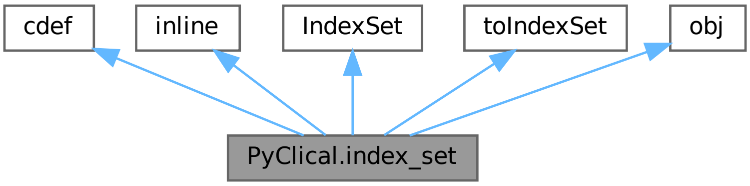
Definition at line 87 of file index_set.h.

The documentation for this class was generated from the following files:

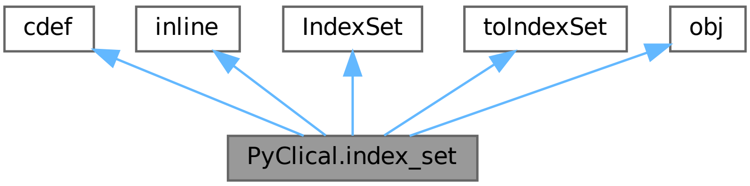
- glucat/index_set.h
- glucat/index_set_imp.h

6.18 PyClical.index_set Class Reference

Inheritance diagram for PyClical.index_set:



Collaboration diagram for PyClical.index_set:



Public Member Functions

- __cinit__ (self, other=0)
- __dealloc__ (self)
- __richcmp__ (lhs, rhs, int, op)
- __setitem__ (self, idx, val)
- __getitem__ (self, idx)
- __contains__ (self, idx)
- __iter__ (self)
- __invert__ (self)
- __xor__ (lhs, rhs)
- __ixor__ (self, rhs)
- __and__ (lhs, rhs)
- __iand__ (self, rhs)
- __or__ (lhs, rhs)
- __ior__ (self, rhs)
- count (self)
- count_neg (self)
- count_pos (self)
- min (self)
- max (self)

- hash_fn (self)
- sign_of_mult (self, rhs)
- sign_of_square (self)
- __repr__ (self)
- __str__ (self)

Public Attributes

- instance = new IndexSet((<index_set>other).unwrap())
- bool instance = True

6.18.1 Detailed Description

```
Return the C++ IndexSet instance wrapped by index_set(obj).

Python class index_set wraps C++ class IndexSet.
```

Definition at line 46 of file PyClical.pyx.

6.18.2 Member Function Documentation

6.18.2.1 __and__()

```
PyClical.index_set.__and__ (
              lhs,
              rhs)
```

```
Set intersection: and.

>>> print(index_set({1}) & index_set({2}))
{}
>>> print(index_set({1,2}) & index_set({2}))
{2}
```

Definition at line 271 of file PyClical.pyx.

6.18.2.2 __cinit__()

```
PyClical.index_set.__cinit__ (
              self,
              other = 0)
```

```
Construct an object of type index_set.

>>> print(index_set(1))
{1}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set(index_set({1,2})))
{1,2}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set("{1,2,1}"))
{1,2}
>>> print(index_set(""))
{}
```

Definition at line 74 of file PyClical.pyx.

6.18.2.3 __contains__()

```
PyClical.index_set.__contains__ (
              self,
              idx)
```

```
Check that an index_set object contains the index idx: idx in self.

>>> 1 in index_set({1})
True
>>> 2 in index_set({1})
False
>>> -1 in index_set({2})
False
>>> 1 in index_set({2})
False
>>> 2 in index_set({2})
True
>>> 33 in index_set({2})
False
```

Definition at line 210 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.18.2.4 __dealloc__()

```
PyClical.index_set.__dealloc__ (
              self)
```

```
Clean up by deallocating the instance of C++ class IndexSet.
```

Definition at line 116 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.18.2.5 __getitem__()

```
PyClical.index_set.__getitem__ (
              self,
              idx)
```

```
Get the value of an index_set object at an index.

>>> index_set({1})[1]
True
>>> index_set({1})[2]
False
>>> index_set({2})[-1]
False
>>> index_set({2})[1]
False
>>> index_set({2})[2]
True
>>> index_set({2})[33]
False
```

Definition at line 191 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.18.2.6 __iand__()

```
PyClical.index_set.__iand__ (
              self,
              rhs)
```

```
Set intersection: and.

>>> x = index_set({1}); x &= index_set({2}); print(x)
{}
>>> x = index_set({1,2}); x &= index_set({2}); print(x)
{2}
```

Definition at line 282 of file PyClical.pyx.

6.18.2.7 __invert__()

```
PyClical.index_set.__invert__ (
              self)
```

```
Set complement: not.

>>> print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,
{-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
```

Definition at line 240 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.18.2.8 __ior__()

```
PyClical.index_set.__ior__ (
              self,
              rhs)
```

```
Set union: or.

>>> x = index_set({1}); x |= index_set({2}); print(x)
{1,2}
>>> x = index_set({1,2}); x |= index_set({2}); print(x)
{1,2}
```

Definition at line 304 of file PyClical.pyx.

6.18.2.9 __iter__()

```
PyClical.index_set.__iter__ (
              self)
```

```
Iterate over the indices of an index_set.

>>> for i in index_set({-3,4,7}):print(i, end=",")
-3,4,7,
```

Definition at line 229 of file PyClical.pyx.

References glucat::index_set< LO, HI >.max(), glucat::index_set< DEFAULT_LO, DEFAULT_HI >.max(), PyClical.index_set.max(), glucat::index_set< LO, HI >.min(), glucat::index_set< DEFAULT_LO, DEFAULT_HI >.min(), and PyClical.index_set.min().

6.18.2.10 __ixor__()

```
PyClical.index_set.__ixor__ (
              self,
              rhs)
```

```
Symmetric set difference: exclusive or.

>>> x = index_set({1}); x ^= index_set({2}); print(x)
{1,2}
>>> x = index_set({1,2}); x ^= index_set({2}); print(x)
{1}
```

Definition at line 260 of file PyClical.pyx.

6.18.2.11 __or__()

```
PyClical.index_set.__or__ (
              lhs,
              rhs)
```

```
Set union: or.

>>> print(index_set({1}) | index_set({2}))
{1,2}
>>> print(index_set({1,2}) | index_set({2}))
{1,2}
```

Definition at line 293 of file PyClical.pyx.

6.18.2.12 __repr__()

```
PyClical.index_set.__repr__ (
              self)
```

```
The "official" string representation of self.

>>> index_set({1,2}).__repr__()
'index_set({1,2})'
>>> repr(index_set({1,2}))
'index_set({1,2})'
```

Definition at line 384 of file PyClical.pyx.

References index_set_to_repr().

6.18.2.13 __richcmp__()

```
PyClical.index_set.__richcmp__ (
              lhs,
              rhs,
              int,
              op)
```

```
Compare two objects of class index_set.

>>> index_set(1) == index_set({1})
True
>>> index_set({1}) != index_set({1})
False
>>> index_set({1}) != index_set({2})
True
>>> index_set({1}) == index_set({2})
False
>>> index_set({1}) < index_set({2})
True
>>> index_set({1}) <= index_set({2})
True
>>> index_set({1}) > index_set({2})
False
>>> index_set({1}) >= index_set({2})
False
```

Definition at line 122 of file PyClical.pyx.

6.18.2.14 __setitem__()

```
PyClical.index_set.__setitem__ (
              self,
              idx,
              val)
```

```
Set the value of an index_set object at index idx to value val.

>>> s=index_set({1}); s[2] = True; print(s)
{1,2}
>>> s=index_set({1,2}); s[1] = False; print(s)
{2}
```

Definition at line 179 of file PyClical.pyx.

References PyClical.clifford.instance, and PyClical.index_set.instance.

6.18.2.15 __str__()

```
PyClical.index_set.__str__ (
              self)
```

```
The "informal" string representation of self.

>>> index_set({1,2}).__str__()
'{1,2}'
>>> str(index_set({1,2}))
'{1,2}'
```

Definition at line 395 of file PyClical.pyx.

References index_set_to_str().

6.18.2.16 __xor__()

```
PyClical.index_set.__xor__ (
              lhs,
              rhs)
```

```
Symmetric set difference: exclusive or.

>>> print(index_set({1}) ^ index_set({2}))
{1,2}
>>> print(index_set({1,2}) ^ index_set({2}))
{1}
```

Definition at line 249 of file PyClical.pyx.

6.18.2.17 count()

```
PyClical.index_set.count (
              self)
```

```
Cardinality: Number of indices included in set.

>>> index_set({-1,1,2}).count()
3
```

Definition at line 315 of file PyClical.pyx.

References PyClical.index_set.count(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.index_set.count().

6.18.2.18 count_neg()

```
PyClical.index_set.count_neg (
              self)
```

```
Number of negative indices included in set.

>>> index_set({-1,1,2}).count_neg()
1
```

Definition at line 324 of file PyClical.pyx.

References PyClical.index_set.count_neg(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.index_set.count_neg().

6.18.2.19 count_pos()

```
PyClical.index_set.count_pos (
              self)
```

```
Number of positive indices included in set.

>>> index_set({-1,1,2}).count_pos()
2
```

Definition at line 333 of file PyClical.pyx.

References PyClical.index_set.count_pos(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.index_set.count_pos().

6.18.2.20 hash_fn()

```
PyClical.index_set.hash_fn (
              self)
```

```
Hash function.
```

Definition at line 360 of file PyClical.pyx.

References PyClical.index_set.hash_fn(), PyClical.clifford.instance, and PyClical.index_set.instance.

Referenced by PyClical.index_set.hash_fn().

6.18.2.21 max()

```
PyClical.index_set.max (
              self)
```

```
Maximum member.

>>> index_set({-1,1,2}).max()
2
```

Definition at line 351 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.index_set.max().

Referenced by PyClical.index_set.__iter__(), and PyClical.index_set.max().

6.18.2.22 min()

```
PyClical.index_set.min (
              self)
```

```
Minimum member.

>>> index_set({-1,1,2}).min()
-1
```

Definition at line 342 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.index_set.min().

Referenced by PyClical.index_set.__iter__(), and PyClical.index_set.min().

6.18.2.23 sign_of_mult()

```
PyClical.index_set.sign_of_mult (
              self,
              rhs)
```

```
Sign of geometric product of two Clifford basis elements.

>>> s = index_set({1,2}); t=index_set({-1}); s.sign_of_mult(t)
1
```

Definition at line 366 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.index_set.sign_of_mult().

Referenced by PyClical.index_set.sign_of_mult().

6.18.2.24 sign_of_square()

```
PyClical.index_set.sign_of_square (
              self)
```

```
Sign of geometric square of a Clifford basis element.

>>> s = index_set({1,2}); s.sign_of_square()
-1
```

Definition at line 375 of file PyClical.pyx.

References PyClical.clifford.instance, PyClical.index_set.instance, and PyClical.index_set.sign_of_square().

Referenced by PyClical.index_set.sign_of_square().

6.18.3 Member Data Documentation

6.18.3.1 instance [1/2]

```
PyClical.index_set.instance = new IndexSet((<index_set>other).unwrap())
```

Definition at line 95 of file PyClical.pyx.

Referenced by PyClical.clifford.__call__(), PyClical.index_set.__contains__(), PyClical.clifford.__dealloc__(), PyClical.index_set.__dealloc__(), PyClical.clifford.__getitem__(), PyClical.index_set.__getitem__(), PyClical.index_set.__invert__(), PyClical.clifford.__neg__(), PyClical.index_set.__setitem__(), PyClical.clifford.conj(), PyClical.index_set.count(), PyClical.index_set.count_neg(), PyClical.index_set.count_pos(), PyClical.clifford.even(), PyClical.clifford.frame(), PyClical.index_set.hash_fn(), PyClical.clifford.inv(), PyClical.clifford.involute(), PyClical.clifford.isinf(), PyClical.clifford.isnan(), PyClical.index_set.max(), PyClical.clifford.max_abs(), PyClical.index_set.min(), PyClical.clifford.norm(), PyClical.clifford.odd(), PyClical.clifford.outer_pow(), PyClical.clifford.pow(), PyClical.clifford.pure(), PyClical.clifford.quad(), PyClical.clifford.reverse(), PyClical.clifford.scalar(), PyClical.index_set.sign_of_mult(), PyClical.index_set.sign_of_square(), PyClical.clifford.truncated(), and PyClical.clifford.vector_part().

6.18.3.2 instance [2/2]

```
bool PyClical.index_set.instance = True
```

Definition at line 100 of file PyClical.pyx.

Referenced by PyClical.clifford.__call__(), PyClical.index_set.__contains__(), PyClical.clifford.__dealloc__(), PyClical.index_set.__dealloc__(), PyClical.clifford.__getitem__(), PyClical.index_set.__getitem__(), PyClical.index_set.__invert__(), PyClical.clifford.__neg__(), PyClical.index_set.__setitem__(), PyClical.clifford.conj(), PyClical.index_set.count(), PyClical.index_set.count_neg(), PyClical.index_set.count_pos(), PyClical.clifford.even(), PyClical.clifford.frame(), PyClical.index_set.hash_fn(), PyClical.clifford.inv(), PyClical.clifford.involute(), PyClical.clifford.isinf(), PyClical.clifford.isnan(), PyClical.index_set.max(), PyClical.clifford.max_abs(), PyClical.index_set.min(), PyClical.clifford.norm(), PyClical.clifford.odd(), PyClical.clifford.outer_pow(), PyClical.clifford.pow(), PyClical.clifford.pure(), PyClical.clifford.quad(), PyClical.clifford.reverse(), PyClical.clifford.scalar(), PyClical.index_set.sign_of_mult(), PyClical.index_set.sign_of_square(), PyClical.clifford.truncated(), and PyClical.clifford.vector_part().

The documentation for this class was generated from the following file:

- pyclical/PyClical.pyx

6.19 glucat::index_set_hash< LO, HI > Class Template Reference

```
#include <framed_multi.h>
```

Public Types

- using index_set_t = index_set<LO, HI>

Public Member Functions

- auto operator() (index_set_t val) const -> size_t

6.19.1 Detailed Description

template<const index_t LO, const index_t HI>
class glucat::index_set_hash< LO, HI >

Definition at line 117 of file framed_multi.h.

6.19.2 Member Typedef Documentation

6.19.2.1 index_set_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set_hash< LO, HI >::index_set_t = index_set<LO, HI>
```

Definition at line 120 of file framed_multi.h.

6.19.3 Member Function Documentation

6.19.3.1 operator()()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set_hash< LO, HI >::operator() (
            index_set_t val) const -> size_t   [inline]
```

Definition at line 121 of file framed_multi.h.

The documentation for this class was generated from the following file:

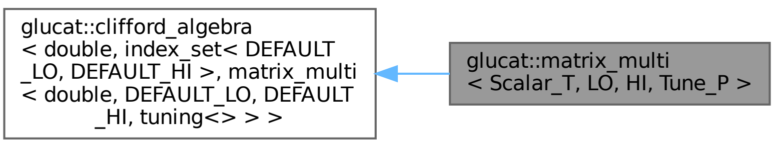
- glucat/framed_multi.h

6.20 glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference

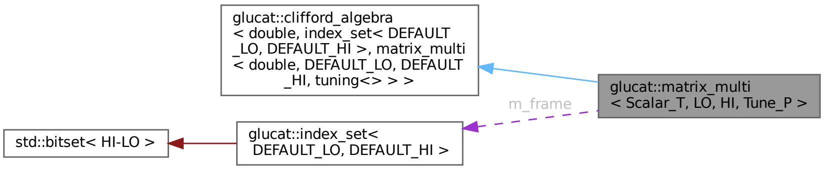
A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector.

```
#include <matrix_multi.h>
```

Inheritance diagram for glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >:



Collaboration diagram for glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >:



Public Types

- using multivector_t = matrix_multi
- using matrix_multi_t = multivector_t
- using scalar_t = Scalar_T
- using tune_p = Tune_P
- using index_set_t = index_set<LO, HI>
- using term_t = std::pair<const index_set_t, Scalar_T>
- using vector_t = std::vector<Scalar_T>
- using error_t = error<multivector_t>
- using framed_multi_t = framed_multi<Scalar_T,LO,HI,Tune_P>

Public Types inherited from glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE

- using scalar_t
- using index_set_t
- using multivector_t
- using pair_t
- using vector_t

Public Member Functions

- ∼matrix_multi () override=default

  Destructor.
- matrix_multi ()

  Default constructor.
- template<typename Other_Scalar_T >
  matrix_multi (const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > &val)

  Construct a multivector from a multivector with a different scalar type.
- template<typename Other_Scalar_T >
  matrix_multi (const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > &val, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given multivector.
- matrix_multi (const multivector_t &val, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given multivector.
- matrix_multi (const index_set_t ist, const Scalar_T &crd=Scalar_T(1))

  Construct a multivector from an index set and a scalar coordinate.
- matrix_multi (const index_set_t ist, const Scalar_T &crd, const index_set_t frm, const bool prechecked=false)

Construct a multivector, within a given frame, from an index set and a scalar coordinate.
- matrix_multi (const Scalar_T &scr, const index_set_t frm=index_set_t())

  Construct a multivector from a scalar (within a frame, if given)
- matrix_multi (const int scr, const index_set_t frm=index_set_t())

  Construct a multivector from an int (within a frame, if given)
- matrix_multi (const vector_t &vec, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a given vector.
- matrix_multi (const std::string &str)

  Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- matrix_multi (const std::string &str, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- matrix_multi (const char ∗str)

  Construct a multivector from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".
- matrix_multi (const char ∗str, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".
- template<typename Other_Scalar_T >
  matrix_multi (const framed_multi< Other_Scalar_T, LO, HI, Tune_P > &val)

  Construct a multivector from a framed_multi_t.
- template<typename Other_Scalar_T >
  matrix_multi (const framed_multi< Other_Scalar_T, LO, HI, Tune_P > &val, const index_set_t frm, const bool prechecked=false)

  Construct a multivector, within a given frame, from a framed_multi_t.
- auto fast_matrix_multi (const index_set_t frm) const -> const matrix_multi_t

  Use generalized FFT to construct a matrix_multi_t.
- template<typename Other_Scalar_T >
  auto fast_framed_multi () const -> const framed_multi< Other_Scalar_T, LO, HI, Tune_P >

  Use inverse generalized FFT to construct a framed_multi_t.
- _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS auto operator= (const multivector_t &rhs) -> multivector_t &

  Assignment operator.
- auto operator+= (const term_t &rhs) -> multivector_t &

  Add a term, if non-zero.

Public Member Functions inherited from glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE

- virtual ∼clifford_algebra ()=default
- virtual auto operator== (const multivector_t &val) const -> bool=0

  Test for equality of multivectors.
- virtual auto operator== (const double &scr) const -> bool=0

  Test for equality of multivector and scalar.
- virtual auto operator+= (const multivector_t &rhs) -> multivector_t &=0

  Geometric sum.
- virtual auto operator+= (const double &scr) -> multivector_t &=0

  Geometric sum of multivector and scalar.
- virtual auto operator-= (const multivector_t &rhs) -> multivector_t &=0

  Geometric difference.
- virtual auto operator-= (const double &scr) -> multivector_t &=0

  Geometric difference of multivector and scalar.
- virtual auto operator- () const -> const multivector_t=0

  Unary -.

- virtual auto operator*= (const double &scr) -> multivector_t &=0

  Product of multivector and scalar.
- virtual auto operator*= (const multivector_t &rhs) -> multivector_t &=0

  Geometric product.
- virtual auto operator%= (const multivector_t &rhs) -> multivector_t &=0

  Contraction.
- virtual auto operator&= (const multivector_t &rhs) -> multivector_t &=0

  Inner product.
- virtual auto operator^= (const multivector_t &rhs) -> multivector_t &=0

  Outer product.
- virtual auto operator/= (const double &scr) -> multivector_t &=0

  Quotient of multivector and scalar.
- virtual auto operator/= (const multivector_t &rhs) -> multivector_t &=0

  Geometric quotient.
- virtual auto operator|= (const multivector_t &rhs) -> multivector_t &=0

  Transformation via twisted adjoint action.
- virtual auto inv () const -> const multivector_t=0

  Geometric multiplicative inverse.
- virtual auto pow (int m) const -> const multivector_t=0

  **this to the m*
- virtual auto outer_pow (int m) const -> const multivector_t=0

  Outer product power.
- virtual auto frame () const -> const index_set_t=0

  Subalgebra generated by all generators of terms of given multivector.
- virtual auto grade () const -> index_t=0

  Maximum of the grades of each term.
- virtual auto operator[ ] (const index_set_t ist) const -> double=0

  Subscripting: map from index set to scalar coordinate.
- virtual auto operator() (index_t grade) const -> const multivector_t=0

  Pure grade-vector part.
- virtual auto scalar () const -> double=0

  Scalar part.
- virtual auto pure () const -> const multivector_t=0

  Pure part.
- virtual auto even () const -> const multivector_t=0

  Even part of multivector, sum of even grade terms.
- virtual auto odd () const -> const multivector_t=0

  Odd part of multivector, sum of odd grade terms.
- virtual auto vector_part () const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frame()
- virtual auto vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t=0

  Vector part of multivector, as a vector_t with respect to frm.
- virtual auto involute () const -> const multivector_t=0

  Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.
- virtual auto reverse () const -> const multivector_t=0

  Reversion, eg. {1}{2} -> {2}*{1}.*
- virtual auto conj () const -> const multivector_t=0

  Conjugation, reverse o involute == involute o reverse.
- virtual auto quad () const -> double=0

  *Scalar_T quadratic form == (rev(x)*x)(0)*
- virtual auto norm () const -> double=0

Scalar_T norm == sum of norm of coordinates.
- virtual auto max_abs () const -> double=0

  Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto truncated (const double &limit=default_truncation) const -> const multivector_t=0

  Remove all terms with relative size smaller than limit.
- virtual auto isinf () const -> bool=0

  Check if a multivector contains any infinite values.
- virtual auto isnan () const -> bool=0

  Check if a multivector contains any IEEE NaN values.
- virtual void write (const std::string &msg="") const=0

  Write formatted multivector to output.
- virtual void write (std::ofstream &ofile, const std::string &msg="") const=0

  Write formatted multivector to file.

Static Public Member Functions

- static auto classname () -> const std::string

  Class name used in messages.
- static auto random (const index_set_t frm, Scalar_T fill=Scalar_T(1)) -> const matrix_multi_t

  Random multivector within a frame.

Static Public Member Functions inherited from glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE

- static auto classname () -> const std::string

Private Types

- using orientation_t = ublas::row_major
- using basis_matrix_t = ublas::compressed_matrix<int, orientation_t>
- using matrix_t = ublas::matrix<Scalar_T, orientation_t>
- using matrix_index_t = typename matrix_t::size_type

Private Member Functions

- template<typename Matrix_T >
  matrix_multi (const Matrix_T &mtx, const index_set_t frm)

  Construct a multivector within a given frame from a given matrix.
- matrix_multi (const matrix_t &mtx, const index_set_t frm)

  Construct a multivector within a given frame from a given matrix.
- auto basis_element (const index_set< LO, HI > &ist) const -> const basis_matrix_t

  Create a basis element matrix within the current frame.

Private Attributes

- index_set_t m_frame

  Index set representing the frame for the subalgebra which contains the multivector.
- matrix_t m_matrix

  Matrix value representing the multivector within the folded frame.

Friends

- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  class framed_multi
- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  class matrix_multi
- auto operator∗ (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto operator^ (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto operator& (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto operator% (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto star (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> Scalar_T
- auto operator/ (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto operator| (const matrix_multi_t &lhs, const matrix_multi_t &rhs) -> const matrix_multi_t
- auto operator>> (std::istream &s, multivector_t &val) -> std::istream &
- auto operator<< (std::ostream &os, const multivector_t &val) -> std::ostream &
- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  auto reframe (const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &lhs, const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &rhs, matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &lhs_reframed, matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &rhs_reframed) -> const index_set< Other_LO, Other_HI >
- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  auto matrix_sqrt (const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &val, const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &i, const index_t level) -> const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P >
- template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  auto matrix_log (const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &val, const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &i, const index_t level) -> const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P >

Additional Inherited Members

Static Public Attributes inherited from glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE

- static const index_t v_lo
- static const index_t v_hi
- static const double default_truncation

  Default for truncation.

6.20.1 Detailed Description

template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
class glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >

A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector.

Definition at line 137 of file matrix_multi.h.

6.20.2 Member Typedef Documentation

6.20.2.1 basis_matrix_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_matrix_t = ublas::compressed_↩
matrix<int, orientation_t>  [private]
```

Definition at line 157 of file matrix_multi.h.

6.20.2.2 error_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::error_t = error<multivector_t>
```

Definition at line 148 of file matrix_multi.h.

6.20.2.3 framed_multi_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::framed_multi_t = framed_multi<Scalar↩
_T,LO,HI,Tune_P>
```

Definition at line 149 of file matrix_multi.h.

6.20.2.4 index_set_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::index_set_t = index_set<LO, HI>
```

Definition at line 145 of file matrix_multi.h.

6.20.2.5 matrix_index_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_index_t = typename matrix_t↩
::size_type  [private]
```

Definition at line 159 of file matrix_multi.h.

6.20.2.6 matrix_multi_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi_t = multivector_t
```

Definition at line 142 of file matrix_multi.h.

6.20.2.7 matrix_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_t = ublas::matrix<Scalar_T,
orientation_t>  [private]
```

Definition at line 158 of file matrix_multi.h.

6.20.2.8 multivector_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::multivector_t = matrix_multi
```

Definition at line 141 of file matrix_multi.h.

6.20.2.9 orientation_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::orientation_t = ublas::row_major
[private]
```

Definition at line 156 of file matrix_multi.h.

6.20.2.10 scalar_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::scalar_t = Scalar_T
```

Definition at line 143 of file matrix_multi.h.

6.20.2.11 term_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::term_t = std::pair<const index_set_t,
Scalar_T>
```

Definition at line 146 of file matrix_multi.h.

6.20.2.12 tune_p

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::tune_p = Tune_P
```

Definition at line 144 of file matrix_multi.h.

6.20.2.13 vector_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::vector_t = std::vector<Scalar_T>
```

Definition at line 147 of file matrix_multi.h.

6.20.3 Constructor & Destructor Documentation

6.20.3.1 ∼matrix_multi()

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::∼matrix_multi () [override], [default]
```

Destructor.

6.20.3.2 matrix_multi() [1/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi ()
```

Default constructor.

Definition at line 106 of file matrix_multi_imp.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix.

6.20.3.3 matrix_multi() [2/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a multivector with a different scalar type.

Definition at line 115 of file matrix_multi_imp.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix.

6.20.3.4 matrix_multi() [3/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const matrix_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 134 of file matrix_multi_imp.h.

References glucat::folded_dim(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame, glucat::matrix_multi< Scalar_T, LO, H
and glucat::numeric_traits< Scalar_T >::to_scalar_t().

6.20.3.5 matrix_multi() [4/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const multivector_t & val,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 159 of file matrix_multi_imp.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame, and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_

6.20.3.6 matrix_multi() [5/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const index_set_t ist,
            const Scalar_T & crd = Scalar_T(1))
```

Construct a multivector from an index set and a scalar coordinate.

Definition at line 171 of file matrix_multi_imp.h.

References glucat::folded_dim(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame, and glucat::matrix_multi< Scalar_T, L

6.20.3.7 matrix_multi() [6/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const index_set_t ist,
            const Scalar_T & crd,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from an index set and a scalar coordinate.

Definition at line 183 of file matrix_multi_imp.h.

References glucat::folded_dim(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix.

6.20.3.8 matrix_multi() [7/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const Scalar_T & scr,
            const index_set_t frm = index_set_t())
```

Construct a multivector from a scalar (within a frame, if given)

Definition at line 197 of file matrix_multi_imp.h.

References glucat::folded_dim(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix.

6.20.3.9 matrix_multi() [8/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const int scr,
            const index_set_t frm = index_set_t())
```

Construct a multivector from an int (within a frame, if given)

Definition at line 209 of file matrix_multi_imp.h.

6.20.3.10 matrix_multi() [9/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const vector_t & vec,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given vector.

Definition at line 215 of file matrix_multi_imp.h.

References glucat::index_set< LO, HI >::count(), glucat::folded_dim(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix, glucat::index_set< LO, HI >::max(), and glucat::index_set< LO, HI >::min().

6.20.3.11 matrix_multi() [10/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const std::string & str)
```

Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 240 of file matrix_multi_imp.h.

6.20.3.12 matrix_multi() [11/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const std::string & str,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 246 of file matrix_multi_imp.h.

6.20.3.13 matrix_multi() [12/17]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const char * str) [inline]
```

Construct a multivector from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 196 of file matrix_multi.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi.

6.20.3.14 matrix_multi() [13/17]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const char * str,
            const index_set_t frm,
            const bool prechecked = false) [inline]
```

Construct a multivector, within a given frame, from a char∗: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 199 of file matrix_multi.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi.

6.20.3.15 matrix_multi() [14/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a framed_multi_t.

Definition at line 253 of file matrix_multi_imp.h.

References glucat::folded_dim(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame, and glucat::matrix_multi< Scalar_T, L

6.20.3.16 matrix_multi() [15/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
            const index_set_t frm,
            const bool prechecked = false)
```

Construct a multivector, within a given frame, from a framed_multi_t.

Definition at line 277 of file matrix_multi_imp.h.

References glucat::folded_dim(), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated().

6.20.3.17 matrix_multi() [16/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Matrix_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const Matrix_T & mtx,
            const index_set_t frm)  [private]
```

Construct a multivector within a given frame from a given matrix.

Definition at line 303 of file matrix_multi_imp.h.

References glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix.

6.20.3.18 matrix_multi() [17/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
            const matrix_t & mtx,
            const index_set_t frm)  [private]
```

Construct a multivector within a given frame from a given matrix.

Definition at line 322 of file matrix_multi_imp.h.

6.20.4 Member Function Documentation

6.20.4.1 basis_element()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element (
            const index_set< LO, HI > & ist) const -> const basis_matrix_t  [private]
```

Create a basis element matrix within the current frame.

Definition at line 1186 of file matrix_multi_imp.h.

References glucat::gen::generator_table< Matrix_T >::generator(), glucat::matrix::mono_prod(), glucat::offset_level(), and glucat::matrix::unit().

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi().

6.20.4.2 classname()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::classname () -> const std::string
[static]
```

Class name used in messages.

Definition at line 78 of file matrix_multi_imp.h.

6.20.4.3 fast_framed_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi () const -> const
framed_multi<Other_Scalar_T,LO,HI,Tune_P>
```

Use inverse generalized FFT to construct a framed_multi_t.

Definition at line 1109 of file matrix_multi_imp.h.

References glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pm4_qp4(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_qp1_pm1(), glucat::fast(), glucat::gen::offset_to_super, glucat::pos_mod(), and glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::unfold().

6.20.4.4 fast_matrix_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_matrix_multi (
             const index_set_t frm) const -> const matrix_multi_t  [inline]
```

Use generalized FFT to construct a matrix_multi_t.

Definition at line 1096 of file matrix_multi_imp.h.

6.20.4.5 operator+=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::operator+= (
             const term_t & rhs) -> multivector_t&  [inline]
```

Add a term, if non-zero.

Geometric sum.

Geometric sum of multivector and scalar.

Definition at line 416 of file matrix_multi_imp.h.

6.20.4.6 operator=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::operator= (
             const multivector_t & rhs) -> multivector_t&
```

Assignment operator.

Definition at line 330 of file matrix_multi_imp.h.

6.20.4.7 random()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::random (
            const index_set_t frm,
            Scalar_T fill = Scalar_T(1)) -> const matrix_multi_t  [static]
```

Random multivector within a frame.

Definition at line 926 of file matrix_multi_imp.h.

References glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::random().

6.20.5 Friends And Related Symbol Documentation

6.20.5.1 framed_multi

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
friend class framed_multi  [friend]
```

Definition at line 151 of file matrix_multi.h.

6.20.5.2 matrix_log

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
auto matrix_log (
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & val,
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & i,
            const index_t level) -> const matrix_multi< Other_Scalar_T, Other_LO, Other_HI,
Other_Tune_P >  [friend]
```

6.20.5.3 matrix_multi

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
friend class matrix_multi  [friend]
```

Definition at line 153 of file matrix_multi.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_F

6.20.5.4 matrix_sqrt

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
auto matrix_sqrt (
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & val,
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & i,
            const index_t level) -> const matrix_multi< Other_Scalar_T, Other_LO, Other_HI,
Other_Tune_P >  [friend]
```

6.20.5.5 operator%

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator% (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.6 operator&

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator& (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.7 operator*

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator* (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.8 operator/

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator/ (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.9 operator<<

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator<< (
            std::ostream & os,
            const multivector_t & val) -> std::ostream &  [friend]
```

6.20.5.10 operator>>

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator>> (
            std::istream & s,
            multivector_t & val) -> std::istream &  [friend]
```

6.20.5.11 operator^

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator^ (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.12 operator" |

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto operator| (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> const matrix_multi_t  [friend]
```

6.20.5.13 reframe

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
auto reframe (
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & lhs,
            const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & rhs,
            matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & lhs_reframed,
            matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & rhs_reframed)
-> const index_set< Other_LO, Other_HI >  [friend]
```

6.20.5.14 star

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
auto star (
            const matrix_multi_t & lhs,
            const matrix_multi_t & rhs) -> Scalar_T  [friend]
```

6.20.6 Member Data Documentation

6.20.6.1 m_frame

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
index_set_t glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame  [private]
```

Index set representing the frame for the subalgebra which contains the multivector.

Definition at line 278 of file matrix_multi.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_mu

6.20.6.2 m_matrix

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
matrix_t glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix  [private]
```

Matrix value representing the multivector within the folded frame.

Definition at line 280 of file matrix_multi.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(),
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(),
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(),
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(),
and glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi().

The documentation for this class was generated from the following files:

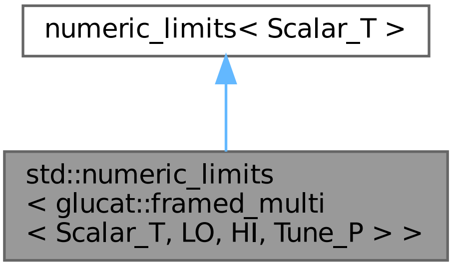
- glucat/framed_multi.h
- glucat/matrix_multi.h
- glucat/matrix_multi_imp.h

6.21 std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference

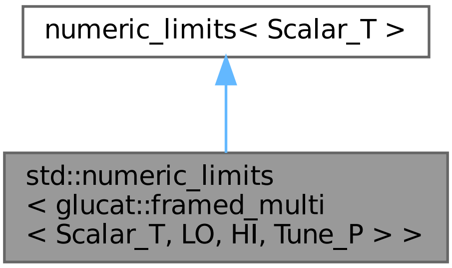
Numeric limits for framed_multi inherit limits for the corresponding scalar type.

```
#include <framed_multi.h>
```

Inheritance diagram for std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >:



Collaboration diagram for std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >:



6.21.1 Detailed Description

template<typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P>
struct std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >

Numeric limits for framed_multi inherit limits for the corresponding scalar type.

Definition at line 345 of file framed_multi.h.

The documentation for this struct was generated from the following file:

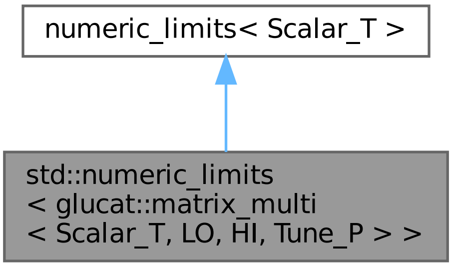
- glucat/framed_multi.h

6.22 std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference

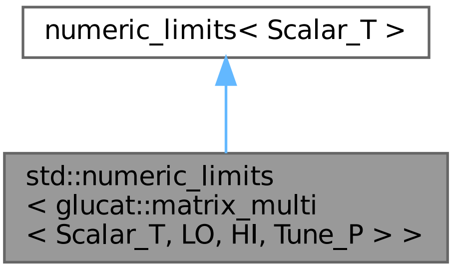
Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

```
#include <matrix_multi.h>
```

Inheritance diagram for std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >:



Collaboration diagram for std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >:



6.22.1 Detailed Description

template<typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P>
struct std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >

Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

Definition at line 296 of file matrix_multi.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi.h

6.23 glucat::numeric_traits< Scalar_T > Class Template Reference

Extra traits which extend numeric limits.

```
#include <scalar.h>
```

Classes

- struct demoted

    Demoted type for long double.
- struct promoted

    Extra traits which extend numeric limits.

Public Member Functions

- auto pi () -> long double

    Pi for long double.
- auto ln_2 () -> long double

    log(2) for long double
- auto to_scalar_t (const Other_Scalar_T &val) -> float

    Extra traits which extend numeric limits.
- auto to_scalar_t (const Other_Scalar_T &val) -> double

    Cast to double.
- auto to_scalar_t (const dd_real &val) -> long double

    Cast to long double.
- auto to_scalar_t (const qd_real &val) -> long double

    Cast to long double.
- auto to_scalar_t (const long double &val) -> dd_real

    Cast to dd_real.
- auto to_scalar_t (const qd_real &val) -> dd_real

    Cast to dd_real.
- auto to_scalar_t (const long double &val) -> qd_real

    Cast to qd_real.
- auto to_scalar_t (const dd_real &val) -> qd_real

    Cast to qd_real.

Static Public Member Functions

- static auto isInf (const Scalar_T &val) -> bool

    Smart isinf.
- static auto isNaN (const Scalar_T &val) -> bool

    Smart isnan.
- static auto isNaN_or_isInf (const Scalar_T &val) -> bool

    Smart isnan or isinf.
- static auto NaN () -> Scalar_T

    Smart NaN.
- static auto to_int (const Scalar_T &val) -> int

    Cast to int.
- static auto to_double (const Scalar_T &val) -> double

Cast to double.

- template<typename Other_Scalar_T >
  static auto to_scalar_t (const Other_Scalar_T &val) -> Scalar_T

  Cast to Scalar_T.
- static auto fmod (const Scalar_T &lhs, const Scalar_T &rhs) -> Scalar_T

  Modulo function for scalar.
- static auto conj (const Scalar_T &val) -> Scalar_T

  Complex conjugate of scalar.
- static auto real (const Scalar_T &val) -> Scalar_T

  Real part of scalar.
- static auto imag (const Scalar_T &val) -> Scalar_T

  Imaginary part of scalar.
- static auto abs (const Scalar_T &val) -> Scalar_T

  Absolute value of scalar.
- static auto pi () -> Scalar_T

  Pi.
- static auto ln_2 () -> Scalar_T

  log(2)
- static auto pow (const Scalar_T &val, int n) -> Scalar_T

  Integer power.
- static auto sqrt (const Scalar_T &val) -> Scalar_T

  Square root of scalar.
- static auto exp (const Scalar_T &val) -> Scalar_T

  Exponential.
- static auto log (const Scalar_T &val) -> Scalar_T

  Logarithm of scalar.
- static auto log2 (const Scalar_T &val) -> Scalar_T

  Log base 2.
- static auto cos (const Scalar_T &val) -> Scalar_T

  Cosine of scalar.
- static auto acos (const Scalar_T &val) -> Scalar_T

  Inverse cosine of scalar.
- static auto cosh (const Scalar_T &val) -> Scalar_T

  Hyperbolic cosine of scalar.
- static auto sin (const Scalar_T &val) -> Scalar_T

  Sine of scalar.
- static auto asin (const Scalar_T &val) -> Scalar_T

  Inverse sine of scalar.
- static auto sinh (const Scalar_T &val) -> Scalar_T

  Hyperbolic sine of scalar.
- static auto tan (const Scalar_T &val) -> Scalar_T

  Tangent of scalar.
- static auto atan (const Scalar_T &val) -> Scalar_T

  Inverse tangent of scalar.
- static auto tanh (const Scalar_T &val) -> Scalar_T

  Hyperbolic tangent of scalar.

Static Private Member Functions

- static auto isInf (const Scalar_T &val, bool_to_type< false >) -> bool
  Smart isinf specialised for Scalar_T without infinity.
- static auto isInf (const Scalar_T &val, bool_to_type< true >) -> bool
  Smart isinf specialised for Scalar_T with infinity.
- static auto isNaN (const Scalar_T &val, bool_to_type< false >) -> bool
  Smart isnan specialised for Scalar_T without quiet NaN.
- static auto isNaN (const Scalar_T &val, bool_to_type< true >) -> bool
  Smart isnan specialised for Scalar_T with quiet NaN.

6.23.1 Detailed Description

template<typename Scalar_T>
class glucat::numeric_traits< Scalar_T >

Extra traits which extend numeric limits.

Definition at line 47 of file scalar.h.

6.23.2 Member Function Documentation

6.23.2.1 abs()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::abs (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Absolute value of scalar.

Definition at line 182 of file scalar.h.

6.23.2.2 acos()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::acos (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Inverse cosine of scalar.

Definition at line 245 of file scalar.h.

6.23.2.3 asin()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::asin (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Inverse sine of scalar.

Definition at line 266 of file scalar.h.

6.23.2.4 atan()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::atan (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Inverse tangent of scalar.

Definition at line 287 of file scalar.h.

6.23.2.5 conj()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::conj (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Complex conjugate of scalar.

Definition at line 161 of file scalar.h.

6.23.2.6 cos()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::cos (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Cosine of scalar.

Definition at line 238 of file scalar.h.

6.23.2.7 cosh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::cosh (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Hyperbolic cosine of scalar.

Definition at line 252 of file scalar.h.

6.23.2.8 exp()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::exp (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Exponential.

Definition at line 217 of file scalar.h.

6.23.2.9 fmod()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::fmod (
            const Scalar_T & lhs,
            const Scalar_T & rhs) -> Scalar_T   [inline], [static]
```

Modulo function for scalar.

Definition at line 154 of file scalar.h.

6.23.2.10 imag()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::imag (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Imaginary part of scalar.

Definition at line 175 of file scalar.h.

6.23.2.11 isInf() [1/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
            const Scalar_T & val) -> bool   [inline], [static]
```

Smart isinf.

Definition at line 83 of file scalar.h.

References glucat::numeric_traits< Scalar_T >::isInf().

6.23.2.12 isInf() [2/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
            const Scalar_T & val,
            bool_to_type< false > ) -> bool   [inline], [static], [private]
```

Smart isinf specialised for Scalar_T without infinity.

Definition at line 54 of file scalar.h.

Referenced by glucat::numeric_traits< Scalar_T >::isInf(), and glucat::numeric_traits< Scalar_T >::isNaN_or_isInf().

6.23.2.13 isInf() [3/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
            const Scalar_T & val,
            bool_to_type< true > ) -> bool   [inline], [static], [private]
```

Smart isinf specialised for Scalar_T with infinity.

Definition at line 61 of file scalar.h.

References _GLUCAT_ISINF.

6.23.2.14 isNaN() [1/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
            const Scalar_T & val) -> bool   [inline], [static]
```

Smart isnan.

Definition at line 93 of file scalar.h.

References glucat::numeric_traits< Scalar_T >::isNaN().

6.23.2.15 isNaN() [2/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
            const Scalar_T & val,
            bool_to_type< false > ) -> bool   [inline], [static], [private]
```

Smart isnan specialised for Scalar_T without quiet NaN.

Definition at line 68 of file scalar.h.

Referenced by glucat::numeric_traits< Scalar_T >::isNaN(), and glucat::numeric_traits< Scalar_T >::isNaN_or_isInf().

6.23.2.16 isNaN() [3/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
            const Scalar_T & val,
            bool_to_type< true > ) -> bool   [inline], [static], [private]
```

Smart isnan specialised for Scalar_T with quiet NaN.

Definition at line 75 of file scalar.h.

References _GLUCAT_ISNAN.

6.23.2.17 isNaN_or_isInf()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN_or_isInf (
            const Scalar_T & val) -> bool   [inline], [static]
```

Smart isnan or isinf.

Definition at line 103 of file scalar.h.

References glucat::numeric_traits< Scalar_T >::isInf(), and glucat::numeric_traits< Scalar_T >::isNaN().

6.23.2.18 ln_2() [1/2]

```
auto glucat::numeric_traits< longdouble >::ln_2 () -> long double   [inline]
```

log(2) for long double

Definition at line 59 of file long_double.h.

References glucat::l_ln2.

6.23.2.19 ln_2() [2/2]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::ln_2 () -> Scalar_T   [inline], [static]
```

log(2)

Definition at line 196 of file scalar.h.

Referenced by glucat::numeric_traits< Scalar_T >::log2().

6.23.2.20 log()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::log (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Logarithm of scalar.

Definition at line 224 of file scalar.h.

Referenced by glucat::numeric_traits< Scalar_T >::log2().

6.23.2.21 log2()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::log2 (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Log base 2.

Definition at line 231 of file scalar.h.

References glucat::numeric_traits< Scalar_T >::ln_2(), and glucat::numeric_traits< Scalar_T >::log().

Referenced by glucat::log2().

6.23.2.22 NaN()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::NaN () -> Scalar_T   [inline], [static]
```

Smart NaN.

Definition at line 115 of file scalar.h.

Referenced by glucat::cr_sqrt(), glucat::db_sqrt(), glucat::matrix::norm_frob2(), glucat::operator*(), and glucat::matrix::trace().

6.23.2.23 pi() [1/2]

```
auto glucat::numeric_traits< longdouble >::pi () -> long double   [inline]
```

Pi for long double.

Definition at line 51 of file long_double.h.

References glucat::l_pi.

6.23.2.24 pi() [2/2]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::pi () -> Scalar_T   [inline], [static]
```

Pi.

Definition at line 189 of file scalar.h.

Referenced by glucat::matrix::classify_eigenvalues().

6.23.2.25 pow()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::pow (
            const Scalar_T & val,
            int n) -> Scalar_T   [inline], [static]
```

Integer power.

Definition at line 203 of file scalar.h.

Referenced by glucat::error_squared_tol().

6.23.2.26 real()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::real (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Real part of scalar.

Definition at line 168 of file scalar.h.

6.23.2.27 sin()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sin (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Sine of scalar.

Definition at line 259 of file scalar.h.

6.23.2.28 sinh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sinh (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Hyperbolic sine of scalar.

Definition at line 273 of file scalar.h.

6.23.2.29 sqrt()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sqrt (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Square root of scalar.

Definition at line 210 of file scalar.h.

Referenced by glucat::abs().

6.23.2.30 tan()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::tan (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Tangent of scalar.

Definition at line 280 of file scalar.h.

6.23.2.31 tanh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::tanh (
            const Scalar_T & val) -> Scalar_T   [inline], [static]
```

Hyperbolic tangent of scalar.

Definition at line 294 of file scalar.h.

6.23.2.32 to_double()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::to_double (
            const Scalar_T & val) -> double   [inline], [static]
```

Cast to double.

Definition at line 133 of file scalar.h.

Referenced by glucat::operator<<(), glucat::numeric_traits< Scalar_T >::to_scalar_t(), and glucat::numeric_traits< Scalar_T >::to_

6.23.2.33 to_int()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::to_int (
            const Scalar_T & val) -> int   [inline], [static]
```

Cast to int.

Definition at line 126 of file scalar.h.

6.23.2.34 to_scalar_t() [1/9]

```
auto glucat::numeric_traits< longdouble >::to_scalar_t (
            const dd_real & val) -> long double   [inline]
```

Cast to long double.

Definition at line 71 of file scalar_imp.h.

6.23.2.35 to_scalar_t() [2/9]

```
auto glucat::numeric_traits< qd_real >::to_scalar_t (
            const dd_real & val) -> qd_real   [inline]
```

Cast to qd_real.

Definition at line 116 of file scalar_imp.h.

6.23.2.36 to_scalar_t() [3/9]

```
auto glucat::numeric_traits< dd_real >::to_scalar_t (
            const long double & val) -> dd_real   [inline]
```

Cast to dd_real.

Definition at line 89 of file scalar_imp.h.

6.23.2.37 to_scalar_t() [4/9]

```
auto glucat::numeric_traits< qd_real >::to_scalar_t (
            const long double & val) -> qd_real   [inline]
```

Cast to qd_real.

Definition at line 107 of file scalar_imp.h.

6.23.2.38 to_scalar_t() [5/9]

```
auto glucat::numeric_traits< double >::to_scalar_t (
            const Other_Scalar_T & val) -> double   [inline]
```

Cast to double.

Definition at line 61 of file scalar_imp.h.

References glucat::numeric_traits< Scalar_T >::to_double().

6.23.2.39 to_scalar_t() [6/9]

```
auto glucat::numeric_traits< float >::to_scalar_t (
            const Other_Scalar_T & val) -> float   [inline]
```

Extra traits which extend numeric limits.

Cast to float

Definition at line 52 of file scalar_imp.h.

References glucat::numeric_traits< Scalar_T >::to_double().

6.23.2.40 to_scalar_t() [7/9]

```
template<typename Scalar_T >
template<typename Other_Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::to_scalar_t (
            const Other_Scalar_T & val) -> Scalar_T   [inline], [static]
```

Cast to Scalar_T.

Definition at line 141 of file scalar.h.

Referenced by glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), glucat::matrix::nork_range(), glucat::to_demote(), and glucat::to_promote().

6.23.2.41 to_scalar_t() [8/9]

```
auto glucat::numeric_traits< dd_real >::to_scalar_t (
            const qd_real & val) -> dd_real   [inline]
```

Cast to dd_real.

Definition at line 98 of file scalar_imp.h.

6.23.2.42 to_scalar_t() [9/9]

```
auto glucat::numeric_traits< longdouble >::to_scalar_t (
            const qd_real & val) -> long double   [inline]
```

Cast to long double.

Definition at line 80 of file scalar_imp.h.

The documentation for this class was generated from the following file:

- glucat/scalar.h

6.24 pade::pade_log_denom< Scalar_T > Struct Template Reference

Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<Scalar_T, 14>

Static Public Attributes

- static const array denom

6.24.1 Detailed Description

template<typename Scalar_T>
struct pade::pade_log_denom< Scalar_T >

Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

Definition at line 1731 of file matrix_multi_imp.h.

6.24.2 Member Typedef Documentation

6.24.2.1 array

```
template<typename Scalar_T >
using pade::pade_log_denom< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1733 of file matrix_multi_imp.h.

6.24.3 Member Data Documentation

6.24.3.1 denom

```
template<typename Scalar_T >
const pade_log_denom< longdouble >::array pade::pade_log_denom< Scalar_T >::denom  [static]
```

Initial value:

```
=
  {
         1.0,                 13.0/2.0,            468.0/25.0,        1573.0/50.0,
      1573.0/46.0,         11583.0/460.0,        10296.0/805.0,       2574.0/575.0,
     11583.0/10925.0,        143.0/874.0,          572.0/37145.0,      117.0/148580.0,
        13.0/742900.0,          1.0/10400600.0
  }
```

Definition at line 1734 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.25 pade::pade_log_denom< dd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<dd_real, 22>

Static Public Attributes

- static const array denom

6.25.1 Detailed Description

Definition at line 1820 of file matrix_multi_imp.h.

6.25.2 Member Typedef Documentation

6.25.2.1 array

```
using pade::pade_log_denom< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1822 of file matrix_multi_imp.h.

6.25.3 Member Data Documentation

6.25.3.1 denom

```
const pade_log_denom< dd_real >::array pade::pade_log_denom< dd_real >::denom  [static]
```

Initial value:

```
=
  {
          dd_real("1"),                              dd_real("21")/dd_real("2"),
       dd_real("2100")/dd_real("41"),             dd_real("12635")/dd_real("82"),
     dd_real("341145")/dd_real("1066"),         dd_real("1037799")/dd_real("2132"),
   dd_real("11069856")/dd_real("19721"),        dd_real("9883800")/dd_real("19721"),
    dd_real("6918660")/dd_real("19721"),         dd_real("293930")/dd_real("1517"),
    dd_real("1410864")/dd_real("16687"),          dd_real("88179")/dd_real("3034"),
     dd_real("734825")/dd_real("94054"),         dd_real("305235")/dd_real("188108"),
     dd_real("348840")/dd_real("1363783"),        dd_real("40698")/dd_real("1363783"),
       dd_real("6783")/dd_real("2727566"),         dd_real("9975")/dd_real("70916716"),
        dd_real("266")/dd_real("53187537"),           dd_real("7")/dd_real("70916716"),
          dd_real("7")/dd_real("8155422340"),         dd_real("1")/dd_real("538257874440")
  }
```

Definition at line 1823 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.26 pade::pade_log_denom< float > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<float, 10>

Static Public Attributes

- static const array denom

6.26.1 Detailed Description

Definition at line 1758 of file matrix_multi_imp.h.

6.26.2 Member Typedef Documentation

6.26.2.1 array

```
using pade::pade_log_denom< float >::array = std::array<float, 10>
```

Definition at line 1760 of file matrix_multi_imp.h.

6.26.3 Member Data Documentation

6.26.3.1 denom

```
const pade_log_denom< float >::array pade::pade_log_denom< float >::denom [static]
```

Initial value:

```
=
  {
      1.0,           9.0/2.0,       144.0/17.0,   147.0/17.0,
    441.0/85.0,     63.0/34.0,       84.0/221.0,    9.0/221.0,
      9.0/4862.0,    1.0/48620.0
  }
```

Definition at line 1761 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.27 pade::pade_log_denom< long double > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<long double, 18>

Static Public Attributes

- static const array denom

6.27.1 Detailed Description

Definition at line 1785 of file matrix_multi_imp.h.

6.27.2 Member Typedef Documentation

6.27.2.1 array

```
using pade::pade_log_denom< long double >::array = std::array<long double, 18>
```

Definition at line 1787 of file matrix_multi_imp.h.

6.27.3 Member Data Documentation

6.27.3.1 denom

```
const array pade::pade_log_denom< long double >::denom [static]
```

Definition at line 1788 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.28 pade::pade_log_denom< qd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<qd_real, 34>

Static Public Attributes

- static const array denom

6.28.1 Detailed Description

Definition at line 1867 of file matrix_multi_imp.h.

6.28.2 Member Typedef Documentation

6.28.2.1 array

```
using pade::pade_log_denom< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1869 of file matrix_multi_imp.h.

6.28.3 Member Data Documentation

6.28.3.1 denom

```
const pade_log_denom< qd_real >::array pade::pade_log_denom< qd_real >::denom [static]
```

Initial value:

```
=
  {
                qd_real("1"),
      qd_real("33")/qd_real("2"),
             qd_real("8448")/qd_real("65"),
      qd_real("42284")/qd_real("65"),
           qd_real("211420")/qd_real("91"),
      qd_real("573562")/qd_real("91"),
          qd_real("32119472")/qd_real("2379"),
      qd_real("92917044")/qd_real("3965"),
         qd_real("603960786")/qd_real("17995"),
      qd_real("144626625")/qd_real("3599"),
        qd_real("2776831200")/qd_real("68381"),
      qd_real("16692542100")/qd_real("478667"),
       qd_real("12241197540")/qd_real("478667"),
      qd_real("1098569010")/qd_real("68381"),
       qd_real("31387686000")/qd_real("3624193"),
      qd_real("9939433900")/qd_real("2479711"),
       qd_real("67091178825")/qd_real("42155087"),
      qd_real("2683647153")/qd_real("4959422"),
       qd_real("19083713088")/qd_real("121505839"),
      qd_real("4708152900")/qd_real("121505839"),
         qd_real("941630580")/qd_real("116546417"),
      qd_real("88704330")/qd_real("62755763"),
          qd_real("12902448")/qd_real("62755763"),
      qd_real("1542684")/qd_real("62755763"),
           qd_real("6427850")/qd_real("2698497809"),
      qd_real("3471039")/qd_real("18889484663"),
           qd_real("8544096")/qd_real("774468871183"),
      qd_real("39556")/qd_real("790274358835"),
            qd_real("118668")/qd_real("7191496660985"),
      qd_real("10230")/qd_real("27327687311743"),
              qd_real("5456")/qd_real("1011124430534491"),
      qd_real("44")/qd_real("1011124430534491"),
                qd_real("11")/qd_real("70778710137414370"),
      qd_real("1")/qd_real("7219428434016265740")
  }
```

Definition at line 1870 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.29 pade::pade_log_numer< Scalar_T > Struct Template Reference

Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<Scalar_T, 14>

Static Public Attributes

- static const array numer

6.29.1 Detailed Description

template<typename Scalar_T>
struct pade::pade_log_numer< Scalar_T >

Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

Definition at line 1714 of file matrix_multi_imp.h.

6.29.2 Member Typedef Documentation

6.29.2.1 array

```
template<typename Scalar_T >
using pade::pade_log_numer< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1716 of file matrix_multi_imp.h.

6.29.3 Member Data Documentation

6.29.3.1 numer

```
template<typename Scalar_T >
const pade_log_numer< longdouble >::array pade::pade_log_numer< Scalar_T >::numer  [static]
```

Initial value:

```
=
  {
        0.0,                      1.0,                     6.0,             4741.0/300.0,
     1441.0/60.0,            107091.0/4600.0,          8638.0/575.0,     263111.0/40250.0,
   153081.0/80500.0,         395243.0/1101240.0,      28549.0/688275.0,  605453.0/228813200.0,
   785633.0/10296594000.0, 1145993.0/1873980108000.0
  }
```

Definition at line 1717 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.30 pade::pade_log_numer< dd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<dd_real, 22>

Static Public Attributes

- static const array numer

6.30.1 Detailed Description

Definition at line 1800 of file matrix_multi_imp.h.

6.30.2 Member Typedef Documentation

6.30.2.1 array

```
using pade::pade_log_numer< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1802 of file matrix_multi_imp.h.

6.30.3 Member Data Documentation

6.30.3.1 numer

```
const pade_log_numer< dd_real >::array pade::pade_log_numer< dd_real >::numer  [static]
```

Initial value:

```
=
  {
          dd_real("0"),                                   dd_real("1"),
         dd_real("10"),                               dd_real("22781")/dd_real("492"),
      dd_real("21603")/dd_real("164"),              dd_real("5492649")/dd_real("21320"),
     dd_real("978724")/dd_real("2665"),             dd_real("4191605")/dd_real("10619"),
   dd_real("12874933")/dd_real("39442"),           dd_real("11473457")/dd_real("54612"),
    dd_real("2406734")/dd_real("22755"),          dd_real("166770367")/dd_real("4004880"),
   dd_real("30653165")/dd_real("2402928"),        dd_real("647746389")/dd_real("215195552"),
   dd_real("25346331")/dd_real("47074027"),       dd_real("278270613")/dd_real("3900419380"),
  dd_real("105689791")/dd_real("15601677520"),    dd_real("606046475")/dd_real("1379188292768"),
     dd_real("969715")/dd_real("53502994116"),     dd_real("11098301")/dd_real("26204577562592"),
     dd_real("118999")/dd_real("26204577562592"), dd_real("18858053")/dd_real("1392249205900512960")
  }
```

Definition at line 1803 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.31 pade::pade_log_numer< float > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<float, 10>

Static Public Attributes

- static const array numer

6.31.1 Detailed Description

Definition at line 1746 of file matrix_multi_imp.h.

6.31.2 Member Typedef Documentation

6.31.2.1 array

```
using pade::pade_log_numer< float >::array = std::array<float, 10>
```

Definition at line 1748 of file matrix_multi_imp.h.

6.31.3 Member Data Documentation

6.31.3.1 numer

```
const pade_log_numer< float >::array pade::pade_log_numer< float >::numer [static]
```

Initial value:

```
=
  {
      0.0,            1.0,             4.0,        1337.0/204.0,
    385.0/68.0,    1879.0/680.0,     193.0/255.0,   197.0/1820.0,
    419.0/61880.0, 7129.0/61261200.0
  }
```

Definition at line 1749 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.32 pade::pade_log_numer< long double > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<long double, 18>

Static Public Attributes

- static const array numer

6.32.1 Detailed Description

Definition at line 1771 of file matrix_multi_imp.h.

6.32.2 Member Typedef Documentation

6.32.2.1 array

```
using pade::pade_log_numer< long double >::array = std::array<long double, 18>
```

Definition at line 1773 of file matrix_multi_imp.h.

6.32.3 Member Data Documentation

6.32.3.1 numer

```
const array pade::pade_log_numer< long double >::numer  [static]
```

Definition at line 1774 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.33 pade::pade_log_numer< qd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<qd_real, 34>

Static Public Attributes

- static const array numer

6.33.1 Detailed Description

Definition at line 1841 of file matrix_multi_imp.h.

6.33.2 Member Typedef Documentation

6.33.2.1 array

```
using pade::pade_log_numer< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1843 of file matrix_multi_imp.h.

6.33.3 Member Data Documentation

6.33.3.1 numer

```
const pade_log_numer< qd_real >::array pade::pade_log_numer< qd_real >::numer [static]
```

Initial value:

```
=
  {
                qd_real("0"),                                                                  qd_real("1"),
                qd_real("16"),
      qd_real("95201")/qd_real("780"),
             qd_real("30721")/qd_real("52"),
      qd_real("7416257")/qd_real("3640"),
           qd_real("1039099")/qd_real("195"),
      qd_real("6097772319")/qd_real("555100"),
        qd_real("1564058073")/qd_real("85400"),
      qd_real("30404640205")/qd_real("1209264"),
         qd_real("725351278")/qd_real("25193"),
      qd_real("4092322670789")/qd_real("147429436"),
     qd_real("4559713849589")/qd_real("201040140"),
      qd_real("5049361751189")/qd_real("320023080"),
       qd_real("74979677195")/qd_real("8000577"),
      qd_real("16569850691873")/qd_real("3481514244"),
     qd_real("1065906022369")/qd_real("515779888"),
      qd_real("335956770855841")/qd_real("438412904800"),
  qd_real("1462444287585964")/qd_real("6041877844275"),
      qd_real("397242326339851")/qd_real("6122436215532"),
    qd_real("64211291334131")/qd_real("4373168725380"),
      qd_real("142322343550859")/qd_real("51080680851480"),
   qd_real("154355972958659")/qd_real("351179680853925"),
      qd_real("167483568676259")/qd_real("2937139148960100"),
     qd_real("4230788929433")/qd_real("704913395750424"),
      qd_real("197968763176019")/qd_real("3929239483719956000"),
    qd_real("10537522306718")/qd_real("319250708052246425"),
      qd_real("236648286272519")/qd_real("14424919747503542550 0"),
   qd_real("260715545088119")/qd_real("4375558990076074573500"),
      qd_real("289596255666839")/qd_real("192874640282553367199880"),
     qd_real("8802625510547")/qd_real("36163995052978756349977 5"),
      qd_real("373831661521439")/qd_real("165920409303066534133696 7700"),
   qd_real("446033437968239")/qd_real("4645771460485862955743509 56000"),
      qd_real("53676090078349")/qd_real("4738686889695580214858379 7512000")
      }
```

Definition at line 1844 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.34 pade::pade_sqrt_denom< Scalar_T > Struct Template Reference

Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

`#include <matrix_multi_imp.h>`

Public Types

- using array = std::array<Scalar_T, 14>

Static Public Attributes

- static const array denom

6.34.1 Detailed Description

template<typename Scalar_T>
struct pade::pade_sqrt_denom< Scalar_T >

Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

Definition at line 1401 of file matrix_multi_imp.h.

6.34.2 Member Typedef Documentation

6.34.2.1 array

```
template<typename Scalar_T >
using pade::pade_sqrt_denom< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1403 of file matrix_multi_imp.h.

6.34.3 Member Data Documentation

6.34.3.1 denom

```
template<typename Scalar_T >
const pade_sqrt_denom< longdouble >::array pade::pade_sqrt_denom< Scalar_T >::denom  [static]
```

Initial value:

```
=
  {
       1.0,              25.0/4.0,         69.0/4.0,        1771.0/64.0,
    7315.0/256.0,      20349.0/1024.0,    4845.0/512.0,    12597.0/4096.0,
   21879.0/32768.0,    12155.0/131072.0,  1001.0/131072.0,  1365.0/4194304.0,
      91.0/16777216.0,     1.0/67108864.0
  }
```

Definition at line 1404 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.35 pade::pade_sqrt_denom< dd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<dd_real, 22>

Static Public Attributes

- static const array denom

6.35.1 Detailed Description

Definition at line 1491 of file matrix_multi_imp.h.

6.35.2 Member Typedef Documentation

6.35.2.1 array

```
using pade::pade_sqrt_denom< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1493 of file matrix_multi_imp.h.

6.35.3 Member Data Documentation

6.35.3.1 denom

```
const pade_sqrt_denom< dd_real >::array pade::pade_sqrt_denom< dd_real >::denom [static]
```

Initial value:

```
=
{
        dd_real("1"),                                  dd_real("41")/dd_real("4"),
      dd_real("195")/dd_real("4"),                   dd_real("9139")/dd_real("64"),
    dd_real("73815")/dd_real("256"),               dd_real("435897")/dd_real("1024"),
   dd_real("121737")/dd_real("256"),               dd_real("840565")/dd_real("2048"),
  dd_real("4539051")/dd_real("16384"),            dd_real("9641775")/dd_real("65536"),
  dd_real("4032015")/dd_real("65536"),           dd_real("84672315")/dd_real("4194304"),
 dd_real("86493225")/dd_real("16777216"),        dd_real("67863915")/dd_real("67108864"),
  dd_real("5014575")/dd_real("33554432"),         dd_real("4345965")/dd_real("268435456"),
  dd_real("5311735")/dd_real("4294967296"),       dd_real("1081575")/dd_real("17179869184"),
    dd_real("33649")/dd_real("17179869184"),         dd_real("8855")/dd_real("274877906944"),
      dd_real("231")/dd_real("1099511627776"),          dd_real("1")/dd_real("4398046511104")
}
```

Definition at line 1494 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.36 pade::pade_sqrt_denom< float > Struct Reference

`#include <matrix_multi_imp.h>`

Public Types

- using array = std::array<float, 10>

Static Public Attributes

- static const array denom

6.36.1 Detailed Description

Definition at line 1428 of file matrix_multi_imp.h.

6.36.2 Member Typedef Documentation

6.36.2.1 array

`using pade::pade_sqrt_denom< float >::array = std::array<float, 10>`

Definition at line 1430 of file matrix_multi_imp.h.

6.36.3 Member Data Documentation

6.36.3.1 denom

`const pade_sqrt_denom< float >::array pade::pade_sqrt_denom< float >::denom [static]`

Initial value:

```
=
  {
       1.0,            17.0/4.0,         15.0/2.0,       455.0/64.0,
    1001.0/256.0,    1287.0/1024.0,    231.0/1024.0,    165.0/8192.0,
      45.0/65536,        1.0/262144.0
  }
```

Definition at line 1431 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.37 pade::pade_sqrt_denom< long double > Struct Reference

`#include <matrix_multi_imp.h>`

Public Types

- using array = std::array<long double, 18>

Static Public Attributes

- static const array denom

6.37.1 Detailed Description

Definition at line 1455 of file matrix_multi_imp.h.

6.37.2 Member Typedef Documentation

6.37.2.1 array

```
using pade::pade_sqrt_denom< long double >::array = std::array<long double, 18>
```

Definition at line 1457 of file matrix_multi_imp.h.

6.37.3 Member Data Documentation

6.37.3.1 denom

```
const array pade::pade_sqrt_denom< long double >::denom  [static]
```

Definition at line 1458 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.38 pade::pade_sqrt_denom< qd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<qd_real, 34>

Static Public Attributes

- static const array denom

6.38.1 Detailed Description

Definition at line 1538 of file matrix_multi_imp.h.

6.38.2 Member Typedef Documentation

6.38.2.1 array

```
using pade::pade_sqrt_denom< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1540 of file matrix_multi_imp.h.

6.38.3 Member Data Documentation

6.38.3.1 denom

```
const pade_sqrt_denom< qd_real >::array pade::pade_sqrt_denom< qd_real >::denom  [static]
```

Initial value:

```
=
  {
      qd_real("1"),
      qd_real("65")/qd_real("4"),
      qd_real("126"),
      qd_real("39711")/qd_real("64"),
      qd_real("557845")/qd_real("256"),
      qd_real("5949147")/qd_real("1024"),
      qd_real("12515965")/qd_real("1024"),
      qd_real("170574723")/qd_real("8192"),
      qd_real("1916797311")/qd_real("65536"),
      qd_real("8996462475")/qd_real("262144"),
      qd_real("4450881435")/qd_real("131072"),
      qd_real("59826782925")/qd_real("2097152"),
      qd_real("171503444385")/qd_real("8388608"),
      qd_real("420696483235")/qd_real("33554432"),
      qd_real("221120793075")/qd_real("33554432"),
      qd_real("797168807855")/qd_real("268435456"),
      qd_real("4923689695575")/qd_real("4294967296"),
      qd_real("6499270398159")/qd_real("17179869184"),
      qd_real("456864812569")/qd_real("4294967296"),
      qd_real("3486599885395")/qd_real("137438953472"),
      qd_real("2804116503573")/qd_real("549755813888"),
      qd_real("1886827875075")/qd_real("2199023255552"),
      qd_real("263012370465")/qd_real("2199023255552"),
      qd_real("240141729555")/qd_real("17592186044416"),
      qd_real("176848560525")/qd_real("140737488355328"),
      qd_real("51538723353")/qd_real("562949953421312"),
      qd_real("1450433115")/qd_real("281474976710656"),
      qd_real("977699359")/qd_real("4503599627370496"),
      qd_real("118183439")/qd_real("18014398509481984"),
      qd_real("9652005")/qd_real("72057594037927936"),
      qd_real("121737")/qd_real("72057594037927936"),
      qd_real("6545")/qd_real("576460752303423488"),
      qd_real("561")/qd_real("18446744073709551616"),
      qd_real("1")/qd_real("73786976294838206464")
  }
```

Definition at line 1541 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.39 pade::pade_sqrt_numer< Scalar_T > Struct Template Reference

Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<Scalar_T, 14>

Static Public Attributes

- static const array numer

6.39.1 Detailed Description

template<typename Scalar_T>
struct pade::pade_sqrt_numer< Scalar_T >

Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

Definition at line 1384 of file matrix_multi_imp.h.

6.39.2 Member Typedef Documentation

6.39.2.1 array

```
template<typename Scalar_T >
using pade::pade_sqrt_numer< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1386 of file matrix_multi_imp.h.

6.39.3 Member Data Documentation

6.39.3.1 numer

```
template<typename Scalar_T >
const pade_sqrt_numer< longdouble >::array pade::pade_sqrt_numer< Scalar_T >::numer  [static]
```

Initial value:

```
=
  {
        1.0,              27.0/4.0,         81.0/4.0,        2277.0/64.0,
    10395.0/256.0,    32319.0/1024.0,    8721.0/512.0,     26163.0/4096.0,
    53703.0/32768.0,  36465.0/131072.0,  3861.0/131072.0,  7371.0/4194304.0,
      819.0/16777216.0,    27.0/67108864.0
  }
```

Definition at line 1387 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.40 pade::pade_sqrt_numer< dd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<dd_real, 22>

Static Public Attributes

- static const array numer

6.40.1 Detailed Description

Definition at line 1471 of file matrix_multi_imp.h.

6.40.2 Member Typedef Documentation

6.40.2.1 array

```
using pade::pade_sqrt_numer< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1473 of file matrix_multi_imp.h.

6.40.3 Member Data Documentation

6.40.3.1 numer

```
const pade_sqrt_numer< dd_real >::array pade::pade_sqrt_numer< dd_real >::numer  [static]
```

Initial value:

```
=
  {
          dd_real("1"),                                  dd_real("43")/dd_real("4"),
        dd_real("215")/dd_real("4"),                   dd_real("10621")/dd_real("64"),
      dd_real("90687")/dd_real("256"),               dd_real("567987")/dd_real("1024"),
     dd_real("168861")/dd_real("256"),              dd_real("1246355")/dd_real("2048"),
    dd_real("7228859")/dd_real("16384"),           dd_real("16583853")/dd_real("65536"),
    dd_real("7538115")/dd_real("65536"),          dd_real("173376645")/dd_real("4194304"),
  dd_real("195747825")/dd_real("16777216"),      dd_real("171655785")/dd_real("67108864"),
   dd_real("14375115")/dd_real("33554432"),       dd_real("14375115")/dd_real("268435456"),
   dd_real("20764055")/dd_real("4294967296"),      dd_real("5167525")/dd_real("17179869184"),
     dd_real("206701")/dd_real("17179869184"),       dd_real("76153")/dd_real("274877906944"),
       dd_real("3311")/dd_real("1099511627776") ,       dd_real("43")/dd_real("4398046511104")
  }
```

Definition at line 1474 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.41 pade::pade_sqrt_numer< float > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<float, 10>

Static Public Attributes

- static const array numer

6.41.1 Detailed Description

Definition at line 1416 of file matrix_multi_imp.h.

6.41.2 Member Typedef Documentation

6.41.2.1 array

```
using pade::pade_sqrt_numer< float >::array = std::array<float, 10>
```

Definition at line 1418 of file matrix_multi_imp.h.

6.41.3 Member Data Documentation

6.41.3.1 numer

```
const pade_sqrt_numer< float >::array pade::pade_sqrt_numer< float >::numer  [static]
```

Initial value:

```
=
  {
       1.0,            19.0/4.0,        19.0/2.0,      665.0/64.0,
    1729.0/256.0,    2717.0/1024.0,    627.0/1024.0,   627.0/8192.0,
     285.0/65536.0,    19.0/262144.0
  }
```

Definition at line 1419 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.42 pade::pade_sqrt_numer< long double > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<long double, 18>

Static Public Attributes

- static const array numer

6.42.1 Detailed Description

Definition at line 1441 of file matrix_multi_imp.h.

6.42.2 Member Typedef Documentation

6.42.2.1 array

```
using pade::pade_sqrt_numer< long double >::array = std::array<long double, 18>
```

Definition at line 1443 of file matrix_multi_imp.h.

6.42.3 Member Data Documentation

6.42.3.1 numer

```
const array pade::pade_sqrt_numer< long double >::numer  [static]
```

Definition at line 1444 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.43 pade::pade_sqrt_numer< qd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using array = std::array<qd_real, 34>

Static Public Attributes

- static const array numer

6.43.1 Detailed Description

Definition at line 1512 of file matrix_multi_imp.h.

6.43.2 Member Typedef Documentation

6.43.2.1 array

```
using pade::pade_sqrt_numer< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1514 of file matrix_multi_imp.h.

6.43.3 Member Data Documentation

6.43.3.1 numer

```
const pade_sqrt_numer< qd_real >::array pade::pade_sqrt_numer< qd_real >::numer  [static]
```

Initial value:

```
=
  {
            qd_real("1"),                                         qd_real("67")/qd_real("4"),
          qd_real("134"),                                      qd_real("43617")/qd_real("64"),
       qd_real("633485")/qd_real("256"),                     qd_real("6992857")/qd_real("1024"),
     qd_real("15246721")/qd_real("1024"),                  qd_real("215632197")/qd_real("8192"),
   qd_real("2518145487")/qd_real("65536"),               qd_real("12301285425")/qd_real("262144"),
   qd_real("6344873535")/qd_real("131072"),              qd_real("89075432355")/qd_real("2097152"),
 qd_real("267226297065")/qd_real("8388608"),            qd_real("687479618945")/qd_real("33554432"),
 qd_real("379874182975")/qd_real("33554432"),          qd_real("1443521895305")/qd_real("268435456"),
qd_real("9425348845815")/qd_real("4294967296"),       qd_real("13195488384141")/qd_real("17179869184"),
 qd_real("987417498133")/qd_real("4294967296"),
    qd_real("8055248011085")/qd_real("137438953472"),
qd_real("6958363175533")/qd_real("549755813888"),
    qd_real("5056698705201")/qd_real("2199023255552"),
 qd_real("766166470485")/qd_real("2199023255552"),
    qd_real("766166470485")/qd_real("17592186044416"),
 qd_real("623623871325")/qd_real("140737488355328"),
    qd_real("203123203803")/qd_real("562949953421312"),
   qd_real("6478601247")/qd_real("281474976710656"),
    qd_real("5038912081")/qd_real("4503599627370496"),
    qd_real("719844583")/qd_real("18014398509481984"),
    qd_real("71853815")/qd_real("72057594037927936"),
      qd_real("1165197")/qd_real("72057594037927936"),
    qd_real("87703")/qd_real("576460752303423488"),
        qd_real("12529")/qd_real("18446744073709551616"),
    qd_real("67")/qd_real("73786976294838206464")
  }
```

Definition at line 1515 of file matrix_multi_imp.h.

The documentation for this struct was generated from the following file:

- glucat/matrix_multi_imp.h

6.44 glucat::numeric_traits< Scalar_T >::promoted Struct Reference

Extra traits which extend numeric limits.

```
#include <promotion.h>
```

Public Types

- using type = double
- using type = long double

6.44.1 Detailed Description

template<typename Scalar_T>
struct glucat::numeric_traits< Scalar_T >::promoted

Extra traits which extend numeric limits.

Promoted type.

Promoted type for long double.

Promoted type for double

Definition at line 145 of file scalar.h.

6.44.2 Member Typedef Documentation

6.44.2.1 type [1/2]

```
template<typename Scalar_T >
typedef double glucat::numeric_traits< Scalar_T >::promoted::type = double
```

Definition at line 72 of file promotion.h.

6.44.2.2 type [2/2]

```
template<typename Scalar_T >
using glucat::numeric_traits< Scalar_T >::promoted::type = long double
```

Definition at line 86 of file promotion.h.

The documentation for this struct was generated from the following files:

- glucat/promotion.h
- glucat/scalar.h

6.45 glucat::random_generator< Scalar_T > Class Template Reference

Random number generator with single instance per Scalar_T.

```
#include <random.h>
```

Public Member Functions

- random_generator (const random_generator &)=delete
- auto operator= (const random_generator &) -> random_generator &=delete
- auto uniform () -> Scalar_T
- auto normal () -> Scalar_T

Static Public Member Functions

- static auto generator () -> random_generator &

    Single instance of Random number generator.

Private Member Functions

- random_generator ()
- ∼random_generator ()=default

Private Attributes

- std::mt19937 uint_gen
- std::uniform_real_distribution< double > uniform_dist
- std::normal_distribution< double > normal_dist

Static Private Attributes

- static const unsigned long seed = 19590921UL

Friends

- class friend_for_private_destructor

6.45.1 Detailed Description

template<typename Scalar_T>
class glucat::random_generator< Scalar_T >

Random number generator with single instance per Scalar_T.

Definition at line 42 of file random.h.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 random_generator() [1/2]

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::random_generator (
            const random_generator< Scalar_T > & )  [delete]
```

6.45.2.2 random_generator() [2/2]

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::random_generator ()  [inline], [private]
```

Definition at line 61 of file random.h.

References glucat::random_generator< Scalar_T >::seed.

6.45.2.3 ∼random_generator()

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::~random_generator ()  [private], [default]
```

6.45.3 Member Function Documentation

6.45.3.1 generator()

```
template<typename Scalar_T >
static auto glucat::random_generator< Scalar_T >::generator () -> random_generator&   [inline],
[static]
```

Single instance of Random number generator.

Definition at line 51 of file random.h.

6.45.3.2 normal()

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::normal () -> Scalar_T   [inline]
```

Definition at line 70 of file random.h.

References glucat::random_generator< Scalar_T >::normal_dist.

6.45.3.3 operator=()

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::operator= (
            const random_generator< Scalar_T > & ) -> random_generator &=delete  [delete]
```

6.45.3.4 uniform()

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::uniform () -> Scalar_T   [inline]
```

Definition at line 68 of file random.h.

References glucat::random_generator< Scalar_T >::uniform_dist.

6.45.4 Friends And Related Symbol Documentation

6.45.4.1 friend_for_private_destructor

```
template<typename Scalar_T >
friend class friend_for_private_destructor  [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE http://doc.ece.uci.edu

Definition at line 48 of file random.h.

6.45.5 Member Data Documentation

6.45.5.1 normal_dist

```
template<typename Scalar_T >
std::normal_distribution<double> glucat::random_generator< Scalar_T >::normal_dist  [private]
```

Definition at line 59 of file random.h.

Referenced by glucat::random_generator< Scalar_T >::normal().

6.45.5.2 seed

```
template<typename Scalar_T >
const unsigned long glucat::random_generator< Scalar_T >::seed = 19590921UL  [static], [private]
```

Definition at line 55 of file random.h.

Referenced by glucat::random_generator< Scalar_T >::random_generator().

6.45.5.3 uint_gen

```
template<typename Scalar_T >
std::mt19937 glucat::random_generator< Scalar_T >::uint_gen  [private]
```

Definition at line 57 of file random.h.

6.45.5.4 uniform_dist

```
template<typename Scalar_T >
std::uniform_real_distribution<double> glucat::random_generator< Scalar_T >::uniform_dist
[private]
```

Definition at line 58 of file random.h.

Referenced by glucat::random_generator< Scalar_T >::uniform().

The documentation for this class was generated from the following file:

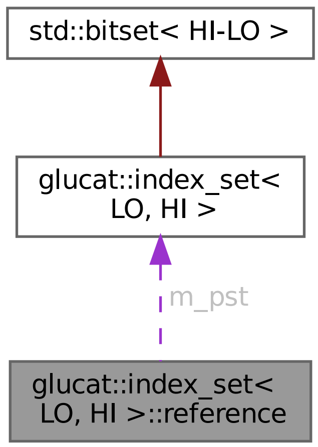
- glucat/random.h

6.46 glucat::index_set< LO, HI >::reference Class Reference

Index set member reference.

```
#include <index_set.h>
```

Collaboration diagram for glucat::index_set< LO, HI >::reference:



Public Member Functions

- reference ()=delete

  Default constructor is deleted.
- reference (index_set_t &ist, index_t idx)

  index_set reference
- ∼reference ()=default
- auto operator== (const reference &c_j) const -> bool

  for b[i] == c[j];
- auto operator= (const bool x) -> reference &

  for b[i] = x;
- auto operator= (const reference &c_j) -> reference &

  for b[i] = c[j];
- auto operator∼ () const -> bool

  Flips a bit.
- operator bool () const

  for x = b[i];
- auto flip () -> reference &

  for b[i].flip();

Private Attributes

- index_set_t ∗ m_pst
- index_t m_idx

Friends

- class index_set

6.46.1 Detailed Description

template<const index_t LO, const index_t HI>
class glucat::index_set< LO, HI >::reference

Index set member reference.

Definition at line 177 of file index_set.h.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 reference() [1/2]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::reference ()  [delete]
```

Default constructor is deleted.

6.46.2.2 reference() [2/2]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::reference (
            index_set_t & ist,
            index_t idx)  [inline]
```

index_set reference

Definition at line 985 of file index_set_imp.h.

6.46.2.3 ∼reference()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::∼reference ()  [default]
```

6.46.3 Member Function Documentation

6.46.3.1 flip()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::flip () -> reference&  [inline]
```

for b[i].flip();

Definition at line 1049 of file index_set_imp.h.

6.46.3.2 operator bool()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::operator bool () const [inline]
```

for x = b[i];

Definition at line 1041 of file index_set_imp.h.

6.46.3.3 operator=() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator= (
            const bool x) -> reference& [inline]
```

for b[i] = x;

Definition at line 1003 of file index_set_imp.h.

6.46.3.4 operator=() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator= (
            const reference & c_j) -> reference& [inline]
```

for b[i] = c[j];

Definition at line 1017 of file index_set_imp.h.

6.46.3.5 operator==()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator== (
            const reference & c_j) const -> bool [inline]
```

for b[i] == c[j];

Definition at line 995 of file index_set_imp.h.

6.46.3.6 operator∼()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator~ () const -> bool [inline]
```

Flips a bit.

flips the bit

Definition at line 1034 of file index_set_imp.h.

6.46.4 Friends And Related Symbol Documentation

6.46.4.1 index_set

```
template<const index_t LO, const index_t HI>
friend class index_set  [friend]
```

Definition at line 178 of file index_set.h.

6.46.5 Member Data Documentation

6.46.5.1 m_idx

```
template<const index_t LO, const index_t HI>
index_t glucat::index_set< LO, HI >::reference::m_idx  [private]
```

Definition at line 200 of file index_set.h.

6.46.5.2 m_pst

```
template<const index_t LO, const index_t HI>
index_set_t* glucat::index_set< LO, HI >::reference::m_pst  [private]
```

Definition at line 199 of file index_set.h.

The documentation for this class was generated from the following files:

- glucat/index_set.h
- glucat/index_set_imp.h

6.47 glucat::sorted_range< Map_T, Sorted_Map_T > Class Template Reference

Sorted range for use with output.

```
#include <framed_multi_imp.h>
```

Public Types

- using map_t = Map_T
- using sorted_map_t = Sorted_Map_T
- using sorted_iterator = typename Sorted_Map_T::const_iterator

Public Member Functions

- sorted_range (Sorted_Map_T &sorted_val, const Map_T &val)

Public Attributes

- sorted_iterator sorted_begin
- sorted_iterator sorted_end

6.47.1 Detailed Description

template<typename Map_T, typename Sorted_Map_T>
class glucat::sorted_range< Map_T, Sorted_Map_T >

Sorted range for use with output.

Definition at line 1112 of file framed_multi_imp.h.

6.47.2 Member Typedef Documentation

6.47.2.1 map_t

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::map_t = Map_T
```

Definition at line 1115 of file framed_multi_imp.h.

6.47.2.2 sorted_iterator

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_iterator = typename Sorted_Map_T↩
::const_iterator
```

Definition at line 1117 of file framed_multi_imp.h.

6.47.2.3 sorted_map_t

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_map_t = Sorted_Map_T
```

Definition at line 1116 of file framed_multi_imp.h.

6.47.3 Constructor & Destructor Documentation

6.47.3.1 sorted_range()

```
template<typename Map_T , typename Sorted_Map_T >
glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range (
            Sorted_Map_T & sorted_val,
            const Map_T & val) [inline]
```

Definition at line 1119 of file framed_multi_imp.h.

References glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_begin, and glucat::sorted_range< Map_T, Sorted_Map_T >::s

6.47.4 Member Data Documentation

6.47.4.1 sorted_begin

```
template<typename Map_T , typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_begin
```

Definition at line 1126 of file framed_multi_imp.h.

Referenced by glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range().

6.47.4.2 sorted_end

```
template<typename Map_T , typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_end
```

Definition at line 1127 of file framed_multi_imp.h.

Referenced by glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range().

The documentation for this class was generated from the following file:

- glucat/framed_multi_imp.h

6.48 glucat::sorted_range< Sorted_Map_T, Sorted_Map_T > Class Template Reference

```
#include <framed_multi_imp.h>
```

Public Types

- using map_t = Sorted_Map_T
- using sorted_map_t = Sorted_Map_T
- using sorted_iterator = typename Sorted_Map_T::const_iterator

Public Member Functions

- sorted_range (Sorted_Map_T &sorted_val, const Sorted_Map_T &val)

Public Attributes

- sorted_iterator sorted_begin
- sorted_iterator sorted_end

6.48.1 Detailed Description

template<typename Sorted_Map_T>
class glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >

Definition at line 1131 of file framed_multi_imp.h.

6.48.2 Member Typedef Documentation

6.48.2.1 map_t

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::map_t = Sorted_Map_T
```

Definition at line 1134 of file framed_multi_imp.h.

6.48.2.2 sorted_iterator

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_iterator = typename Sorted_←
Map_T::const_iterator
```

Definition at line 1136 of file framed_multi_imp.h.

6.48.2.3 sorted_map_t

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_map_t = Sorted_Map_T
```

Definition at line 1135 of file framed_multi_imp.h.

6.48.3 Constructor & Destructor Documentation

6.48.3.1 sorted_range()

```
template<typename Sorted_Map_T >
glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_range (
            Sorted_Map_T & sorted_val,
            const Sorted_Map_T & val )  [inline]
```

Definition at line 1138 of file framed_multi_imp.h.

6.48.4 Member Data Documentation

6.48.4.1 sorted_begin

```
template<typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_begin
```

Definition at line 1142 of file framed_multi_imp.h.

6.48.4.2 sorted_end

```
template<typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_end
```

Definition at line 1143 of file framed_multi_imp.h.

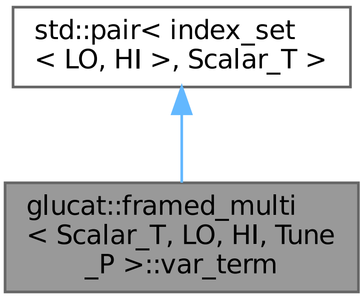
The documentation for this class was generated from the following file:

- glucat/framed_multi_imp.h

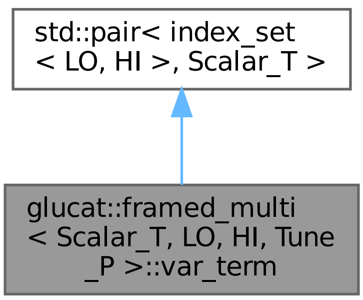
6.49 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term Class Reference

Variable term.

Inheritance diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term:



Collaboration diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term:



Public Types

- using var_pair_t = std::pair<index_set<LO, HI>, Scalar_T>

Public Member Functions

- ~var_term ()=default
  Destructor.
- var_term ()
  Default constructor.
- var_term (const index_set_t ist, const Scalar_T &crd=Scalar_T(1))
  Construct a variable term from an index set and a scalar coordinate.
- auto operator*= (const term_t &rhs) -> var_term_t &
  Product of variable term and term.

Static Public Member Functions

- static auto classname () -> const std::string
  Class name used in messages.

6.49.1 Detailed Description

template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P>
class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term

Variable term.

Definition at line 279 of file framed_multi.h.

6.49.2 Member Typedef Documentation

6.49.2.1 var_pair_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_pair_t = std::pair<index_set<LO,
HI>, Scalar_T>
```

Definition at line 283 of file framed_multi.h.

6.49.3 Constructor & Destructor Documentation

6.49.3.1 ~var_term()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::~var_term ()  [default]
```

Destructor.

6.49.3.2 var_term() [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_term ()  [inline]
```

Default constructor.

Definition at line 291 of file framed_multi.h.

6.49.3.3 var_term() [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_term (
            const index_set_t ist,
            const Scalar_T & crd = Scalar_T(1))  [inline]
```

Construct a variable term from an index set and a scalar coordinate.

Definition at line 295 of file framed_multi.h.

6.49.4 Member Function Documentation

6.49.4.1 classname()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::classname () -> const
std::string  [inline], [static]
```

Class name used in messages.

Definition at line 286 of file framed_multi.h.

6.49.4.2 operator*=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::operator*= (
            const term_t & rhs) -> var_term_t&  [inline]
```

Product of variable term and term.

Definition at line 299 of file framed_multi.h.

The documentation for this class was generated from the following file:

- glucat/framed_multi.h

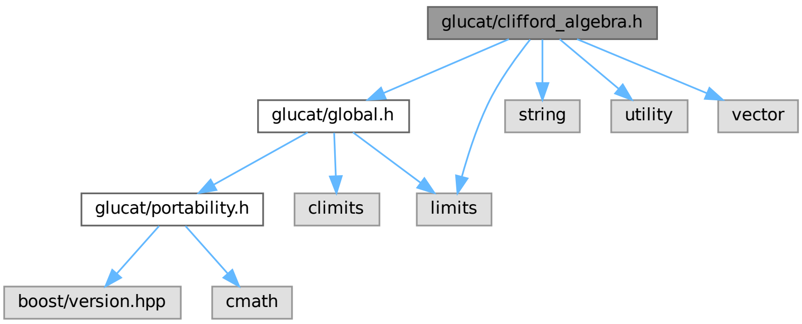
Chapter 7

File Documentation

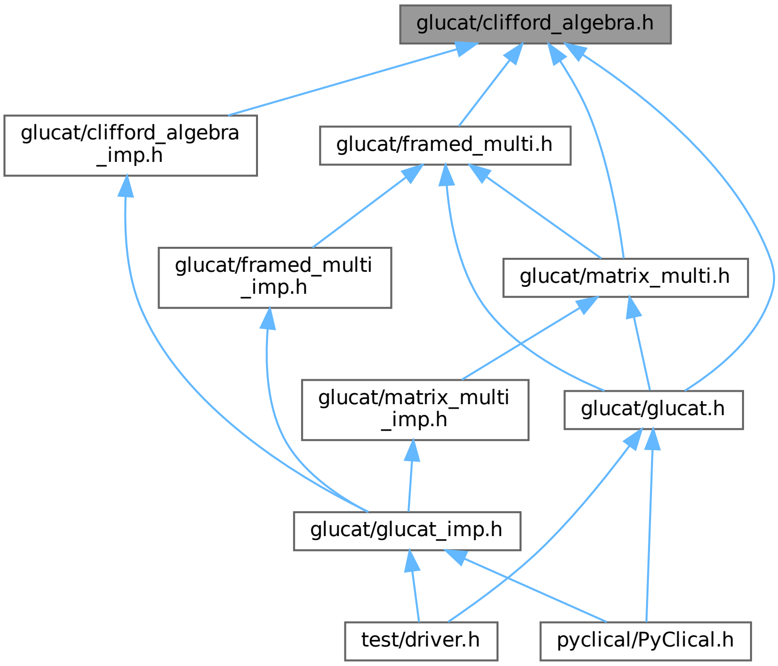
7.1 glucat/clifford_algebra.h File Reference

```
#include "glucat/global.h"
#include <limits>
#include <string>
#include <utility>
#include <vector>
```

Include dependency graph for clifford_algebra.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >

    clifford_algebra<> declares the operations of a Clifford algebra

Namespaces

- namespace glucat

Macros

- #define _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS

Functions

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

    Test for inequality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool

Test for inequality of multivector and scalar.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator!= (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

  Test for inequality of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::error_squared_tol (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Quadratic norm error tolerance relative to a specific multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::error_squared (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T

  Relative or absolute error using the quadratic norm.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool

  Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

  Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric sum.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator- (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inner product.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Left contraction.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune↩
_P > &rhs) -> Scalar_T

  Hestenes scalar product.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator| (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Transformation via twisted adjoint action.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric multiplicative inverse.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Integer power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune↩ _P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Multivector power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::outer_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Real part: synonym for scalar part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Imaginary part: deprecated (always 0)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Pure part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Even part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Odd part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::vector_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar↩ _T >

  Vector part of multivector, as a vector_t with respect to frame()
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Main involution, each {i} is replaced by -{i} in each term, eg. {1}∗{2} -> (-{2})∗(-{1})

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Reversion, eg. {1}∗{2} -> {2}∗{1}.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Conjugation, rev o invo == invo o rev.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar_T quadratic form == (rev(x)∗x)(0)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::norm (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar_T norm == sum of norm of coordinates.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Absolute value == sqrt(norm)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::max_abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Maximum of absolute values of components of multivector: multivector infinity norm.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::complexifier (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of -1 which commutes with all members of the frame of the given multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::elliptic (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::clifford_exp (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Natural logarithm of multivector with specified complexifier.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::cosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Inverse sine of multivector with specified complexifier.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::tanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic tangent of multivector.

7.1.1 Macro Definition Documentation

7.1.1.1 _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS

```
#define _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
```

Definition at line 145 of file clifford_algebra.h.

7.2 clifford_algebra.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_CLIFFORD_ALGEBRA_H
#define _GLUCAT_CLIFFORD_ALGEBRA_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    clifford_algebra.h : Declare the operations of a Clifford algebra
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"

#include <limits>
#include <string>
#include <utility>
#include <vector>

namespace glucat
{
  template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
  class clifford_algebra
  {
  public:
    using scalar_t = Scalar_T;
    using index_set_t = Index_Set_T;
    static const index_t v_lo = index_set_t::v_lo;
    static const index_t v_hi = index_set_t::v_hi;
    using multivector_t = Multivector_T;
    using pair_t = std::pair<const index_set_t, Scalar_T>;
    using vector_t = std::vector<Scalar_T>;

    static  auto   classname() -> const std::string;

    static const Scalar_T default_truncation;

    virtual ~clifford_algebra() = default;

  // clifford_algebra operations
    virtual auto operator==  (const multivector_t& val) const -> bool = 0;
    virtual auto operator==  (const Scalar_T& scr) const -> bool = 0;
    virtual auto operator+=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator+=  (const Scalar_T& scr) -> multivector_t& = 0;
    virtual auto operator-=  (const multivector_t& rhs) -> multivector_t& = 0;
```

```
    virtual auto operator-=  (const Scalar_T& scr) -> multivector_t& = 0;
    virtual auto operator-   () const -> const multivector_t = 0;
    virtual auto operator*=  (const Scalar_T& scr) -> multivector_t& = 0;
    virtual auto operator*=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator%=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator&=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator^=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator/=  (const Scalar_T& scr) -> multivector_t& = 0;
    virtual auto operator/=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto operator|=  (const multivector_t& rhs) -> multivector_t& = 0;
    virtual auto inv         () const -> const multivector_t = 0;
    virtual auto pow         (int m) const -> const multivector_t = 0;
    virtual auto outer_pow   (int m) const -> const multivector_t = 0;
    virtual auto frame       () const -> const index_set_t = 0;
    virtual auto grade       () const -> index_t = 0;
    virtual auto operator[]  (const index_set_t ist) const -> Scalar_T = 0;
    virtual auto operator()  (index_t grade) const -> const multivector_t = 0;
    virtual auto scalar      () const -> Scalar_T = 0;
    virtual auto pure        () const -> const multivector_t = 0;
    virtual auto even        () const -> const multivector_t = 0;
    virtual auto odd         () const -> const multivector_t = 0;
    virtual auto vector_part () const -> const vector_t = 0;
    virtual auto vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t = 0;
    virtual auto involute    () const -> const multivector_t = 0;
    virtual auto reverse     () const -> const multivector_t = 0;
    virtual auto conj        () const -> const multivector_t = 0;
    virtual auto quad        () const -> Scalar_T = 0;
    virtual auto norm        () const -> Scalar_T = 0;
    virtual auto max_abs     () const -> Scalar_T = 0;
    virtual auto truncated   (const Scalar_T& limit = default_truncation) const -> const multivector_t = 0;
    virtual auto isinf       () const -> bool = 0;
    virtual auto isnan       () const -> bool = 0;
    virtual void write       (const std::string& msg="") const = 0;
    virtual void write       (std::ofstream& ofile, const std::string& msg="") const = 0;
  };

#ifndef _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
#define _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS \
    auto operator==  (const multivector_t& val) const  -> bool                override;\
    auto operator==  (const Scalar_T& scr) const       -> bool                override;\
    auto operator+=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator+=  (const Scalar_T& scr)             -> multivector_t&      override;\
    auto operator-=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator-=  (const Scalar_T& scr)             -> multivector_t&      override;\
    auto operator-   () const                          -> const multivector_t override;\
    auto operator*=  (const Scalar_T& scr)             -> multivector_t&      override;\
    auto operator*=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator%=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator&=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator^=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator/=  (const Scalar_T& scr)             -> multivector_t&      override;\
    auto operator/=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto operator|=  (const multivector_t& rhs)        -> multivector_t&      override;\
    auto inv         () const                          -> const multivector_t override;\
    auto pow         (int m) const                     -> const multivector_t override;\
    auto outer_pow   (int m) const                     -> const multivector_t override;\
    auto frame       () const                          -> const index_set_t   override;\
    auto grade       () const                          -> index_t             override;\
    auto operator[]  (const index_set_t ist) const     -> Scalar_T            override;\
    auto operator()  (index_t grade) const             -> const multivector_t override;\
    auto scalar      () const                          -> Scalar_T            override;\
    auto pure        () const                          -> const multivector_t override;\
    auto even        () const                          -> const multivector_t override;\
    auto odd         () const                          -> const multivector_t override;\
    auto vector_part () const                          -> const vector_t      override;\
    auto vector_part (const index_set_t frm, const bool prechecked = false) const     \
                                                       -> const vector_t      override;\
    auto involute    () const                          -> const multivector_t override;\
    auto reverse     () const                          -> const multivector_t override;\
    auto conj        () const                          -> const multivector_t override;\
    auto quad        () const                          -> Scalar_T            override;\
    auto norm        () const                          -> Scalar_T            override;\
    auto max_abs     () const                          -> Scalar_T            override;\
    auto truncated   (const Scalar_T& limit = multivector_t::default_truncation) const \
                                                       -> const multivector_t override;\
    auto isinf       () const                          -> bool                override;\
    auto isnan       () const                          -> bool                override;\
    void write       (const std::string& msg="") const                        override;\
    void write       (std::ofstream& ofile, const std::string& msg="") const  override;
#endif // _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
```

```
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> bool;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator!= (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> bool;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  error_squared_tol(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  error_squared(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
                const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
                const Scalar_T threshold) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
               const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
               const Scalar_T threshold,
               const Scalar_T tolerance) -> bool;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
               const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator+ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
```

```
  auto
  operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator- (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator* (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator^ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator& (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
```

```
  operator% (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  star (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator/ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  operator| (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  inv(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  outer_pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
```

```
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
scalar(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
real(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
imag(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

template
<
```

```
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  norm(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  complexifier(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
       const Multivector<Scalar_T,LO,HI,Tune_P>& i,
       const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  // Transcendental functions

  template
  < template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  clifford_exp(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  log(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
      const Multivector<Scalar_T,LO,HI,Tune_P>& i,
      const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
```

```
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
    const Multivector<Scalar_T,LO,HI,Tune_P>& i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
     const Multivector<Scalar_T,LO,HI,Tune_P>& i,
     const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
      const Multivector<Scalar_T,LO,HI,Tune_P>& i,
      const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
    const Multivector<Scalar_T,LO,HI,Tune_P>& i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
     const Multivector<Scalar_T,LO,HI,Tune_P>& i,
     const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
```

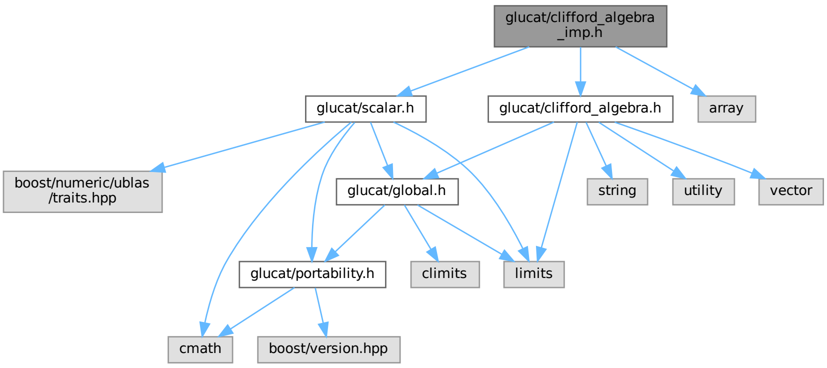
```
template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
sinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
      const Multivector<Scalar_T,LO,HI,Tune_P>& i,
      const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
    const Multivector<Scalar_T,LO,HI,Tune_P>& i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
     const Multivector<Scalar_T,LO,HI,Tune_P>& i,
     const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
tanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
auto
atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
      const Multivector<Scalar_T,LO,HI,Tune_P>& i,
```

```
        const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
}
#endif  // _GLUCAT_CLIFFORD_ALGEBRA_H
```

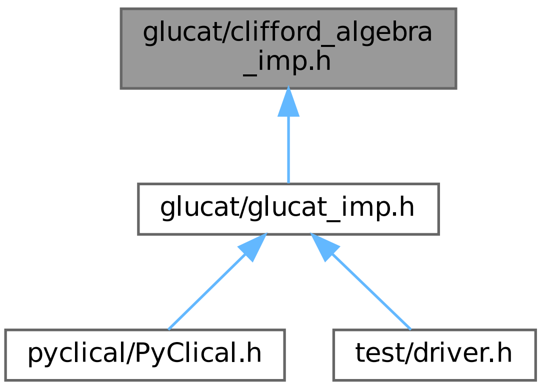
7.3 glucat/clifford_algebra_imp.h File Reference

```
#include "glucat/clifford_algebra.h"
#include "glucat/scalar.h"
#include <array>
```

Include dependency graph for clifford_algebra_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Functions

- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

    Test for inequality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool

    Test for inequality of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!= (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

    Test for inequality of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared_tol (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

    Quadratic norm error tolerance relative to a specific multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T

    Relative or absolute error using the quadratic norm.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool

    Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool

    Test for approximate equality of multivectors.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

    Geometric sum of multivector and scalar.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

    Geometric sum of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

    Geometric sum.

- template< template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of multivector and scalar.
- template< template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference of scalar and multivector.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric difference.
- template< template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of multivector and scalar.
- template< template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Product of scalar and multivector.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inner product.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Left contraction.
- template< template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

  Hestenes scalar product.
- template< template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of multivector and scalar.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator/ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Quotient of scalar and multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator| (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Transformation via twisted adjoint action.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Geometric multiplicative inverse.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Integer power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune↩ _P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Multivector power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::outer_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Outer product power of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Real part: synonym for scalar part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Imaginary part: deprecated (always 0)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Pure part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Even part.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Odd part.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::vector_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar_T >

  Vector part of multivector, as a vector_t with respect to frame()
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Main involution, each {i} is replaced by -{i} in each term, eg. {1}∗{2} -> (-{2})∗(-{1})
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Reversion, eg. {1}∗{2} -> {2}∗{1}.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Conjugation, rev o invo == invo o rev.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar_T quadratic form == (rev(x)∗x)(0)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::norm (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Scalar_T norm == sum of norm of coordinates.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Absolute value == sqrt(norm)
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::max_abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

  Maximum of absolute values of components of multivector: multivector infinity norm.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::complexifier (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of -1 which commutes with all members of the frame of the given multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::elliptic (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
static void glucat::check_complex (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false)

Check that i is a valid complexifier for val.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Square root of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::clifford_exp (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Cosine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Inverse cosine of multivector with specified complexifier.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse cosine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse sine of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse sine of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::tanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Hyperbolic tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse hyperbolic tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Inverse hyperbolic tangent of multivector.

- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Tangent of multivector.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector with specified complexifier.
- template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

  Inverse tangent of multivector.

7.4 clifford_algebra_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_CLIFFORD_ALGEBRA_IMP_H
#define _GLUCAT_CLIFFORD_ALGEBRA_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    clifford_algebra_imp.h : Implement common Clifford algebra functions
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

// References for algorithms:
// [AS]:
// Milton Abramowicz and Irene A. Stegun, "Handbook of mathematical functions",
// Dover 1972, first published 1965.
// [CHKL]:
// Sheung Hun Cheng, Nicholas J. Higham, Charles S. Kenney and Alan J. Laub,
// "Approximating the Logarithm of a Matrix to Specified Accuracy", 1999.
// ftp://ftp.ma.man.ac.uk/pub/narep/narep353.ps.gz
// [GL]:
// Gene H. Golub and Charles F. van Loan,
// "Matrix Computations", 3rd ed., Johns Hopkins UP, 1996.
```

```
// [GW]:
// C.F. Gerald and P.O. Wheatley, "Applied Numerical Analysis",
// 6th Edition, Addison-Wesley, 1999.
// [H]:
// Nicholas J. Higham
// "The Scaling and Squaring Method for the Matrix Exponential Revisited",
// SIAM Journal on Matrix Analysis and Applications,
// Vol. 26,  Issue 4 (2005), pp. 1179-1193.
// [Z]:
// Doron Zeilberger, "PADE" (Maple code), 2002.
// http://www.math.rutgers.edu/~zeilberg/tokhniot/PADE

#include "glucat/clifford_algebra.h"
#include "glucat/scalar.h"

#include <array>

namespace glucat
{
  template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
  auto
  clifford_algebra<Scalar_T,Index_Set_T,Multivector_T>::
  classname() -> const std::string
  { return "clifford_algebra"; }

  template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
  const
  Scalar_T
  clifford_algebra<Scalar_T,Index_Set_T,Multivector_T>::
  default_truncation = std::numeric_limits<Scalar_T>::epsilon();

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool
  { return !(lhs == rhs); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> bool
  { return !(lhs == scr); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator!= (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> bool
  { return !(rhs == scr); }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  auto
  error_squared_tol(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    static const auto scalar_eps  = std::numeric_limits<Scalar_T>::epsilon();
    static const auto nbr_different_bits =
      std::numeric_limits<Scalar_T>::digits / Tune_P::denom_different_bits + Tune_P::extra_different_bits;
    static const auto abs_tol = scalar_eps *
      numeric_traits<Scalar_T>::pow(Scalar_T(2), nbr_different_bits);
    using framed_multi_t = typename multivector_t::framed_multi_t;
    const auto nbr_terms = double(framed_multi_t(val).truncated(scalar_eps).nbr_terms());
    return abs_tol * abs_tol * std::max(Scalar_T(nbr_terms), Scalar_T(1));
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  error_squared(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
                const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
```

```
                const Scalar_T threshold) -> Scalar_T
  {
    const auto relative = norm(rhs) > threshold;
    const auto abs_norm_diff = norm(rhs-lhs);
    return (relative)
      ? abs_norm_diff/norm(rhs)
      : abs_norm_diff;
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
               const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
               const Scalar_T threshold,
               const Scalar_T tolerance) -> bool
  { return error_squared(lhs, rhs, threshold) < tolerance; }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
               const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool
  {
    const Scalar_T rhs_tol = error_squared_tol(rhs);
    return approx_equal(lhs, rhs, rhs_tol, rhs_tol);
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result += scr;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator+ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    return rhs + scr;
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result += rhs;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result -= scr;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
```

```
  inline
  auto
  operator- (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return -rhs + scr; }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result -= rhs;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result *= scr;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator* (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { // Note: this assumes that scalar commutes with multivector.
    // This excludes Clifford algebras over non-commuting rings.
    return rhs * scr;
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs * multivector_t(rhs);
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator^ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs ^ multivector_t(rhs);
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator& (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs & multivector_t(rhs);
  }
```

```
  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator% (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs % multivector_t(rhs);
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  star (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> Scalar_T
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return star(lhs, multivector_t(rhs));
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    auto result = lhs;
    return result /= scr;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  operator/ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    Multivector<Scalar_T,LO,HI,Tune_P> result = scr;
    return result /= rhs;
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs / multivector_t(rhs);
  }

  template
  <
    template<typename, const index_t, const index_t, typename> class Multivector,
    template<typename, const index_t, const index_t, typename> class RHS,
    typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
  >
  inline
  auto
  operator| (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    return lhs | multivector_t(rhs);
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
```

```
inv(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.inv(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
  if (lhs == Scalar_T(0))
  {
    using traits_t = numeric_traits<Scalar_T>;
    return
      (rhs < 0)
      ? traits_t::NaN()
      : (rhs == 0)
        ? Scalar_T(1)
        : Scalar_T(0);
  }
  auto result = multivector_t(Scalar_T(1));
  auto power =
    (rhs < 0)
    ? lhs.inv()
    : lhs;
  for (auto
      k = std::abs(rhs);
      k != 0;
      k /= 2)
  {
    if (k % 2)
      result *= power;
    power *= power;
  }
  return result;
}

template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  template<typename, const index_t, const index_t, typename> class RHS,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
inline
auto
pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;

  if (lhs == Scalar_T(0))
  {
    const Scalar_T m = rhs.scalar();
    if (rhs == m)
      return
        (m < 0)
        ? traits_t::NaN()
        : (m == 0)
          ? Scalar_T(1)
          : Scalar_T(0);
    else
      return Scalar_T(0);
  }
  return exp(log(lhs) * rhs);
}

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
outer_pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return lhs.outer_pow(rhs); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
scalar(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return val.scalar(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
real(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return val.scalar(); }

```

```
template
<
  template<typename, const index_t, const index_t, typename> class Multivector,
  typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
>
inline
auto
imag(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return Scalar_T(0); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val - val.scalar(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.even(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.odd(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>
{ return val.vector_part(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.involute(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.reverse(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return val.conj(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return val.quad(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
norm(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return val.norm(); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return numeric_traits<Scalar_T>::sqrt(val.norm()); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
{ return val.max_abs(); }
```

```
template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
complexifier(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
  using traits_t = numeric_traits<Scalar_T>;

  auto frm = val.frame();
  using array_t = std::array<index_t, 4>;
  auto incp = array_t{0, 2, 1, 0};
  auto incq = array_t{1, 0, 0, 0};
  auto bott = pos_mod((frm.count_pos() - frm.count_neg()), 4);
  for (auto
      k = index_t(0);
      k != incp[bott];
      k++)
    for (auto
        idx = index_t(1);
        idx != HI+1;
        ++idx)
      if (!frm[idx])
      {
        frm.set(idx);
        break;
      }
  for (auto
      k = index_t(0);
      k != incq[bott];
      k++)
    for (auto
        idx = index_t(-1);
        idx != LO-1;
        --idx)
      if (!frm[idx])
      {
        frm.set(idx);
        break;
      }
  auto new_bott = pos_mod(frm.count_pos() - frm.count_neg(), 4);

  if ((incp[new_bott] == 0) && (incq[new_bott] == 0))
    return multivector_t(frm, Scalar_T(1));
  else
    // Return IEEE NaN or -Inf
    return traits_t::NaN();
}

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return complexifier(val); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
static
void
check_complex(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
              const Multivector<Scalar_T,LO,HI,Tune_P>& i, const bool prechecked = false)
{
  if (!prechecked)
  {
    using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
    using index_set_t = typename multivector_t::index_set_t;
    using error_t = typename multivector_t::error_t;

    const auto i_frame = i.frame();
    // We need i to be a complexifier whose frame is large enough to represent val
    if (complexifier(i) != i ||
        (val.frame() | i_frame) != i_frame ||
        complexifier(val).frame().count() > i_frame.count())
      throw error_t("check_complex(val, i): i is not a valid complexifier for val");
  }
}

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return sqrt(val, i, prechecked); }
```

```
template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return sqrt(val, complexifier(val), true); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
clifford_exp(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
{
  // Scaling and squaring Pade' approximation of matrix exponential
  // Reference: [GL], Section 11.3, p572-576
  // Reference: [H]

  using traits_t = numeric_traits<Scalar_T>;

  const auto scalar_val = val.scalar();
  const auto scalar_exp = traits_t::exp(scalar_val);
  if (traits_t::isNaN_or_isInf(scalar_exp))
    return traits_t::NaN();
  if (val == scalar_val)
    return scalar_exp;

  using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
  auto A = val - scalar_val;
  const auto pure_scale2 = A.norm();

  if (traits_t::isNaN_or_isInf(pure_scale2))
    return traits_t::NaN();
  if (pure_scale2 == Scalar_T(0))
    return scalar_exp;

  const auto ilog2_scale =
    std::max(0, traits_t::to_int(ceil((log2(pure_scale2) +
Scalar_T(A.frame().count()))/Scalar_T(2))) - 3);
  const auto i_scale = traits_t::pow(Scalar_T(2), ilog2_scale);
  if (traits_t::isNaN_or_isInf(i_scale))
    return traits_t::NaN();

  A /= i_scale;
  multivector_t pure_exp;
  {
    using limits_t = std::numeric_limits<Scalar_T>;
    const auto nbr_even_powers = 2*(limits_t::digits / 32) + 4;
    using nbr_t = decltype(nbr_even_powers);

    // Create an array of coefficients
    const auto max_power = 2*nbr_even_powers + 1;
    static std::array<Scalar_T, max_power+1> c;
    if (c[0] != Scalar_T(1))
    {
      c[0] = Scalar_T(1);
      for (auto
          k = decltype(max_power)(0);
          k != max_power;
          ++k)
        c[k+1] = c[k]*(max_power-k) / ((2*max_power-k)*(k+1));
    }

    // Create an array of even powers
    std::array<multivector_t, nbr_even_powers> AA;
    AA[0] = A * A;
    AA[1] = AA[0] * AA[0];
    for (auto
      k = nbr_t(2);
      k != nbr_even_powers;
      ++k)
      AA[k] = AA[k-2] * AA[1];

    // Use compensated summation to calculate U and AV
    auto residual = multivector_t();
    auto U = multivector_t(c[0]);
    for (auto
        k = nbr_t(0);
        k != nbr_even_powers;
        ++k)
    {
      const auto& term = AA[k]*c[2*k + 2] - residual;
      const auto& sum = U + term;
      residual = (sum - U) - term;
      U = sum;
    }
    residual = multivector_t();
    auto AV = multivector_t(c[1]);
```

```
      for (auto
          k = nbr_t(0);
          k != nbr_even_powers;
          ++k)
      {
        const auto& term = AA[k]*c[2*k + 3] - residual;
        const auto& sum = AV + term;
        residual = (sum - AV) - term;
        AV = sum;
      }
      AV *= A;
      pure_exp = (U+AV) / (U-AV);
    }
    for (auto
        k = decltype(ilog2_scale)(0);
        k != ilog2_scale;
        ++k)
      pure_exp *= pure_exp;
    return pure_exp * scalar_exp;
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  log(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return log(val, i, prechecked); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return log(val, complexifier(val), true); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s)
      return traits_t::cosh(s);
    return (exp(val)+exp(-val)) / Scalar_T(2);
  }

  // Reference: [AS], Section 4.6, p86-89
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    check_complex(val, i, prechecked);
    if (val.isnan())
      return traits_t::NaN();

    const auto radical = sqrt(val*val - Scalar_T(1), i, true);
    return (norm(val + radical) >= norm(val))
           ?  log(val + radical, i, true)
           : -log(val - radical, i, true);
  }

  // Reference: [AS], Section 4.6, p86-89
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return acosh(val, complexifier(val), true); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
```

```
    return traits_t::NaN();

  const auto& s = val.scalar();
  if (val == s)
    return traits_t::cos(s);

  check_complex(val, i, prechecked);

  static const auto& twopi = Scalar_T(2) * traits_t::pi();
  const auto& z = i *
    (val - s + traits_t::fmod(s, twopi));
  return (exp(z)+exp(-z)) / Scalar_T(2);
}

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return cos(val, complexifier(val), true); }

// Reference: [AS], Section 4.4, p79-83
template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;
  if (val.isnan())
    return traits_t::NaN();

  const auto& s = val.scalar();
  if (val == s && traits_t::abs(s) <= Scalar_T(1))
    return traits_t::acos(s);

  check_complex(val, i, prechecked);
  return i * acosh(val, i, true);
}

// Reference: [AS], Section 4.4, p79-83
template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return acos(val, complexifier(val), true); }

template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
sinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;
  if (val.isnan())
    return traits_t::NaN();

  const auto& s = val.scalar();
  if (val == s)
    return traits_t::sinh(s);

  return (exp(val)-exp(-val)) / Scalar_T(2);
}

// Reference: [AS], Section 4.6, p86-89
template< template<typename, const index_t, const index_t, typename> class Multivector,
          typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;
  check_complex(val, i, prechecked);
  if (val.isnan())
    return traits_t::NaN();

  const auto radical = sqrt(val*val + Scalar_T(1), i, true);
  return (norm(val + radical) >= norm(val))
         ?  log( val + radical, i, true)
         : -log(-val + radical, i, true);
}

// Reference: [AS], Section 4.6, p86-89
template< template<typename, const index_t, const index_t, typename> class Multivector,
```

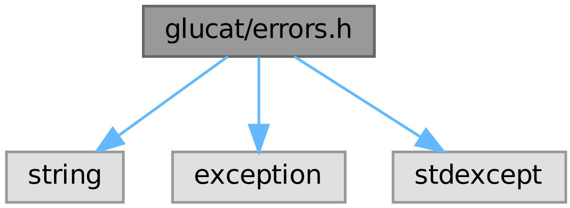
```
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return asinh(val, complexifier(val), true); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s)
      return traits_t::sin(s);

    check_complex(val, i, prechecked);

    static const auto& twopi = Scalar_T(2) * traits_t::pi();
    const auto& z = i *
      (val - s + traits_t::fmod(s, twopi));
    return i * (exp(-z)-exp(z)) / Scalar_T(2);
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return sin(val, complexifier(val), true); }

  // Reference: [AS], Section 4.4, p79-83
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s && traits_t::abs(s) <= Scalar_T(1))
      return traits_t::asin(s);

    check_complex(val, i, prechecked);
    return -i * asinh(i * val, i, true);
  }

  // Reference: [AS], Section 4.4, p79-83
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return asin(val, complexifier(val), true); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  tanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s)
      return traits_t::tanh(s);

    return sinh(val) / cosh(val);
  }

  // Reference: [AS], Section 4.6, p86-89
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

```
  {
    using traits_t = numeric_traits<Scalar_T>;
    check_complex(val, i, prechecked);
    return val.isnan()
        ? traits_t::NaN()
        : (norm(val + Scalar_T(1)) > norm(val - Scalar_T(1)))
          ? (log(val + Scalar_T(1), i, true) - log(-val + Scalar_T(1), i, true)) / Scalar_T(2)
          : log((val + Scalar_T(1)) / (-val + Scalar_T(1)), i, true) / Scalar_T(2);
  }

  // Reference: [AS], Section 4.6, p86-89
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return atanh(val, complexifier(val), true); }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s)
      return traits_t::tan(s);

    check_complex(val, i, prechecked);
    return sin(val, i, true) / cos(val, i, true);
  }

  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return tan(val, complexifier(val), true); }

  // Reference: [AS], Section 4.4, p79-83
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto& s = val.scalar();
    if (val == s)
      return traits_t::atan(s);

    check_complex(val, i, prechecked);
    return -i * atanh(i * val, i, true);
  }

  // Reference: [AS], Section 4.4, p79-83
  template< template<typename, const index_t, const index_t, typename> class Multivector,
            typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
  { return atan(val, complexifier(val), true); }

}
#endif  // _GLUCAT_CLIFFORD_ALGEBRA_IMP_H
```

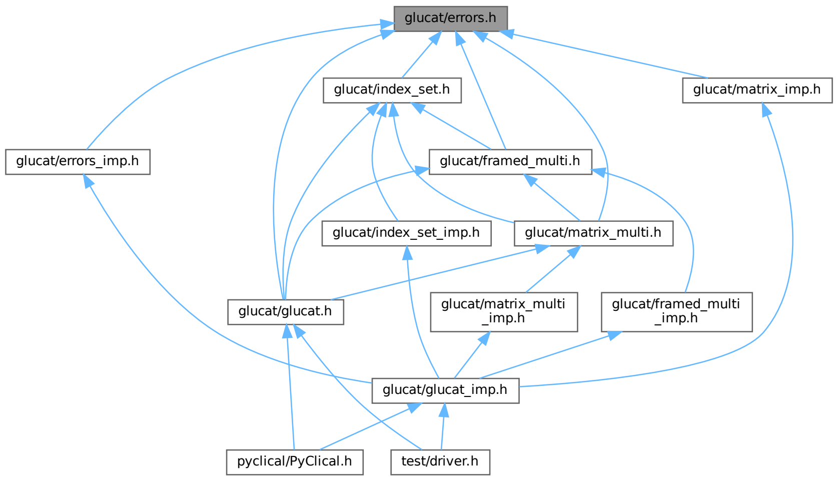
7.5 glucat/errors.h File Reference

```
#include <string>
#include <exception>
```

`#include <stdexcept>`
Include dependency graph for errors.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::glucat_error

    Abstract exception class.
- class glucat::error< Class_T >

    Specific exception class.

Namespaces

- namespace glucat

7.6 errors.h

Go to the documentation of this file.

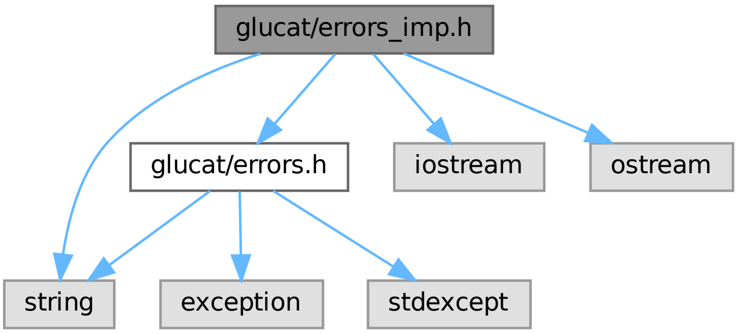
```
#ifndef _GLUCAT_ERRORS_H
#define _GLUCAT_ERRORS_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    errors.h : Declare error classes and functions
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include <string>
#include <exception>
#include <stdexcept>

namespace glucat
{
  class glucat_error : public std::logic_error
  {
  public:
    glucat_error(const std::string& context, const std::string& msg)
    : logic_error(msg), name(context)
    { }
    ~glucat_error() noexcept override = default;
    virtual auto heading() const noexcept -> const std::string =0;
    virtual auto classname() const noexcept -> const std::string =0;
    virtual void print_error_msg() const =0;
    std::string name;
  };

  template< class Class_T >
  class error : public glucat_error
  {
  public:
    error(const std::string& msg);
    error(const std::string& context, const std::string& msg);
    auto heading() const noexcept -> const std::string override;
    auto classname() const noexcept -> const std::string override;
    void print_error_msg() const override;
  };
}
#endif // _GLUCAT_ERRORS_H
```

7.7 glucat/errors_imp.h File Reference

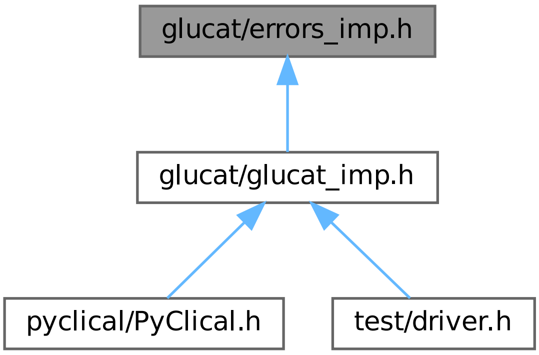
```
#include "glucat/errors.h"
#include <string>
#include <iostream>
```

`#include <ostream>`

Include dependency graph for errors_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

7.8 errors_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_ERRORS_IMP_H
#define _GLUCAT_ERRORS_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    errors_imp.h : Define error functions
                             -------------------
```

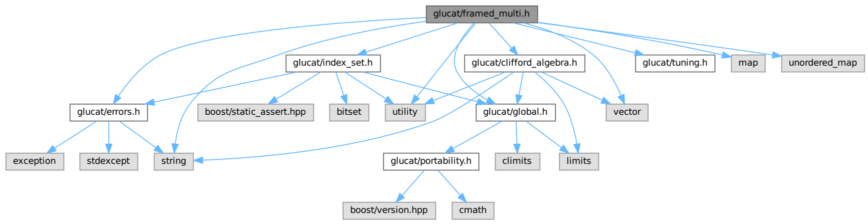
```
    begin                : Sun 2001-12-20
    copyright            : (C) 2001-2007 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/errors.h"

#include <string>
#include <iostream>
#include <ostream>

namespace glucat
{
  template< class Class_T >
  error<Class_T>::
  error(const std::string& msg)
  : glucat_error(Class_T::classname(), msg)
  { }

  template< class Class_T >
  error<Class_T>::
  error(const std::string& context, const std::string& msg)
  : glucat_error(context, msg)
  { }

  template< class Class_T >
  auto
  error<Class_T>::
  heading() const noexcept -> const std::string
  { return "Error in glucat::"; }

  template< class Class_T >
  auto
  error<Class_T>::
  classname() const noexcept -> const std::string
  { return name; }

  template< class Class_T >
  void
  error<Class_T>::
  print_error_msg() const
  { std::cerr « heading() « classname() « std::endl « what() « std::endl; }
}
#endif // _GLUCAT_ERRORS_IMP_H
```

7.9 glucat/framed_multi.h File Reference

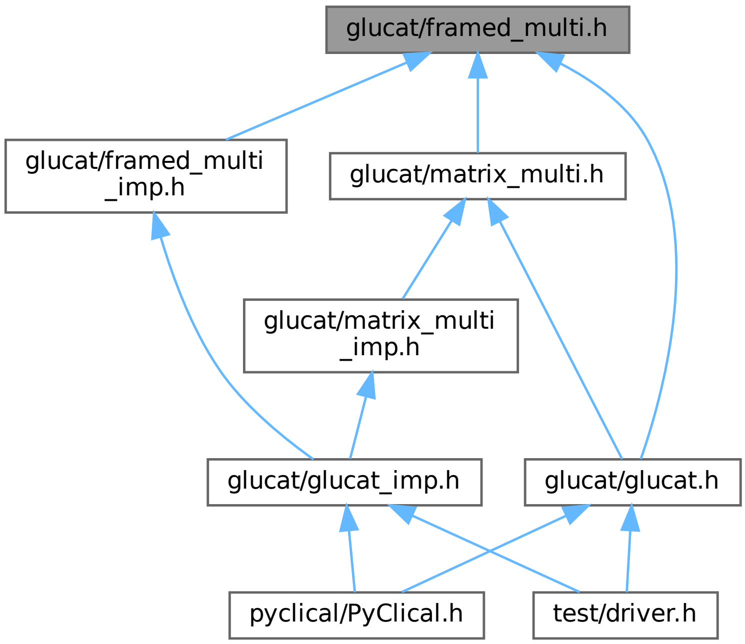
```
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include <string>
#include <utility>
#include <map>
```

```
#include <unordered_map>
#include <vector>
```

Include dependency graph for framed_multi.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::index_set_hash< LO, HI >
- class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >

  A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.
- class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t
- class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term

  Variable term.
- struct std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >

  Numeric limits for framed_multi inherit limits for the corresponding scalar type.

Namespaces

- namespace glucat
- namespace std

Functions

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator* (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator^ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Outer product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator& (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Inner product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator% (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Left contraction.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::star (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

  Hestenes scalar product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator/ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator| (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Transformation via twisted adjoint action.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator>> (std::istream &s, framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

  Read multivector from input.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator<< (std::ostream &os, const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

  Write multivector to output.
- template<typename Scalar_T , const index_t LO, const index_t HI>
  auto glucat::operator<< (std::ostream &os, const std::pair< const index_set< LO, HI >, Scalar_T > &term) -> std::ostream &

  Write term to output.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::exp (const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.
- template<typename Scalar_T , const index_t LO, const index_t HI>
  static auto glucat::crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T

Coordinate of product of terms.

- template<typename Scalar_T , const index_t LO, const index_t HI>
  auto glucat::operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >

  Product of terms.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sqrt (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::log (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.

7.10 framed_multi.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_FRAMED_MULTI_H
#define _GLUCAT_FRAMED_MULTI_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    framed_multi.h : Declare a class for the framed representation of a multivector
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"

#if defined(_GLUCAT_USE_BOOST_POOL_ALLOC)
// Use the Boost pool allocator
#include <boost/pool/poolfwd.hpp>
#endif

#include <string>
#include <utility>
#include <map>
#include <unordered_map>
#include <vector>

namespace glucat
{
  // Forward declarations for friends

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  class framed_multi; // forward

```

```
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
class matrix_multi; // forward
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator* (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator^ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator& (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator% (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
star(const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> Scalar_T;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator/ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator| (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator» (std::istream& s, framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator« (std::ostream& os, const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&;
template< typename Scalar_T, const index_t LO, const index_t HI >
auto
operator« (std::ostream& os, const std::pair< const index_set<LO,HI>, Scalar_T >& term) -> std::ostream&;
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
template< const index_t LO, const index_t HI>
class index_set_hash
{
public:
  using index_set_t = index_set<LO, HI>;
  inline auto operator()(index_set_t val) const -> size_t { return val.hash_fn(); }
};
template< typename Scalar_T = double,  const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<> >
class framed_multi :
public clifford_algebra< Scalar_T, index_set<LO,HI>, framed_multi<Scalar_T,LO,HI,Tune_P> >,
private std::unordered_map< index_set<LO,HI>, Scalar_T, index_set_hash<LO,HI> >
{
public:
  using multivector_t = framed_multi;
  using framed_multi_t = multivector_t;
  using scalar_t = Scalar_T;
  using tune_p = Tune_P;
  using index_set_t = index_set<LO, HI>;
  using term_t = std::pair<const index_set_t, Scalar_T>;
  using vector_t = std::vector<Scalar_T>;
  using error_t = error<multivector_t>;
  using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P >;
  template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  friend class matrix_multi;
  template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
  friend class framed_multi;
private:
```

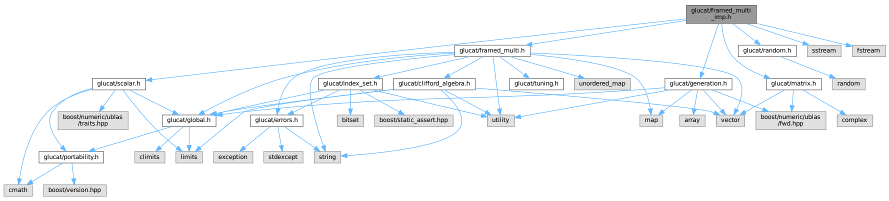
```
    class var_term; // forward
    using var_term_t = class var_term;
    using matrix_t = typename matrix_multi_t::matrix_t;
    using sorted_map_t = std::map< index_set_t, Scalar_T, std::less<const index_set_t> >;
    using map_t = std::unordered_map<index_set_t, Scalar_T, index_set_hash<LO, HI»;

    class hash_size_t
    {
    public:
      hash_size_t(size_t hash_size)
      : n(hash_size)
      { };
      auto operator()() const -> size_t
      { return n; }
    private:
      size_t n;
    };

    using framed_pair_t = std::pair<const multivector_t, const multivector_t>;
    using size_type = typename map_t::size_type;
    using iterator = typename map_t::iterator;
    using const_iterator = typename map_t::const_iterator;

  public:
    static auto classname() -> const std::string;
    ~framed_multi() override = default;
    framed_multi();

  private:
    framed_multi(const hash_size_t& hash_size);
  public:
    template< typename Other_Scalar_T >
    framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
    template< typename Other_Scalar_T >
    framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
                 const index_set_t frm, const bool prechecked = false);
    framed_multi(const framed_multi_t& val,
                 const index_set_t frm, const bool prechecked = false);
    framed_multi(const index_set_t ist, const Scalar_T& crd = Scalar_T(1));
    framed_multi(const index_set_t ist, const Scalar_T& crd,
                 const index_set_t frm, const bool prechecked = false);
    framed_multi(const Scalar_T& scr, const index_set_t frm = index_set_t());
    framed_multi(const int scr, const index_set_t frm = index_set_t());
    framed_multi(const vector_t& vec,
                 const index_set_t frm, const bool prechecked = false);
    framed_multi(const std::string& str);
    framed_multi(const std::string& str,
                 const index_set_t frm, const bool prechecked = false);
    framed_multi(const char* str)
    { *this = framed_multi(std::string(str)); };
    framed_multi(const char* str,
                 const index_set_t frm, const bool prechecked = false)
    { *this = framed_multi(std::string(str), frm, prechecked); };
    template< typename Other_Scalar_T >
    framed_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P >& val);
    template< typename Other_Scalar_T >
    auto fast_matrix_multi(const index_set_t frm) const -> const
  matrix_multi<Other_Scalar_T,LO,HI,Tune_P >;
    auto fast_framed_multi() const -> const framed_multi_t;

    _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS

    auto nbr_terms() const -> unsigned long;

    static auto random(const index_set_t frm, Scalar_T fill = Scalar_T(1)) -> const multivector_t;

    // Friend declarations

    friend auto
      operator* <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
    friend auto
      operator^ <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
    friend auto
      operator& <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
    friend auto
      operator% <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
    friend auto
      star      <>(const multivector_t& lhs, const multivector_t& rhs) -> Scalar_T;
    friend auto
      operator/ <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
    friend auto
      operator| <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;

    friend auto
      operator» <>(std::istream& s, multivector_t& val) -> std::istream&;
    friend auto
      operator« <>(std::ostream& os, const multivector_t& val) -> std::ostream&;
```

```
    friend auto
      operator« <>(std::ostream& os, const term_t& term) -> std::ostream&;

    friend auto
      exp <>(const multivector_t& val) -> const multivector_t;

    auto      operator+= (const term_t& term) -> multivector_t&;

  private:
    auto       fold(const index_set_t frm) const -> multivector_t;
    auto       unfold(const index_set_t frm) const -> multivector_t;
    auto      centre_pm4_qp4(index_t& p, index_t& q) -> multivector_t&;
    auto      centre_pp4_qm4(index_t& p, index_t& q) -> multivector_t&;
    auto      centre_qp1_pm1(index_t& p, index_t& q) -> multivector_t&;
    auto      divide(const index_set_t ist) const -> const framed_pair_t;
    auto      fast(const index_t level, const bool odd) const -> const matrix_t;

    class var_term :
    public std::pair<index_set<LO,HI>, Scalar_T>
    {
    public:
      using var_pair_t = std::pair<index_set<LO, HI>, Scalar_T>;

      static auto classname() -> const std::string
      { return "var_term"; }
      ~var_term() = default;
      var_term()
      : var_pair_t(index_set_t(), Scalar_T(1))
      { };
      var_term(const index_set_t ist, const Scalar_T& crd = Scalar_T(1))
      : var_pair_t(ist, crd)
      { };
      auto operator*= (const term_t& rhs) -> var_term_t&
      {
        this->second *= rhs.second * this->first.sign_of_mult(rhs.first);
        this->first  ^= rhs.first;
        return *this;
      }
    };
  };

  // Non-members

  template< typename Scalar_T, const index_t LO, const index_t HI >
  inline
  static
  auto
  crd_of_mult(const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
              const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> Scalar_T;

  template< typename Scalar_T, const index_t LO, const index_t HI >
  auto
  operator*
   (const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
    const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> const std::pair<const index_set<LO,HI>,
Scalar_T>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  sqrt(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  log(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
}

namespace std
{
  template < typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P >
  struct numeric_limits< glucat::framed_multi<Scalar_T,LO,HI,Tune_P> > :
  public numeric_limits<Scalar_T>
  { };
}
#endif  // _GLUCAT_FRAMED_MULTI_H
```

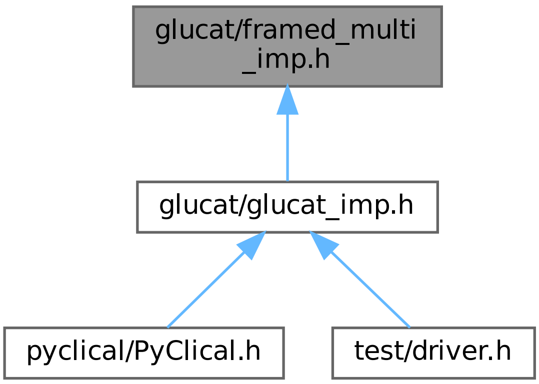
7.11 glucat/framed_multi_imp.h File Reference

```
#include "glucat/framed_multi.h"
#include "glucat/scalar.h"
#include "glucat/random.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include <sstream>
#include <fstream>
```

Include dependency graph for framed_multi_imp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::sorted_range< Map_T, Sorted_Map_T >

  Sorted range for use with output.
- class glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >

Namespaces

- namespace glucat

Macros

- #define _GLUCAT_HASH_N(x)
- #define _GLUCAT_HASH_SIZE_T(x)

Functions

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Outer product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Inner product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Left contraction.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::star (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

    Hestenes scalar product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator/ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Geometric quotient.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator| (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

    Transformation via twisted adjoint action.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator<< (std::ostream &os, const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

    Write multivector to output.
- template<typename Scalar_T , const index_t LO, const index_t HI>
auto glucat::operator<< (std::ostream &os, const std::pair< const index_set< LO, HI >, Scalar_T > &term) -> std::ostream &

    Write term to output.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator>> (std::istream &s, framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

    Read multivector from input.
- template<typename Scalar_T , const index_t LO, const index_t HI>
static auto glucat::crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T

    Coordinate of product of terms.
- template<typename Scalar_T , const index_t LO, const index_t HI>
auto glucat::operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >

Product of terms.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sqrt (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::exp (const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::log (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.

7.11.1 Macro Definition Documentation

7.11.1.1 _GLUCAT_HASH_N

```
#define _GLUCAT_HASH_N(
            x)
```

Value:
```
(x)
```

Definition at line 54 of file framed_multi_imp.h.

7.11.1.2 _GLUCAT_HASH_SIZE_T

```
#define _GLUCAT_HASH_SIZE_T(
            x)
```

Value:
```
(typename multivector_t::hash_size_t)(x)
```

Definition at line 55 of file framed_multi_imp.h.

Referenced by glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), glucat::operator%(), glucat::operator&(), glucat::operator∗(), and glucat::operator^().

7.12 framed_multi_imp.h

Go to the documentation of this file.
```
00001 #ifndef _GLUCAT_FRAMED_MULTI_IMP_H
00002 #define _GLUCAT_FRAMED_MULTI_IMP_H
00003 /***************************************************************************
00004     GluCat : Generic library of universal Clifford algebra templates
00005     framed_multi_imp.h : Implement the coordinate map representation of a
00006     Clifford algebra element
00007                              -------------------
00008     begin                : Sun 2001-12-09
00009     copyright            : (C) 2001-2021 by Paul C. Leopardi
00010  ***************************************************************************
00011 
00012     This library is free software: you can redistribute it and/or modify
00013     it under the terms of the GNU Lesser General Public License as published
00014     by the Free Software Foundation, either version 3 of the License, or
```

```
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/framed_multi.h"

#include "glucat/scalar.h"
#include "glucat/random.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"

#include <sstream>
#include <fstream>

namespace glucat
{
  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  classname() -> const std::string
  { return "framed_multi"; }

#define _GLUCAT_HASH_N(x) (x)
#define _GLUCAT_HASH_SIZE_T(x) (typename multivector_t::hash_size_t)(x)

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  framed_multi()
  : map_t(_GLUCAT_HASH_N(0))
  { }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  framed_multi(const hash_size_t& hash_size)
  : map_t(_GLUCAT_HASH_N(hash_size()))
  { }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  template< typename Other_Scalar_T >
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
  : map_t(_GLUCAT_HASH_N(val.size()))
  {
    for (auto& val_term : val)
      this->insert(term_t(val_term.first, numeric_traits<Scalar_T>::to_scalar_t(val_term.second)));
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  template< typename Other_Scalar_T >
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
               const index_set_t frm, const bool prechecked)
  : map_t(_GLUCAT_HASH_N(val.size()))
  {
    if (!prechecked && (val.frame() | frm) != frm)
      throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
    for (auto& val_term : val)
      this->insert(term_t(val_term.first, numeric_traits<Scalar_T>::to_scalar_t(val_term.second)));
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  framed_multi(const multivector_t& val,
               const index_set_t frm, const bool prechecked)
  : map_t(_GLUCAT_HASH_N(val.size()))
  {
    if (!prechecked && (val.frame() | frm) != frm)
      throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
    for (auto& val_term : val)
      this->insert(val_term);
  }
```

```

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const index_set_t ist, const Scalar_T& crd)
: map_t(_GLUCAT_HASH_N(1))
{
  if (crd != Scalar_T(0))
    this->insert(term_t(ist, crd));
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const index_set_t ist, const Scalar_T& crd,
             const index_set_t frm, const bool prechecked)
: map_t(_GLUCAT_HASH_N(1))
{
  if (!prechecked && (ist | frm) != frm)
    throw error_t("multivector_t(ist,crd,frm): cannot initialize with value outside of frame");
  if (crd != Scalar_T(0))
    this->insert(term_t(ist, crd));
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const Scalar_T& scr, const index_set_t frm)
: map_t(_GLUCAT_HASH_N(1))
{
  if (scr != Scalar_T(0))
    this->insert(term_t(index_set_t(), scr));
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const int scr, const index_set_t frm)
: map_t(_GLUCAT_HASH_N(1))
{
  if (scr != Scalar_T(0))
    this->insert(term_t(index_set_t(), Scalar_T(scr)));
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const vector_t& vec,
             const index_set_t frm, const bool prechecked)
: map_t(_GLUCAT_HASH_N(vec.size()))
{
  if (!prechecked && index_t(vec.size()) != frm.count())
    throw error_t("multivector_t(vec,frm): cannot initialize with vector not matching frame");
  auto idx = frm.min();
  const auto frm_end = frm.max()+1;
  for (auto& crd : vec)
  {
    *this += term_t(index_set_t(idx), crd);
    for (
      ++idx;
      idx != frm_end && !frm[idx];
      ++idx)
      ;
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const std::string& str)
: map_t(_GLUCAT_HASH_N(0))
{
  std::istringstream ss(str);
  ss » *this;
  if (!ss)
    throw error_t("multivector_t(str): could not parse string");
  // Peek to see if the end of the string has been reached.
  ss.peek();
  if (!ss.eof())
    throw error_t("multivector_t(str): could not parse entire string");
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const std::string& str, const index_set_t frm, const bool prechecked)
: map_t(_GLUCAT_HASH_N(0))
{
  if (prechecked)
    *this = multivector_t(str);
  else
    *this = multivector_t(multivector_t(str), frm, false);
}

```

```
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
template< typename Other_Scalar_T >
framed_multi<Scalar_T,LO,HI,Tune_P>::
framed_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
: map_t(_GLUCAT_HASH_N(1))
{
  if (val == Other_Scalar_T(0))
    return;

  const auto dim = val.m_matrix.size1();
  using traits_t = numeric_traits<Scalar_T>;
  if (dim == 1)
  {
    this->insert(term_t(index_set_t(), traits_t::to_scalar_t(val.m_matrix(0, 0))));
    return;
  }
  if (dim >= Tune_P::inv_fast_dim_threshold)
    try
    {
      *this = (val.template fast_framed_multi<Scalar_T>()).truncated();
      return;
    }
    catch (const glucat_error& e)
    { }

  const auto val_norm = traits_t::to_scalar_t(val.norm());
  if (traits_t::isNaN_or_isInf(val_norm))
  {
    *this = traits_t::NaN();
    return;
  }
  const auto frm = val.frame();
  const auto algebra_dim = set_value_t(1) « frm.count();
  auto result = multivector_t(
    _GLUCAT_HASH_SIZE_T(std::min<size_t>(algebra_dim, matrix::nnz(val.m_matrix))));
  for (auto
      stv = set_value_t(0);
      stv != algebra_dim;
      stv++)
  {
    const auto ist = index_set_t(stv, frm, true);
    const auto crd =
      traits_t::to_scalar_t(matrix::inner<Other_Scalar_T>(val.basis_element(ist), val.m_matrix));
    if (crd != Scalar_T(0))
      result.insert(term_t(ist, crd));
  }
  *this = result.truncated();
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator== (const multivector_t& rhs) const -> bool
{
  if (this->size() != rhs.size())
    return false;
  const auto rhs_end = rhs.end();
  for (auto& this_term : *this)
  {
    const const_iterator& rhs_it = rhs.find(this_term.first);
    if (rhs_it == rhs_end || rhs_it->second != this_term.second)
      return false;
  }
  return true;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator== (const Scalar_T& scr) const -> bool
{
  switch (this->size())
  {
  case 0:
    return scr == Scalar_T(0);
  case 1:
    {
      const auto& this_it = this->begin();
      return this_it->first == index_set_t() && this_it->second == scr;
    }
  default:
    return false;
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
```

```
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const Scalar_T& scr) -> multivector_t&
{
  *this += term_t(index_set_t(), scr);
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const multivector_t& rhs) -> multivector_t&
{ // simply add terms
  for (auto& rhs_term : rhs)
    *this += rhs_term;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator-= (const Scalar_T& scr) -> multivector_t&
{
  *this += term_t(index_set_t(), -scr);
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator-= (const multivector_t& rhs) -> multivector_t&
{
  for (auto& rhs_term : rhs)
    *this += term_t(rhs_term.first, -(rhs_term.second));
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator- () const -> const multivector_t
{ // multiply coordinates of all terms by -1
  auto result = *this;
  for (auto& result_term : result)
    result_term.second *= Scalar_T(-1);
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator*= (const Scalar_T& scr) -> multivector_t&
{ // multiply coordinates of all terms by scalar
  using traits_t = numeric_traits<Scalar_T>;

  if (traits_t::isNaN_or_isInf(scr))
    return *this = traits_t::NaN();
  if (scr == Scalar_T(0))
    if (this->isnan())
      *this = traits_t::NaN();
    else
      this->clear();
  else
    for (auto& this_term : *this)
      this_term.second *= scr;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator* (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  using traits_t = numeric_traits<Scalar_T>;

  if (lhs.isnan() || rhs.isnan())
    return traits_t::NaN();

  const double lhs_size = lhs.size();
  const double rhs_size = rhs.size();
  const auto our_frame = lhs.frame() | rhs.frame();
```

```
    const auto frm_count = our_frame.count();
    const auto algebra_dim = set_value_t(1) « frm_count;
    const auto direct_mult = lhs_size * rhs_size <= double(algebra_dim);
    if (direct_mult)
    { // If we have a sparse multiply, store the result directly
      auto result = multivector_t(
        _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
      for (auto& lhs_term : lhs)
        for (auto& rhs_term : rhs)
          result += lhs_term * rhs_term;
      return result;
    }
    else
    { // Past a certain threshold, the matrix algorithm is fastest
      using matrix_multi_t = typename multivector_t::matrix_multi_t;
      return matrix_multi_t(lhs, our_frame, true) *
             matrix_multi_t(rhs, our_frame, true);
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  operator*= (const multivector_t& rhs) -> multivector_t&
  { return *this = *this * rhs; }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator^ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
  { // Arvind Raja's original reference:
    // "old clical, outerproduct(p,q:pterm):pterm in file compmod.pas"

    if (lhs.empty() || rhs.empty())
      return Scalar_T(0);

    using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
    using index_set_t = typename multivector_t::index_set_t;
    using term_t = typename multivector_t::term_t;

    const auto empty_set = index_set_t();

    const double lhs_size = lhs.size();
    const double rhs_size = rhs.size();
    const auto lhs_frame = lhs.frame();
    const auto rhs_frame = rhs.frame();
    const auto our_frame = lhs_frame | rhs_frame;
    const auto algebra_dim = set_value_t(1) « our_frame.count();
    auto result = multivector_t(
      _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
    const auto lhs_end = lhs.end();
    const auto rhs_end = rhs.end();

    if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
    {
      for (auto
          result_stv = set_value_t(0);
          result_stv != algebra_dim;
          ++result_stv)
      {
        const auto result_ist = index_set_t(result_stv, our_frame, true);
        const auto lhs_result_frame = lhs_frame & result_ist;
        const auto lhs_result_dim = set_value_t(1) « lhs_result_frame.count();
        auto result_crd = Scalar_T(0);
        for (auto
            lhs_stv = set_value_t(0);
            lhs_stv != lhs_result_dim;
            ++lhs_stv)
        {
          const auto lhs_ist = index_set_t(lhs_stv, lhs_result_frame, true);
          const auto rhs_ist = result_ist ^ lhs_ist;
          if ((rhs_ist | rhs_frame) == rhs_frame)
          {
            const auto lhs_it = lhs.find(lhs_ist);
            if (lhs_it != lhs_end)
            {
              const auto rhs_it = rhs.find(rhs_ist);
              if (rhs_it != rhs_end)
                result_crd += crd_of_mult(*lhs_it, *rhs_it);
            }
          }
        }
        if (result_crd != Scalar_T(0))
          result.insert(term_t(result_ist, result_crd));
      }
      return result;
```

```
    }
    else
    {
      for (auto& lhs_term : lhs)
        for (auto& rhs_term : rhs)
          if ((lhs_term.first & rhs_term.first) == empty_set)
            result += lhs_term * rhs_term;
      return result;
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  operator^= (const multivector_t& rhs) -> multivector_t&
  { return *this = *this ^ rhs; }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator& (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
  { // Arvind Raja's original reference:
    // "old clical, innerproduct(p,q:pterm):pterm in file compmod.pas"

    if (lhs.empty() || rhs.empty())
      return Scalar_T(0);

    using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
    using index_set_t = typename multivector_t::index_set_t;
    using term_t = typename multivector_t::term_t;

    const auto lhs_end = lhs.end();
    const auto rhs_end = rhs.end();
    const double lhs_size = lhs.size();
    const double rhs_size = rhs.size();

    const auto lhs_frame = lhs.frame();
    const auto rhs_frame = rhs.frame();

    const auto our_frame = lhs_frame | rhs_frame;
    const auto algebra_dim = set_value_t(1) << our_frame.count();
    auto result = multivector_t(
      _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
    if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
    {
      for (auto
          result_stv = set_value_t(0);
          result_stv != algebra_dim;
          ++result_stv)
      {
        const auto result_ist = index_set_t(result_stv, our_frame, true);
        const auto comp_frame = our_frame & ~result_ist;
        const auto comp_dim = set_value_t(1) << comp_frame.count();
        auto result_crd = Scalar_T(0);
        for (auto
            comp_stv = set_value_t(1);
            comp_stv != comp_dim;
            ++comp_stv)
        {
          const auto comp_ist = index_set_t(comp_stv, comp_frame, true);
          const auto our_ist = result_ist ^ comp_ist;
          if ((our_ist | lhs_frame) == lhs_frame)
          {
            const auto lhs_it = lhs.find(our_ist);
            if (lhs_it != lhs_end)
            {
              const auto rhs_it = rhs.find(comp_ist);
              if (rhs_it != rhs_end)
                result_crd += crd_of_mult(*lhs_it, *rhs_it);
            }
          }
          if (result_stv != 0)
          {
            if ((our_ist | rhs_frame) == rhs_frame)
            {
              const auto rhs_it = rhs.find(our_ist);
              if (rhs_it != rhs_end)
              {
                const auto lhs_it = lhs.find(comp_ist);
                if (lhs_it != lhs_end)
                  result_crd += crd_of_mult(*lhs_it, *rhs_it);
              }
            }
          }
        }
        if (result_crd != Scalar_T(0))
```

```
        result.insert(term_t(result_ist, result_crd));
    }
  }
  else
  {
    const auto empty_set = index_set_t();
    for (auto& lhs_term : lhs)
    {
      const auto lhs_ist = lhs_term.first;
      if (lhs_ist != empty_set)
        for (auto& rhs_term : rhs)
        {
          const auto rhs_ist = rhs_term.first;
          if (rhs_ist != empty_set)
          {
            const auto our_ist = lhs_ist | rhs_ist;
            if ((lhs_ist == our_ist) || (rhs_ist == our_ist))
              result += lhs_term * rhs_term;
          }
        }
    }
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator&= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this & rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator% (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
{
  // Reference: Leo Dorst, "Honing geometric algebra for its use in the computer sciences",
  // in Geometric Computing with Clifford Algebras, ed. G. Sommer,
  // Springer 2001, Chapter 6, pp. 127-152.
  // http://staff.science.uva.nl/~leo/clifford/index.html

  if (lhs.empty() || rhs.empty())
    return Scalar_T(0);

  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  using index_set_t = typename multivector_t::index_set_t;
  using term_t = typename multivector_t::term_t;
  using map_t = typename multivector_t::map_t;

  const auto lhs_end   = lhs.end();
  const auto rhs_end   = rhs.end();
  const double lhs_size = lhs.size();
  const double rhs_size = rhs.size();
  const auto lhs_frame = lhs.frame();
  const auto rhs_frame = rhs.frame();

  const auto our_frame = lhs_frame | rhs_frame;
  const auto algebra_dim = set_value_t(1) << our_frame.count();
  auto result = multivector_t(
    _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));

  if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
  {
    for (auto
        result_stv = set_value_t(0);
        result_stv != algebra_dim;
        ++result_stv)
    {
      const auto result_ist = index_set_t(result_stv, our_frame, true);
      const auto comp_frame = lhs_frame & ~result_ist;
      const auto comp_dim = set_value_t(1) << comp_frame.count();
      auto result_crd = Scalar_T(0);
      for (auto
          comp_stv = set_value_t(0);
          comp_stv != comp_dim;
          ++comp_stv)
      {
        const auto comp_ist = index_set_t(comp_stv, comp_frame, true);
        const auto rhs_ist = result_ist ^ comp_ist;
        if ((rhs_ist | rhs_frame) == rhs_frame)
        {
          const auto rhs_it = rhs.find(rhs_ist);
          if (rhs_it != rhs_end)
          {
            const auto lhs_it = lhs.find(comp_ist);
            if (lhs_it != lhs_end)
```

```
              result_crd += crd_of_mult(*lhs_it, *rhs_it);
          }
        }
      }
      if (result_crd != Scalar_T(0))
        result.insert(term_t(result_ist, result_crd));
    }
  }
  else
  {
    for (auto& rhs_term : rhs)
    {
      const auto rhs_ist = rhs_term.first;
      for (auto& lhs_term : lhs)
      {
        const index_set_t lhs_ist = lhs_term.first;
        if ((lhs_ist | rhs_ist) == rhs_ist)
          result += lhs_term * rhs_term;
      }
    }
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator%= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this % rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
star(const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T
{
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;

  auto result = Scalar_T(0);
  const auto small_star_large = lhs.size() < rhs.size();
  const auto* smallp =
    small_star_large
    ? &lhs
    : &rhs;
  const auto* largep =
    small_star_large
    ? &rhs
    : &lhs;

  for (auto& small_term : *smallp)
  {
    const auto small_ist = small_term.first;
    const auto large_crd = (*largep)[small_ist];
    if (large_crd != Scalar_T(0))
      result += small_ist.sign_of_square() * small_term.second * large_crd;
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator/= (const Scalar_T& scr) -> multivector_t&
{ // Divide coordinates of all terms by scr
  using traits_t = numeric_traits<Scalar_T>;

  if (traits_t::isNaN(scr))
    return *this = traits_t::NaN();
  if (traits_t::isInf(scr))
    if (this->isnan())
      *this = traits_t::NaN();
    else
      this->clear();
  else
    for (auto& this_term : *this)
      this_term.second /= scr;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator/ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  using traits_t = numeric_traits<Scalar_T>;
```

```
  using index_set_t = typename multivector_t::index_set_t;
  using matrix_multi_t = typename multivector_t::matrix_multi_t;

  if (rhs == Scalar_T(0))
    return traits_t::NaN();

  const auto our_frame = lhs.frame() | rhs.frame();
  return matrix_multi_t(lhs, our_frame, true) / matrix_multi_t(rhs, our_frame, true);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator/= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this / rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator| (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
   framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  using matrix_multi_t = typename multivector_t::matrix_multi_t;

  return matrix_multi_t(rhs) * matrix_multi_t(lhs) / matrix_multi_t(rhs.involute());
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator|= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this | rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
inv() const -> const multivector_t
{
  auto result = matrix_multi_t(Scalar_T(1), this->frame());
  return result /= matrix_multi_t(*this);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
pow(int m) const -> const multivector_t
{ return glucat::pow(*this, m); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
outer_pow(int m) const -> const multivector_t
{
  if (m < 0)
    throw error_t("outer_pow(int): negative exponent");
  auto result = multivector_t(Scalar_T(1));
  auto a = *this;
  for (;
      m != 0;
      m »= 1, a = a ^ a)
    if (m & 1)
      result ^= a;
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
frame() const -> const index_set_t
{
  auto result = index_set_t();
  for (auto& this_term : *this)
    result |= this_term.first;
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
grade() const -> index_t
{
```

```
    auto result = index_t(0);
    for (auto& this_term : *this)
      result = std::max(result, this_term.first.count());
    return result;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  operator[] (const index_set_t ist) const -> Scalar_T
  {
    const auto& this_it = this->find(ist);
    if (this_it == this->end())
      return Scalar_T(0);
    else
      return this_it->second;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  operator() (index_t grade) const -> const multivector_t
  {
    if ((grade < 0) || (grade > HI-LO))
      return Scalar_T(0);
    else
    {
      auto result = multivector_t();
      for (auto& this_term : *this)
        if (this_term.first.count() == grade)
          result += this_term;
      return result;
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  scalar() const -> Scalar_T
  { return (*this)[index_set_t()]; }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  pure() const -> const multivector_t
  { return *this - this->scalar(); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  even() const -> const multivector_t
  { // even part of x, sum of the pure(count) with even count
    auto result = multivector_t();
    for (auto& this_term : *this)
      if ((this_term.first.count() % 2) == 0)
        result.insert(this_term);
    return result;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  odd() const -> const multivector_t
  { // even part of x, sum of the pure(count) with even count
    auto result = multivector_t();
    for (auto& this_term : *this)
      if ((this_term.first.count() % 2) == 1)
        result.insert(this_term);
    return result;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  vector_part() const -> const vector_t
  { return this->vector_part(this->frame(), true); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
  {
    if (!prechecked && (this->frame() | frm) != frm)
      throw error_t("vector_part(frm): value is outside of requested frame");
```

```
  auto result = vector_t();
  result.reserve(frm.count());
  const auto frm_end = frm.max()+1;
  for (auto
      idx  = frm.min();
      idx != frm_end;
      ++idx)
    // Frame may contain indices which do not correspond to a grade 1 term but
    // frame cannot omit any index corresponding to a grade 1 term
    if (frm[idx])
      result.push_back((*this)[index_set_t(idx)]);
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
involute() const -> const multivector_t
{
  auto result = *this;
  for (auto& result_term : result)
  { // for a k-vector u, involute(u) == (-1)^k * u
    if ((result_term.first.count() % 2) == 1)
      result_term.second *= Scalar_T(-1);
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
reverse() const -> const multivector_t
{
  auto result = *this;
  for (auto& result_term : result)
    // For a k-vector u, reverse(u) = { -u, k == 2,3 (mod 4)
    //                                {  u, k == 0,1 (mod 4)
    switch (result_term.first.count() % 4)
    {
    case 2:
    case 3:
      result_term.second *= Scalar_T(-1);
      break;
    default:
      break;
    }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
conj() const -> const multivector_t
{
  auto result = *this;
  for (auto& result_term : result)
    // For a k-vector u, conj(u) = { -u, k == 1,2 (mod 4)
    //                             {  u, k == 0,3 (mod 4)
    switch (result_term.first.count() % 4)
    {
    case 1:
    case 2:
      result_term.second *= Scalar_T(-1);
      break;
    default:
      break;
    }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
quad() const -> Scalar_T
{
  // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
  // ref: old clical: quadfunction(p:pter):pterm in file compmod.pas
  auto result = Scalar_T(0);
  for (auto& this_term : *this)
  {
    const auto sign =
      (this_term.first.count_neg() % 2)
      ? -Scalar_T(1)
      :  Scalar_T(1);
    result += sign * (this_term.second) * (this_term.second);
  }
  return result;
```

```
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
norm() const -> Scalar_T
{
  using traits_t = numeric_traits<Scalar_T>;

  auto result = Scalar_T(0);
  for (auto& this_term : *this)
  {
    const auto abs_crd = traits_t::abs(this_term.second);
    result +=  abs_crd * abs_crd;
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
max_abs() const -> Scalar_T
{
  using traits_t = numeric_traits<Scalar_T>;

  auto result = Scalar_T(0);
  for (auto& this_term : *this)
  {
    const auto abs_crd = traits_t::abs(this_term.second);
    if (abs_crd > result)
      result = abs_crd;
  }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
random(const index_set_t frm, Scalar_T fill) -> const multivector_t
{
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  using index_set_t = typename multivector_t::index_set_t;
  using term_t = typename multivector_t::term_t;

  using random_generator_t = random_generator<Scalar_T>;
  auto& generator = random_generator_t::generator();

  fill =
    (fill < Scalar_T(0))
    ? Scalar_T(0)
    : (fill > Scalar_T(1))
      ? Scalar_T(1)
      : fill;
  const auto algebra_dim = set_value_t(1) « frm.count();
  using traits_t = numeric_traits<Scalar_T>;
  const auto mean_abs = traits_t::sqrt(Scalar_T(double(algebra_dim)));
  auto result = multivector_t();
  for (auto
      stv = set_value_t(0);
      stv != algebra_dim;
      ++stv)
    if (generator.uniform() < fill)
    {
      const auto& result_crd = generator.normal() / mean_abs;
      result.insert(term_t(index_set_t(stv, frm, true), result_crd));
    }
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
void
framed_multi<Scalar_T,LO,HI,Tune_P>::
write(const std::string& msg) const
{ std::cout « msg « std::endl « "  " « (*this) « std::endl; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
void
framed_multi<Scalar_T,LO,HI,Tune_P>::
write(std::ofstream& ofile, const std::string& msg) const
{
  if (!ofile)
    throw error_t("write(ofile,msg): cannot write to output file");
  ofile « msg « std::endl « "  " « (*this) « std::endl;
}
```

```
template< typename Map_T,typename Sorted_Map_T >
class sorted_range
{
public:
  using map_t = Map_T;
  using sorted_map_t = Sorted_Map_T;
  using sorted_iterator = typename Sorted_Map_T::const_iterator;

  sorted_range (Sorted_Map_T &sorted_val, const Map_T& val)
  {
    for (auto& val_term : val)
      sorted_val.insert(val_term);
    sorted_begin = sorted_val.begin();
    sorted_end   = sorted_val.end();
  }
  sorted_iterator sorted_begin;
  sorted_iterator sorted_end;
};

template< typename Sorted_Map_T >
class sorted_range< Sorted_Map_T, Sorted_Map_T >
{
public:
  using map_t = Sorted_Map_T;
  using sorted_map_t = Sorted_Map_T;
  using sorted_iterator = typename Sorted_Map_T::const_iterator;

  sorted_range (Sorted_Map_T &sorted_val, const Sorted_Map_T& val)
  : sorted_begin( val.begin() ),
    sorted_end( val.end() )
  { }
  sorted_iterator sorted_begin;
  sorted_iterator sorted_end;
};

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator« (std::ostream& os, const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
{
  using limits_t = std::numeric_limits<Scalar_T>;
  if (val.empty())
    os « 0;
  else if (val.isnan())
    os « limits_t::quiet_NaN();
  else if (val.isinf())
  {
    const Scalar_T& inf = limits_t::infinity();
    os « (scalar(val) < 0.0 ? -inf : inf);
  }
  else
  {
    using traits_t = numeric_traits<Scalar_T>;
    using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
    Scalar_T truncation;
    switch (os.flags() & std::ios::floatfield)
    {
      case std::ios_base::scientific:
        truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()) + 1);
        break;
      case std::ios_base::fixed:
        truncation = Scalar_T(1) / (traits_t::pow(Scalar_T(10), int(os.precision())) *
val.max_abs());
        break;
      case std::ios_base::fixed | std::ios_base::scientific:
        truncation = multivector_t::default_truncation;
        break;
      default:
        truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()));
        break;
    }
    auto truncated_val = val.truncated(truncation);
    if (truncated_val.empty())
      os « 0;
    else
    {
      using map_t = typename multivector_t::map_t;
      using sorted_map_t = typename multivector_t::sorted_map_t;
      using sorted_iterator = typename sorted_map_t::const_iterator;
      auto sorted_val = sorted_map_t();
      const auto sorted_val_range = sorted_range< map_t, sorted_map_t >(sorted_val, truncated_val);
      auto sorted_it = sorted_val_range.sorted_begin;
      os « *sorted_it;
      for (++sorted_it;
          sorted_it != sorted_val_range.sorted_end;
          ++sorted_it)
      {
        const Scalar_T& scr = sorted_it->second;
```

```
        if (scr >= 0.0)
          os « '+';
        os « *sorted_it;
      }
    }
  }
  return os;
}

template< typename Scalar_T, const index_t LO, const index_t HI >
auto
operator« (std::ostream& os, const std::pair< const index_set<LO,HI>, Scalar_T >& term) ->
std::ostream&
{
  const auto second_as_double = numeric_traits<Scalar_T>::to_double(term.second);
  const auto use_double =
    (os.precision() <= std::numeric_limits<double>::digits10) ||
    (term.second == Scalar_T(second_as_double));
  if (term.first.count() == 0)
    if (use_double)
      os « second_as_double;
    else
      os « term.second;
  else if (term.second == Scalar_T(-1))
  {
    os « '-';
    os « term.first;
  }
  else if (term.second != Scalar_T(1))
  {
    if (use_double)
    {
      auto tol = std::pow(10.0,-os.precision());
      if ( std::fabs(second_as_double + 1.0) < tol )
        os « '-';
      else if ( std::fabs(second_as_double - 1.0) >= tol )
        os « second_as_double;
    }
    else
      os « term.second;
    os « term.first;
  }
  else
    os « term.first;
  return os;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator» (std::istream& s, framed_multi<Scalar_T,LO,HI,Tune_P> & val) -> std::istream&
{ // Input looks like 1.0-2.0{1,2}+3.2{3,4}.
  using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  // Parsing variables.
  auto local_val = multivector_t();
  auto c = 0;
  // Parsing control variables.
  auto negative = false;
  auto expect_term = true;
  // The multivector may begin with '+' or '-'. Check for this.
  c = s.peek();
  if (s.good() && (c == int('+') || c == int('-')))
  { // A '-' here negates the following term.
    negative = (c == int('-'));
    // Consume the '+' or '-'.
    s.get();
  }
  while (s.good())
  { // Parse a term.
    // A term consists of an optional scalar, followed by an optional index set.
    // At least one of the two must be present.
    // Default coordinate is Scalar_T(1).
    auto coordinate = Scalar_T(1);
    // Default index set is empty.
    auto ist = index_set<LO,HI>();
    // First, check for an opening brace.
    c = s.peek();
    if (s.good())
    { // If the character is not an opening brace,
      // a coordinate value is expected here.
      if (c != int('{'))
      { // Try to read a coordinate value.
        double coordinate_as_double;
        s » coordinate_as_double;
        // Reading the coordinate may have resulted in an end of file condition.
        // This is not a failure.
        if (s)
          coordinate = Scalar_T(coordinate_as_double);
```

```
      }
    }
    else
    { // End of file here ends parsing while a term may still be expected.
      break;
    }
    // Coordinate is now Scalar_T(1) or a Scalar_T value.
    // Parse an optional index set.
    if (s.good())
    {
      c = s.peek();
      if (s.good() && c == int('{'))
      { // Try to read index set.
        s » ist;
      }
    }
    // Reading the term may have resulted in an end of file condition.
    // This is not a failure.
    if (s)
    {
      // Immediately after parsing a term, another term is not expected.
      expect_term = false;
      if (coordinate != Scalar_T(0))
      {
        // Add the term to the local multivector.
        coordinate =
          negative
          ? -coordinate
          :  coordinate;
        using term_t = typename multivector_t::term_t;
        local_val += term_t(ist, coordinate);
      }
    }
    // Check if anything follows the current term.
    if (s.good())
    {
      c = s.peek();
      if (s.good())
      { // Only '+' and '-' are valid here.
        if (c == int('+') || c == int('-'))
        { // A '-' here negates the following term.
          negative = (c == int('-'));
          // Consume the '+' or '-'.
          s.get();
          // Immediately after '+' or '-',
          // expect another term.
          expect_term = true;
        }
        else
        { // Any other character here is a not failure,
          // but still ends the parsing of the multivector.
          break;
        }
      }
    }
  }
  // If a term is still expected, this is a failure.
  if (expect_term)
    s.clear(std::istream::failbit);
  // End of file is not a failure.
  if (s)
  { // The multivector has been successfully parsed.
    val = local_val;
  }
  return s;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
nbr_terms () const -> unsigned long
{ return this->size(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const term_t& term) -> multivector_t&
{ // Do not insert terms with 0 coordinate
  if (term.second != Scalar_T(0))
  {
    const auto& this_it = this->find(term.first);
    if (this_it == this->end())
      this->insert(term);
    else if (this_it->second + term.second == Scalar_T(0))
      // Erase term if resulting coordinate is 0
      this->erase(this_it);
```

```
      else
        this_it->second += term.second;
    }
    return *this;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  isinf() const -> bool
  {
    using traits_t = numeric_traits<Scalar_T>;

    if (std::numeric_limits<Scalar_T>::has_infinity)
      for (auto& this_term : *this)
        if (traits_t::isInf(this_term.second))
          return true;
    return false;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  isnan() const -> bool
  {
    using traits_t = numeric_traits<Scalar_T>;

    if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
      for (auto& this_term : *this)
        if (traits_t::isNaN(this_term.second))
          return true;
    return false;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  truncated(const Scalar_T& limit) const -> const multivector_t
  {
    using traits_t = numeric_traits<Scalar_T>;

    if (this->isnan() || this->isinf())
      return *this;
    const auto truncation = traits_t::abs(limit);
    const auto top = max_abs();
    auto result = multivector_t();
    if (top != Scalar_T(0))
      for (auto& this_term : *this)
        if (traits_t::abs(this_term.second) > top * truncation)
          result.insert(this_term);
    return result;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  fold(const index_set_t frm) const -> multivector_t
  {
    if (frm.is_contiguous())
      return *this;
    else
    {
      auto result = multivector_t();
      for (auto& this_term : *this)
        result.insert(term_t(this_term.first.fold(frm), this_term.second));
      return result;
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  unfold(const index_set_t frm) const -> multivector_t
  {
    if (frm.is_contiguous())
      return *this;
    else
    {
      auto result = multivector_t();
      for (auto& this_term : *this)
        result.insert(term_t(this_term.first.unfold(frm), this_term.second));
      return result;
    }
  }

  // Reference: [L] 16.4 Periodicity of 8, p216
  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
```

```
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
centre_pm4_qp4(index_t& p, index_t& q) -> multivector_t&
{
  // We add 4 to q by subtracting 4 from p
  if (q+4 > -LO)
    throw error_t("centre_pm4_qp4(p,q): LO is too high to represent this value");
  if (this->frame().max() > p-4)
  {
    using index_pair_t = typename index_set_t::index_pair_t;
    const auto pm3210 = index_set_t(index_pair_t(p-3,p), true);
    const auto qm4321 = index_set_t(index_pair_t(-q-4,-q-1), true);
    const auto& tqm4321 = term_t(qm4321, Scalar_T(1));
    auto result = multivector_t();
    for (auto& this_term : *this)
    {
      const auto ist = this_term.first;
      if (ist.max() > p-4)
      {
        auto var_term = var_term_t();
        for (auto
            n = index_t(0);
            n != index_t(4);
            ++n)
          if (ist[n+p-3])
            var_term *= term_t(index_set_t(n-q-4), Scalar_T(1)) * tqm4321;
        // Mask out {p-3}..{p}
        result.insert(term_t(ist & ~pm3210, this_term.second) *
                      term_t(var_term.first, var_term.second));
      }
      else
        result.insert(this_term);
    }
    *this = result;
  }
  p -=4; q += 4;
  return *this;
}

// Reference: [L] 16.4 Periodicity of 8, p216
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
centre_pp4_qm4(index_t& p, index_t& q) -> multivector_t&
{
  // We add 4 to p by subtracting 4 from q
  if (p+4 > HI)
    throw error_t("centre_pp4_qm4(p,q): HI is too low to represent this value");
  if (this->frame().min() < -q+4)
  {
    using index_pair_t = typename index_set_t::index_pair_t;
    const auto qp0123 = index_set_t(index_pair_t(-q,-q+3), true);
    const auto pp1234 = index_set_t(index_pair_t(p+1,p+4), true);
    const auto& tpp1234 = term_t(pp1234, Scalar_T(1));
    auto result = multivector_t();
    for (auto& this_term : *this)
    {
      index_set_t ist = this_term.first;
      if (ist.min() < -q+4)
      {
        auto var_term = var_term_t();
        for (auto
            n = index_t(0);
            n != index_t(4);
            ++n)
          if (ist[n-q])
            var_term *= term_t(index_set_t(n+p+1), Scalar_T(1)) * tpp1234;
        // Mask out {-q}..{-q+3}
        result.insert(term_t(var_term.first, var_term.second) *
                      term_t(ist & ~qp0123, this_term.second));
      }
      else
        result.insert(this_term);
    }
    *this = result;
  }
  p +=4; q -= 4;
  return *this;
}

// Reference: [P] Proposition 15.20, p 131
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
centre_qp1_pm1(index_t& p, index_t& q) -> multivector_t&
{
  if (q+1 > HI)
```

```
    throw error_t("centre_qp1_pm1(p,q): HI is too low to represent this value");
  if (p-1 > -LO)
    throw error_t("centre_qp1_pm1(p,q): LO is too high to represent this value");
  const auto qp1 = index_set_t(q+1);
  const auto& tqp1 = term_t(qp1, Scalar_T(1));
  auto result = multivector_t();
  for (auto& this_term : *this)
  {
    const auto ist = this_term.first;
    auto var_term = var_term_t(index_set_t(), this_term.second);
    for (auto
        n = -q;
        n != p;
        ++n)
      if (n != 0 && ist[n])
        var_term *= term_t(index_set_t(-n) | qp1, Scalar_T(1));
    if (p != 0 && ist[p])
      var_term *= tqp1;
    result.insert(term_t(var_term.first, var_term.second));
  }
  index_t orig_p = p;
  p = q+1;
  q = orig_p-1;
  return *this = result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
divide(const index_set_t ist) const -> const framed_pair_t
{
  auto quo = multivector_t();
  auto rem = multivector_t();
  for (auto& this_term : *this)
    if ((this_term.first | ist) == this_term.first)
      quo.insert(term_t(this_term.first ^ ist, this_term.second));
    else
      rem.insert(this_term);
  return framed_pair_t(quo, rem);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
framed_multi<Scalar_T,LO,HI,Tune_P>::
fast(const index_t level, const bool odd) const -> const matrix_t
{
  // Assume val is already folded and centred
  if (this->empty())
  {
    using matrix_index_t = typename matrix_multi_t::matrix_index_t;
    const auto dim = matrix_index_t(1) << level;
    auto result =matrix_t(dim, dim);
    result.clear();
    return result;
  }
  if (level == 0)
    return matrix::unit<matrix_t>(1) * this->scalar();

  using basis_matrix_t = typename matrix_multi_t::basis_matrix_t;
  using basis_scalar_t = typename basis_matrix_t::value_type;

  const auto&  I = matrix::unit<basis_matrix_t>(2);
  auto J = basis_matrix_t(2,2,2);
  J.clear();
  J(0,1)  = basis_scalar_t(-1);
  J(1,0)  = basis_scalar_t( 1);
  auto K = J;
  K(0,1)  = basis_scalar_t( 1);
  auto JK = I;
  JK(0,0) = basis_scalar_t(-1);

  const auto ist_mn = index_set_t(-level);
  const auto ist_pn = index_set_t(level);
  if (level == 1)
  {
    if (odd)
      return matrix_t(J) * (*this)[ist_mn] + matrix_t(K)  * (*this)[ist_pn];
    else
      return matrix_t(I) * this->scalar()  + matrix_t(JK) * (*this)[ist_mn ^ ist_pn];
  }
  else
  {
    const auto& pair_mn = this->divide(ist_mn);
    const auto& quo_mn = pair_mn.first;
    const auto& rem_mn = pair_mn.second;
    const auto& pair_quo_mnpn = quo_mn.divide(ist_pn);
    const auto& val_mnpn = pair_quo_mnpn.first;
```

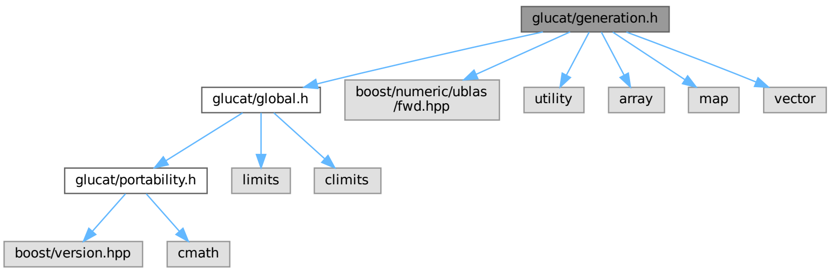
```
      const auto& val_mn    = pair_quo_mnpn.second;
      const auto& pair_rem_mnpn = rem_mn.divide(ist_pn);
      const auto& val_pn    = pair_rem_mnpn.first;
      const auto& val_1     = pair_rem_mnpn.second;
      using matrix::kron;
      if (odd)
        return - kron(JK, val_1.fast    (level-1, 1))
               + kron(I,  val_mnpn.fast(level-1, 1))
               + kron(J,  val_mn.fast  (level-1, 0))
               + kron(K,  val_pn.fast  (level-1, 0));
      else
        return   kron(I,  val_1.fast    (level-1, 0))
               + kron(JK, val_mnpn.fast(level-1, 0))
               + kron(K,  val_mn.fast  (level-1, 1))
               - kron(J,  val_pn.fast  (level-1, 1));
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  template< typename Other_Scalar_T >
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  fast_matrix_multi(const index_set_t frm) const -> const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>
  {
    // Fold val
    auto val = this->fold(frm);
    auto p = frm.count_pos();
    auto q = frm.count_neg();
    const auto bott_offset = gen::offset_to_super[pos_mod(p - q, 8)];
    p += std::max(bott_offset,index_t(0));
    q -= std::min(bott_offset,index_t(0));
    if (p > HI)
      throw error_t("fast_matrix_multi(frm): HI is too low to represent this value");
    if (q > -LO)
      throw error_t("fast_matrix_multi(frm): LO is too high to represent this value");
    // Centre val
    while (p - q > 4)
      val.centre_pm4_qp4(p, q);
    while (p - q < -3)
      val.centre_pp4_qm4(p, q);
    if (p - q > 1)
      val.centre_qp1_pm1(p, q);
    const index_t level = (p + q)/2;

    // Do the fast transform
    const auto& ev_val = val.even();
    const auto& od_val = val.odd();
    return matrix_multi<Other_Scalar_T,LO,HI,Tune_P>(ev_val.fast(level, 0) + od_val.fast(level, 1),
  frm);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  framed_multi<Scalar_T,LO,HI,Tune_P>::
  fast_framed_multi() const -> const multivector_t
  { return *this; }

  template< typename Scalar_T, const index_t LO, const index_t HI >
  inline
  static
  auto
  crd_of_mult(const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
              const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> Scalar_T
  { return lhs.first.sign_of_mult(rhs.first) * lhs.second * rhs.second; }

  template< typename Scalar_T, const index_t LO, const index_t HI >
  inline
  auto
  operator* (const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
             const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> const std::pair<const
  index_set<LO,HI>, Scalar_T>
  {
    using term_t = std::pair<const index_set<LO,HI>, Scalar_T>;
    return term_t(lhs.first ^ rhs.first, crd_of_mult(lhs, rhs));
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  sqrt(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
  bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    check_complex(val, i, prechecked);
```

```

    const auto realval = val.scalar();
    if (val == realval)
    {
      if (realval < Scalar_T(0))
        return i * traits_t::sqrt(-realval);
      else
        return traits_t::sqrt(realval);
    }
    using matrix_multi_t = typename framed_multi<Scalar_T,LO,HI,Tune_P>::matrix_multi_t;
    return sqrt(matrix_multi_t(val), matrix_multi_t(i), prechecked);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto s = scalar(val);
    if (val == s)
      return traits_t::exp(s);

    const double size = val.size();
    const auto frm_count = val.frame().count();
    const auto algebra_dim = set_value_t(1) « frm_count;

    if( (size * size <= double(algebra_dim)) || (frm_count < Tune_P::mult_matrix_threshold))
    {
      switch (Tune_P::function_precision)
      {
      case precision_demoted:
        {
          using demoted_scalar_t = typename traits_t::demoted::type;
          using demoted_multivector_t = framed_multi<demoted_scalar_t,LO,HI,Tune_P>;

          const auto& demoted_val = demoted_multivector_t(val);
          return clifford_exp(demoted_val);
        }
        break;
      case precision_promoted:
        {
          using promoted_scalar_t = typename traits_t::promoted::type;
          using promoted_multivector_t = framed_multi<promoted_scalar_t,LO,HI,Tune_P>;

          const auto& promoted_val = promoted_multivector_t(val);
          return clifford_exp(promoted_val);
        }
        break;
      default:
        return clifford_exp(val);
      }
    }
    else
    {
      using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
      return exp(matrix_multi_t(val));
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  log(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val == Scalar_T(0) || val.isnan())
      return traits_t::NaN();

    check_complex(val, i, prechecked);

    const auto realval = val.scalar();
    if (val == realval)
    {
      if (realval < Scalar_T(0))
        return i * traits_t::pi() + traits_t::log(-realval);
      else
        return traits_t::log(realval);
    }
    using matrix_multi_t = typename framed_multi<Scalar_T,LO,HI,Tune_P>::matrix_multi_t;
    return log(matrix_multi_t(val), matrix_multi_t(i), prechecked);
  }
}
#endif  // _GLUCAT_FRAMED_MULTI_IMP_H
```

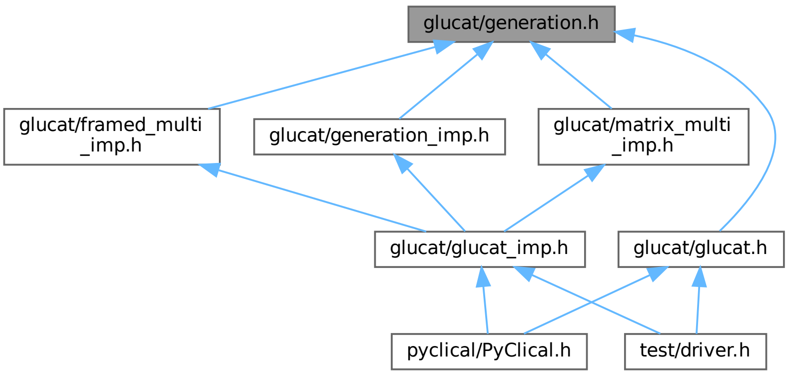
7.13 glucat/generation.h File Reference

```
#include "glucat/global.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <utility>
#include <array>
#include <map>
#include <vector>
```

Include dependency graph for generation.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::gen::generator_table< Matrix_T >

    Table of generators for specific signatures.

Namespaces

- namespace glucat
- namespace glucat::gen

Typedefs

- using glucat::gen::signature_t = std::pair< index_t, index_t >

  A signature is a pair of indices, p, q, with p == frame.max(), q == -frame.min()

Variables

- static const std::array< index_t, 8 > glucat::gen::offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1}

  Offsets between the current signature and that of the real superalgebra.

7.14 generation.h

Go to the documentation of this file.

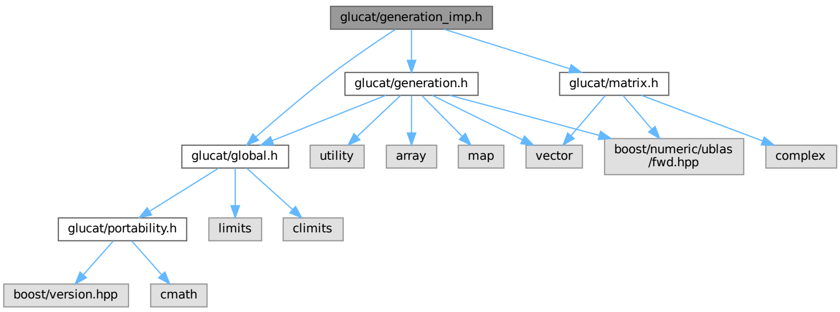
```
#ifndef _GLUCAT_GENERATION_H
#define _GLUCAT_GENERATION_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    generation.h : Declare functions for generation of the matrix representation
                             -------------------
    begin                : Wed Jan 23 2002
    copyright            : (C) 2002-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"

#include <boost/numeric/ublas/fwd.hpp>

#include <utility>
#include <array>
#include <map>
#include <vector>

namespace glucat { namespace gen
{
  namespace ublas = boost::numeric::ublas;

  using signature_t = std::pair<index_t, index_t>;

  template< class Matrix_T >
  class generator_table :
  private std::map< signature_t, std::vector<Matrix_T> >
  {
  public:
    auto operator() (const index_t p, const index_t q) -> const Matrix_T*;
    static auto generator() -> generator_table<Matrix_T>&;
  private:
    auto gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&;
    void gen_from_pm1_qm1(const std::vector<Matrix_T>& old, const signature_t sig);
    void gen_from_pm4_qp4(const std::vector<Matrix_T>& old, const signature_t sig);
    void gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig);
    void gen_from_qp1_pm1(const std::vector<Matrix_T>& old, const signature_t sig);

```

```
    friend class friend_for_private_destructor;
    // Enforce singleton
    // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
    generator_table() = default;
    ~generator_table() = default;
  public:
    generator_table(const generator_table&) = delete;
    auto operator= (const generator_table&) -> generator_table& = delete;
  };

  static const std::array<index_t, 8> offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1};

} }
#endif  // _GLUCAT_GENERATION_H
```

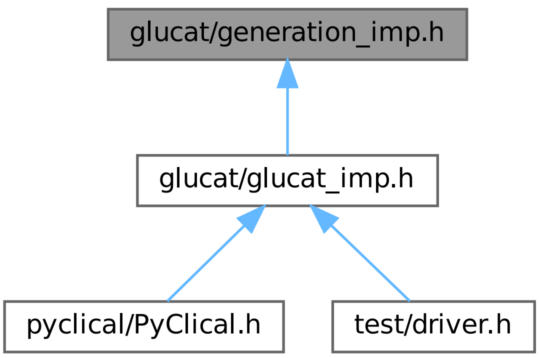
7.15 glucat/generation_imp.h File Reference

```
#include "glucat/global.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
```

Include dependency graph for generation_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat
- namespace glucat::gen

7.16 generation_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_GENERATION_IMP_H
#define _GLUCAT_GENERATION_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    generation_imp.h : Implement functions for generation of the matrix representation
                             -------------------
    begin                : Wed Jan 23 2002
    copyright            : (C) 2002-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"

namespace glucat { namespace gen
{
  // References for algorithms:
  // [M]: Scott Meyers, "Effective C++" Second Edition, Addison-Wesley, 1998.
  // [P]: Ian R. Porteous, "Clifford algebras and the classical groups", Cambridge UP, 1995.
  // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.

  // Reference: [M] Item 47
  template< class Matrix_T >
  auto
  generator_table<Matrix_T>::
  generator() -> generator_table<Matrix_T>&
  { static generator_table<Matrix_T> g; return g;}

  // Reference: [P] Table 15.27, p 133
  template< class Matrix_T >
  inline
  auto
  generator_table<Matrix_T>::
  operator() (const index_t p, const index_t q) -> const Matrix_T*
  {
    const auto bott = pos_mod(p-q, 8);
    switch(bott)
    {
    case 0:
    case 2:
      // Construct generators
      return &(gen_vector(p, q)[q]);
    default:
      // Select generators from the vector for a larger frame
      const auto super_p = p + std::max(offset_to_super[bott],index_t(0));
      const auto super_q = q - std::min(offset_to_super[bott],index_t(0));
      return &(gen_vector(super_p, super_q)[super_q]);
    }
  }
```

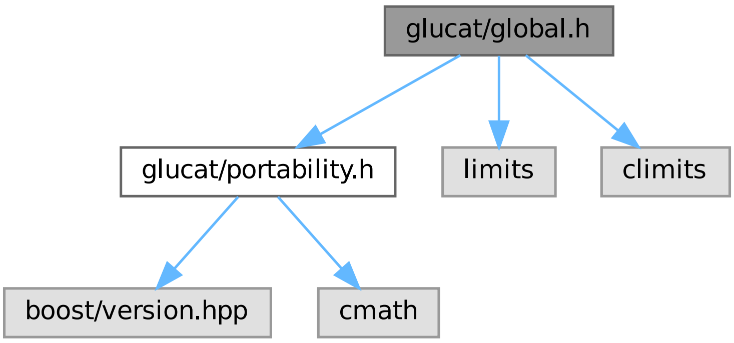
```
template< class Matrix_T >
auto
generator_table<Matrix_T>::
gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&
{
  using result_t = std::vector<Matrix_T>;
  const auto card = p + q;
  const auto bias = p - q;
  const auto bott = pos_mod(bias, 8);
  const auto sig = signature_t(p, q);
  if (this->find(sig) == this->end())
    switch(bott)
    {
    case 0:
      if (bias < 0)
        // Construct generators for p,q given generators for p+4,q-4
        gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
      else if (bias > 0)
        // Construct generators for p,q given generators for p-4,q+4
        gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
      else if (card == 0)
      { // Base case. Save a generator vector containing one matrix, size 1.
        auto result = result_t(1, matrix::unit<Matrix_T>(1));
        this->insert(make_pair(sig, result));
      }
      else
        // Construct generators for p,q given generators for p-1,q-1
        gen_from_pm1_qm1(gen_vector(p-1, q-1), sig);
      break;
    case 2:
      if (bias < 2)
        // Construct generators for p,q given generators for p+4,q-4
        gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
      else if (bias > 2)
        // Construct generators for p,q given generators for p-4,q+4
        gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
      else
        // Construct generators for p,q given generators for q+1,p-1
        gen_from_qp1_pm1(gen_vector(q+1, p-1), sig);
      break;
    default:
      break;
    }
  return (*this)[sig];
}

// Reference: [P] Proposition 15.17, p 131
template< class Matrix_T >
void
generator_table<Matrix_T>::
gen_from_pm1_qm1(const std::vector<Matrix_T>& old, const signature_t sig)
{
  const auto new_size = old.size() + 2;
  using size_t = decltype(new_size);
  using result_t = std::vector<Matrix_T>;
  auto result = result_t(new_size);

  const auto old_dim = old[0].size1();
  const auto& eye = matrix::unit<Matrix_T>(old_dim);

  auto neg = Matrix_T(2,2,2);
  neg(0,1) =    -1;
  neg(1,0) = 1;

  auto pos = neg;
  pos(0,1) =     1;

  auto dup = Matrix_T(2,2,2);
  dup(0,0) = 1;
  dup(1,1) =    -1;

  result[0] = matrix::mono_kron(neg, eye);
  for (auto
      k = size_t(1);
      k != new_size-1;
      ++k)
    result[k] = matrix::mono_kron(dup, old[k-1]);
  result[new_size-1] = matrix::mono_kron(pos, eye);

  // Save the resulting generator array.
  this->insert(make_pair(sig, result));
}

// Reference: [L] 16.4 Periodicity of 8, p216
template< class Matrix_T >
void
```

```
  generator_table<Matrix_T>::
  gen_from_pm4_qp4(const std::vector<Matrix_T>& old, const signature_t sig)
  {
    const auto old_size = old.size();
    using size_t = decltype(old_size);
    using result_t = std::vector<Matrix_T>;
    auto result = result_t(old_size);

    auto h = old[0];
    for (auto
        k = size_t(1);
        k != size_t(4);
        ++k)
      h = matrix::mono_prod(old[k], h);

    for (auto
        k = size_t(0);
        k != old_size-4;
        ++k)
      result[k] = old[k+4];
    for (auto
        k = old_size-4;
        k != old_size;
        ++k)
      result[k] = matrix::mono_prod(old[k+4-old_size], h);
    // Save the resulting generator array.
    this->insert(make_pair(sig, result));
  }

  // Reference: [L] 16.4 Periodicity of 8, p216
  template< class Matrix_T >
  void
  generator_table<Matrix_T>::
  gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig)
  {
    const auto old_size = old.size();
    using size_t = decltype(old_size);
    using result_t = std::vector<Matrix_T>;
    auto result = result_t(old_size);

    auto h = old[old_size-1];
    for (auto
        k = size_t(1);
        k != size_t(4);
        ++k)
      h = matrix::mono_prod(old[old_size-1-k], h);

    for (auto
        k = size_t(0);
        k != size_t(4);
        ++k)
      result[k] = matrix::mono_prod(old[k+old_size-4], h);
    for (auto
        k = size_t(4);
        k != old_size;
        ++k)
      result[k] = old[k-4];
    // Save the resulting generator array.
    this->insert(make_pair(sig, result));
  }

  // Reference: [P] Proposition 15.20, p 131
  template< class Matrix_T >
  void
  generator_table<Matrix_T>::
  gen_from_qp1_pm1(const std::vector<Matrix_T>& old, const signature_t sig)
  {
    const auto old_size = old.size();
    using size_t = decltype(old_size);
    using result_t = std::vector<Matrix_T>;
    auto result = result_t(old_size);

    const auto& h = old[old_size-1];
    for (auto
        k = size_t(0);
        k != old_size-1;
        ++k)
      result[k] = matrix::mono_prod(old[old_size-2-k], h);
    result[old_size-1] = h;

    // Save the resulting generator array.
    this->insert(make_pair(sig, result));
  }

} }
#endif  // _GLUCAT_GENERATION_IMP_H
```

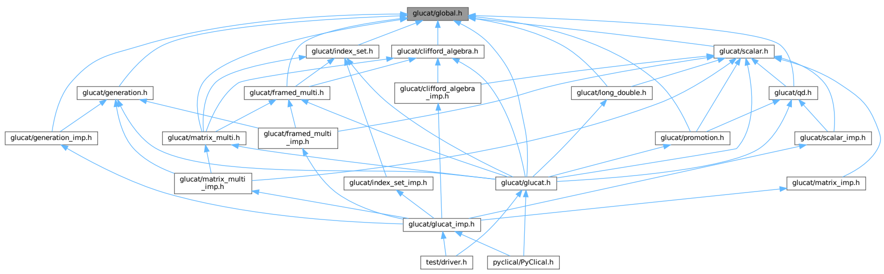
7.17 glucat/global.h File Reference

```
#include "glucat/portability.h"
#include <limits>
#include <climits>
```

Include dependency graph for global.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct glucat::CTAssertion< true >
- class glucat::compare_types< LHS_T, RHS_T >

    Type comparison.
- class glucat::compare_types< T, T >
- class glucat::bool_to_type< truth_value >

    Bool to type.

Namespaces

- namespace glucat

Macros

- #define _GLUCAT_CTAssert(expr, msg)

Typedefs

- using glucat::index_t = int

  Size of index_t should be enough to represent LO, HI.
- using glucat::set_value_t = unsigned long

  Size of set_value_t should be enough to contain index_set<LO,HI>

Functions

- glucat::_GLUCAT_CTAssert (std::numeric_limits< unsigned char >::radix==2, CannotDetermineBitsPer↩ Char) const index_t BITS_PER_CHAR

  If radix of unsigned char is not 2, we can't easily determine number of bits from sizeof.
- glucat::_GLUCAT_CTAssert (_GLUCAT_BITS_PER_ULONG==BITS_PER_SET_VALUE, BitsPerULong↩ DoesNotMatchSetValueT) const index_t DEFAULT_LO

  Default lowest index in an index set.
- template<typename LHS_T , typename RHS_T >
  auto glucat::pos_mod (LHS_T lhs, RHS_T rhs) -> LHS_T

  Modulo function which works reliably for lhs < 0.

Variables

- const double glucat::MS_PER_S = 1000.0

  Timing constant: deprecated here - moved to test/timing.h.
- const index_t glucat::BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits

  Number of bits in set_value_t.
- const index_t glucat::DEFAULT_HI = index_t(BITS_PER_SET_VALUE / 2)

  Default highest index in an index set.

7.17.1 Macro Definition Documentation

7.17.1.1 _GLUCAT_CTAssert

```
#define _GLUCAT_CTAssert(
              expr,
              msg)
```

Value:

```
namespace { struct msg { glucat::CTAssertion<(expr)> ERROR_##msg; }; }
```

Definition at line 48 of file global.h.

7.18 global.h

Go to the documentation of this file.

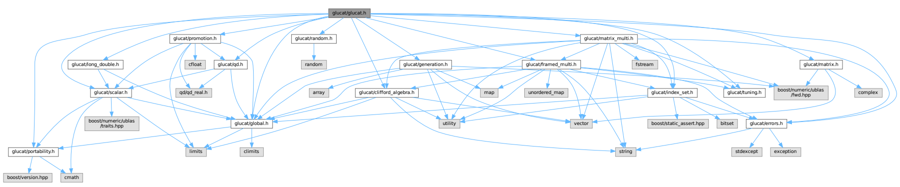
```
#ifndef _GLUCAT_GLOBAL_H
#define _GLUCAT_GLOBAL_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    global.h : Global declarations
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include "glucat/portability.h"

#include <limits>
#include <climits>

namespace glucat
{
  // References:
  // [AA]: A. Alexandrescu, "Modern C++ Design", Addison-Wesley, 2001.

  // Reference: [AA], p. 25
  template<bool> struct CTAssertion;
  template<> struct CTAssertion<true> { };
  #define _GLUCAT_CTAssert(expr, msg) \
      namespace { struct msg { glucat::CTAssertion<(expr)> ERROR_##msg; }; }

  // Reference: [AA], pp. 34--37
  template < typename LHS_T, typename RHS_T >
  class compare_types
  {
  public:
    enum { are_same = false };
  };
  template < typename T >
  class compare_types<T, T>
  {
  public:
    enum { are_same = true };
  };

  // Reference: [AA], 2.4, p. 29
  template< bool truth_value >
  class bool_to_type
  {
  private:
    enum { value = truth_value };
  };

  // Global types which determine sizes
  using index_t = int;
  using set_value_t = unsigned long;

  // Global constants
  const double MS_PER_S = 1000.0;

  // Constants which determine sizes

  // Bits per unsigned long
  #if  (ULONG_MAX == (4294967295UL))
```

```
#define _GLUCAT_BITS_PER_ULONG 32
#elif (ULONG_MAX == (18446744073709551615UL))
#define _GLUCAT_BITS_PER_ULONG 64
#elif defined(__WORDSIZE)
#define _GLUCAT_BITS_PER_ULONG __WORDSIZE
#endif

_GLUCAT_CTAssert(std::numeric_limits<unsigned char>::radix == 2, CannotDetermineBitsPerChar)


const index_t BITS_PER_CHAR = std::numeric_limits<unsigned char>::digits;

const index_t BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits;

_GLUCAT_CTAssert(_GLUCAT_BITS_PER_ULONG == BITS_PER_SET_VALUE, BitsPerULongDoesNotMatchSetValueT)

// Constants which are determined by size
const index_t DEFAULT_LO = -index_t(BITS_PER_SET_VALUE / 2);
const index_t DEFAULT_HI =  index_t(BITS_PER_SET_VALUE / 2);

template< typename LHS_T, typename RHS_T >
inline
auto
pos_mod(LHS_T lhs, RHS_T rhs) -> LHS_T
{ return lhs > 0? lhs % rhs : (-lhs) % rhs == 0 ? 0 : rhs - (-lhs) % rhs; }

}
#endif // _GLUCAT_GLOBAL_H
```

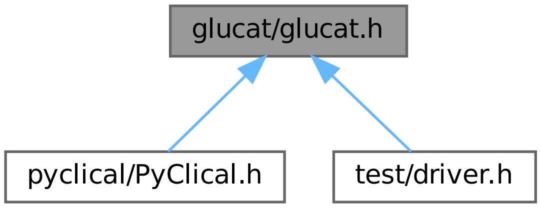
7.19 glucat/glucat.h File Reference

```
#include "glucat/portability.h"
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/scalar.h"
#include "glucat/long_double.h"
#include "glucat/qd.h"
#include "glucat/promotion.h"
#include "glucat/random.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include "glucat/matrix_multi.h"
```

Include dependency graph for glucat.h:



This graph shows which files directly or indirectly include this file:



7.20 glucat.h

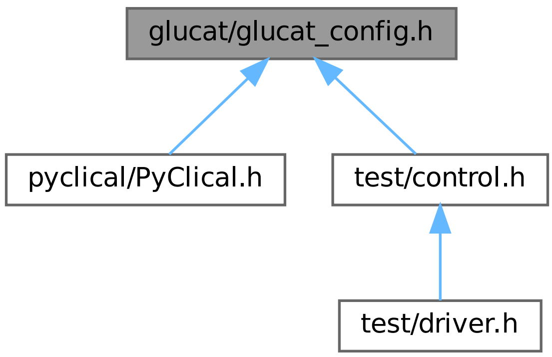
Go to the documentation of this file.

```
#ifndef _GLUCAT_GLUCAT_H
#define _GLUCAT_GLUCAT_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    glucat.h : Organize GluCat header files for applications
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     Arvind Raja's original header comments and references follow.
 ***************************************************************************
// clifford algebra package,  Arvind.Raja@hut.fi
// ref: Press et.al. "Numerical Recipes in C", 2nd ed., C.U.P., 1992.
// ref: LEDA, v 3.0, Stefan N\"aher, Max-Planck-Institut f\"ur Informatik
// ref: Stroustrup B., "The C++ Programming Language", 2nd ed.,
//      Addison-Wesley, 1991.
// ref: R. Sedgewick, "Algorithms in C++", Addison-Wesley, 1992.
// ref: S. Meyers, "Effective C++ ", Addison-Wesley, 1992.
 ***************************************************************************/

#include "glucat/portability.h"

#include "glucat/global.h"

#include "glucat/errors.h"

#include "glucat/index_set.h"

#include "glucat/scalar.h"

#include "glucat/long_double.h"

#include "glucat/qd.h"
```

```

#include "glucat/promotion.h"

#include "glucat/random.h"

#include "glucat/clifford_algebra.h"

#include "glucat/tuning.h"

#include "glucat/framed_multi.h"

#include "glucat/generation.h"

#include "glucat/matrix.h"

#include "glucat/matrix_multi.h"

#endif  // _GLUCAT_GLUCAT_H
```

7.21 glucat/glucat_config.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define GLUCAT_HAVE_CXX11 1
- #define GLUCAT_HAVE_INTTYPES_H 1
- #define GLUCAT_HAVE_STDINT_H 1
- #define GLUCAT_HAVE_STDIO_H 1
- #define GLUCAT_HAVE_STDLIB_H 1
- #define GLUCAT_HAVE_STRINGS_H 1
- #define GLUCAT_HAVE_STRING_H 1
- #define GLUCAT_HAVE_SYS_STAT_H 1
- #define GLUCAT_HAVE_SYS_TYPES_H 1
- #define GLUCAT_HAVE_UNISTD_H 1
- #define GLUCAT_PACKAGE "glucat"
- #define GLUCAT_PACKAGE_BUGREPORT ""
- #define GLUCAT_PACKAGE_NAME "glucat"
- #define GLUCAT_PACKAGE_STRING "glucat 0.12.0"
- #define GLUCAT_PACKAGE_TARNAME "glucat"

- #define GLUCAT_PACKAGE_URL ""
- #define GLUCAT_PACKAGE_VERSION "0.12.0"
- #define GLUCAT_STDC_HEADERS 1
- #define GLUCAT_VERSION "0.12.0"

7.21.1 Macro Definition Documentation

7.21.1.1 GLUCAT_HAVE_CXX11

```
#define GLUCAT_HAVE_CXX11 1
```

Definition at line 20 of file glucat_config.h.

7.21.1.2 GLUCAT_HAVE_INTTYPES_H

```
#define GLUCAT_HAVE_INTTYPES_H 1
```

Definition at line 28 of file glucat_config.h.

7.21.1.3 GLUCAT_HAVE_STDINT_H

```
#define GLUCAT_HAVE_STDINT_H 1
```

Definition at line 39 of file glucat_config.h.

7.21.1.4 GLUCAT_HAVE_STDIO_H

```
#define GLUCAT_HAVE_STDIO_H 1
```

Definition at line 44 of file glucat_config.h.

7.21.1.5 GLUCAT_HAVE_STDLIB_H

```
#define GLUCAT_HAVE_STDLIB_H 1
```

Definition at line 49 of file glucat_config.h.

7.21.1.6 GLUCAT_HAVE_STRING_H

```
#define GLUCAT_HAVE_STRING_H 1
```

Definition at line 59 of file glucat_config.h.

7.21.1.7 GLUCAT_HAVE_STRINGS_H

```
#define GLUCAT_HAVE_STRINGS_H 1
```

Definition at line 54 of file glucat_config.h.

7.21.1.8 GLUCAT_HAVE_SYS_STAT_H

```
#define GLUCAT_HAVE_SYS_STAT_H 1
```

Definition at line 64 of file glucat_config.h.

7.21.1.9 GLUCAT_HAVE_SYS_TYPES_H

```
#define GLUCAT_HAVE_SYS_TYPES_H 1
```

Definition at line 69 of file glucat_config.h.

7.21.1.10 GLUCAT_HAVE_UNISTD_H

```
#define GLUCAT_HAVE_UNISTD_H 1
```

Definition at line 74 of file glucat_config.h.

7.21.1.11 GLUCAT_PACKAGE

```
#define GLUCAT_PACKAGE "glucat"
```

Definition at line 79 of file glucat_config.h.

7.21.1.12 GLUCAT_PACKAGE_BUGREPORT

```
#define GLUCAT_PACKAGE_BUGREPORT ""
```

Definition at line 84 of file glucat_config.h.

7.21.1.13 GLUCAT_PACKAGE_NAME

```
#define GLUCAT_PACKAGE_NAME "glucat"
```

Definition at line 89 of file glucat_config.h.

Referenced by glucat::control_t::control_t().

7.21.1.14 GLUCAT_PACKAGE_STRING

```
#define GLUCAT_PACKAGE_STRING "glucat 0.12.0"
```

Definition at line 94 of file glucat_config.h.

7.21.1.15 GLUCAT_PACKAGE_TARNAME

```
#define GLUCAT_PACKAGE_TARNAME "glucat"
```

Definition at line 99 of file glucat_config.h.

7.21.1.16 GLUCAT_PACKAGE_URL

```
#define GLUCAT_PACKAGE_URL ""
```

Definition at line 104 of file glucat_config.h.

7.21.1.17 GLUCAT_PACKAGE_VERSION

```
#define GLUCAT_PACKAGE_VERSION "0.12.0"
```

Definition at line 109 of file glucat_config.h.

7.21.1.18 GLUCAT_STDC_HEADERS

```
#define GLUCAT_STDC_HEADERS 1
```

Definition at line 116 of file glucat_config.h.

7.21.1.19 GLUCAT_VERSION

```
#define GLUCAT_VERSION "0.12.0"
```

Definition at line 121 of file glucat_config.h.

Referenced by glucat::control_t::control_t().

7.22 glucat_config.h

Go to the documentation of this file.

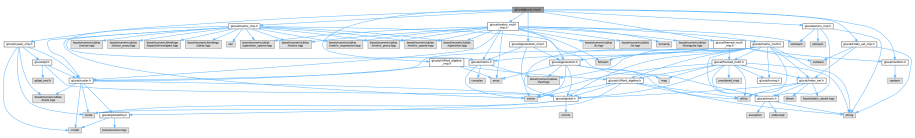
```
#ifndef _GLUCAT_GLUCAT_CONFIG_H
#define _GLUCAT_GLUCAT_CONFIG_H 1

/* glucat/glucat_config.h. Generated automatically at end of configure. */
/* config.h.  Generated from config.h.in by configure.  */
/* config.h.in.  Generated from configure.ac by autoheader.  */

/* Define to dummy `main' function (if any) required to link to the Fortran
   libraries. */
/* #undef F77_DUMMY_MAIN */

/* Define if F77 and FC dummy `main' functions are identical. */
/* #undef FC_DUMMY_MAIN_EQ_F77 */

/* Define if you have a BLAS library. */
/* #undef HAVE_BLAS */

/* define if the compiler supports basic C++11 syntax */
#ifndef GLUCAT_HAVE_CXX11
#define GLUCAT_HAVE_CXX11 1
#endif

/* define if the compiler supports basic C++14 syntax */
/* #undef HAVE_CXX14 */

/* Define to 1 if you have the <inttypes.h> header file. */
#ifndef GLUCAT_HAVE_INTTYPES_H
#define GLUCAT_HAVE_INTTYPES_H 1
#endif

/* Define if you have LAPACK library. */
/* #undef HAVE_LAPACK */

/* Define to 1 if you have the `imf' library (-limf). */
/* #undef HAVE_LIBIMF */

/* Define to 1 if you have the <stdint.h> header file. */
#ifndef GLUCAT_HAVE_STDINT_H
#define GLUCAT_HAVE_STDINT_H 1
#endif

/* Define to 1 if you have the <stdio.h> header file. */
#ifndef GLUCAT_HAVE_STDIO_H
#define GLUCAT_HAVE_STDIO_H 1
#endif

/* Define to 1 if you have the <stdlib.h> header file. */
#ifndef GLUCAT_HAVE_STDLIB_H
#define GLUCAT_HAVE_STDLIB_H 1
#endif

/* Define to 1 if you have the <strings.h> header file. */
#ifndef GLUCAT_HAVE_STRINGS_H
#define GLUCAT_HAVE_STRINGS_H 1
#endif

/* Define to 1 if you have the <string.h> header file. */
#ifndef GLUCAT_HAVE_STRING_H
#define GLUCAT_HAVE_STRING_H 1
#endif

/* Define to 1 if you have the <sys/stat.h> header file. */
#ifndef GLUCAT_HAVE_SYS_STAT_H
#define GLUCAT_HAVE_SYS_STAT_H 1
#endif

/* Define to 1 if you have the <sys/types.h> header file. */
#ifndef GLUCAT_HAVE_SYS_TYPES_H
#define GLUCAT_HAVE_SYS_TYPES_H 1
#endif

/* Define to 1 if you have the <unistd.h> header file. */
#ifndef GLUCAT_HAVE_UNISTD_H
#define GLUCAT_HAVE_UNISTD_H 1
#endif

/* Name of package */
#ifndef GLUCAT_PACKAGE
#define GLUCAT_PACKAGE "glucat"
#endif

/* Define to the address where bug reports for this package should be sent. */
```

```
#ifndef GLUCAT_PACKAGE_BUGREPORT
#define GLUCAT_PACKAGE_BUGREPORT ""
#endif

/* Define to the full name of this package. */
#ifndef GLUCAT_PACKAGE_NAME
#define GLUCAT_PACKAGE_NAME "glucat"
#endif

/* Define to the full name and version of this package. */
#ifndef GLUCAT_PACKAGE_STRING
#define GLUCAT_PACKAGE_STRING "glucat 0.12.0"
#endif

/* Define to the one symbol short name of this package. */
#ifndef GLUCAT_PACKAGE_TARNAME
#define GLUCAT_PACKAGE_TARNAME "glucat"
#endif

/* Define to the home page for this package. */
#ifndef GLUCAT_PACKAGE_URL
#define GLUCAT_PACKAGE_URL ""
#endif

/* Define to the version of this package. */
#ifndef GLUCAT_PACKAGE_VERSION
#define GLUCAT_PACKAGE_VERSION "0.12.0"
#endif

/* Define to 1 if all of the C90 standard headers exist (not just the ones
   required in a freestanding environment). This macro is provided for
   backward compatibility; new code need not use it. */
#ifndef GLUCAT_STDC_HEADERS
#define GLUCAT_STDC_HEADERS 1
#endif

/* Version number of package */
#ifndef GLUCAT_VERSION
#define GLUCAT_VERSION "0.12.0"
#endif

/* once: _GLUCAT_GLUCAT_CONFIG_H */
#endif
```

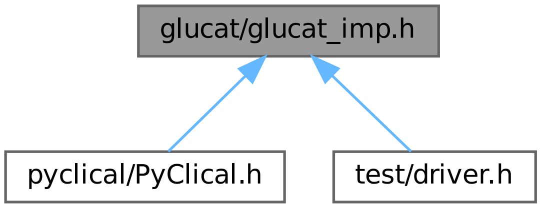
7.23 glucat/glucat_imp.h File Reference

```
#include "glucat/errors_imp.h"
#include "glucat/index_set_imp.h"
#include "glucat/scalar_imp.h"
#include "glucat/clifford_algebra_imp.h"
#include "glucat/random.h"
#include "glucat/framed_multi_imp.h"
#include "glucat/matrix_imp.h"
#include "glucat/generation_imp.h"
#include "glucat/matrix_multi_imp.h"
```

Include dependency graph for glucat_imp.h:



This graph shows which files directly or indirectly include this file:



7.24 glucat_imp.h

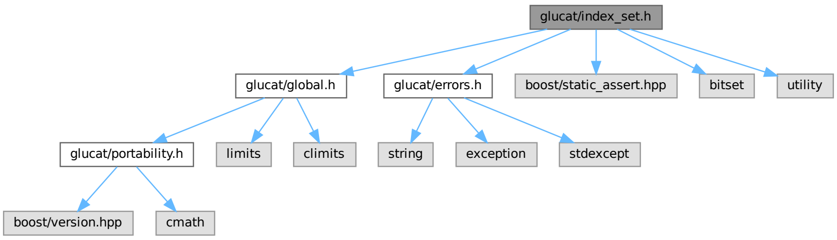
Go to the documentation of this file.

```
#ifndef _GLUCAT_GLUCAT_IMP_H
#define _GLUCAT_GLUCAT_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    glucat_imp.h : Organize GluCat template definitions which cannot be compiled separately
                             -------------------
    begin                : Sun 2001-12-25
    copyright            : (C) 2001-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     For Arvind Raja's original header comments, see glucat.h
 ***************************************************************************/

// Template definitions which cannot be compiled separately

#include "glucat/errors_imp.h"

#include "glucat/index_set_imp.h"

#include "glucat/scalar_imp.h"

#include "glucat/clifford_algebra_imp.h"

#include "glucat/random.h"

#include "glucat/framed_multi_imp.h"

#include "glucat/matrix_imp.h"

#include "glucat/generation_imp.h"

#include "glucat/matrix_multi_imp.h"

#endif  // _GLUCAT_GLUCAT_IMP_H
```

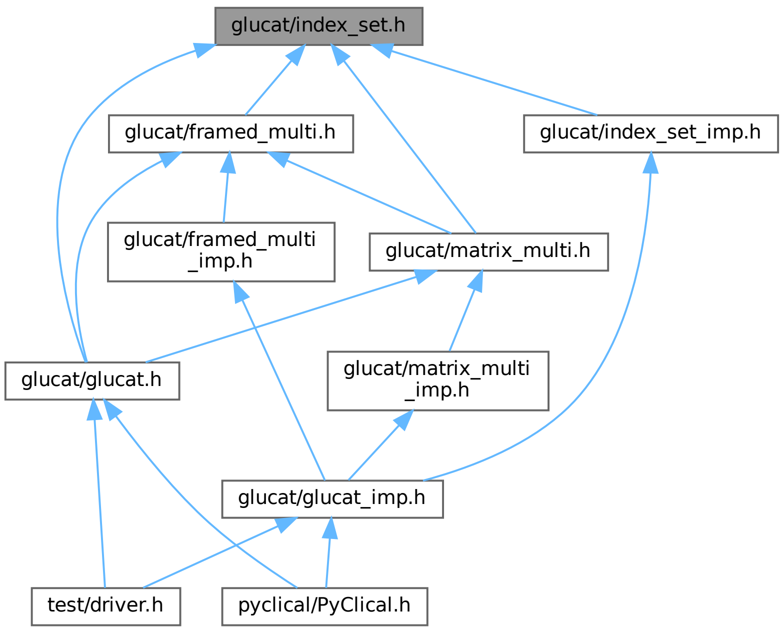
7.25 glucat/index_set.h File Reference

```
#include "glucat/global.h"
#include "glucat/errors.h"
#include <boost/static_assert.hpp>
#include <bitset>
#include <utility>
```

Include dependency graph for index_set.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::index_set< LO, HI >

    Index set class based on std::bitset<> in Gnu standard C++ library.
- class glucat::index_set< LO, HI >::reference

    Index set member reference.

Namespaces

- namespace glucat

Functions

- template<const index_t LO, const index_t HI>
auto glucat::operator^ (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Symmetric set difference: exclusive or.
- template<const index_t LO, const index_t HI>
auto glucat::operator& (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Set intersection: and.
- template<const index_t LO, const index_t HI>
auto glucat::operator| (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Set union: or.
- template<const index_t LO, const index_t HI>
auto glucat::compare (const index_set< LO, HI > &a, const index_set< LO, HI > &b) -> int

  "lexicographic compare" eg. {3,4,5} is less than {3,7,8}
- glucat::_GLUCAT_CTAssert (sizeof(set_value_t) >=sizeof(std::bitset< DEFAULT_HI-DEFAULT_LO >), Default_index_set_too_big_for_value) template< const index_t LO

  Size of set_value_t should be enough to contain bitset<DEFAULT_HI-DEFAULT_LO>
- const index_t HI auto glucat::operator<< (std::ostream &os, const index_set< LO, HI > &ist) -> std::ostream &
- template<const index_t LO, const index_t HI>
auto glucat::operator>> (std::istream &s, index_set< LO, HI > &ist) -> std::istream &

  Read in index set.
- auto glucat::sign_of_square (index_t j) -> int

  Square of generator {j}.
- template<const index_t LO, const index_t HI>
auto glucat::min_neg (const index_set< LO, HI > &ist) -> index_t

  Minimum negative index, or 0 if none.
- template<const index_t LO, const index_t HI>
auto glucat::max_pos (const index_set< LO, HI > &ist) -> index_t

  Maximum positive index, or 0 if none.

7.26 index_set.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_INDEX_SET_H
#define _GLUCAT_INDEX_SET_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    index_set.h : Declare a class for a set of non-zero integer indices
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
```

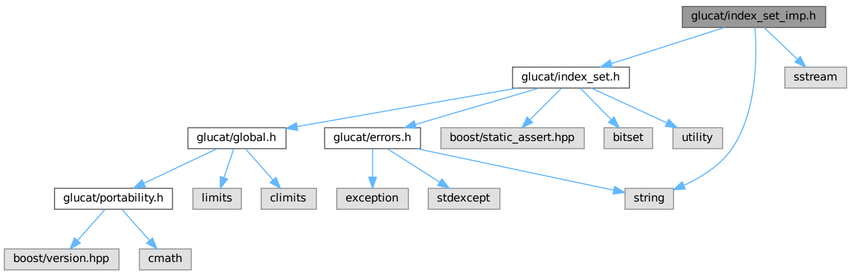
```
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public License
   along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/errors.h"

#include <boost/static_assert.hpp>

#include <bitset>
#include <utility>

namespace glucat
{
  template<const index_t LO, const index_t HI>
  class index_set; // forward

  template<const index_t LO, const index_t HI>
  auto
  operator^ (const index_set<LO,HI>& lhs,
             const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;

  template<const index_t LO, const index_t HI>
  auto
  operator& (const index_set<LO,HI>& lhs,
             const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;

  template<const index_t LO, const index_t HI>
  auto
  operator| (const index_set<LO,HI>& lhs,
             const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;

  // -1 if a<b, +1 if a>b, 0 if a==b
  template<const index_t LO, const index_t HI>
  auto
  compare(const index_set<LO,HI>& a, const index_set<LO,HI>& b) -> int;

  template<const index_t LO, const index_t HI>
  class index_set :
  private std::bitset<HI-LO>
  {
  private:
    BOOST_STATIC_ASSERT((LO <= 0) && (0 <= HI) && (LO < HI) && \
                        (-LO    <  _GLUCAT_BITS_PER_ULONG)    && \
                        ( HI    <  _GLUCAT_BITS_PER_ULONG)    && \
                        ( HI-LO <= _GLUCAT_BITS_PER_ULONG));
    using bitset_t = std::bitset<HI - LO>;
    using error_t = error<index_set>;
  public:
    using index_set_t = index_set;
    using index_pair_t = std::pair<index_t, index_t>;

    static const index_t v_lo = LO;
    static const index_t v_hi = HI;

    static auto classname() -> const std::string;
    index_set     () = default;
    index_set     (const bitset_t bst);
    index_set     (const index_t idx);
    index_set     (const set_value_t folded_val, const index_set_t frm, const bool prechecked = false);
    index_set     (const index_pair_t& range, const bool prechecked = false);
    index_set     (const std::string& str);

    auto operator==  (const index_set_t rhs) const -> bool;
    auto operator!=  (const index_set_t rhs) const -> bool;
    auto operator~   () const -> index_set_t;
    auto operator^=  (const index_set_t rhs) -> index_set_t&;
    auto operator&=  (const index_set_t rhs) -> index_set_t&;
    auto operator|=  (const index_set_t rhs) -> index_set_t&;
    auto operator[]  (const index_t idx) const -> bool;
    auto test(const index_t idx) const -> bool;
    auto set() -> index_set_t&;
    auto set(const index_t idx) -> index_set_t&;
    auto set(const index_t idx, const int val) -> index_set_t&;
```

```
    auto reset() -> index_set_t&;
    auto reset(const index_t idx) -> index_set_t&;
    auto flip() -> index_set_t&;
    auto flip(const index_t idx) -> index_set_t&;
    auto count() const -> index_t;
    auto count_neg() const -> index_t;
    auto count_pos() const -> index_t;
    auto min() const -> index_t;
    auto max() const -> index_t;

  // Functions which support Clifford algebra operations
    auto operator<     (const index_set_t rhs) const -> bool;
    auto is_contiguous () const -> bool;
    auto fold          () const -> const index_set_t;
    auto fold          (const index_set_t frm, const bool prechecked = false) const -> const index_set_t;
    auto unfold        (const index_set_t frm, const bool prechecked = false) const -> const index_set_t;
    auto value_of_fold (const index_set_t frm) const -> set_value_t;
    auto sign_of_mult  (const index_set_t ist) const -> int;
    auto sign_of_square()                      const -> int;

    auto hash_fn       ()                      const -> size_t;

  // Friends
    friend auto operator^<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
    friend auto operator&<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
    friend auto operator|<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
    friend auto compare<>                   (const index_set_t& lhs, const index_set_t& rhs) -> int;

  // Member reference:
    class reference;
    friend class reference;

    class reference {
      friend class index_set;

    public:
      reference() = delete;
      reference    (index_set_t& ist, index_t idx);
      ~reference   () = default;
      auto operator== (const reference& c_j) const -> bool;
      auto operator= (const bool x) -> reference&;
      auto operator= (const reference& c_j) -> reference&;
      auto operator~ () const -> bool;
           operator bool () const;
      auto flip() -> reference&;

    private:
      index_set_t* m_pst;
      index_t      m_idx;
    };
    auto operator[](index_t idx) -> reference;
  private:
    auto lex_less_than (const index_set_t rhs) const -> bool;
  };

  _GLUCAT_CTAssert(sizeof(set_value_t) >= sizeof(std::bitset<DEFAULT_HI-DEFAULT_LO>),
           Default_index_set_too_big_for_value)

  // non-members

  template<const index_t LO, const index_t HI>
  auto
  operator« (std::ostream& os, const index_set<LO,HI>& ist) -> std::ostream&;

  template<const index_t LO, const index_t HI>
  auto
  operator» (std::istream& s, index_set<LO,HI>& ist) -> std::istream&;

  // Functions which support Clifford algebra operations
  auto sign_of_square(index_t j) -> int;

  template<const index_t LO, const index_t HI>
  auto
  min_neg(const index_set<LO,HI>& ist) -> index_t;

  template<const index_t LO, const index_t HI>
  auto
  max_pos(const index_set<LO,HI>& ist) -> index_t;
}
#endif // _GLUCAT_INDEX_SET_H
```

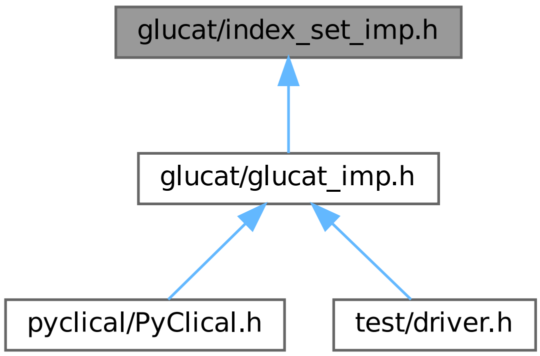
7.27 glucat/index_set_imp.h File Reference

```
#include "glucat/index_set.h"
#include <string>
#include <sstream>
```

Include dependency graph for index_set_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Functions

- template<const index_t LO, const index_t HI>
  auto glucat::operator^ (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

  Symmetric set difference: exclusive or.
- template<const index_t LO, const index_t HI>
  auto glucat::operator& (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

Set intersection: and.

- template<const index_t LO, const index_t HI>
auto glucat::operator| (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >

    Set union: or.

- template<const index_t LO, const index_t HI>
auto glucat::compare (const index_set< LO, HI > &a, const index_set< LO, HI > &b) -> int

    "lexicographic compare" eg. {3,4,5} is less than {3,7,8}

- template<const index_t LO, const index_t HI>
auto glucat::operator<< (std::ostream &os, const index_set< LO, HI > &ist) -> std::ostream &

    Write out index set.

- template<const index_t LO, const index_t HI>
auto glucat::operator>> (std::istream &s, index_set< LO, HI > &ist) -> std::istream &

    Read in index set.

- static auto glucat::inverse_reversed_gray (unsigned long x) -> unsigned long

    Inverse reversed Gray code.

- static auto glucat::inverse_gray (unsigned long x) -> unsigned long

    Inverse Gray code.

- auto glucat::sign_of_square (index_t j) -> int

    Square of generator {j}.

- template<const index_t LO, const index_t HI>
auto glucat::min_neg (const index_set< LO, HI > &ist) -> index_t

    Minimum negative index, or 0 if none.

- template<const index_t LO, const index_t HI>
auto glucat::max_pos (const index_set< LO, HI > &ist) -> index_t

    Maximum positive index, or 0 if none.

7.28 index_set_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_INDEX_SET_IMP_H
#define _GLUCAT_INDEX_SET_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    index_set_imp.h : Implement a class for a set of non-zero integer indices
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2016 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/index_set.h"
```

```

#include <string>
#include <sstream>

namespace glucat
{
  // References for algorithms:
  // [JA]: Joerg Arndt,  "Algorithms for programmers", http://www.jjj.de/fxt/fxtbook.pdf
  //      Chapter 1, Bit wizardry, http://www.jjj.de/bitwizardry/bitwizardrypage.html
  // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  classname() -> const std::string
  { return "index_set"; }

  template<const index_t LO, const index_t HI>
  index_set<LO,HI>::
  index_set(const index_t idx)
  { this->set(idx); }

  template<const index_t LO, const index_t HI>
  index_set<LO,HI>::
  index_set(const bitset_t bst):
  bitset_t(bst)
  { }

  template<const index_t LO, const index_t HI>
  index_set<LO,HI>::
  index_set(const set_value_t folded_val, const index_set_t frm, const bool prechecked)
  {
    if (!prechecked && folded_val >= (set_value_t(1) « frm.count()))
        throw error_t("index_set(val,frm): cannot create: value gives an index set outside of frame");
    const index_set_t folded_frame = frm.fold();
    const index_t min_index = folded_frame.min();
    const index_t skip = min_index > 0 ? 1 : 0;
    const index_set_t folded_set = index_set_t(bitset_t(folded_val) « (min_index - skip - LO));
    *this = folded_set.unfold(frm);
  }

  template<const index_t LO, const index_t HI>
  index_set<LO,HI>::
  index_set(const index_pair_t& range, const bool prechecked)
  {
    if (!prechecked && (range.first < LO || range.second > HI))
        throw error_t("index_set(range): cannot create: range is too large");
    const index_t begin_bit = (range.first < 0)
                            ? range.first-LO
                            : range.first-LO-1;
    const index_t end_bit = (range.second < 0)
                            ? range.second-LO+1
                            : range.second-LO;
    unsigned long mask = ( (end_bit == _GLUCAT_BITS_PER_ULONG)
                           ? -1UL
                           : (1UL « end_bit)-1UL)
                         & ~((1UL « begin_bit)-1UL);
    *this = bitset_t(mask);
  }

  template<const index_t LO, const index_t HI>
  index_set<LO,HI>::
  index_set(const std::string& str)
  {
    std::istringstream ss(str);
    ss » *this;
    if (!ss)
      throw error_t("index_set_t(str): could not parse string");
    // Peek to see if the end of the string has been reached.
    ss.peek();
    if (!ss.eof())
      throw error_t("index_set_t(str): could not parse entire string");
  }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  operator== (const index_set_t rhs) const -> bool
  {
    const auto* pthis = static_cast<const bitset_t*>(this);
    return *pthis == static_cast<bitset_t>(rhs);
  }

  template<const index_t LO, const index_t HI>
  inline
```

```
auto
index_set<LO,HI>::
operator!= (const index_set_t rhs) const -> bool
{
  const auto* pthis = static_cast<const bitset_t*>(this);
  return *pthis != static_cast<bitset_t>(rhs);
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator~ () const -> index_set_t
{ return bitset_t::operator~(); }

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator^= (const index_set_t rhs) -> index_set_t&
{
  bitset_t* pthis = this;
  *pthis ^= static_cast<bitset_t>(rhs);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
operator^ (const index_set<LO,HI>& lhs,
           const index_set<LO,HI>& rhs) -> const
index_set<LO,HI>
{
  using index_set_t = index_set<LO, HI>;
  using bitset_t = typename index_set_t::bitset_t;
  return static_cast<bitset_t>(lhs) ^ static_cast<bitset_t>(rhs);
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator&= (const index_set_t rhs) -> index_set_t&
{
  bitset_t* pthis = this;
  *pthis &= static_cast<bitset_t>(rhs);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
operator& (const index_set<LO,HI>& lhs,
           const index_set<LO,HI>& rhs) -> const
index_set<LO,HI>
{
  using index_set_t = index_set<LO, HI>;
  using bitset_t = typename index_set_t::bitset_t;
  return static_cast<bitset_t>(lhs) & static_cast<bitset_t>(rhs);
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator|= (const index_set_t rhs) -> index_set_t&
{
  bitset_t* pthis = this;
  *pthis |= static_cast<bitset_t>(rhs);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
operator| (const index_set<LO,HI>& lhs,
           const index_set<LO,HI>& rhs) -> const
index_set<LO,HI>
{
  using index_set_t = index_set<LO, HI>;
  using bitset_t = typename index_set_t::bitset_t;
  return static_cast<bitset_t>(lhs) | static_cast<bitset_t>(rhs);
}

template<const index_t LO, const index_t HI>
inline
auto
```

```
index_set<LO,HI>::
operator[] (const index_t idx) -> reference
{ return reference(*this, idx); }

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator[] (const index_t idx) const -> bool
{ return this->test(idx); }

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
test(const index_t idx) const -> bool
{
  // Reference: [JA], 1.2.1
  return (idx < 0)
         ?   bool(bitset_t::to_ulong() & (1UL « (idx - LO)))
         : (idx > 0)
           ? bool(bitset_t::to_ulong() & (1UL « (idx - LO - 1)))
           : false;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
set() -> index_set_t&
{
  bitset_t::set();
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
set(index_t idx) -> index_set_t&
{
  if (idx > 0)
    bitset_t::set(idx-LO-1);
  else if (idx < 0)
    bitset_t::set(idx-LO);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
set(const index_t idx, const int val) -> index_set_t&
{
  if (idx > 0)
    bitset_t::set(idx-LO-1, val);
  else if (idx < 0)
    bitset_t::set(idx-LO, val);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
reset() -> index_set_t&
{
  bitset_t::reset();
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
reset(const index_t idx) -> index_set_t&
{
  if (idx > 0)
    bitset_t::reset(idx-LO-1);
  else if (idx < 0)
    bitset_t::reset(idx-LO);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
```

```
index_set<LO,HI>::
flip() -> index_set<LO,HI>&
{
  bitset_t::flip();
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
flip(const index_t idx) -> index_set_t&
{
  if (idx > 0)
    bitset_t::flip(idx-LO-1);
  else if (idx < 0)
    bitset_t::flip(idx-LO);
  return *this;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
count() const -> index_t
{
  unsigned long val = bitset_t::to_ulong();
  // Reference: [JA], 1.3
  if (val == 0)
    return 0;
  else
  {
    index_t result = 1;
    while (val &= val-1)
      ++result;
    return result;
  }
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
count_neg() const -> index_t
{
  static const index_set_t lo_mask = bitset_t((1UL « -LO) - 1UL);
  const index_set_t neg_part = *this & lo_mask;
  return neg_part.count();
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
count_pos() const -> index_t
{
  const auto* pthis = static_cast<const bitset_t*>(this);
  const index_set_t pos_part = *pthis » -LO;
  return pos_part.count();
}

#if (_GLUCAT_BITS_PER_ULONG == 64)
template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
min() const -> index_t
{
  // Reference: [JA], 1.3
  unsigned long val = bitset_t::to_ulong();
  if (val == 0)
    return 0;
  else
  {
    val -= val & (val-1); // isolate lowest bit

    index_t idx = 0;
    const index_t nbits = HI - LO;

    if (nbits > 8)
    {
      if (val & 0xffffffff00000000ul)
        idx += 32;
      if (val & 0xffff0000ffff0000ul)
        idx += 16;
      if (val & 0xff00ff00ff00ff00ul)
        idx +=  8;
```

```
      }
        if (val & 0xf0f0f0f0f0f0f0f0ul)
          idx +=  4;
        if (val & 0xccccccccccccccccul)
          idx +=  2;
        if (val & 0xaaaaaaaaaaaaaaaaul)
          idx +=  1;

      return idx + ((idx < -LO) ? LO : LO+1);
    }
  }
#elif (_GLUCAT_BITS_PER_ULONG == 32)
  template<const index_t LO, const index_t HI>
  inline
  index_t
  index_set<LO,HI>::
  min() const
  {
    // Reference: [JA], 1.3
    unsigned long val = bitset_t::to_ulong();
    if (val == 0)
      return 0;
    else
    {
      val -= val & (val-1); // isolate lowest bit

      index_t idx = 0;
      const index_t nbits = HI - LO;
      if (nbits > 8)
      {
        if (val & 0xffff0000ul)
          idx += 16;
        if (val & 0xff00ff00ul)
          idx +=  8;
      }
        if (val & 0xf0f0f0f0ul)
          idx +=  4;
        if (val & 0xccccccccul)
          idx +=  2;
        if (val & 0xaaaaaaaaul)
          idx +=  1;

      return idx + ((idx < -LO) ? LO : LO+1);
    }
  }
#else
  template<const index_t LO, const index_t HI>
  auto
  index_set<LO,HI>::
  min() const -> index_t
  {
    for (auto
        idx = LO;
        idx != 0;
        ++idx)
      if (this->test(idx))
        return idx;
    for (auto
        idx = index_t(1);
        idx <= HI;
        ++idx)
      if (this->test(idx))
        return idx;
    return 0;
  }
#endif

#if (_GLUCAT_BITS_PER_ULONG == 64)
  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  max() const -> index_t
  {
    // Reference: [JA], 1.6
    auto val = bitset_t::to_ulong();
    if (val == 0)
      return 0;
    else
    {
      auto idx = index_t(0);
      const auto nbits = HI - LO;
      if (nbits > 8)
      {
        if (val & 0xffffffff00000000ul)
          { val >>= 32; idx += 32; }
        if (val & 0x00000000ffff0000ul)
```

```
          { val >>= 16; idx += 16; }
        if (val & 0x000000000000ff00ul)
          { val >>=  8; idx +=  8; }
      }
        if (val & 0x00000000000000f0ul)
          { val >>=  4; idx +=  4; }
        if (val & 0x000000000000000cul)
          { val >>=  2; idx +=  2; }
        if (val & 0x0000000000000002ul)
          {              idx +=  1; }
      return idx + ((idx < -LO) ? LO : LO+1);
    }
  }
#elif (_GLUCAT_BITS_PER_ULONG == 32)
  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  max() const -> index_t
  {
    // Reference: [JA], 1.6
    auto val = bitset_t::to_ulong();
    if (val == 0)
      return 0;
    else
    {
      auto idx = index_t(0);
      const auto nbits = HI - LO;
      if (nbits > 8)
      {
        if (val & 0xffff0000ul)
          { val >>= 16; idx += 16; }
        if (val & 0x0000ff00ul)
          { val >>=  8; idx +=  8; }
      }
        if (val & 0x000000f0ul)
          { val >>=  4; idx +=  4; }
        if (val & 0x0000000cul)
          { val >>=  2; idx +=  2; }
        if (val & 0x00000002ul)
          {              idx +=  1; }
      return idx + ((idx < -LO) ? LO : LO+1);
    }
  }
#else
  template<const index_t LO, const index_t HI>
  auto
  index_set<LO,HI>::
  max() const -> index_t
  {
    for (auto
        idx = HI;
        idx != 0;
        --idx)
      if (this->test(idx))
        return idx;
    for (auto
        idx = index_t(-1);
        idx >= LO;
        --idx)
      if (this->test(idx))
        return idx;
    return 0;
  }
#endif

  //  eg. {3,4,5} is less than {3,7,8}
  template<const index_t LO, const index_t HI>
  inline
  auto
  compare(const index_set<LO,HI>& a, const index_set<LO,HI>& b) -> int
  {
    return (a == b)
         ? 0
         : a.lex_less_than(b)
           ? -1
           :  1;
  }

  //  eg. {3,4,5} is less than {3,7,8}
  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  lex_less_than(const index_set_t rhs) const -> bool
  { return bitset_t::to_ulong() < rhs.bitset_t::to_ulong(); }
```

```
// Order by count, then order lexicographically within the equivalence class of count.
template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
operator< (const index_set_t rhs) const -> bool
{
  const auto this_grade = this->count();
  const auto rhs_grade  = rhs.count();
  return (this_grade < rhs_grade)
         ? true
         : (this_grade > rhs_grade)
           ? false
           : this->lex_less_than(rhs);
}

template<const index_t LO, const index_t HI>
auto
operator« (std::ostream& os, const index_set<LO,HI>& ist) -> std::ostream&
{
  index_t i;
  os « '{';
  for (i = LO;
      (i <= HI) && !(ist[i]);
      ++i)
  { }
  if (i <= HI)
    os « i;
  for (++i;
      i <= HI;
      ++i)
    if (ist[i])
      os « ',' « i;
  os « '}';
  return os;
}

template<const index_t LO, const index_t HI>
auto
operator» (std::istream& s, index_set<LO,HI>& ist) -> std::istream&
{
  // Parsing variables.
  auto i = index_t(0);
  using index_set_t = index_set<LO,HI>;
  auto local_ist = index_set_t();
  // Parsing control variables.
  auto parse_index_list = true;
  auto expect_closing_brace = false;
  auto expect_index = false;
  // Parse an optional opening brace.
  auto c = s.peek();
  // If there is a failure or end of file, this ends parsing.
  if (!s.good())
    parse_index_list = false;
  else
  { // Check for an opening brace.
    expect_closing_brace = (c == int('{'));
    if (expect_closing_brace)
    { // Consume the opening brace.
      s.get();
      // The next character may be a closing brace,
      // indicating the empty index set.
      c = s.peek();
      if (s.good() && (c == int('}')))
      { // A closing brace has been parsed and is no longer expected.
        expect_closing_brace = false;
        // Consume the closing brace.
        s.get();
        // This ends parsing.
        parse_index_list = false;
      }
    }
  }
  if (s.good() && parse_index_list)
  { // Parse an optional index list.
    // The index list starts with a first index.
    for (s » i;
        !s.fail();
        s » i)
    { // An index has been parsed. Check to see if it is in range.
      if ((i < LO) || (i > HI))
      { // An index out of range is a failure.
        s.clear(std::istream::failbit);
        break;
      }
      // Add the index to the index set local_ist.
      local_ist.set(i);
```

```
      // Immediately after parsing an index, an index is no longer expected.
      expect_index = false;
      // Reading the index may have resulted in an end of file condition.
      // If so, this ends the index list.
      if (s.eof())
        break;
      // The index list continues with a comma, and
      // may be ended by a closing brace, if it was begun with an opening brace.
      // Parse a possible comma or closing brace.
      c = s.peek();
      if (!s.good())
        break;
      // First, test for a closing brace, if expected.
      if (expect_closing_brace && (c == int('}')))
      { // Consume the closing brace.
        s.get();
        // Immediately after parsing the closing brace, it is no longer expected.
        expect_closing_brace = false;
        // A closing brace ends the index list.
        break;
      }
      // Now test for a comma.
      if (c == int(','))
      { // Consume the comma.
        s.get();
        // A index is expected after the comma.
        expect_index = true;
      }
      else
      { // Any other character here is a failure.
        s.clear(std::istream::failbit);
        break;
      }
    }
  }
  // If an index or a closing brace is still expected, this is a failure.
  if (expect_index || expect_closing_brace)
    s.clear(std::istream::failbit);
  // End of file is not a failure.
  if (s)
  { // The index set has been successfully parsed.
    ist = local_ist;
  }
  return s;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
is_contiguous () const -> bool
{
  const auto min_index = this->min();
  const auto max_index = this->max();
  return (min_index < 0 && max_index > 0)
       ?  max_index - min_index == this->count()
       : (min_index == 1 || max_index == -1) &&
         (max_index - min_index == this->count() - 1);
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
fold() const -> const
index_set<LO,HI>
{ return this->fold(*this, true); }

template<const index_t LO, const index_t HI>
auto
index_set<LO,HI>::
fold(const index_set_t frm, const bool prechecked) const -> const
index_set<LO,HI>
{
  if (!prechecked && ((*this | frm) != frm))
    throw error_t("fold(frm): cannot fold from outside of frame");
  const auto frm_min = frm.min();
  const auto frm_max = frm.max();
  auto result = index_set_t();
  auto fold_idx = index_t(-1);
  for (auto
      unfold_idx = fold_idx;
      unfold_idx >= frm_min;
      --unfold_idx)
    if (frm.test(unfold_idx))
      // result.set(fold_idx--, this->test(unfold_idx));
    {
```

```
      if (this->test(unfold_idx))
        result.set(fold_idx);
      --fold_idx;
    }
  fold_idx = index_t(1);
  for (auto
      unfold_idx = fold_idx;
      unfold_idx <= frm_max;
      ++unfold_idx)
    if (frm.test(unfold_idx))
      // result.set(fold_idx++, this->test(unfold_idx));
    {
      if (this->test(unfold_idx))
        result.set(fold_idx);
      ++fold_idx;
    }
  return result;
}

template<const index_t LO, const index_t HI>
auto
index_set<LO,HI>::
unfold(const index_set_t frm, const bool prechecked) const -> const index_set_t
{
  const char* msg =
    "unfold(frm): cannot unfold into a smaller frame";
  const auto frm_min = frm.min();
  const auto frm_max = frm.max();
  auto result = index_set_t();
  auto fold_idx = index_t(-1);
  for (auto
      unfold_idx = fold_idx;
      unfold_idx >= frm_min;
      --unfold_idx)
    if (frm.test(unfold_idx))
      if (this->test(fold_idx--))
        result.set(unfold_idx);
  if (!prechecked && ((fold_idx+1) > this->min()))
    throw error_t(msg);
  fold_idx = index_t(1);
  for (auto
      unfold_idx = fold_idx;
      unfold_idx <= frm_max;
      ++unfold_idx)
    if (frm.test(unfold_idx))
      if (this->test(fold_idx++))
        result.set(unfold_idx);
  if (!prechecked && ((fold_idx-1) < this->max()))
    throw error_t(msg);
  return result;
}

template<const index_t LO, const index_t HI>
inline
auto
index_set<LO,HI>::
value_of_fold(const index_set_t frm) const -> set_value_t
{
  const auto min_index = frm.fold().min();
  if (min_index == 0)
    return 0;
  else
  {
    const auto folded_set = this->fold(frm);
    const auto skip = min_index > 0 ? index_t(1) : index_t(0);
    return folded_set.bitset_t::to_ulong() » (min_index-LO-skip);
  }
}

inline
static
auto inverse_reversed_gray(unsigned long x) -> unsigned long
{
  // Reference: [JA]
#if (_GLUCAT_BITS_PER_ULONG >= 64)
  x ^= x « 32; // for 64-bit words
#endif
  x ^= x « 16; // reversed_gray ** 16
  x ^= x «  8; // reversed_gray **  8
  x ^= x «  4; // reversed_gray **  4
  x ^= x «  2; // reversed_gray **  2
  x ^= x «  1; // reversed_gray **  1
  return x;
}

inline
static
```

```
  auto inverse_gray(unsigned long x) -> unsigned long
  {
    // Reference: [JA]
#if (_GLUCAT_BITS_PER_ULONG >= 64)
    x ^= x » 32; // for 64-bit words
#endif
    x ^= x » 16; // gray ** 16
    x ^= x »  8; // gray **  8
    x ^= x »  4; // gray **  4
    x ^= x »  2; // gray **  2
    x ^= x »  1; // gray **  1
    return x;
  }

  template<const index_t LO, const index_t HI>
  auto
  index_set<LO,HI>::
  sign_of_mult(const index_set_t rhs) const -> int
  {
    // Implemented using Walsh functions and Gray codes.
    // Reference: [L] Chapter 21, 21.3
    // Reference: [JA]
    const auto uthis = this->bitset_t::to_ulong();
    const auto urhs  =   rhs.bitset_t::to_ulong();
    const auto nbits = HI - LO;
    auto negative = 0UL;
    if (nbits > 8)
    {
      // Set h to be the inverse reversed Gray code of rhs.
      // This sets each bit of h to be the cumulative ^ of
      // the same and lower bits of rhs.
      const auto h = inverse_reversed_gray(urhs);
      // Set k to be the inverse Gray code of *this & h.
      // This sets the low bit of k to be parity(*this & h).
      const auto k = inverse_gray(uthis & h);
      // Set q to be the inverse Gray code of the positive part of *this & rhs.
      const auto q = inverse_gray((uthis & urhs) » -LO);
      negative = k ^ q;
    }
    else
    {
      auto h = 0UL;
      for (auto
          j = index_t(0);
          j < -LO;
          ++j)
      {
        h ^= urhs » j;
        negative ^= h & (uthis » j);
      }
      for (auto
          j = index_t(-LO);
          j < nbits;
          ++j)
      {
        negative ^= h & (uthis » j);
        h ^= urhs » j;
      }
    }
    return 1 - int((negative & 1) « 1);
  }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  sign_of_square() const -> int
  {
    auto result = 1 - int((this->count_neg() % 2) « 1);
    switch (this->count() % 4)
    {
      case 2:
      case 3:
        result *= -1;
        break;
      default:
        break;
    }
    return result;
  }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::
  hash_fn() const -> size_t
  {
```

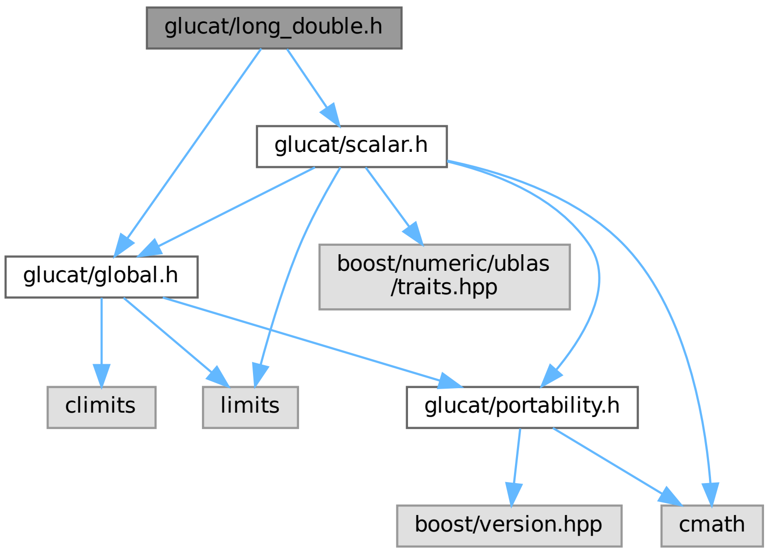
```
    static const auto lo_mask = (1UL « -LO) - 1UL;
    const auto uthis = bitset_t::to_ulong();
    const auto neg_part = uthis & lo_mask;
    const auto pos_part = uthis » -LO;
    return size_t(neg_part ^ pos_part);
  }

  inline
  auto
  sign_of_square(index_t j) -> int
  { return (j < 0) ? -1 : 1; }

  template<const index_t LO, const index_t HI>
  inline
  auto
  min_neg(const index_set<LO,HI>& ist) -> index_t
  { return std::min(ist.min(), 0); }

  template<const index_t LO, const index_t HI>
  inline
  auto
  max_pos(const index_set<LO,HI>& ist) -> index_t
  { return std::max(ist.max(), 0); }

// index_set reference

  template<const index_t LO, const index_t HI>
  inline
  index_set<LO,HI>::reference::
  reference( index_set_t& ist, index_t idx ) :
    m_pst(&ist),
    m_idx(idx)
  { }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::reference::
  operator== (const reference& c_j) const -> bool
  { return m_pst == c_j.m_pst && m_idx == c_j.m_idx; }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::reference::
  operator= (bool x) -> reference&
  {
    if ( x )
      m_pst->set(m_idx);
    else
      m_pst->reset(m_idx);
    return *this;
  }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::reference::
  operator= (const reference& c_j) -> reference&
  {
    if (&c_j != this && c_j != *this)
    {
      if ( (c_j.m_pst)[c_j.m_idx] )
        m_pst->set(m_idx);
      else
        m_pst->reset(m_idx);
    }
    return *this;
  }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::reference::
  operator~ () const -> bool
  { return !(m_pst->test(m_idx)); }

  template<const index_t LO, const index_t HI>
  inline
  index_set<LO,HI>::reference::
  operator bool () const
  { return m_pst->test(m_idx); }

  template<const index_t LO, const index_t HI>
  inline
  auto
  index_set<LO,HI>::reference::
```

```
  flip() -> reference&
  {
    m_pst->flip(m_idx);
    return *this;
  }
}
#endif // _GLUCAT_INDEX_SET_IMP_H
```

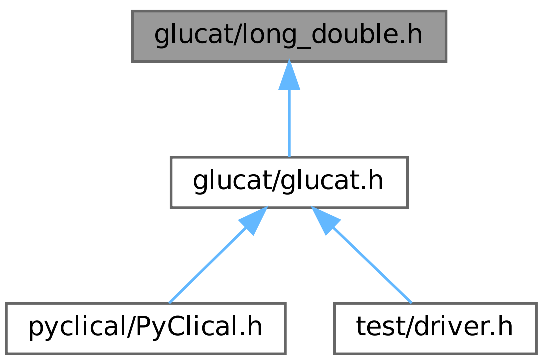
7.29 glucat/long_double.h File Reference

```
#include "glucat/global.h"
#include "glucat/scalar.h"
```

Include dependency graph for long_double.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Variables

- static const long double glucat::l_pi = 3.14159265358979323846264338327950 29L
- static const long double glucat::l_ln2 = 0.69314718055994530941723212145817 66L

7.30 long_double.h

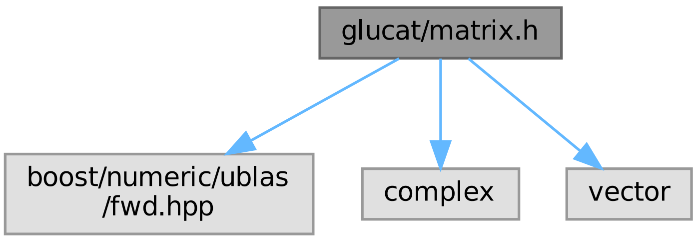
Go to the documentation of this file.

```
#ifndef _GLUCAT_LONG_DOUBLE_H
#define _GLUCAT_LONG_DOUBLE_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    long_double.h : Define std functions for long double
                             -------------------
    begin                : 2001-12-18
    copyright            : (C) 2001-2016 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/scalar.h"

namespace glucat
{
#if defined(__USE_GNU)
  static const long double l_pi   = M_PIl;
  static const long double l_ln2 = M_LN2l;
#else
  static const long double l_pi = 3.1415926535897932384626433832795029L;
  static const long double l_ln2 = 0.6931471805599453094172321214581766L;
#endif

  template<>
  inline
  auto
  numeric_traits<long double>::
  pi() -> long double
  { return l_pi; }

  template<>
  inline
  auto
  numeric_traits<long double>::
  ln_2() -> long double
  { return l_ln2; }
}
#endif // _GLUCAT_LONG_DOUBLE_H
```

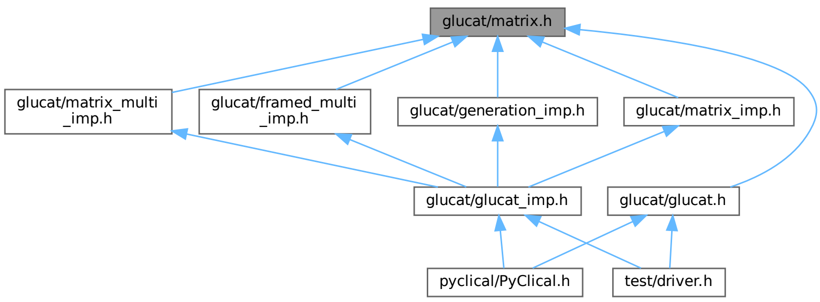
7.31 glucat/matrix.h File Reference

```
#include <boost/numeric/ublas/fwd.hpp>
#include <complex>
#include <vector>
```

Include dependency graph for matrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct glucat::matrix::eig_genus< Matrix_T >

  Structure containing classification of eigenvalues.

Namespaces

- namespace glucat
- namespace glucat::matrix

Typedefs

- using glucat::matrix::eig_case_t

  Classification of eigenvalues of a matrix.

Functions

- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Kronecker tensor product of matrices - as per Matlab kron.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::mono_kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Sparse Kronecker tensor product of monomial matrices.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::nork (const LHS_T &lhs, const RHS_T &rhs, const bool mono=true) -> const RHS_T

  Left inverse of Kronecker product.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::signed_perm_nork (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Left inverse of Kronecker product where lhs is a signed permutation matrix.
- template< typename Matrix_T >
  auto glucat::matrix::nnz (const Matrix_T &m) -> typename Matrix_T::size_type

  Number of non-zeros.
- template< typename Matrix_T >
  auto glucat::matrix::isinf (const Matrix_T &m) -> bool

  Infinite.
- template< typename Matrix_T >
  auto glucat::matrix::isnan (const Matrix_T &m) -> bool

  Not a Number.
- template< typename Matrix_T >
  auto glucat::matrix::unit (const typename Matrix_T::size_type n) -> const Matrix_T

  Unit matrix - as per Matlab eye.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::mono_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↩expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of monomial matrices.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::sparse_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↩expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of sparse matrices.
- template< typename LHS_T , typename RHS_T >
  auto glucat::matrix::prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

  Product of matrices.
- template< typename Scalar_T , typename LHS_T , typename RHS_T >
  auto glucat::matrix::inner (const LHS_T &lhs, const RHS_T &rhs) -> Scalar_T

  *Inner product: sum(x(i,j)*y(i,j))/x.nrows()*
- template< typename Matrix_T >
  auto glucat::matrix::norm_frob2 (const Matrix_T &val) -> typename Matrix_T::value_type

  Square of Frobenius norm.
- template< typename Matrix_T >
  auto glucat::matrix::trace (const Matrix_T &val) -> typename Matrix_T::value_type

  Matrix trace.
- template< typename Matrix_T >
  auto glucat::matrix::eigenvalues (const Matrix_T &val) -> std::vector< std::complex< double > >

  Eigenvalues of a matrix.
- template< typename Matrix_T >
  auto glucat::matrix::classify_eigenvalues (const Matrix_T &val) -> eig_genus< Matrix_T >

  Classify the eigenvalues of a matrix.

7.32 matrix.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_MATRIX_H
#define _GLUCAT_MATRIX_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    matrix.h : Declare common matrix functions
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2012 by Paul C. Leopardi
                         : uBLAS interface contributed by Joerg Walter
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include <boost/numeric/ublas/fwd.hpp>

#include <complex>
#include <vector>

namespace glucat
{
  namespace ublas = boost::numeric::ublas;

  namespace matrix
  {
    template< typename LHS_T, typename RHS_T >
    auto
    kron(const LHS_T& lhs, const RHS_T& rhs) -> const
    RHS_T;

    template< typename LHS_T, typename RHS_T >
    auto
    mono_kron(const LHS_T& lhs, const RHS_T& rhs) -> const
    RHS_T;

    template< typename LHS_T, typename RHS_T >
    auto
    nork(const LHS_T& lhs, const RHS_T& rhs, const bool mono = true) -> const
    RHS_T;

    template< typename LHS_T, typename RHS_T >
    auto
    signed_perm_nork(const LHS_T& lhs, const RHS_T& rhs) -> const
    RHS_T;

    template< typename Matrix_T >
    auto
    nnz(const Matrix_T& m) -> typename Matrix_T::size_type;

    template< typename Matrix_T >
    auto
    isinf(const Matrix_T& m) -> bool;

    template< typename Matrix_T >
    auto
    isnan(const Matrix_T& m) -> bool;

    template< typename Matrix_T >
    auto
    unit(const typename Matrix_T::size_type n) -> const
    Matrix_T;

```

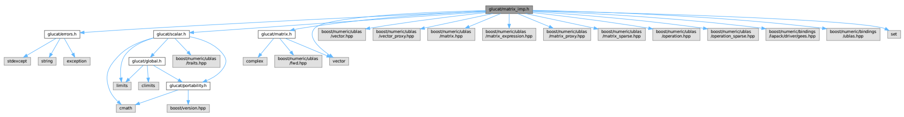
```
    template< typename LHS_T, typename RHS_T >
    auto
    mono_prod(const ublas::matrix_expression<LHS_T>& lhs,
              const ublas::matrix_expression<RHS_T>& rhs) -> const
    typename RHS_T::expression_type;

    template< typename LHS_T, typename RHS_T >
    auto
    sparse_prod(const ublas::matrix_expression<LHS_T>& lhs,
                const ublas::matrix_expression<RHS_T>& rhs) -> const
    typename RHS_T::expression_type;

    template< typename LHS_T, typename RHS_T >
    auto
    prod(const ublas::matrix_expression<LHS_T>& lhs,
         const ublas::matrix_expression<RHS_T>& rhs) -> const
    typename RHS_T::expression_type;

    template< typename Scalar_T, typename LHS_T, typename RHS_T >
    auto
    inner(const LHS_T& lhs, const RHS_T& rhs) -> Scalar_T;

    template< typename Matrix_T >
    auto
    norm_frob2(const Matrix_T& val) -> typename Matrix_T::value_type;

    template< typename Matrix_T >
    auto
    trace(const Matrix_T& val) -> typename Matrix_T::value_type;

    template< typename Matrix_T >
    auto
    eigenvalues(const Matrix_T& val) -> std::vector< std::complex<double> >;

    using eig_case_t = enum {
      safe_eigs,
      neg_real_eigs,
      both_eigs};

    template< typename Matrix_T >
    struct eig_genus
    {
      using Scalar_T = typename Matrix_T::value_type;
      bool m_is_singular = false;
      eig_case_t m_eig_case = safe_eigs;
      Scalar_T   m_safe_arg = Scalar_T(0);
    };

    template< typename Matrix_T >
    auto
    classify_eigenvalues(const Matrix_T& val) -> eig_genus<Matrix_T>;
  }
}

#endif  // _GLUCAT_MATRIX_H
```

7.33 glucat/matrix_imp.h File Reference

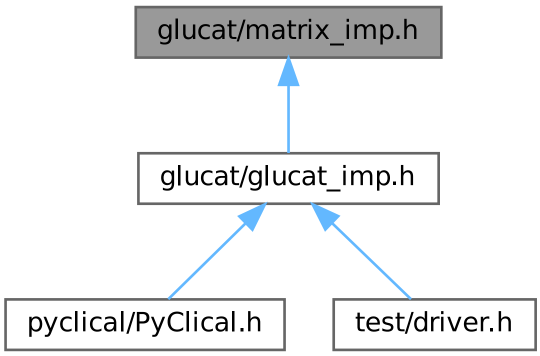
```
#include "glucat/errors.h"
#include "glucat/scalar.h"
#include "glucat/matrix.h"
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/vector_proxy.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>
#include <boost/numeric/bindings/lapack/driver/gees.hpp>
#include <boost/numeric/bindings/ublas.hpp>
#include <set>
```

`#include <vector>`

Include dependency graph for matrix_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat
- namespace glucat::matrix

Functions

- template<typename LHS_T , typename RHS_T >
  auto glucat::matrix::kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Kronecker tensor product of matrices - as per Matlab kron.
- template<typename LHS_T , typename RHS_T >
  auto glucat::matrix::mono_kron (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Sparse Kronecker tensor product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
  void glucat::matrix::nork_range (RHS_T &result, const typename LHS_T::const_iterator2 lhs_it2, const RHS_T &rhs, const typename RHS_T::size_type res_s1, const typename RHS_T::size_type res_s2)

  Utility routine for nork: calculate result for a range of indices.
- template<typename LHS_T , typename RHS_T >
  auto glucat::matrix::nork (const LHS_T &lhs, const RHS_T &rhs, const bool mono=true) -> const RHS_T

  Left inverse of Kronecker product.
- template<typename LHS_T , typename RHS_T >
  auto glucat::matrix::signed_perm_nork (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T

  Left inverse of Kronecker product where lhs is a signed permutation matrix.

- template<typename Matrix_T >
auto glucat::matrix::nnz (const Matrix_T &m) -> typename Matrix_T::size_type

    Number of non-zeros.
- template<typename Matrix_T >
auto glucat::matrix::isinf (const Matrix_T &m) -> bool

    Infinite.
- template<typename Matrix_T >
auto glucat::matrix::isnan (const Matrix_T &m) -> bool

    Not a Number.
- template<typename Matrix_T >
auto glucat::matrix::unit (const typename Matrix_T::size_type n) -> const Matrix_T

    Unit matrix - as per Matlab eye.
- template<typename LHS_T , typename RHS_T >
auto glucat::matrix::mono_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↩ expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

    Product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
auto glucat::matrix::sparse_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↩ expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

    Product of sparse matrices.
- template<typename LHS_T , typename RHS_T >
auto glucat::matrix::prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type

    Product of matrices.
- template<typename Scalar_T , typename LHS_T , typename RHS_T >
auto glucat::matrix::inner (const LHS_T &lhs, const RHS_T &rhs) -> Scalar_T

    Inner product: sum(x(i,j)∗y(i,j))/x.nrows()
- template<typename Matrix_T >
auto glucat::matrix::norm_frob2 (const Matrix_T &val) -> typename Matrix_T::value_type

    Square of Frobenius norm.
- template<typename Matrix_T >
auto glucat::matrix::trace (const Matrix_T &val) -> typename Matrix_T::value_type

    Matrix trace.
- template<typename Matrix_T >
static auto glucat::matrix::to_lapack (const Matrix_T &val) -> ublas::matrix< double, ublas::column_major >

    Convert matrix to LAPACK format.
- template<typename Matrix_T >
auto glucat::matrix::eigenvalues (const Matrix_T &val) -> std::vector< std::complex< double > >

    Eigenvalues of a matrix.
- template<typename Matrix_T >
auto glucat::matrix::classify_eigenvalues (const Matrix_T &val) -> eig_genus< Matrix_T >

    Classify the eigenvalues of a matrix.

7.34 matrix_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_MATRIX_IMP_H
#define _GLUCAT_MATRIX_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    matrix_imp.h : Implement common matrix functions
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2012 by Paul C. Leopardi
                         : uBLAS interface contributed by Joerg Walter
```

```
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/errors.h"
#include "glucat/scalar.h"
#include "glucat/matrix.h"

# if  defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
#  pragma GCC diagnostic push
#  pragma GCC diagnostic ignored "-Wunused-local-typedefs"
# endif
# if  defined(_GLUCAT_HAVE_BOOST_SERIALIZATION_ARRAY_WRAPPER_H)
#  include <boost/serialization/array_wrapper.hpp>
# endif
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/vector_proxy.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>

#if defined(_GLUCAT_USE_BINDINGS)
# include <boost/numeric/bindings/lapack/driver/gees.hpp>
# include <boost/numeric/bindings/ublas.hpp>
#endif

#if defined(_GLUCAT_USE_BLAZE)
#include <blaze/Math.h>
#include <blaze/math/DynamicMatrix.h>
#include <blaze/math/DynamicVector.h>
#endif

# if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
#  pragma GCC diagnostic pop
# endif

#include <set>
#include <vector>

namespace glucat { namespace matrix
{
  // References for algorithms:
  // [v]: C. F. van Loan and N. Pitsianis, "Approximation with Kronecker products",
  // in Linear Algebra for Large Scale and Real-Time Applications, Marc S. Moonen,
  // Gene H. Golub, and Bart L. R. Moor (eds.), 1993, pp. 293--314.

  template< typename LHS_T, typename RHS_T >
  auto
  kron(const LHS_T& lhs, const RHS_T& rhs) -> const
  RHS_T
  {
    const auto rhs_s1 = rhs.size1();
    const auto rhs_s2 = rhs.size2();
    auto result = RHS_T(lhs.size1()*rhs_s1, lhs.size2()*rhs_s2);
    result.clear();

    for (auto
        lhs_it1 = lhs.begin1();
        lhs_it1 != lhs.end1();
        ++lhs_it1)
      for (auto
          lhs_it2 = lhs_it1.begin();
          lhs_it2 != lhs_it1.end();
```

```
        ++lhs_it2)
    {
      const auto start1 = rhs_s1 * lhs_it2.index1();
      const auto start2 = rhs_s2 * lhs_it2.index2();
      const auto& lhs_val = *lhs_it2;
      for (auto
          rhs_it1 = rhs.begin1();
          rhs_it1 != rhs.end1();
          ++rhs_it1)
        for (auto
            rhs_it2 = rhs_it1.begin();
            rhs_it2 != rhs_it1.end();
            ++rhs_it2)
          result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
    }
  return result;
}

template< typename LHS_T, typename RHS_T >
auto
mono_kron(const LHS_T& lhs, const RHS_T& rhs) -> const
RHS_T
{
  const auto rhs_s1 = rhs.size1();
  const auto rhs_s2 = rhs.size2();
  const auto dim = lhs.size1()*rhs_s1;
  auto result = RHS_T(dim, dim, dim);
  result.clear();

  for (auto
      lhs_it1 = lhs.begin1();
      lhs_it1 != lhs.end1();
      ++lhs_it1)
  {
    const auto lhs_it2 = lhs_it1.begin();
    const auto start1 = rhs_s1 * lhs_it2.index1();
    const auto start2 = rhs_s2 * lhs_it2.index2();
    const auto& lhs_val = *lhs_it2;
    for (auto
        rhs_it1 = rhs.begin1();
        rhs_it1 != rhs.end1();
        ++rhs_it1)
    {
      const auto rhs_it2 = rhs_it1.begin();
      result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
    }
  }
  return result;
}

template< typename LHS_T, typename RHS_T >
void
nork_range(RHS_T& result,
           const typename LHS_T::const_iterator2 lhs_it2,
           const RHS_T& rhs,
           const typename RHS_T::size_type res_s1,
           const typename RHS_T::size_type res_s2)
{
  // Definition matches [v] Section 4, Theorem 4.1.
  const auto start1 = res_s1 * lhs_it2.index1();
  const auto start2 = res_s2 * lhs_it2.index2();
  using ublas::range;
  const auto& range1 = range(start1, start1 + res_s1);
  const auto& range2 = range(start2, start2 + res_s2);
  using matrix_range_t = ublas::matrix_range<const RHS_T>;
  const auto& rhs_range = matrix_range_t(rhs, range1, range2);
  using Scalar_T = typename RHS_T::value_type;
  const auto lhs_val = numeric_traits<Scalar_T>::to_scalar_t(*lhs_it2);
  for (auto
      rhs_it1 = rhs_range.begin1();
      rhs_it1 != rhs_range.end1();
      ++rhs_it1)
    for (auto
        rhs_it2 = rhs_it1.begin();
        rhs_it2 != rhs_it1.end();
        ++rhs_it2)
      result(rhs_it2.index1(), rhs_it2.index2()) += lhs_val * *rhs_it2;
 }

template< typename LHS_T, typename RHS_T >
auto
nork(const LHS_T& lhs, const RHS_T& rhs, const bool mono) -> const
RHS_T
{
  // nork(A, kron(A, B)) is close to B
  // Definition matches [v] Section 4, Theorem 4.1.
  const auto lhs_s1 = lhs.size1();
```

```
  const auto lhs_s2 = lhs.size2();
  const auto rhs_s1 = rhs.size1();
  const auto rhs_s2 = rhs.size2();
  const auto res_s1 = rhs_s1 / lhs_s1;
  const auto res_s2 = rhs_s2 / lhs_s2;
  using Scalar_T = typename RHS_T::value_type;
  const auto norm_frob2_lhs = norm_frob2(lhs);
  if (!mono)
  {
    using error_t = error<RHS_T>;
    if (rhs_s1 == 0)
      throw error_t("matrix", "nork: number of rows must not be 0");
    if (rhs_s2 == 0)
      throw error_t("matrix", "nork: number of cols must not be 0");
    if (res_s1 * lhs_s1 != rhs_s1)
      throw error_t("matrix", "nork: incompatible numbers of rows");
    if (res_s2 * lhs_s2 != rhs_s2)
      throw error_t("matrix", "nork: incompatible numbers of cols");
    if (norm_frob2_lhs == Scalar_T(0))
      throw error_t("matrix", "nork: LHS must not be 0");
  }
  auto result = RHS_T(res_s1, res_s2);
  result.clear();
  for (auto
      lhs_it1 = lhs.begin1();
      lhs_it1 != lhs.end1();
      ++lhs_it1)
    for (auto
        lhs_it2 = lhs_it1.begin();
        lhs_it2 != lhs_it1.end();
        ++lhs_it2)
      if (*lhs_it2  != Scalar_T(0))
        nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
  result /= norm_frob2_lhs;
  return result;
}

template< typename LHS_T, typename RHS_T >
auto
signed_perm_nork(const LHS_T& lhs, const RHS_T& rhs) -> const
RHS_T
{
  // signed_perm_nork(A, kron(A, B)) is close to B
  // Definition matches [v] Section 4, Theorem 4.1.
  const auto lhs_s1 = lhs.size1();
  const auto lhs_s2 = lhs.size2();
  const auto rhs_s1 = rhs.size1();
  const auto rhs_s2 = rhs.size2();
  const auto res_s1 = rhs_s1 / lhs_s1;
  const auto res_s2 = rhs_s2 / lhs_s2;
  using Scalar_T = typename RHS_T::value_type;
  const auto norm_frob2_lhs = Scalar_T( double(lhs_s1) );
  auto result = RHS_T(res_s1, res_s2);
  result.clear();
  for (auto
      lhs_it1 = lhs.begin1();
      lhs_it1 != lhs.end1();
      ++lhs_it1)
  {
    const auto lhs_it2 = lhs_it1.begin();
    nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
  }
  result /= norm_frob2_lhs;
  return result;
}

template< typename Matrix_T >
auto
nnz(const Matrix_T& m) -> typename Matrix_T::size_type
{
  using size_t = typename Matrix_T::size_type;
  auto result = size_t(0);
  for (auto
      it1 = m.begin1();
      it1 != m.end1();
      ++it1)
    for (auto& entry : it1)
      if (entry != 0)
        ++result;
  return result;
}

template< typename Matrix_T >
auto
isinf(const Matrix_T& m) -> bool
{
  using Scalar_T = typename Matrix_T::value_type;
```

```
  for (auto
      it1 = m.begin1();
      it1 != m.end1();
      ++it1)
    for (auto& entry : it1)
      if (numeric_traits<Scalar_T>::isInf(entry))
        return true;

  return false;
}

template< typename Matrix_T >
auto
isnan(const Matrix_T& m) -> bool
{
  using Scalar_T = typename Matrix_T::value_type;
  for (auto
      it1 = m.begin1();
      it1 != m.end1();
      ++it1)
    for (auto& entry : it1)
      if (numeric_traits<Scalar_T>::isNaN(entry))
        return true;

  return false;
}

template< typename Matrix_T >
inline
auto
unit(const typename Matrix_T::size_type dim) -> const
Matrix_T
{
  using Scalar_T = typename Matrix_T::value_type;
  return ublas::identity_matrix<Scalar_T>(dim);
}

template< typename LHS_T, typename RHS_T >
auto
mono_prod(const ublas::matrix_expression<LHS_T>& lhs,
          const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
{
  using rhs_expression_t = const RHS_T;
  using matrix_row_t = typename ublas::matrix_row<rhs_expression_t>;

  const auto dim = lhs().size1();
  // The following assumes that RHS_T is a sparse matrix type.
  auto result = RHS_T(dim, dim, dim);
  for (auto
      lhs_row = lhs().begin1();
      lhs_row != lhs().end1();
      ++lhs_row)
  {
    const auto& lhs_it = lhs_row.begin();
    if (lhs_it != lhs_row.end())
    {
      const auto& rhs_row = matrix_row_t(rhs(), lhs_it.index2());
      const auto& rhs_it = rhs_row.begin();
      if (rhs_it != rhs_row.end())
        result(lhs_it.index1(), rhs_it.index()) = (*lhs_it) * (*rhs_it);
    }
  }
  return result;
}

template< typename LHS_T, typename RHS_T >
inline
auto
sparse_prod(const ublas::matrix_expression<LHS_T>& lhs,
            const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
{
  using expression_t = typename RHS_T::expression_type;
  return ublas::sparse_prod<expression_t>(lhs(), rhs());
}

template< typename LHS_T, typename RHS_T >
inline
auto
prod(const ublas::matrix_expression<LHS_T>& lhs,
     const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
{
  const auto dim = lhs().size1();
  RHS_T result(dim, dim);
  ublas::axpy_prod(lhs, rhs, result, true);
  return result;
}
```

```
  template< typename Scalar_T, typename LHS_T, typename RHS_T >
  auto
  inner(const LHS_T& lhs, const RHS_T& rhs) -> Scalar_T
  {
    auto result = Scalar_T(0);
    for (auto
        lhs_it1 = lhs.begin1();
        lhs_it1 != lhs.end1();
        ++lhs_it1)
      for (auto
          lhs_it2 = lhs_it1.begin();
          lhs_it2 != lhs_it1.end();
          ++lhs_it2)
      {
        const auto& rhs_val = rhs(lhs_it2.index1(),lhs_it2.index2());
        if (rhs_val != Scalar_T(0))
          result += (*lhs_it2) * rhs_val;
      }
    return result / lhs.size1();
  }

  template< typename Matrix_T >
  auto
  norm_frob2(const Matrix_T& val) -> typename Matrix_T::value_type
  {
    using Scalar_T = typename Matrix_T::value_type;

    auto result = Scalar_T(0);
    for (auto
        val_it1 = val.begin1();
        val_it1 != val.end1();
        ++val_it1)
      for (auto& val_entry : val_it1)
      {
        if (numeric_traits<Scalar_T>::isNaN(val_entry))
          return numeric_traits<Scalar_T>::NaN();
        result += val_entry * val_entry;
      }
    return result;
  }

  template< typename Matrix_T >
  auto
  trace(const Matrix_T& val) -> typename Matrix_T::value_type
  {
    using Scalar_T = typename Matrix_T::value_type;

    auto result = Scalar_T(0);
    auto dim = val.size1();
    for (auto
        ndx = decltype(dim)(0);
        ndx != dim;
        ++ndx)
    {
      const Scalar_T crd = val(ndx, ndx);
      if (numeric_traits<Scalar_T>::isNaN(crd))
        return numeric_traits<Scalar_T>::NaN();
      result += crd;
    }
    return result;
  }

#if defined(_GLUCAT_USE_BINDINGS)
  template< typename Matrix_T >
  static
  auto
  to_lapack(const Matrix_T& val) -> ublas::matrix<double, ublas::column_major>
  {
    const auto s1 = val.size1();
    const auto s2 = val.size2();

    using lapack_matrix_t = typename ublas::matrix<double, ublas::column_major>;
    auto result = lapack_matrix_t(s1, s2);
    result.clear();

    using Scalar_T = typename Matrix_T::value_type;
    using traits_t = numeric_traits<Scalar_T>;

    for (auto
        val_it1 = val.begin1();
        val_it1 != val.end1();
        ++val_it1)
      for (auto
          val_it2 = val_it1.begin();
          val_it2 != val_it1.end();
          ++val_it2)
        result(val_it2.index1(), val_it2.index2()) = traits_t::to_double(*val_it2);
```

```

    return result;
  }
#endif

#if defined(_GLUCAT_USE_BLAZE)
  template< typename Matrix_T >
  static
  auto
  to_blaze(const Matrix_T& val) -> blaze::DynamicMatrix<double,blaze::rowMajor>
  {
    const auto s1 = val.size1();
    const auto s2 = val.size2();

    using blaze_matrix_t = typename blaze::DynamicMatrix<double,blaze::rowMajor>;
    auto result = blaze_matrix_t(s1, s2);

    using Scalar_T = typename Matrix_T::value_type;
    using traits_t = numeric_traits<Scalar_T>;

    for (auto
        val_it1 = val.begin1();
        val_it1 != val.end1();
        ++val_it1)
      for (auto
          val_it2 = val_it1.begin();
          val_it2 != val_it1.end();
          ++val_it2)
        result(val_it2.index1(), val_it2.index2()) = traits_t::to_double(*val_it2);

    return result;
  }

#endif

  template< typename Matrix_T >
  auto
  eigenvalues(const Matrix_T& val) -> std::vector< std::complex<double> >
  {
    using complex_t = std::complex<double>;
    using complex_vector_t = typename std::vector<complex_t>;

    const auto dim = val.size1();
    auto lambda = complex_vector_t(dim);

#if defined(_GLUCAT_USE_BINDINGS)
    namespace lapack = boost::numeric::bindings::lapack;
    using lapack_matrix_t = typename ublas::matrix<double, ublas::column_major>;

    auto T = to_lapack(val);
    auto V = T;
    using vector_t = typename ublas::vector<double>;
    auto real_lambda = vector_t(dim);
    auto imag_lambda = vector_t(dim);
    fortran_int_t sdim = 0;

    lapack::gees('N', 'N', nullptr, T, sdim, real_lambda, imag_lambda, V);

    for (auto
        k = decltype(dim)(0);
        k != dim;
        ++k)
      lambda[k] = complex_t(real_lambda[k], imag_lambda[k]);
#endif
#if defined(_GLUCAT_USE_BLAZE)
    using blaze_matrix_t = typename blaze::DynamicMatrix<double, blaze::rowMajor>;
    using complex_t = std::complex<double>;
    using blaze_complex_vector_t = blaze::DynamicVector<complex_t, blaze::columnVector>;

    auto blaze_val = to_blaze(val);
    auto blaze_lambda = blaze_complex_vector_t(dim);
    blaze::geev(blaze_val, blaze_lambda);

    for (auto
        k = decltype(dim)(0);
        k != dim;
        ++k)
      lambda[k] = blaze_lambda[k];
#endif
    return lambda;
  }

  template< typename Matrix_T >
  auto
  classify_eigenvalues(const Matrix_T& val) -> eig_genus<Matrix_T>
  {
    using Scalar_T = typename Matrix_T::value_type;
```

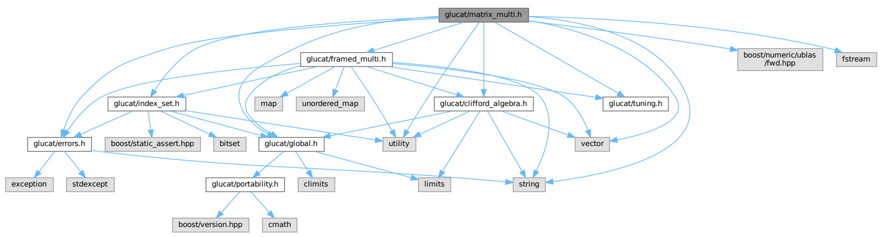
```
  eig_genus<Matrix_T> result;

  using complex_t = std::complex<double>;
  using complex_vector_t = typename std::vector<complex_t>;
  auto lambda = eigenvalues(val);

  std::set<double> arg_set;

  using vector_index_t = typename complex_vector_t::size_type;
  const auto dim = lambda.size();
  static const auto epsilon =
    std::max(std::numeric_limits<double>::epsilon(),
             numeric_traits<Scalar_T>::to_double(std::numeric_limits<Scalar_T>::epsilon()));
  static const auto zero_eig_tol = 4096.0*epsilon;

  bool neg_real_eig_found = false;
  bool imag_eig_found = false;
  bool zero_eig_found = false;

  for (auto
      k = decltype(dim)(0);
      k != dim;
      ++k)
  {
    const auto lambda_k = lambda[k];
    arg_set.insert(std::arg(lambda_k));

    const auto real_lambda_k = std::real(lambda_k);
    const auto imag_lambda_k = std::imag(lambda_k);
    const auto norm_tol = 4096.0*epsilon*std::norm(lambda_k);

    if (!neg_real_eig_found &&
        real_lambda_k < -epsilon &&
       (imag_lambda_k == 0.0 ||
        imag_lambda_k * imag_lambda_k < norm_tol))
      neg_real_eig_found = true;
    if (!imag_eig_found &&
        imag_lambda_k > epsilon &&
       (real_lambda_k == 0.0 ||
        real_lambda_k * real_lambda_k < norm_tol))
      imag_eig_found = true;
    if (!zero_eig_found &&
      std::norm(lambda_k) < zero_eig_tol)
      zero_eig_found = true;
  }

  if (zero_eig_found)
      result.m_is_singular = true;

  static const auto pi = numeric_traits<double>::pi();
  if (neg_real_eig_found)
  {
    if (imag_eig_found)
      result.m_eig_case = both_eigs;
    else
    {
      result.m_eig_case = neg_real_eigs;
      result.m_safe_arg = Scalar_T(-pi / 2.0);
    }
  }

  if (result.m_eig_case == both_eigs)
  {
    auto arg_it = arg_set.begin();
    auto first_arg = *arg_it;
    auto best_arg = first_arg;
    auto best_diff = 0.0;
    auto previous_arg = first_arg;
    for (++arg_it;
        arg_it != arg_set.end();
        ++arg_it)
    {
      const auto arg_diff = *arg_it - previous_arg;
      if (arg_diff > best_diff)
      {
        best_diff = arg_diff;
        best_arg = previous_arg;
      }
      previous_arg = *arg_it;
    }
    const auto arg_diff = first_arg + 2.0*pi - previous_arg;
    if (arg_diff > best_diff)
    {
      best_diff = arg_diff;
      best_arg = previous_arg;
    }
    result.m_safe_arg = Scalar_T(pi - (best_arg + best_diff / 2.0));
```

```
    }
    return result;
  }
} }

#endif  // _GLUCAT_MATRIX_IMP_H
```

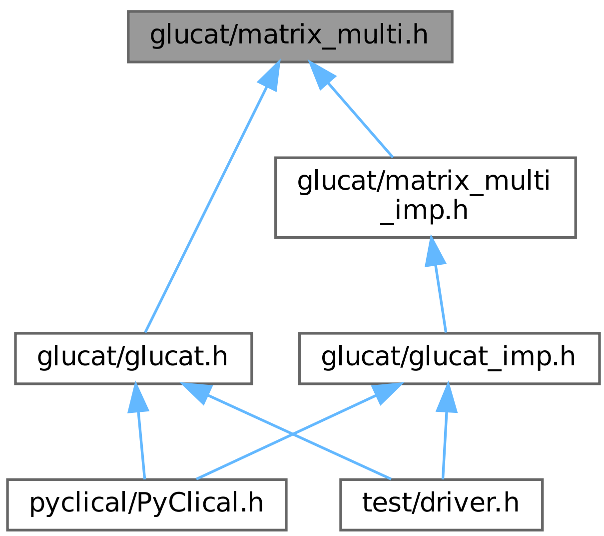
7.35 glucat/matrix_multi.h File Reference

```
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <fstream>
#include <string>
#include <utility>
#include <vector>
```

Include dependency graph for matrix_multi.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >

    A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector.
- struct std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >

    Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

Namespaces

- namespace glucat
- namespace std

Functions

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator* (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator^ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Outer product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator& (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Inner product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator% (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Left contraction.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::star (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

    Hestenes scalar product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator/ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Geometric quotient.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator| (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

    Transformation via twisted adjoint action.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator>> (std::istream &s, matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

    Read multivector from input.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator<< (std::ostream &os, const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

    Write multivector to output.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::reframe (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs, matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs_reframed, matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs_reframed) -> const index_set< LO, HI >

Find a common frame for operands of a binary operator.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::matrix_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::matrix_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::exp (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.

7.36 matrix_multi.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_MATRIX_MULTI_H
#define _GLUCAT_MATRIX_MULTI_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    matrix_multi.h : Declare a class for the matrix representation of a multivector
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"

#include <boost/numeric/ublas/fwd.hpp>

#include <fstream>
#include <string>
```

```
#include <utility>
#include <vector>

namespace glucat
{
  namespace ublas = boost::numeric::ublas;

  // Forward declarations for friends

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  class framed_multi;  // forward

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  class matrix_multi;  // forward

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator* (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator^ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator& (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator% (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> Scalar_T;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator/ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator| (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator» (std::istream& s, matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  operator« (std::ostream& os, const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  reframe (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs,    const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs,
                 matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs_reframed, matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs_reframed) -> const index_set<LO,HI>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
              const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
              const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_log( const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
              const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
              const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

```

```
template< typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<> >
class matrix_multi :
public clifford_algebra< Scalar_T, index_set<LO,HI>, matrix_multi<Scalar_T,LO,HI,Tune_P> >
{
public:
  using multivector_t = matrix_multi;
  using matrix_multi_t = multivector_t;
  using scalar_t = Scalar_T;
  using tune_p = Tune_P;
  using index_set_t = index_set<LO, HI>;
  using term_t = std::pair<const index_set_t, Scalar_T>;
  using vector_t = std::vector<Scalar_T>;
  using error_t = error<multivector_t>;
  using framed_multi_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
  template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
  friend class framed_multi;
  template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
  friend class matrix_multi;

private:
  using orientation_t = ublas::row_major;
  using basis_matrix_t = ublas::compressed_matrix<int, orientation_t>;
  using matrix_t = ublas::matrix<Scalar_T, orientation_t>;
  using matrix_index_t = typename matrix_t::size_type;

public:
  static auto classname() -> const std::string;
  ~matrix_multi() override = default;
  matrix_multi();
  template< typename Other_Scalar_T >
  matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
  template< typename Other_Scalar_T >
  matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
               const index_set_t frm, const bool prechecked = false);
  matrix_multi(const multivector_t& val,
               const index_set_t frm, const bool prechecked = false);
  matrix_multi(const index_set_t ist, const Scalar_T& crd = Scalar_T(1));
  matrix_multi(const index_set_t ist, const Scalar_T& crd,
               const index_set_t frm, const bool prechecked = false);
  matrix_multi(const Scalar_T& scr, const index_set_t frm = index_set_t());
  matrix_multi(const int scr, const index_set_t frm = index_set_t());
  matrix_multi(const vector_t& vec,
               const index_set_t frm, const bool prechecked = false);
  matrix_multi(const std::string& str);
  matrix_multi(const std::string& str,
               const index_set_t frm, const bool prechecked = false);
  matrix_multi(const char* str)
  { *this = matrix_multi(std::string(str)); };
  matrix_multi(const char* str,
               const index_set_t frm, const bool prechecked = false)
  { *this = matrix_multi(std::string(str), frm, prechecked); };
  template< typename Other_Scalar_T >
  matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
  template< typename Other_Scalar_T >
  matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
               const index_set_t frm, const bool prechecked = false);
  auto fast_matrix_multi(const index_set_t frm) const -> const matrix_multi_t;
  template< typename Other_Scalar_T >
  auto fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>;

private:
  template< typename Matrix_T >
  matrix_multi(const Matrix_T& mtx, const index_set_t frm);
  matrix_multi(const matrix_t& mtx, const index_set_t frm);
  auto basis_element(const index_set<LO,HI>& ist) const -> const basis_matrix_t;

public:
  _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS

  auto     operator= (const multivector_t& rhs) -> multivector_t&;

  static auto random(const index_set_t frm, Scalar_T fill = Scalar_T(1)) -> const matrix_multi_t;

  // Friend declarations

  friend auto
    operator* <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
  friend auto
    operator^ <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
  friend auto
    operator& <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
  friend auto
    operator% <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
  friend auto
```

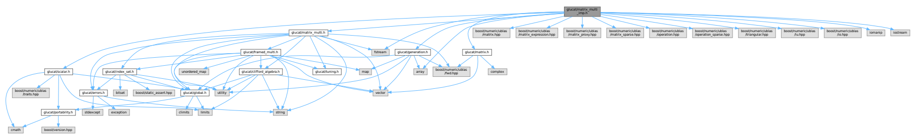
```
      star      <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> Scalar_T;
    friend auto
      operator/ <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
    friend auto
      operator| <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;

    friend auto
      operator» <>(std::istream& s, multivector_t& val) -> std::istream&;
    friend auto
      operator« <>(std::ostream& os, const multivector_t& val) -> std::ostream&;
    template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
    friend auto
      reframe  (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs,    const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs,
                  matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs_reframed, matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs_reframed) -> const index_set<Other_LO,Other_HI>;
    template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
    friend auto
      matrix_sqrt(const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
                  const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
                  const index_t level)
               -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;
    template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
    friend auto
      matrix_log(const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
                 const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
                 const index_t level)
              -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;

    auto     operator+= (const term_t& rhs) -> multivector_t&;

   private:
    // Data members

    index_set_t         m_frame;
    matrix_t            m_matrix;
  };

  // Non-members

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  exp(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;

}

namespace std
{
  template < typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P >
  struct numeric_limits< glucat::matrix_multi<Scalar_T,LO,HI,Tune_P> > :
  public numeric_limits<Scalar_T>
  { };
}
#endif  // _GLUCAT_MATRIX_MULTI_H
```

7.37 glucat/matrix_multi_imp.h File Reference

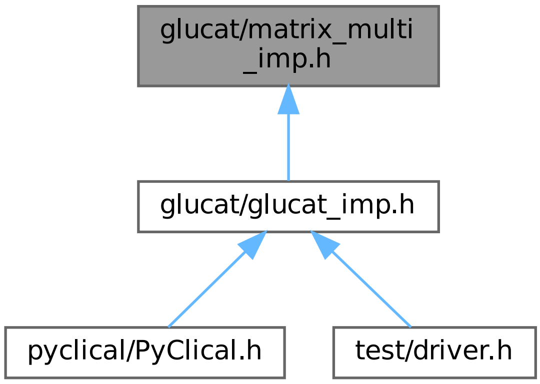
```
#include "glucat/matrix_multi.h"
#include "glucat/scalar.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>
#include <boost/numeric/ublas/triangular.hpp>
#include <boost/numeric/ublas/lu.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <fstream>
```

```
#include <iomanip>
#include <array>
#include <iostream>
```

Include dependency graph for matrix_multi_imp.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::basis_table< Scalar_T, LO, HI, Matrix_T >

    Table of basis elements used as a cache by basis_element()
- struct pade::pade_sqrt_numer< Scalar_T >

    Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)
- struct pade::pade_sqrt_denom< Scalar_T >

    Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)
- struct pade::pade_sqrt_numer< float >
- struct pade::pade_sqrt_denom< float >
- struct pade::pade_sqrt_numer< long double >
- struct pade::pade_sqrt_denom< long double >
- struct pade::pade_sqrt_numer< dd_real >
- struct pade::pade_sqrt_denom< dd_real >
- struct pade::pade_sqrt_numer< qd_real >
- struct pade::pade_sqrt_denom< qd_real >
- struct pade::pade_log_numer< Scalar_T >

    Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)
- struct pade::pade_log_denom< Scalar_T >

    Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

- struct pade::pade_log_numer< float >
- struct pade::pade_log_denom< float >
- struct pade::pade_log_numer< long double >
- struct pade::pade_log_denom< long double >
- struct pade::pade_log_numer< dd_real >
- struct pade::pade_log_denom< dd_real >
- struct pade::pade_log_numer< qd_real >
- struct pade::pade_log_denom< qd_real >

Namespaces

- namespace glucat
- namespace pade

Functions

- auto glucat::offset_level (const index_t p, const index_t q) -> index_t

  Determine the log2 dim corresponding to signature p, q.
- template<typename Matrix_Index_T , const index_t LO, const index_t HI>
  static auto glucat::folded_dim (const index_set< LO, HI > &sub) -> Matrix_Index_T

  Determine the matrix dimension of the fold of a subalegbra.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::reframe (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs, matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs_reframed, matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs_reframed) -> const index_set< LO, HI >

  Find a common frame for operands of a binary operator.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator* (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Geometric product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator^ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Outer product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator& (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Inner product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator% (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar↩
  _T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Left contraction.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::star (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T

  Hestenes scalar product.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator/ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Geometric quotient.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator| (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

Transformation via twisted adjoint action.

- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator<< (std::ostream &os, const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream &

  Write multivector to output.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::operator>> (std::istream &s, matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &

  Read multivector from input.
- template<typename Multivector_T , typename Matrix_T , typename Basis_Matrix_T >
  static auto glucat::fast (const Matrix_T &X, index_t level) -> Multivector_T

  Inverse generalized Fast Fourier Transform.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P , const size_t Size>
  static auto glucat::pade_approx (const std::array< Scalar_T, Size > &numer, const std::array< Scalar_T, Size > &denom, const matrix_multi< Scalar_T, LO, HI, Tune_P > &X) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Pade' approximation.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static void glucat::db_step (matrix_multi< Scalar_T, LO, HI, Tune_P > &M, matrix_multi< Scalar_T, LO, HI, Tune_P > &Y)

  Single step of product form of Denman-Beavers square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto glucat::db_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 4)) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Product form of Denman-Beavers square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto glucat::cr_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_Y_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 1)) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Cyclic reduction square root iteration.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::matrix_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Square root of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto glucat::pade_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Pade' approximation of log.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  static auto glucat::cascade_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Incomplete square root cascade and Pade' approximation of log.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::matrix_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Natural logarithm of multivector with specified complexifier.
- template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
  auto glucat::exp (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >

  Exponential of multivector.

7.38 matrix_multi_imp.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_MATRIX_MULTI_IMP_H
#define _GLUCAT_MATRIX_MULTI_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    matrix_multi_imp.h : Implement the matrix representation of a multivector
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/matrix_multi.h"

#include "glucat/scalar.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"

# if  defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
#  pragma GCC diagnostic push
#  pragma GCC diagnostic ignored "-Wunused-local-typedefs"
# endif
# if  defined(_GLUCAT_HAVE_BOOST_SERIALIZATION_ARRAY_WRAPPER_H)
#  include <boost/serialization/array_wrapper.hpp>
# endif
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>
#include <boost/numeric/ublas/triangular.hpp>
#include <boost/numeric/ublas/lu.hpp>
#include <boost/numeric/ublas/io.hpp>
# if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
#  pragma GCC diagnostic pop
# endif

#include <fstream>
#include <iomanip>
#include <array>
#include <iostream>

namespace glucat
{
  // References for algorithms:
  // [CHKL]:
  // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.
  // [MB]: Beatrice Meini, "The Matrix Square Root From a New Functional Perspective:
  // Theoretical Results and Computational Issues", SIAM Journal on
  // Matrix Analysis and Applications 26(2):362-376, 2004.
  // [P]: Ian R. Porteous, "Clifford algebras and the classical groups", Cambridge UP, 1995.

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  classname() -> const std::string
  { return "matrix_multi"; }

  // Reference: [P] Table 15.27, p 133
  inline
```

```
auto
offset_level(const index_t p, const index_t q) -> index_t
{
  // Offsets between the log2 of the matrix dimension for the current signature
  // and that of the real superalgebra
  static const std::array<int, 8> offset_log2_dim = {0, 1, 0, 1, 1, 2, 1, 1};
  const index_t bott = pos_mod(p-q, 8);
  return (p+q)/2 + offset_log2_dim[bott];
}

// Reference: [P] Table 15.27, p 133
template< typename Matrix_Index_T, const index_t LO, const index_t HI >
inline
static
auto
folded_dim( const index_set<LO,HI>& sub ) -> Matrix_Index_T
{ return 1 « offset_level(sub.count_pos(), sub.count_neg()); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi()
: m_frame( index_set_t() ),
  m_matrix( matrix_t( 1, 1 ) )
{ this->m_matrix.clear(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
template< typename Other_Scalar_T >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
: m_frame( val.m_frame ), m_matrix( val.m_matrix.size1(), val.m_matrix.size2() )
{
  this->m_matrix.clear();
  for (auto
      val_it1 = val.m_matrix.begin1();
      val_it1 != val.m_matrix.end1();
      ++val_it1)
    for (auto
        val_it2 = val_it1.begin();
        val_it2 != val_it1.end();
        ++val_it2)
      this->m_matrix(val_it2.index1(), val_it2.index2()) =
numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
template< typename Other_Scalar_T >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val, const index_set_t frm, const bool
prechecked)
: m_frame( frm )
{
  if (frm != val.m_frame)
    *this = multivector_t(framed_multi_t(val), frm);
  else
  {
    const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
    this->m_matrix.resize(dim, dim, false);
    this->m_matrix.clear();
    for (auto
        val_it1 = val.m_matrix.begin1();
        val_it1 != val.m_matrix.end1();
        ++val_it1)
      for (auto
          val_it2 = val_it1.begin();
          val_it2 != val_it1.end();
          ++val_it2)
        this->m_matrix(val_it2.index1(), val_it2.index2()) =
numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const multivector_t& val, const index_set_t frm, const bool prechecked)
: m_frame( frm )
{
  if (frm != val.m_frame)
    *this = multivector_t(framed_multi_t(val), frm);
  else
    this->m_matrix = val.m_matrix;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const index_set_t ist, const Scalar_T& crd)
: m_frame( ist )
{
```

```
  const auto dim = folded_dim<matrix_index_t>(this->m_frame);
  this->m_matrix.resize(dim, dim, false);
  this->m_matrix.clear();
  *this += term_t(ist, crd);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const index_set_t ist, const Scalar_T& crd, const index_set_t frm, const bool prechecked)
: m_frame( frm )
{
  if (!prechecked && (ist | frm) != frm)
    throw error_t("multivector_t(ist,crd,frm): cannot initialize with value outside of frame");
  const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
  this->m_matrix.resize(dim, dim, false);
  this->m_matrix.clear();
  *this += term_t(ist, crd);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const Scalar_T& scr, const index_set_t frm)
: m_frame( frm )
{
  const auto dim = folded_dim<matrix_index_t>(frm);
  this->m_matrix.resize(dim, dim, false);
  this->m_matrix.clear();
  *this += term_t(index_set_t(), scr);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const int scr, const index_set_t frm)
{ *this = multivector_t(Scalar_T(scr), frm); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const vector_t& vec,
             const index_set_t frm, const bool prechecked)
: m_frame( frm )
{
  if (!prechecked && index_t(vec.size()) != frm.count())
    throw error_t("multivector_t(vec,frm): cannot initialize with vector not matching frame");
  const auto dim = folded_dim<matrix_index_t>(frm);
  this->m_matrix.resize(dim, dim, false);
  this->m_matrix.clear();
  auto idx = frm.min();
  const auto frm_end = frm.max()+1;
  for (auto& crd : vec)
  {
    *this += term_t(index_set_t(idx), crd);
    for (
      ++idx;
      idx != frm_end && !frm[idx];
      ++idx)
      ;
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const std::string& str)
{ *this = framed_multi_t(str); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const std::string& str, const index_set_t frm, const bool prechecked)
{ *this = multivector_t(framed_multi_t(str), frm, prechecked); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
template< typename Other_Scalar_T >
matrix_multi<Scalar_T,LO,HI,Tune_P>::
matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
: m_frame( val.frame() )
{
  if (val.size() >= Tune_P::fast_size_threshold)
    try
    {
      *this = val.template fast_matrix_multi<Scalar_T>(this->m_frame);
      return;
    }
    catch (const glucat_error& e)
    { }
  const auto dim = folded_dim<matrix_index_t>(this->m_frame);
  this->m_matrix.resize(dim, dim, false);
  this->m_matrix.clear();
```

```
    using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
    for (auto& val_term : val)
      *this += val_term;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  template< typename Other_Scalar_T >
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& framed_val, const index_set_t frm, const bool prechecked)
  {
    const auto val = framed_val.truncated();
    const auto our_frame = val.frame() | frm;
    if (val.size() >= Tune_P::fast_size_threshold)
      try
      {
        *this = val.template fast_matrix_multi<Scalar_T>(our_frame);
        return;
      }
      catch (const glucat_error& e)
      { }
    this->m_frame = our_frame;
    const auto dim = folded_dim<matrix_index_t>(our_frame);
    this->m_matrix.resize(dim, dim, false);
    this->m_matrix.clear();

    using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
    for (auto& val_term : val)
      *this += val_term;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  template< typename Matrix_T >
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  matrix_multi(const Matrix_T& mtx, const index_set_t frm)
  : m_frame( frm ), m_matrix( mtx.size1(), mtx.size2() )
  {
    this->m_matrix.clear();

    for (auto
        mtx_it1 = mtx.begin1();
        mtx_it1 != mtx.end1();
        ++mtx_it1)
      for (auto
          mtx_it2 = mtx_it1.begin();
          mtx_it2 != mtx_it1.end();
          ++mtx_it2)
        this->m_matrix(mtx_it2.index1(), mtx_it2.index2()) = numeric_traits<Scalar_T>::to_scalar_t(*mtx_it2);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  matrix_multi(const matrix_t& mtx, const index_set_t frm)
  : m_frame( frm ), m_matrix( mtx )
  { }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  operator= (const multivector_t& rhs) -> multivector_t&
  {
    // Check for assignment to self
    if (this == &rhs)
      return *this;
    this->m_frame = rhs.m_frame;
    this->m_matrix = rhs.m_matrix;
    return *this;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  reframe (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs,    const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs,
                   matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs_reframed, matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs_reframed) -> const index_set<LO,HI>
  {
    using index_set_t = index_set<LO, HI>;
    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
    using framed_multi_t = typename multivector_t::framed_multi_t;
    // Determine the initial common frame
    index_set_t our_frame = lhs.m_frame | rhs.m_frame;
    framed_multi_t framed_lhs;
    framed_multi_t framed_rhs;
    if ((lhs.m_frame != our_frame) || (rhs.m_frame != our_frame))
```

```
  {
    // The common frame may expand as a result of the transform to framed_multi_t
    framed_lhs = framed_multi_t(lhs);
    framed_rhs = framed_multi_t(rhs);
    our_frame |= framed_lhs.frame() | framed_rhs.frame();
  }
  // Do the reframing only where necessary
  if (lhs.m_frame != our_frame)
    lhs_reframed = multivector_t(framed_lhs, our_frame, true);
  if (rhs.m_frame != our_frame)
    rhs_reframed = multivector_t(framed_rhs, our_frame, true);
  return our_frame;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator== (const multivector_t& rhs) const -> bool
{
  // Ensure that there is no aliasing
  if (this == &rhs)
    return true;

  // Operate only within a common frame
  multivector_t lhs_reframed;
  multivector_t rhs_reframed;
  const index_set_t our_frame = reframe(*this, rhs, lhs_reframed, rhs_reframed);
  const multivector_t& lhs_ref = (this->m_frame == our_frame)
    ? *this
    : lhs_reframed;
  const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
    ? rhs
    : rhs_reframed;

  return ublas::norm_inf(lhs_ref.m_matrix - rhs_ref.m_matrix) == 0;
}

// Test for equality of multivector and scalar
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator== (const Scalar_T& scr) const -> bool
{
  if (scr != Scalar_T(0))
    return *this == multivector_t(framed_multi_t(scr), this->m_frame, true);
  else if (ublas::norm_inf(this->m_matrix) != 0)
    return false;
  else
  {
    const matrix_index_t dim = this->m_matrix.size1();
    return !(dim == 1 && this->isnan());
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const Scalar_T& scr) -> multivector_t&
{ return *this += term_t(index_set_t(), scr); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const multivector_t& rhs) -> multivector_t&
{
  // Ensure that there is no aliasing
  if (this == &rhs)
    return *this *= Scalar_T(2);

  // Operate only within a common frame
  multivector_t rhs_reframed;
  const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
  const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
    ? rhs
    : rhs_reframed;

  noalias(this->m_matrix) += rhs_ref.m_matrix;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
```

```
operator-= (const Scalar_T& scr) -> multivector_t&
{ return *this += term_t(index_set_t(), -scr); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator-= (const multivector_t& rhs) -> multivector_t&
{
  // Ensure that there is no aliasing
  if (this == &rhs)
    return *this = Scalar_T(0);

  // Operate only within a common frame
  multivector_t rhs_reframed;
  const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
  const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
    ? rhs
    : rhs_reframed;

  noalias(this->m_matrix) -= rhs_ref.m_matrix;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator- () const -> const multivector_t
{ return multivector_t(-(this->m_matrix), this->m_frame); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator*= (const Scalar_T& scr) -> multivector_t&
{ // multiply coordinates of all terms by scalar

  using traits_t = numeric_traits<Scalar_T>;
  if (traits_t::isNaN_or_isInf(scr) || this->isnan())
    return *this = traits_t::NaN();
  if (scr == Scalar_T(0))
    *this = Scalar_T(0);
  else
    this->m_matrix *= scr;
  return *this;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator* (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
  using index_set_t = typename multivector_t::index_set_t;

  if (lhs.isnan() || rhs.isnan())
    return numeric_traits<Scalar_T>::NaN();

  // Operate only within a common frame
  multivector_t lhs_reframed;
  multivector_t rhs_reframed;
  const index_set_t our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
  const multivector_t& lhs_ref = (lhs.m_frame == our_frame)
    ? lhs
    : lhs_reframed;
  const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
    ? rhs
    : rhs_reframed;

  using matrix_t = typename multivector_t::matrix_t;
  using matrix_index_t = typename matrix_t::size_type;

  const matrix_index_t dim = lhs_ref.m_matrix.size1();
  multivector_t result = multivector_t(matrix_t(dim, dim), our_frame);
  result.m_matrix.clear();
  ublas::axpy_prod(lhs_ref.m_matrix, rhs_ref.m_matrix, result.m_matrix, true);
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator*= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this * rhs; }
```

```
template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator^ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
  using framed_multi_t = typename multivector_t::framed_multi_t;
  return framed_multi_t(lhs) ^ framed_multi_t(rhs);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator^= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this ^ rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator& (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
  using framed_multi_t = typename multivector_t::framed_multi_t;
  return framed_multi_t(lhs) & framed_multi_t(rhs);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator&= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this & rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator% (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
{
  using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
  using framed_multi_t = typename multivector_t::framed_multi_t;
  return framed_multi_t(lhs) % framed_multi_t(rhs);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator%= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this % rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T
{ return (lhs * rhs).scalar(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator/= (const Scalar_T& scr) -> multivector_t&
{ return *this *= Scalar_T(1)/scr; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
operator/ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;

  if (lhs.isnan() || rhs.isnan())
    return traits_t::NaN();

  if (rhs == Scalar_T(0))
    return traits_t::NaN();

  using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
  using index_set_t = typename multivector_t::index_set_t;

  // Operate only within a common frame
  multivector_t lhs_reframed;
```

```
 multivector_t rhs_reframed;
 const auto our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
 const auto& lhs_ref = (lhs.m_frame == our_frame)
   ? lhs
   : lhs_reframed;
 const auto& rhs_ref = (rhs.m_frame == our_frame)
   ? rhs
   : rhs_reframed;

 // Solve result == lhs_ref/rhs_ref <=> result*rhs_ref == lhs_ref
 // We now solve X == B/A
 // (where X == result, B == lhs_ref.m_matrix and A == rhs_ref.m_matrix)
 // X == B/A <=> X*A == B <=> AT*XT == BT
 // So, we solve AT*XT == BT

 using matrix_t = typename multivector_t::matrix_t;
 using matrix_index_t = typename matrix_t::size_type;

 const auto& AT = matrix_t(ublas::trans(rhs_ref.m_matrix));
 auto LU = AT;

 using permutation_t = ublas::permutation_matrix<matrix_index_t>;

 auto pvector = permutation_t(AT.size1());
 if (! ublas::lu_factorize(LU, pvector))
 {
   const auto& BT = matrix_t(ublas::trans(lhs_ref.m_matrix));
   auto XT = BT;
   ublas::lu_substitute(LU, pvector, XT);
   if (matrix::isnan(XT))
     return traits_t::NaN();

   // Iterative refinement.
   // Reference: Nicholas J. Higham, "Accuracy and Stability of Numerical Algorithms",
   // SIAM, 1996, ISBN 0-89871-355-2, Chapter 11
   if (Tune_P::div_max_steps > 0)
   {
     // matrix_t R = ublas::prod(AT, XT) - BT;
     auto R = matrix_t(-BT);
     ublas::axpy_prod(AT, XT, R, false);
     if (matrix::isnan(R))
       return traits_t::NaN();

     auto nr = Scalar_T(ublas::norm_inf(R));
     if ( nr != Scalar_T(0) && !traits_t::isNaN_or_isInf(nr) )
     {
       auto XTnew = XT;
       auto nrold = nr + Scalar_T(1);
       for (auto
           step = 0;
           step != Tune_P::div_max_steps &&
           nr < nrold &&
           nr != Scalar_T(0) &&
           nr == nr;
           ++step)
       {
         nrold = nr;
         if (step != 0)
           XT = XTnew;
         auto& D = R;
         ublas::lu_substitute(LU, pvector, D);
         XTnew -= D;
         // noalias(R) = ublas::prod(AT, XTnew) - BT;
         R = -BT;
         ublas::axpy_prod(AT, XTnew, R, false);
         nr = ublas::norm_inf(R);
       }
     }
   }
   return multivector_t(ublas::trans(XT), our_frame);
 }
 else
   // AT is singular. Return NaN
   return traits_t::NaN();
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator/= (const multivector_t& rhs) -> multivector_t&
{ return *this = *this / rhs; }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator| (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
```

```
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  { return rhs * lhs / rhs.involute(); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  operator|= (const multivector_t& rhs) -> multivector_t&
  { return *this = rhs * *this / rhs.involute(); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  inv() const -> const multivector_t
  { return multivector_t(Scalar_T(1), this->m_frame) / *this; }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  pow(int m) const -> const multivector_t
  { return glucat::pow(*this, m); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  outer_pow(int m) const -> const multivector_t
  {
    if (m < 0)
      throw error_t("outer_pow(m): negative exponent");
    framed_multi_t a = *this;
    return a.outer_pow(m);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  grade() const -> index_t
  { return framed_multi_t(*this).grade(); }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  frame() const -> const index_set_t
  { return this->m_frame; }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  operator[] (const index_set_t ist) const -> Scalar_T
  {
    // Use matrix inner product only if ist is in frame
    if ( (ist | this->m_frame) == this->m_frame)
      return matrix::inner<Scalar_T>(this->basis_element(ist), this->m_matrix);
    else
      return Scalar_T(0);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  operator() (index_t grade) const -> const multivector_t
  {
    if ((grade < 0) || (grade > HI-LO))
      return 0;
    else
      return (framed_multi_t(*this))(grade);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
  matrix_multi<Scalar_T,LO,HI,Tune_P>::
  scalar() const -> Scalar_T
  {
    const matrix_index_t dim = this->m_matrix.size1();
    return matrix::trace(this->m_matrix) / Scalar_T( double(dim) );
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  auto
```

```
matrix_multi<Scalar_T,LO,HI,Tune_P>::
pure() const -> const multivector_t
{ return *this - this->scalar(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
even() const -> const multivector_t
{ return framed_multi_t(*this).even(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
odd() const -> const multivector_t
{ return framed_multi_t(*this).odd(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
vector_part() const -> const vector_t
{ return this->vector_part(this->frame(), true); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
{
  if (!prechecked && (this->frame() | frm) != frm)
    throw error_t("vector_part(frm): value is outside of requested frame");
  vector_t result;
  // If we need to enlarge the frame we may as well use a framed_multi_t
  if (this->frame() != frm)
    return framed_multi_t(*this).vector_part(frm, true);

  const auto begin_index = frm.min();
  const auto end_index = frm.max()+1;
  for (auto
      idx = begin_index;
      idx != end_index;
      ++idx)
    if (frm[idx])
      // Frame may contain indices which do not correspond to a grade 1 term but
      // frame cannot omit any index corresponding to a grade 1 term
      result.push_back(
        matrix::inner<Scalar_T>(this->basis_element(index_set_t(idx)),
        this->m_matrix));
  return result;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
involute() const -> const multivector_t
{ return framed_multi_t(*this).involute(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
reverse() const -> const multivector_t
{ return framed_multi_t(*this).reverse(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
conj() const -> const multivector_t
{ return framed_multi_t(*this).conj(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
quad() const -> Scalar_T
{ // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
  // Arvind Raja ref: "old clical: quadfunction(p:pter):pterm in file compmod.pas"
  return framed_multi_t(*this).quad();
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
norm() const -> Scalar_T
```

```
{
  const matrix_index_t dim = this->m_matrix.size1();
  return matrix::norm_frob2(this->m_matrix) / Scalar_T( double(dim) );
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
max_abs() const -> Scalar_T
{ return framed_multi_t(*this).max_abs(); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
random(const index_set<LO,HI> frm, Scalar_T fill) -> const multivector_t
{
  return framed_multi<Scalar_T,LO,HI,Tune_P>::random(frm, fill);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
void
matrix_multi<Scalar_T,LO,HI,Tune_P>::
write(const std::string& msg) const
{ framed_multi_t(*this).write(msg); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
void
matrix_multi<Scalar_T,LO,HI,Tune_P>::
write(std::ofstream& ofile, const std::string& msg) const
{
  if (!ofile)
    throw error_t("write(ofile,msg): cannot write to output file");
  framed_multi_t(*this).write(ofile, msg);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator« (std::ostream& os, const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
{
  os « typename matrix_multi<Scalar_T,LO,HI,Tune_P>::framed_multi_t(val);
  return os;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
operator» (std::istream& s, matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&
{ // Input looks like 1.0-2.0{1,2}+3.2{3,4}
  framed_multi<Scalar_T,LO,HI,Tune_P> local;
  s » local;
  // If s.bad() then we have a corrupt input
  // otherwise we are fine and can copy the resulting matrix_multi
  if (!s.bad())
    val = local;
  return s;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
isinf() const -> bool
{
  if (std::numeric_limits<Scalar_T>::has_infinity)
    return matrix::isinf(this->m_matrix);
  else
    return false;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
isnan() const -> bool
{
  if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
    return matrix::isnan(this->m_matrix);
  else
    return false;
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
```

```
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
truncated(const Scalar_T& limit) const -> const multivector_t
{ return framed_multi_t(*this).truncated(limit); }

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
operator+= (const term_t& term) -> multivector_t&
{
  if (term.second != Scalar_T(0))
    this->m_matrix.plus_assign(matrix_t(this->basis_element(term.first)) * term.second);
  return *this;
}

template< typename Multivector_T, typename Matrix_T, typename Basis_Matrix_T >
static
auto
fast(const Matrix_T& X, index_t level) -> Multivector_T
{
  using framed_multi_t = Multivector_T;

  using index_set_t = typename framed_multi_t::index_set_t;
  using Scalar_T = typename framed_multi_t::scalar_t;
  using matrix_t = Matrix_T;
  using basis_matrix_t = Basis_Matrix_T;
  using basis_scalar_t = typename basis_matrix_t::value_type;
  using traits_t = numeric_traits<Scalar_T>;

  if (level == 0)
    return framed_multi_t(traits_t::to_scalar_t(X(0,0)));

  if (ublas::norm_inf(X) == 0)
    return Scalar_T(0);

  const basis_matrix_t&  I = matrix::unit<basis_matrix_t>(2);
  basis_matrix_t J(2,2,2);
  J.clear();
  J(0,1)  = basis_scalar_t(-1);
  J(1,0)  = basis_scalar_t( 1);
  basis_matrix_t K = J;
  K(0,1)  = basis_scalar_t( 1);
  basis_matrix_t JK = I;
  JK(0,0) = basis_scalar_t(-1);

  using matrix::signed_perm_nork;
  const index_set_t ist_mn   = index_set_t(-level);
  const index_set_t ist_pn   = index_set_t(level);
  const index_set_t ist_mnpn = ist_mn | ist_pn;
  if (level == 1)
  {
    using term_t = typename framed_multi_t::term_t;
    const Scalar_T i_x  = traits_t::to_scalar_t(signed_perm_nork(I, X)(0, 0));
    const Scalar_T j_x  = traits_t::to_scalar_t(signed_perm_nork(J, X)(0, 0));
    const Scalar_T k_x  = traits_t::to_scalar_t(signed_perm_nork(K, X)(0, 0));
    const Scalar_T jk_x = traits_t::to_scalar_t(signed_perm_nork(JK,X)(0, 0));
    framed_multi_t
           result  = i_x;
           result += term_t(ist_mn,   j_x);  // j_x *  mn;
           result += term_t(ist_pn,   k_x);  // k_x *  pn;
    return result += term_t(ist_mnpn, jk_x); // jk_x * mnpn;
  }
  else
  {
    const framed_multi_t& mn   = framed_multi_t(ist_mn);
    const framed_multi_t& pn   = framed_multi_t(ist_pn);
    const framed_multi_t& mnpn = framed_multi_t(ist_mnpn);
    const framed_multi_t& i_x  = fast<framed_multi_t, matrix_t, basis_matrix_t>
                                      (signed_perm_nork(I, X), level-1);
    const framed_multi_t& j_x  = fast<framed_multi_t, matrix_t, basis_matrix_t>
                                      (signed_perm_nork(J, X), level-1);
    const framed_multi_t& k_x  = fast<framed_multi_t, matrix_t, basis_matrix_t>
                                      (signed_perm_nork(K, X), level-1);
    const framed_multi_t& jk_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
                                      (signed_perm_nork(JK,X), level-1);
    framed_multi_t
           result  =  i_x.even() - jk_x.odd();
           result += (j_x.even() - k_x.odd()) * mn;
           result += (k_x.even() - j_x.odd()) * pn;
    return result += (jk_x.even() - i_x.odd()) * mnpn;
  }
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
inline
auto
```

```
matrix_multi<Scalar_T,LO,HI,Tune_P>::
fast_matrix_multi(const index_set_t frm) const -> const multivector_t
{
  if (this->m_frame == frm)
    return *this;
  else
    return (this->template fast_framed_multi<Scalar_T>()).template fast_matrix_multi<Scalar_T>(frm);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
template <typename Other_Scalar_T>
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>
{
  // Determine the amount of off-centering needed
  index_t p = this->m_frame.count_pos();
  index_t q = this->m_frame.count_neg();

  const index_t bott = pos_mod(p-q, 8);
  p += std::max(gen::offset_to_super[bott],index_t(0));
  q -= std::min(gen::offset_to_super[bott],index_t(0));

  const index_t orig_p = p;
  const index_t orig_q = q;
  while (p-q > 4)
    { p -= 4; q += 4; }
  while (p-q < -3)
    { p += 4; q -= 4; }
  if (p-q > 1)
  {
    index_t old_p = p;
    p = q+1;
    q = old_p-1;
  }
  const index_t level = (p+q)/2;

  // Do the inverse fast transform
  using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
  framed_multi_t val = fast<framed_multi_t, matrix_t, basis_matrix_t>(this->m_matrix, level);

  // Off-centre val
  switch (pos_mod(orig_p-orig_q, 8))
  {
  case 2:
  case 3:
  case 4:
    val.centre_qp1_pm1(p, q);
    break;
  default:
    break;
  }
  if (orig_p-orig_q > 4)
    while (p != orig_p)
      val.centre_pp4_qm4(p, q);
  if (orig_p-orig_q < -3)
    while (p != orig_p)
      val.centre_pm4_qp4(p, q);

  // Return unfolded val
  return val.unfold(this->m_frame);
}

template< typename Scalar_T, const index_t LO, const index_t HI, typename Matrix_T >
class basis_table :
public std::map< std::pair< const index_set<LO,HI>, const index_set<LO,HI> >,
                 Matrix_T* >
{
public:
  static auto basis() -> basis_table& { static basis_table b; return b;}
private:
  friend class friend_for_private_destructor;
  // Enforce singleton
  // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
  basis_table() = default;
  ~basis_table() = default;
public:
  basis_table(const basis_table&) = delete;
  auto operator= (const basis_table&) -> basis_table& = delete;
};

template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
auto
matrix_multi<Scalar_T,LO,HI,Tune_P>::
basis_element(const index_set_t& ist) const -> const basis_matrix_t
{
  using index_set_pair_t = std::pair<const index_set_t, const index_set_t>;
```

```
    const auto& unfolded_pair = index_set_pair_t(ist, this->m_frame);

    using basis_table_t = basis_table<Scalar_T, LO, HI, basis_matrix_t>;
    auto& basis_cache = basis_table_t::basis();

    const auto frame_count = this->m_frame.count();
    const auto use_cache = frame_count <= index_t(Tune_P::basis_max_count);

    if (use_cache)
    {
      const auto basis_it = basis_cache.find(unfolded_pair);
      if (basis_it != basis_cache.end())
        return *(basis_it->second);
    }
    const auto folded_set = ist.fold(this->m_frame);
    const auto folded_frame = this->m_frame.fold();
    const auto& folded_pair = index_set_pair_t(folded_set, folded_frame);
    using basis_pair_t = std::pair<const index_set_pair_t, basis_matrix_t *>;
    if (use_cache)
    {
      const auto basis_it = basis_cache.find(folded_pair);
      if (basis_it != basis_cache.end())
      {
        auto* result_ptr = basis_it->second;
        basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
        return *result_ptr;
      }
    }
    const auto folded_max = folded_frame.max();
    const auto folded_min = folded_frame.min();
    const auto p = std::max(folded_max,           index_t(0));
    const auto q = std::max(index_t(-folded_min), index_t(0));
    const auto* e = (gen::generator_table<basis_matrix_t>::generator())(p, q);
    const auto dim = matrix_index_t(1) << offset_level(p, q);
    auto result = matrix::unit<basis_matrix_t>(dim);
    for (auto
        k = folded_min;
        k <= folded_max;
        ++k)
      if (folded_set[k])
        result = matrix::mono_prod(result, e[k]);
    if (use_cache)
    {
      auto* result_ptr = new basis_matrix_t(result);
      basis_cache.insert(basis_pair_t(folded_pair, result_ptr));
      basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
    }
    return result;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P, const size_t Size
>
  inline
  static
  auto
  pade_approx(
    const std::array<Scalar_T, Size>& numer,
    const std::array<Scalar_T, Size>& denom,
    const matrix_multi<Scalar_T,LO,HI,Tune_P>& X) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Pade' approximation
    // Reference: [GW], Section 4.3, pp318-322
    // Reference: [GL], Section 11.3, p572-576.

    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
    using traits_t = numeric_traits<Scalar_T>;

    if (X.isnan())
      return traits_t::NaN();

    // Array size is assumed to be even
    const auto nbr_even_powers = Size/2 - 1;

    // Create an array of even powers
    auto XX = std::vector<multivector_t>(nbr_even_powers);
    XX[0] = X * X;
    XX[1] = XX[0] * XX[0];
    for (auto
      k = size_t(2);
      k != nbr_even_powers;
      ++k)
      XX[k] = XX[k-2] * XX[1];

    // Calculate numerator N and denominator D
    auto N = multivector_t(numer[1]);
    for (auto
        k = size_t(0);
```

```
        k != nbr_even_powers;
        ++k)
      N += XX[k] * numer[2*k + 3];
    N *= X;
    N += numer[0];
    for (auto
        k = size_t(0);
        k != nbr_even_powers;
        ++k)
      N += XX[k] * numer[2*k + 2];
    auto D = multivector_t(denom[1]);
    for (auto
        k = size_t(0);
        k != nbr_even_powers;
        ++k)
      D += XX[k] * denom[2*k + 3];
    D *= X;
    D += denom[0];
    for (auto
        k = size_t(0);
        k != nbr_even_powers;
        ++k)
      D += XX[k] * denom[2*k + 2];
    return N / D;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  inline
  static
  void
  db_step(matrix_multi<Scalar_T,LO,HI,Tune_P>& M, matrix_multi<Scalar_T,LO,HI,Tune_P>& Y)
  {
    // Reference: [CHKL]
    const auto& invM = inv(M);
    M = ((M + invM)/Scalar_T(2) + Scalar_T(1)) / Scalar_T(2);
    Y *= (invM + Scalar_T(1)) / Scalar_T(2);
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  static
  auto
  db_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
          Scalar_T norm_tol=std::pow(std::numeric_limits<Scalar_T>::epsilon(), 4)) -> const
matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [CHKL]
    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;

    if (val == Scalar_T(0))
      return val;

    static const auto sqrt_max_steps = Tune_P::db_sqrt_max_steps;
    auto M = val;
    auto Y = val;

    for (auto
        step = 0;
        step != sqrt_max_steps && norm(M - Scalar_T(1)) > norm_tol;
        ++step)
    {
      if (Y.isnan())
        return numeric_traits<Scalar_T>::NaN();
      db_step(M, Y);
    }
    return Y;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  static
  auto
  cr_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
          Scalar_T norm_Y_tol=std::pow(std::numeric_limits<Scalar_T>::epsilon(), 1)) -> const
matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [MB]
    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;

    if (val == Scalar_T(0))
      return val;

    static const auto sqrt_max_steps = Tune_P::cr_sqrt_max_steps;
    auto Z = Scalar_T(2) * (Scalar_T(1) + val);
    auto Y = Scalar_T(1) - val;
    using traits_t = numeric_traits<Scalar_T>;
    auto norm_Y = norm(Y);
    for (auto
        step = 0;
```

```
        step != sqrt_max_steps && norm_Y > norm_Y_tol;
        ++step)
    {
      const auto old_norm_Y = norm_Y;
      Y = (-Y / Z) * Y;
      norm_Y = norm(Y);
      if (Y.isnan() || (norm_Y > old_norm_Y * Scalar_T(2)))
        return numeric_traits<Scalar_T>::NaN();

      Z += Y * Scalar_T(2);
    }
    return Z / Scalar_T(4);
  }
}

namespace pade {
  // Reference: [Z], Pade1
  template< typename Scalar_T >
  struct pade_sqrt_numer
  {
    using array = std::array<Scalar_T, 14>;
    static const array numer;
  };
  template< typename Scalar_T >
  const typename pade_sqrt_numer<Scalar_T>::array pade_sqrt_numer<Scalar_T>::numer =
  {
        1.0,              27.0/4.0,         81.0/4.0,       2277.0/64.0,
    10395.0/256.0,      32319.0/1024.0,    8721.0/512.0,    26163.0/4096.0,
    53703.0/32768.0,    36465.0/131072.0,  3861.0/131072.0,  7371.0/4194304.0,
      819.0/16777216.0,    27.0/67108864.0
  };

  // Reference: [Z], Pade1
  template< typename Scalar_T >
  struct pade_sqrt_denom
  {
    using array = std::array<Scalar_T, 14>;
    static const array denom;
  };
  template< typename Scalar_T >
  const typename pade_sqrt_denom<Scalar_T>::array pade_sqrt_denom<Scalar_T>::denom =
  {
        1.0,              25.0/4.0,         69.0/4.0,       1771.0/64.0,
     7315.0/256.0,      20349.0/1024.0,    4845.0/512.0,    12597.0/4096.0,
    21879.0/32768.0,    12155.0/131072.0,  1001.0/131072.0,  1365.0/4194304.0,
       91.0/16777216.0,     1.0/67108864.0
  };

  template< >
  struct pade_sqrt_numer<float>
  {
    using array = std::array<float, 10>;
    static const array numer;
  };
  const typename pade_sqrt_numer<float>::array pade_sqrt_numer<float>::numer =
  {
       1.0,            19.0/4.0,        19.0/2.0,      665.0/64.0,
    1729.0/256.0,    2717.0/1024.0,    627.0/1024.0,   627.0/8192.0,
     285.0/65536.0,    19.0/262144.0
  };
  template< >
  struct pade_sqrt_denom<float>
  {
    using array = std::array<float, 10>;
    static const array denom;
  };
  const typename pade_sqrt_denom<float>::array pade_sqrt_denom<float>::denom =
  {
       1.0,            17.0/4.0,        15.0/2.0,      455.0/64.0,
    1001.0/256.0,    1287.0/1024.0,    231.0/1024.0,   165.0/8192.0,
      45.0/65536,       1.0/262144.0
  };

  template< >
  struct pade_sqrt_numer<long double>
  {
    using array = std::array<long double, 18>;
    static const array numer;
  };
  const typename pade_sqrt_numer<long double>::array pade_sqrt_numer<long double>::numer =
  {
        1.0L,                 35.0L/4.0L,             35.0L,              5425.0L/64.0L,
    35525.0L/256.0L,      166257.0L/1024.0L,      143325.0L/1024.0L,     740025.0L/8192.0L,
  2877875.0L/65536.0L,    4206125.0L/262144.0L,    572033.0L/131072.0L,  1820105.0L/2097152.0L,
  1028755.0L/8388608.0L,    395675.0L/33554432.0L,    24225.0L/33554432.0L,    6783.0L/268435456.0L,
     1785.0L/4294967296.0L,     35.0L/17179869184.0L
  };
```

```
  template< >
  struct pade_sqrt_denom<long double>
  {
    using array = std::array<long double, 18>;
    static const array denom;
  };
  const typename pade_sqrt_denom<long double>::array pade_sqrt_denom<long double>::denom =
  {
        1.0L,                   33.0L/4.0L,             31.0L,                4495.0L/64.0L,
    27405.0L/256.0L,        118755.0L/1024.0L,       94185.0L/1024.0L,     444015.0L/8192.0L,
  1562275.0L/65536.0L,     2042975.0L/262144.0L,    245157.0L/131072.0L,   676039.0L/2097152.0L,
   323323.0L/8388608.0L,    101745.0L/33554432.0L,    4845.0L/33554432.0L,    969.0L/268435456.0L,
      153.0L/4294967296.0L,      1.0L/17179869184.0L
  };

#if defined(_GLUCAT_USE_QD)
  template< >
  struct pade_sqrt_numer<dd_real>
  {
    using array = std::array<dd_real, 22>;
    static const array numer;
  };
  const typename pade_sqrt_numer<dd_real>::array pade_sqrt_numer<dd_real>::numer =
  {
          dd_real("1"),                            dd_real("43")/dd_real("4"),
        dd_real("215")/dd_real("4"),             dd_real("10621")/dd_real("64"),
      dd_real("90687")/dd_real("256"),         dd_real("567987")/dd_real("1024"),
     dd_real("168861")/dd_real("256"),        dd_real("1246355")/dd_real("2048"),
    dd_real("7228859")/dd_real("16384"),     dd_real("16583853")/dd_real("65536"),
    dd_real("7538115")/dd_real("65536"),    dd_real("173376645")/dd_real("4194304"),
  dd_real("195747825")/dd_real("16777216"),  dd_real("171655785")/dd_real("67108864"),
   dd_real("14375115")/dd_real("33554432"),   dd_real("14375115")/dd_real("268435456"),
   dd_real("20764055")/dd_real("4294967296"),  dd_real("5167525")/dd_real("17179869184"),
     dd_real("206701")/dd_real("17179869184"),   dd_real("76153")/dd_real("274877906944"),
       dd_real("3311")/dd_real("1099511627776") ,   dd_real("43")/dd_real("4398046511104")
  };
  template< >
  struct pade_sqrt_denom<dd_real>
  {
    using array = std::array<dd_real, 22>;
    static const array denom;
  };
  const typename pade_sqrt_denom<dd_real>::array pade_sqrt_denom<dd_real>::denom =
  {
          dd_real("1"),                            dd_real("41")/dd_real("4"),
        dd_real("195")/dd_real("4"),              dd_real("9139")/dd_real("64"),
      dd_real("73815")/dd_real("256"),          dd_real("435897")/dd_real("1024"),
     dd_real("121737")/dd_real("256"),          dd_real("840565")/dd_real("2048"),
    dd_real("4539051")/dd_real("16384"),       dd_real("9641775")/dd_real("65536"),
    dd_real("4032015")/dd_real("65536"),      dd_real("84672315")/dd_real("4194304"),
   dd_real("86493225")/dd_real("16777216"),   dd_real("67863915")/dd_real("67108864"),
    dd_real("5014575")/dd_real("33554432"),    dd_real("4345965")/dd_real("268435456"),
    dd_real("5311735")/dd_real("4294967296"),  dd_real("1081575")/dd_real("17179869184"),
      dd_real("33649")/dd_real("17179869184"),    dd_real("8855")/dd_real("274877906944"),
        dd_real("231")/dd_real("1099511627776"),     dd_real("1")/dd_real("4398046511104")
  };

  template< >
  struct pade_sqrt_numer<qd_real>
  {
    using array = std::array<qd_real, 34>;
    static const array numer;
  };
  const typename pade_sqrt_numer<qd_real>::array pade_sqrt_numer<qd_real>::numer =
  {
              qd_real("1"),                                          qd_real("67")/qd_real("4"),
            qd_real("134"),                                       qd_real("43617")/qd_real("64"),
         qd_real("633485")/qd_real("256"),                     qd_real("6992857")/qd_real("1024"),
       qd_real("15246721")/qd_real("1024"),                  qd_real("215632197")/qd_real("8192"),
     qd_real("2518145487")/qd_real("65536"),
qd_real("12301285425")/qd_real("262144"),
     qd_real("6344873535")/qd_real("131072"),
qd_real("89075432355")/qd_real("2097152"),
   qd_real("267226297065")/qd_real("8388608"),
qd_real("687479618945")/qd_real("33554432"),
   qd_real("379874182975")/qd_real("33554432"),
qd_real("1443521895305")/qd_real("268435456"),
  qd_real("9425348845815")/qd_real("4294967296"),
qd_real("13195488384141")/qd_real("17179869184"),
   qd_real("987417498133")/qd_real("4294967296"),
qd_real("8055248011085")/qd_real("137438953472"),
  qd_real("6958363175533")/qd_real("549755813888"),
qd_real("5056698705201")/qd_real("2199023255552"),
   qd_real("766166470485")/qd_real("2199023255552"),
qd_real("766166470485")/qd_real("17592186044416"),
   qd_real("623623871325")/qd_real("140737488355328"),
qd_real("203123203803")/qd_real("562949953421312"),
```

```
     qd_real("6478601247")/qd_real("281474976710656"),
qd_real("5038912081")/qd_real("4503599627370496"),
      qd_real("719844583")/qd_real("18014398509481984"),
qd_real("71853815")/qd_real("72057594037927936"),
        qd_real("1165197")/qd_real("72057594037927936"),
qd_real("87703")/qd_real("576460752303423488"),
          qd_real("12529")/qd_real("18446744073709551616"),
qd_real("67")/qd_real("73786976294838206464")
  };
  template< >
  struct pade_sqrt_denom<qd_real>
  {
    using array = std::array<qd_real, 34>;
    static const array denom;
  };
  const typename pade_sqrt_denom<qd_real>::array pade_sqrt_denom<qd_real>::denom =
  {
            qd_real("1"),                                               qd_real("65")/qd_real("4"),
          qd_real("126"),                                            qd_real("39711")/qd_real("64"),
       qd_real("557845")/qd_real("256"),                           qd_real("5949147")/qd_real("1024"),
     qd_real("12515965")/qd_real("1024"),                        qd_real("170574723")/qd_real("8192"),
   qd_real("1916797311")/qd_real("65536"),
qd_real("8996462475")/qd_real("262144"),
   qd_real("4450881435")/qd_real("131072"),
qd_real("59826782925")/qd_real("2097152"),
 qd_real("171503444385")/qd_real("8388608"),
qd_real("420696483235")/qd_real("33554432"),
 qd_real("221120793075")/qd_real("33554432"),
qd_real("797168807855")/qd_real("268435456"),
qd_real("4923689695575")/qd_real("4294967296"),
qd_real("6499270398159")/qd_real("17179869184"),
 qd_real("456864812569")/qd_real("4294967296"),
qd_real("3486599885395")/qd_real("137438953472"),
qd_real("2804116503573")/qd_real("549755813888"),
qd_real("1886827875075")/qd_real("2199023255552"),
 qd_real("263012370465")/qd_real("2199023255552"),
qd_real("240141729555")/qd_real("17592186044416"),
 qd_real("176848560525")/qd_real("140737488355328"),
qd_real("51538723353")/qd_real("5629499534213312"),
   qd_real("1450433115")/qd_real("281474976710656"),
qd_real("977699359")/qd_real("4503599627370496"),
    qd_real("118183439")/qd_real("18014398509481984"),
qd_real("9652005")/qd_real("72057594037927936"),
       qd_real("121737")/qd_real("72057594037927936"),
qd_real("6545")/qd_real("576460752303423488"),
          qd_real("561")/qd_real("18446744073709551616"),
qd_real("1")/qd_real("73786976294838206464")
  };
#endif
}

namespace glucat
{
  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
              const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
              const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [GW], Section 4.3, pp318-322
    // Reference: [GL], Section 11.3, p572-576
    // Reference: [Z], Pade1

    using traits_t = numeric_traits<Scalar_T>;

    if (val.isnan())
      return traits_t::NaN();

    const auto scr_val = val.scalar();
    if (val == scr_val)
    {
      if (scr_val < Scalar_T(0))
        return i * traits_t::sqrt(-scr_val);
      else
        return traits_t::sqrt(scr_val);
    }

    // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
    const auto scale =
      (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < Scalar_T(1))
      ? scr_val
      : (scr_val < Scalar_T(0))
        ? -abs(val)
        :  abs(val);
    const auto sqrt_scale = traits_t::sqrt(traits_t::abs(scale));
    if (traits_t::isNaN_or_isInf(sqrt_scale))
      return traits_t::NaN();
```

```

    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
    auto rescale = multivector_t(sqrt_scale);
    if (scale < Scalar_T(0))
      rescale = i * sqrt_scale;

    const auto& unitval = val / scale;
    static const auto max_norm = Scalar_T(1.0/4.0);
    auto use_approx_sqrt = true;
    auto use_cr_sqrt = false;
    auto scaled_result = multivector_t();
#if defined(_GLUCAT_USE_EIGENVALUES)
    static const auto sqrt_2 = traits_t::sqrt(Scalar_T(2));
    if (level == 0)
    {
      using matrix_t = typename multivector_t::matrix_t;

      // What kind of eigenvalues does the matrix contain?
      const auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
      const index_t next_level =
        (genus.m_is_singular)
        ? level
        : level + 1;
      switch (genus.m_eig_case)
      {
      case matrix::neg_real_eigs:
        scaled_result = matrix_sqrt(-i * unitval, i, next_level) * (i + Scalar_T(1)) / sqrt_2;
        use_approx_sqrt = false;
        break;
      case matrix::both_eigs:
        {
          const auto safe_arg = genus.m_safe_arg;
          scaled_result = matrix_sqrt(exp(i*safe_arg) * unitval, i, next_level) * exp(-i*safe_arg / Scalar_T(2));
        }
        use_approx_sqrt = false;
        break;
      default:
        break;
      }
      use_cr_sqrt = genus.m_is_singular;
    }
#endif
    if (use_approx_sqrt)
    {
      scaled_result =
        (norm(unitval - Scalar_T(1)) < max_norm)
          // Pade' approximation of square root
        ? pade_approx(pade::pade_sqrt_numer<Scalar_T>::numer,
                      pade::pade_sqrt_denom<Scalar_T>::denom,
                      unitval - Scalar_T(1))
          // Product form of Denman-Beavers square root iteration
        : (use_cr_sqrt)
          ? cr_sqrt(unitval)
          : db_sqrt(unitval);
    }
    return (scaled_result.isnan() ||
        !approx_equal(pow(scaled_result, 2), unitval))
      ? traits_t::NaN()
      : scaled_result * rescale;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [GW], Section 4.3, pp318-322
    // Reference: [GL], Section 11.3, p572-576
    // Reference: [Z], Pade1

    using traits_t = numeric_traits<Scalar_T>;

    if (val.isnan())
      return traits_t::NaN();

    check_complex(val, i, prechecked);

    switch (Tune_P::function_precision)
    {
    case precision_demoted:
      {
        using demoted_scalar_t = typename traits_t::demoted::type;
        using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;

        const auto& demoted_val = demoted_multivector_t(val);
        const auto& demoted_i = demoted_multivector_t(i);
```

```

        return matrix_sqrt(demoted_val, demoted_i, 0);
      }
      break;
    case precision_promoted:
      {
        using promoted_scalar_t = typename traits_t::promoted::type;
        using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;

        const auto& promoted_val = promoted_multivector_t(val);
        const auto& promoted_i = promoted_multivector_t(i);

        return matrix_sqrt(promoted_val, promoted_i, 0);
      }
      break;
    default:
      return matrix_sqrt(val, i, 0);
    }
  }
}

namespace pade {
  // Reference: [Z], Pade1
  template< typename Scalar_T >
  struct pade_log_numer
  {
    using array = std::array<Scalar_T, 14>;
    static const array numer;
  };
  template< typename Scalar_T >
  const typename pade_log_numer<Scalar_T>::array pade_log_numer<Scalar_T>::numer =
  {
         0.0,                     1.0,                     6.0,             4741.0/300.0,
      1441.0/60.0,           107091.0/4600.0,           8638.0/575.0,    263111.0/40250.0,
    153081.0/80500.0,        395243.0/1101240.0,       28549.0/688275.0, 605453.0/228813200.0,
    785633.0/10296594000.0, 1145993.0/1873980108000.0
  };

  // Reference: [Z], Pade1
  template< typename Scalar_T >
  struct pade_log_denom
  {
    using array = std::array<Scalar_T, 14>;
    static const array denom;
  };
  template< typename Scalar_T >
  const typename pade_log_denom<Scalar_T>::array pade_log_denom<Scalar_T>::denom =
  {
         1.0,                    13.0/2.0,              468.0/25.0,        1573.0/50.0,
      1573.0/46.0,            11583.0/460.0,          10296.0/805.0,       2574.0/575.0,
     11583.0/10925.0,           143.0/874.0,            572.0/37145.0,      117.0/148580.0,
        13.0/742900.0,            1.0/10400600.0
  };

  template< >
  struct pade_log_numer<float>
  {
    using array = std::array<float, 10>;
    static const array numer;
  };
  const typename pade_log_numer<float>::array pade_log_numer<float>::numer =
  {
      0.0,            1.0,             4.0,       1337.0/204.0,
    385.0/68.0,    1879.0/680.0,     193.0/255.0,  197.0/1820.0,
    419.0/61880.0, 7129.0/61261200.0
  };
  template< >
  struct pade_log_denom<float>
  {
    using array = std::array<float, 10>;
    static const array denom;
  };
  const typename pade_log_denom<float>::array pade_log_denom<float>::denom =
  {
      1.0,            9.0/2.0,       144.0/17.0,   147.0/17.0,
    441.0/85.0,      63.0/34.0,       84.0/221.0,    9.0/221.0,
      9.0/4862.0,     1.0/48620.0
  };

  template< >
  struct pade_log_numer<long double>
  {
    using array = std::array<long double, 18>;
    static const array numer;
  };
  const typename pade_log_numer<long double>::array pade_log_numer<long double>::numer =
  {
```

```
        0.0L,                       1.0L,                          8.0L,
3835.0L/132.0L,
     8365.0L/132.0L,        11363807.0L/122760.0L,          162981.0L/1705.0L,
9036157.0L/125860.0L,
  18009875.0L/453096.0L,      44211925.0L/2718576.0L,         4149566.0L/849555.0L,
16973929.0L/16020180.0L,
    172459.0L/1068012.0L,    116317061.0L/7025382936.0L,      19679783.0L/18441630207.0L,
23763863.0L/614721006900.0L,
     50747.0L/79318839600.0L, 42142223.0L/14295951736466400.0L
  };
  template< >
  struct pade_log_denom<long double>
  {
    using array = std::array<long double, 18>;
    static const array denom;
  };
  const typename pade_log_denom<long double>::array pade_log_denom<long double>::denom =
  {
        1.0L,                      17.0L/2.0L,                   1088.0L/33.0L,
850.0L/11.0L,
     41650.0L/341.0L,          140777.0L/1023.0L,            1126216.0L/9889.0L,
63206.0L/899.0L,
    790075.0L/24273.0L,         60775.0L/5394.0L,              38896.0L/13485.0L,
21658.0L/40455.0L,
     21658.0L/310155.0L,         4165.0L/682341.0L,              680.0L/2047023.0L,
34.0L/3411705.0L,
        17.0L/129644790.0L,         1.0L/2333606220
  };
#if defined(_GLUCAT_USE_QD)
  template< >
  struct pade_log_numer<dd_real>
  {
    using array = std::array<dd_real, 22>;
    static const array numer;
  };
  const typename pade_log_numer<dd_real>::array pade_log_numer<dd_real>::numer =
  {
         dd_real("0"),                                dd_real("1"),
        dd_real("10"),                            dd_real("22781")/dd_real("492"),
    dd_real("21603")/dd_real("164"),            dd_real("5492649")/dd_real("21320"),
   dd_real("978724")/dd_real("2665"),           dd_real("4191605")/dd_real("10619"),
  dd_real("12874933")/dd_real("39442"),        dd_real("11473457")/dd_real("54612"),
   dd_real("2406734")/dd_real("22755"),        dd_real("166770367")/dd_real("4004880"),
  dd_real("30653165")/dd_real("2402928"),      dd_real("647746389")/dd_real("215195552"),
  dd_real("25346331")/dd_real("47074027"),     dd_real("278270613")/dd_real("3900419380"),
 dd_real("105689791")/dd_real("15601677520"),  dd_real("606046475")/dd_real("1379188292768"),
   dd_real("969715")/dd_real("53502994116"),    dd_real("11098301")/dd_real("26204577562592"),
   dd_real("118999")/dd_real("26204577562592"), dd_real("18858053")/dd_real("1392249205900512960")
  };
  template< >
  struct pade_log_denom<dd_real>
  {
    using array = std::array<dd_real, 22>;
    static const array denom;
  };
  const typename pade_log_denom<dd_real>::array pade_log_denom<dd_real>::denom =
  {
         dd_real("1"),                               dd_real("21")/dd_real("2"),
      dd_real("2100")/dd_real("41"),                dd_real("12635")/dd_real("82"),
    dd_real("341145")/dd_real("1066"),            dd_real("1037799")/dd_real("2132"),
  dd_real("11069856")/dd_real("19721"),           dd_real("9883800")/dd_real("19721"),
   dd_real("6918660")/dd_real("19721"),            dd_real("293930")/dd_real("1517"),
   dd_real("1410864")/dd_real("16687"),             dd_real("88179")/dd_real("3034"),
    dd_real("734825")/dd_real("94054"),            dd_real("305235")/dd_real("188108"),
    dd_real("348840")/dd_real("1363783"),           dd_real("40698")/dd_real("1363783"),
      dd_real("6783")/dd_real("2727566"),             dd_real("9975")/dd_real("70916716"),
       dd_real("266")/dd_real("53187537"),               dd_real("7")/dd_real("70916716"),
         dd_real("7")/dd_real("8155422340"),             dd_real("1")/dd_real("538257874440")
  };

  template< >
  struct pade_log_numer<qd_real>
  {
    using array = std::array<qd_real, 34>;
    static const array numer;
  };
  const typename pade_log_numer<qd_real>::array pade_log_numer<qd_real>::numer =
  {
               qd_real("0"),                                                          qd_real("1"),
               qd_real("16"),
qd_real("95201")/qd_real("780"),
            qd_real("30721")/qd_real("52"),
qd_real("7416257")/qd_real("3640"),
          qd_real("1039099")/qd_real("195"),
qd_real("6097772319")/qd_real("555100"),
       qd_real("1564058073")/qd_real("85400"),
qd_real("30404640205")/qd_real("1209264"),
```

```
          qd_real("725351278")/qd_real("25193"),
      qd_real("4092322670789")/qd_real("147429436"),
      qd_real("4559713849589")/qd_real("201040140"),
      qd_real("5049361751189")/qd_real("320023080"),
        qd_real("74979677195")/qd_real("8000577"),
      qd_real("16569850691873")/qd_real("3481514244"),
      qd_real("1065906022369")/qd_real("515779888"),
      qd_real("335956770855841")/qd_real("438412904800"),
  qd_real("1462444287585964")/qd_real("6041877844275"),
      qd_real("397242326339851")/qd_real("6122436215532"),
    qd_real("64211291334131")/qd_real("4373168725380"),
      qd_real("142322343550859")/qd_real("51080680851480"),
   qd_real("154355972958659")/qd_real("351179680853925"),
      qd_real("167483568676259")/qd_real("2937139148960100"),
     qd_real("4230788929433")/qd_real("704913395750424"),
      qd_real("197968763176019")/qd_real("3929239483719956 00"),
    qd_real("10537522306718")/qd_real("319250708052246425"),
      qd_real("236648286272519")/qd_real("1442491974750354255 00"),
   qd_real("260715545088119")/qd_real("4375558990076074573500"),
      qd_real("289596255666839")/qd_real("192874640282553367199880"),
     qd_real("8802625510547")/qd_real("361639950529787563499775"),
      qd_real("373831661521439")/qd_real("16592040930306653413369677 00"),
   qd_real("446033437968239")/qd_real("46457714604858629557435095600 0"),
      qd_real("53676090078349")/qd_real("4738686889695580214858379751200 0")
      };
  template< >
  struct pade_log_denom<qd_real>
  {
    using array = std::array<qd_real, 34>;
    static const array denom;
  };
  const typename pade_log_denom<qd_real>::array pade_log_denom<qd_real>::denom =
  {
                qd_real("1"),
      qd_real("33")/qd_real("2"),
             qd_real("8448")/qd_real("65"),
      qd_real("42284")/qd_real("65"),
           qd_real("211420")/qd_real("91"),
      qd_real("573562")/qd_real("91"),
         qd_real("32119472")/qd_real("2379"),
      qd_real("92917044")/qd_real("3965"),
        qd_real("603960786")/qd_real("17995"),
      qd_real("144626625")/qd_real("3599"),
       qd_real("2776831200")/qd_real("68381"),
      qd_real("16692542100")/qd_real("478667"),
      qd_real("12241197540")/qd_real("478667"),
      qd_real("1098569010")/qd_real("68381"),
      qd_real("31387686000")/qd_real("3624193"),
      qd_real("9939433900")/qd_real("2479711"),
      qd_real("67091178825")/qd_real("42155087"),
      qd_real("2683647153")/qd_real("4959422"),
      qd_real("19083713088")/qd_real("121505839"),
      qd_real("4708152900")/qd_real("121505839"),
        qd_real("941630580")/qd_real("116546417"),
      qd_real("88704330")/qd_real("62755763"),
         qd_real("12902448")/qd_real("62755763"),
      qd_real("1542684")/qd_real("62755763"),
          qd_real("6427850")/qd_real("2698497809"),
      qd_real("3471039")/qd_real("18889484663"),
          qd_real("8544096")/qd_real("774468871183"),
      qd_real("39556")/qd_real("79027435835"),
           qd_real("118668")/qd_real("7191496660985"),
      qd_real("10230")/qd_real("27327687311743"),
             qd_real("5456")/qd_real("1011124430534491"),
      qd_real("44")/qd_real("1011124430534491"),
               qd_real("11")/qd_real("70778710137414370"),
      qd_real("1")/qd_real("7219428434016265740")
  };
#endif
}

namespace glucat{
  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  static
  auto
  pade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
      matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [GW], Section 4.3, pp318-322
    // Reference: [CHKL]
    // Reference: [GL], Section 11.3, p572-576
    // Reference: [Z], Pade1

    using traits_t = numeric_traits<Scalar_T>;
    if (val == Scalar_T(0) || val.isnan())
      return traits_t::NaN();
    else
```

```
      return pade_approx(pade::pade_log_numer<Scalar_T>::numer,
                         pade::pade_log_denom<Scalar_T>::denom,
                         val - Scalar_T(1));
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  static
  auto
  cascade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Reference: [CHKL]
    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
    using traits_t = numeric_traits<Scalar_T>;
    if (val == Scalar_T(0) || val.isnan())
      return traits_t::NaN();

    using limits_t = std::numeric_limits<Scalar_T>;
    static const auto epsilon = limits_t::epsilon();
    static const auto max_inner_norm = traits_t::pow(epsilon, 2);
    static const auto max_outer_norm = Scalar_T(6.0/limits_t::digits);
    auto Y = val;
    auto E = multivector_t(Scalar_T(0));
    Scalar_T norm_Y_1;
    auto pow_2_outer_step = Scalar_T(1);
    auto pow_4_outer_step = Scalar_T(1);
    int outer_step;
    for (outer_step = 0, norm_Y_1 = norm(Y - Scalar_T(1));
        outer_step != Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm;
        ++outer_step,    norm_Y_1 = norm(Y - Scalar_T(1)))
    {
      if (Y == Scalar_T(0) || Y.isnan())
        return traits_t::NaN();

      // Incomplete product form of Denman-Beavers square root iteration
      auto M = Y;
      for (auto
          inner_step = 0;
          inner_step != Tune_P::log_max_inner_steps &&
            norm(M - Scalar_T(1)) * pow_4_outer_step > max_inner_norm;
          ++inner_step)
        db_step(M, Y);

      E += (M - Scalar_T(1)) * pow_2_outer_step;
      pow_2_outer_step *= Scalar_T(2);
      pow_4_outer_step *= Scalar_T(4);
    }
    if (outer_step == Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm)
      return traits_t::NaN();
    else
      return pade_log(Y) * pow_2_outer_step - E;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  matrix_log( const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
              const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
              const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Scaled incomplete square root cascade and scaled Pade' approximation of log
    // Reference: [CHKL]

    using traits_t = numeric_traits<Scalar_T>;
    if (val == Scalar_T(0) || val.isnan())
      return traits_t::NaN();

    static const auto pi = traits_t::pi();
    const auto scr_val = val.scalar();
    if (val == scr_val)
    {
      if (scr_val < Scalar_T(0))
        return i * pi + traits_t::log(-scr_val);
      else
        return traits_t::log(scr_val);
    }

    // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
    const auto max_norm = Scalar_T(1.0/9.0);
    const auto scale =
      (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < max_norm)
      ? scr_val
      : (scr_val < Scalar_T(0))
        ? -abs(val)
        :  abs(val);
    if (scale == Scalar_T(0))
      return traits_t::NaN();

```

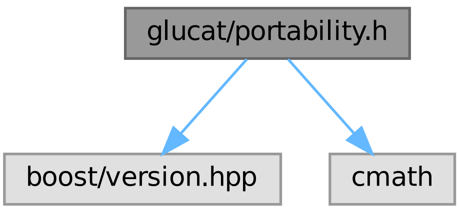
```
    using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
    const auto log_scale = traits_t::log(traits_t::abs(scale));
    auto rescale = multivector_t(log_scale);
    if (scale < Scalar_T(0))
      rescale = i * pi + log_scale;
    const auto unitval = val/scale;
    if (inv(unitval).isnan())
      return traits_t::NaN();

#if defined(_GLUCAT_USE_EIGENVALUES)
    auto scaled_result = multivector_t();
    if (level == 0)
    {
      using matrix_t = typename multivector_t::matrix_t;

      // What kind of eigenvalues does the matrix contain?
      auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
      switch (genus.m_eig_case)
      {
      case matrix::neg_real_eigs:
        scaled_result = matrix_log(-i * unitval, i, level + 1) + i * pi/Scalar_T(2);
        break;
      case matrix::both_eigs:
        {
          const Scalar_T safe_arg = genus.m_safe_arg;
          scaled_result = matrix_log(exp(i*safe_arg) * unitval, i, level + 1) - i * safe_arg;
        }
        break;
      default:
        scaled_result = cascade_log(unitval);
        break;
      }
    }
    else
      scaled_result = cascade_log(unitval);
#else
    auto scaled_result = cascade_log(unitval);
#endif
    return (scaled_result.isnan())
      ? traits_t::NaN()
      : scaled_result + rescale;
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    using traits_t = numeric_traits<Scalar_T>;

    if (val == Scalar_T(0) || val.isnan())
      return traits_t::NaN();

    check_complex(val, i, prechecked);

    switch (Tune_P::function_precision)
    {
    case precision_demoted:
      {
        using demoted_scalar_t = typename traits_t::demoted::type;
        using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;

        const auto& demoted_val = demoted_multivector_t(val);
        const auto& demoted_i = demoted_multivector_t(i);

        return matrix_log(demoted_val, demoted_i, 0);
      }
      break;
    case precision_promoted:
      {
        using promoted_scalar_t = typename traits_t::promoted::type;
        using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;

        const auto& promoted_val = promoted_multivector_t(val);
        const auto& promoted_i = promoted_multivector_t(i);

        return matrix_log(promoted_val, promoted_i, 0);
      }
      break;
    default:
      return matrix_log(val, i, 0);
    }
  }

  template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
  auto
  exp(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

```
  {
    using traits_t = numeric_traits<Scalar_T>;
    if (val.isnan())
      return traits_t::NaN();

    const auto scr_val = val.scalar();
    if (val == scr_val)
      return traits_t::exp(scr_val);

    switch (Tune_P::function_precision)
    {
    case precision_demoted:
      {
        using demoted_scalar_t = typename traits_t::demoted::type;
        using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;

        const auto& demoted_val = demoted_multivector_t(val);
        return clifford_exp(demoted_val);
      }
      break;
    case precision_promoted:
      {
        using promoted_scalar_t = typename traits_t::promoted::type;
        using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;

        const auto& promoted_val = promoted_multivector_t(val);
        return clifford_exp(promoted_val);
      }
      break;
    default:
      return clifford_exp(val);
    }
  }
}
#endif  // _GLUCAT_MATRIX_MULTI_IMP_H
```

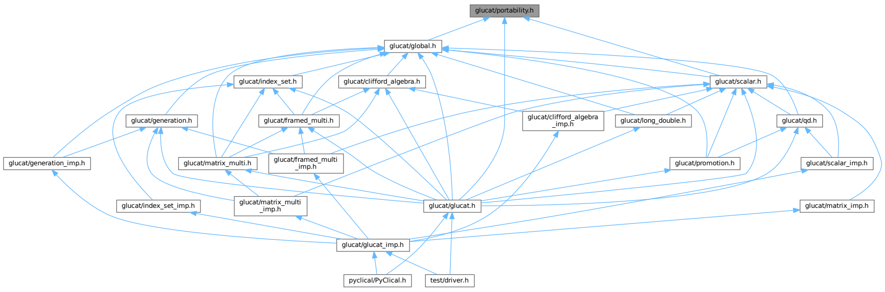
7.39 glucat/portability.h File Reference

```
#include <boost/version.hpp>
#include <cmath>
```

Include dependency graph for portability.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define _GLUCAT_ISNAN(x)
- #define _GLUCAT_ISINF(x)
- #define UBLAS_ABS abs
- #define UBLAS_SQRT sqrt

7.39.1 Macro Definition Documentation

7.39.1.1 _GLUCAT_ISINF

```
#define _GLUCAT_ISINF(
              x)
```

Value:
```
(!_GLUCAT_ISNAN(x) && _GLUCAT_ISNAN(x-x))
```

Definition at line 43 of file portability.h.

Referenced by glucat::numeric_traits< Scalar_T >::isInf().

7.39.1.2 _GLUCAT_ISNAN

```
#define _GLUCAT_ISNAN(
              x)
```

Value:
```
(x != x)
```

Definition at line 42 of file portability.h.

Referenced by glucat::numeric_traits< Scalar_T >::isNaN().

7.39.1.3 UBLAS_ABS

```
#define UBLAS_ABS abs
```

Definition at line 51 of file portability.h.

7.39.1.4 UBLAS_SQRT

```
#define UBLAS_SQRT sqrt
```

Definition at line 52 of file portability.h.

7.40 portability.h

Go to the documentation of this file.

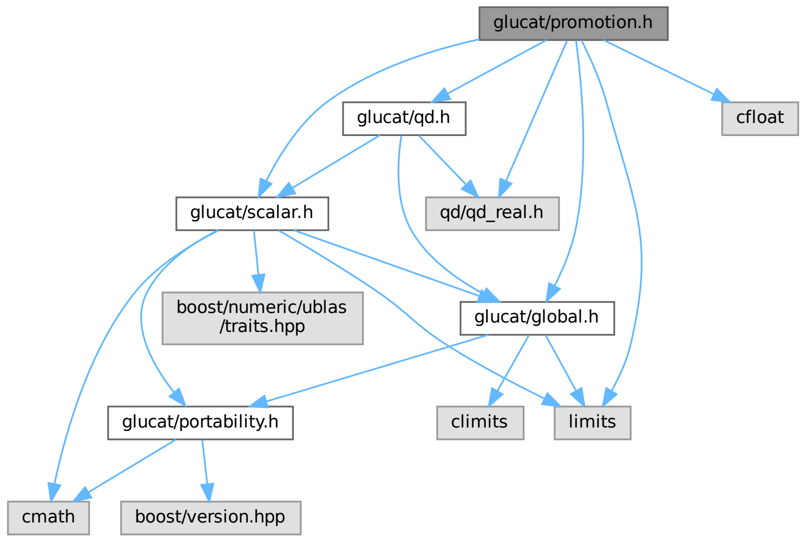
```
00001 #ifndef _GLUCAT_PORTABILITY_H
00002 #define _GLUCAT_PORTABILITY_H
00003 /***************************************************************************
00004     GluCat : Generic library of universal Clifford algebra templates
00005     portability.h : Work around non-standard compilers and libraries
00006                              -------------------
00007     begin                : Sun 2001-08-18
00008     copyright            : (C) 2001-2016 by Paul C. Leopardi
00009  ***************************************************************************
00010 
00011     This library is free software: you can redistribute it and/or modify
00012     it under the terms of the GNU Lesser General Public License as published
00013     by the Free Software Foundation, either version 3 of the License, or
00014     (at your option) any later version.
00015 
00016     This library is distributed in the hope that it will be useful,
00017     but WITHOUT ANY WARRANTY; without even the implied warranty of
00018     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019     GNU Lesser General Public License for more details.
00020 
00021     You should have received a copy of the GNU Lesser General Public License
00022     along with this library.  If not, see <http://www.gnu.org/licenses/>.
00023 
00024  ***************************************************************************
00025  This library is based on a prototype written by Arvind Raja and was
00026  licensed under the LGPL with permission of the author. See Arvind Raja,
00027  "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028  in Ablamowicz, Lounesto and Parra (eds.)
00029  "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030  ***************************************************************************
00031      See also Arvind Raja's original header comments in glucat.h
00032  ***************************************************************************/
00033 
00034 #include <boost/version.hpp>
00035 #include <cmath>
00036 
00037 // Workaround for isnan and isinf
00038 #if __cplusplus > 199711L
00039 # define _GLUCAT_ISNAN(x)  (std::isnan(x))
00040 # define _GLUCAT_ISINF(x)  (std::isinf(x))
00041 #else
00042 # define _GLUCAT_ISNAN(x)  (x != x)
00043 # define _GLUCAT_ISINF(x)  (!_GLUCAT_ISNAN(x) && _GLUCAT_ISNAN(x-x))
00044 #endif
00045 
00046 // Workaround for abs and sqrt
00047 #if BOOST_VERSION >= 103400
00048 # define UBLAS_ABS  type_abs
00049 # define UBLAS_SQRT type_sqrt
00050 #else
00051 # define UBLAS_ABS  abs
00052 # define UBLAS_SQRT sqrt
00053 #endif
00054 
00055 // Use with Cygwin gcc to obtain __WORDSIZE
00056 #if defined(HAVE_BITS_WORDSIZE_H)
00057 # include <bits/wordsize.h>
00058 #endif
00059 
00060 #endif // _GLUCAT_PORTABILITY_H
```

7.41 glucat/promotion.h File Reference

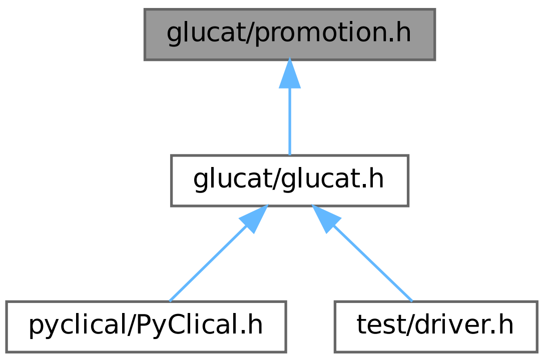
```
#include "glucat/global.h"
#include "glucat/scalar.h"
```

```
#include "glucat/qd.h"
#include <cfloat>
#include <limits>
#include <qd/qd_real.h>
```

Include dependency graph for promotion.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct glucat::numeric_traits< Scalar_T >::promoted

  Extra traits which extend numeric limits.
- struct glucat::numeric_traits< Scalar_T >::demoted

  Demoted type for long double.

Namespaces

- namespace glucat

7.42 promotion.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_PROMOTION_H
#define _GLUCAT_PROMOTION_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    promotion.h : Define promotion and demotion for specific scalar types
                             -------------------
    begin                : 2021-11-13
    copyright            : (C) 2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include "glucat/global.h"
#include "glucat/scalar.h"
#include "glucat/qd.h"

#include <cfloat>
#include <limits>

#if defined(_GLUCAT_USE_QD)
# include <qd/qd_real.h>
#endif

namespace glucat
{
  // Reference: [AA], 2.4, p. 30-31

#if !defined(_GLUCAT_USE_QD) || !defined(QD_API)

# if DBL_MANT_DIG < LDBL_MANT_DIG

  template<>
  struct
  numeric_traits<double>::
  promoted {using type = long double;};

  template<>
  struct
  numeric_traits<long double>::
  demoted {using type = double;};

# else

  template<>
  struct
  numeric_traits<double>::
  promoted {using type = double;};

  template<>
  struct
  numeric_traits<long double>::
  demoted {using type = float;};
```

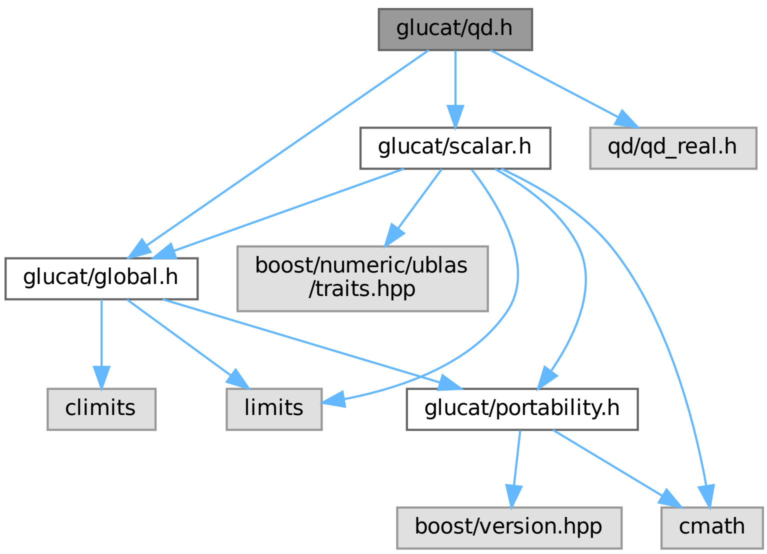
```
# endif // DBL_MANT_DIG < LDBL_MANT_DIG

  template<>
  struct
  numeric_traits<long double>::
  promoted {using type = long double;};

#else

# if (DBL_MANT_DIG < LDBL_MANT_DIG) && (LDBL_MANT_DIG < DBL_MANT_DIG*2)

  template<>
  struct
  numeric_traits<double>::
  promoted {using type = long double;};

  template<>
  struct
  numeric_traits<long double>::
  demoted {using type = double;};

  template<>
  struct
  numeric_traits<long double>::
  promoted {using type = dd_real;};

  template<>
  struct
  numeric_traits<dd_real>::
  demoted {using type = long double;};

  template<>
  struct
  numeric_traits<dd_real>::
  promoted {using type = qd_real;};

  template<>
  struct
  numeric_traits<qd_real>::
  demoted {using type = dd_real;};

# elif (LDBL_MANT_DIG < DBL_MANT_DIG*2)

  template<>
  struct
  numeric_traits<double>::
  promoted {using type = dd_real;};

  template<>
  struct
  numeric_traits<long double>::
  demoted {using type = float;};

  template<>
  struct
  numeric_traits<long double>::
  promoted {using type = dd_real;};

  template<>
  struct
  numeric_traits<dd_real>::
  demoted {using type = double;};

  template<>
  struct
  numeric_traits<dd_real>::
  promoted {using type = qd_real;};

  template<>
  struct
  numeric_traits<qd_real>::
  demoted {using type = dd_real;};

# else

  template<>
  struct
  numeric_traits<double>::
  promoted {using type = dd_real;};

  template<>
  struct
  numeric_traits<dd_real>::
  demoted {using type = double;};

  template<>
```

```
struct
numeric_traits<dd_real>::
promoted {using type = long double;};
template<>
struct
numeric_traits<long double>::
demoted {using type = dd_real;};
template<>
struct
numeric_traits<long double>::
promoted {using type = qd_real;};
template<>
struct
numeric_traits<qd_real>::
demoted {using type = long double;};
# endif // (DBL_MANT_DIG < LDBL_MANT_DIG) && (LDBL_MANT_DIG < DBL_MANT_DIG*2)
template<>
struct
numeric_traits<qd_real>::
promoted {using type = qd_real;};
#endif // !defined(_GLUCAT_USE_QD) || !defined(QD_API)
} // namespace glucat
#endif // _GLUCAT_PROMOTION_H
```

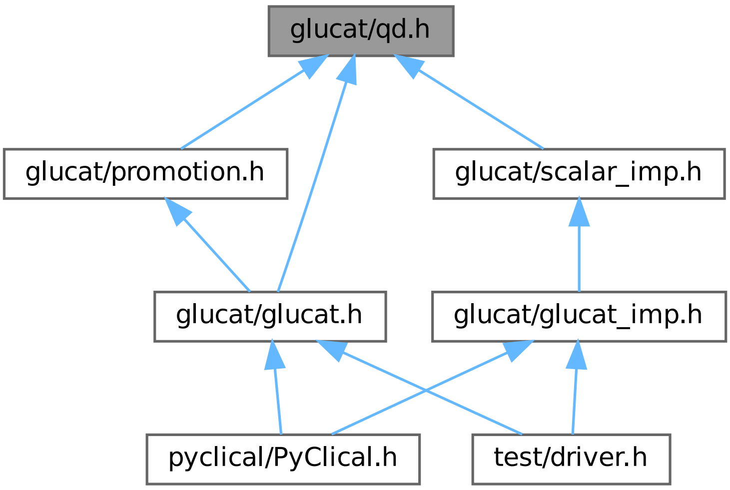
7.43 glucat/qd.h File Reference

```
#include "glucat/global.h"
#include "glucat/scalar.h"
#include <qd/qd_real.h>
```

Include dependency graph for qd.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

7.44 qd.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_QD_H
#define _GLUCAT_QD_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    qd.h : Define functions for dd_real and qd_real as scalar_t
                             -------------------
    begin                : 2010-03-23
    copyright            : (C) 2010-2016 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

```

```
#include "glucat/global.h"
#include "glucat/scalar.h"

#if defined(_GLUCAT_USE_QD)
# include <qd/qd_real.h>
#endif

namespace glucat
{
  // Reference: [AA], 2.4, p. 30-31

#if defined(_GLUCAT_USE_QD) && defined(QD_API)

# define _GLUCAT_QD_F(_T, _F) \
  template<>              \
  inline                  \
  auto                    \
  numeric_traits<_T>::    \
  _F(const _T& val) -> _T \
  { return ::_F(val); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  isNaN(const dd_real& val) -> bool
  { return val.isnan(); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  isInf(const dd_real& val) -> bool
  { return val.isinf(); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  isNaN_or_isInf(const dd_real& val) -> bool
  { return val.isnan() || val.isinf(); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  to_int(const dd_real& val) -> int
  { return ::to_int(val); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  to_double(const dd_real& val) -> double
  { return ::to_double(val); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  fmod(const dd_real& lhs, const dd_real& rhs) -> dd_real
  { return ::fmod(lhs, rhs); }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  pow(const dd_real& val, int n) -> dd_real
  {
    if (val == dd_real(0))
    {
      return
        (n < 0)
        ? NaN()
        : (n == 0)
          ? dd_real(1)
          : dd_real(0);
    }
    auto result = dd_real(1);
    auto power =
      (n < 0)
      ? dd_real(1)/val
      : val;
    for (auto
        k = std::abs(n);
        k != 0;
        k /= 2)
```

```
    {
      if (k % 2)
        result *= power;
      power *= power;
    }
    return result;
  }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  pi() -> dd_real
  { return dd_real::_pi; }

  template<>
  inline
  auto
  numeric_traits<dd_real>::
  ln_2() -> dd_real
  { return dd_real::_log2; }

  _GLUCAT_QD_F(dd_real, exp)


  _GLUCAT_QD_F(dd_real, log)


  _GLUCAT_QD_F(dd_real, cos)


  _GLUCAT_QD_F(dd_real, acos)


  _GLUCAT_QD_F(dd_real, cosh)


  _GLUCAT_QD_F(dd_real, sin)


  _GLUCAT_QD_F(dd_real, asin)


  _GLUCAT_QD_F(dd_real, sinh)


  _GLUCAT_QD_F(dd_real, tan)


  _GLUCAT_QD_F(dd_real, atan)


  _GLUCAT_QD_F(dd_real, tanh)


  template<>
  inline
  auto
  numeric_traits<qd_real>::
  isNaN(const qd_real& val) -> bool
  { return val.isnan(); }

  template<>
  inline
  auto
  numeric_traits<qd_real>::
  isInf(const qd_real& val) -> bool
  { return val.isinf(); }

  template<>
  inline
  auto
  numeric_traits<qd_real>::
  isNaN_or_isInf(const qd_real& val) -> bool
  { return val.isnan() || val.isinf(); }

  template<>
  inline
  auto
  numeric_traits<qd_real>::
  to_int(const qd_real& val) -> int
  { return ::to_int(val); }

  template<>
  inline
  auto
  numeric_traits<qd_real>::
```

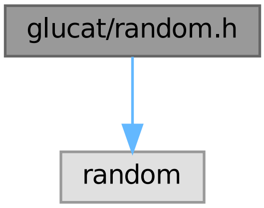
```
to_double(const qd_real& val) -> double
{ return ::to_double(val); }

template<>
inline
auto
numeric_traits<qd_real>::
fmod(const qd_real& lhs, const qd_real& rhs) -> qd_real
{ return ::fmod(lhs, rhs); }

template<>
inline
auto
numeric_traits<qd_real>::
pow(const qd_real& val, int n) -> qd_real
{
  if (val == qd_real(0))
  {
    return
      (n < 0)
      ? NaN()
      : (n == 0)
        ? qd_real(1)
        : qd_real(0);
  }
  auto result = qd_real(1);
  auto power =
    (n < 0)
    ? qd_real(1)/val
    : val;
  for (auto
      k = std::abs(n);
      k != 0;
      k /= 2)
  {
    if (k % 2)
      result *= power;
    power *= power;
  }
  return result;
}

template<>
inline
auto
numeric_traits<qd_real>::
pi() -> qd_real
{ return qd_real::_pi; }

template<>
inline
auto
numeric_traits<qd_real>::
ln_2() -> qd_real
{ return qd_real::_log2; }

_GLUCAT_QD_F(qd_real, exp)


_GLUCAT_QD_F(qd_real, log)


_GLUCAT_QD_F(qd_real, cos)


_GLUCAT_QD_F(qd_real, acos)


_GLUCAT_QD_F(qd_real, cosh)


_GLUCAT_QD_F(qd_real, sin)


_GLUCAT_QD_F(qd_real, asin)


_GLUCAT_QD_F(qd_real, sinh)


_GLUCAT_QD_F(qd_real, tan)


_GLUCAT_QD_F(qd_real, atan)


_GLUCAT_QD_F(qd_real, tanh)
```

```

#endif // !defined(_GLUCAT_USE_QD) || !defined(QD_API)

} // namespace glucat

#endif // _GLUCAT_QD_H
```

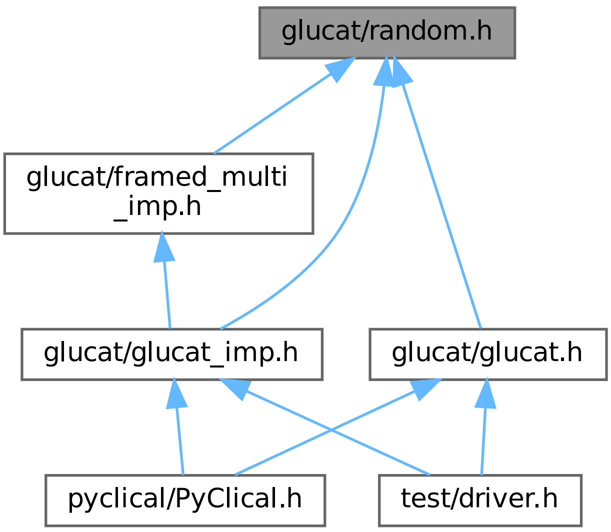
7.45 glucat/random.h File Reference

```
#include <random>
```

Include dependency graph for random.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::random_generator< Scalar_T >

    Random number generator with single instance per Scalar_T.

Namespaces

- namespace glucat

7.46 random.h

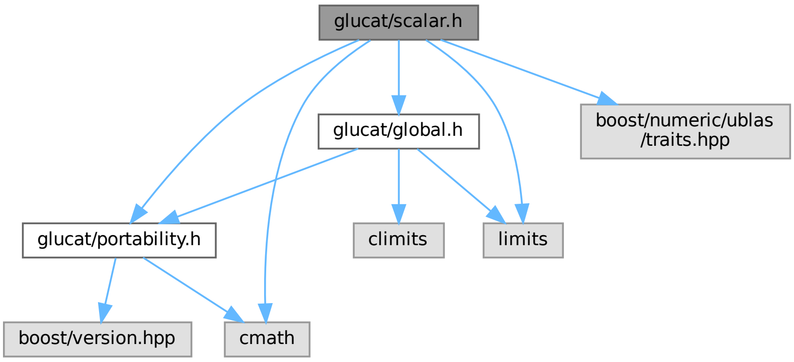
Go to the documentation of this file.

```
#ifndef _GLUCAT_RANDOM_H
#define _GLUCAT_RANDOM_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    random.h : Random number generator with single instance per Scalar_T
                             -------------------
    begin                : 2010-03-28
    copyright            : (C) 2001-2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include <random>

namespace glucat
{
  // Enforce singleton
  // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
  template< typename Scalar_T >
  class random_generator
  {
  private:
    friend class friend_for_private_destructor;
  public:
    static auto generator() -> random_generator& { static random_generator g; return g;}
    random_generator(const random_generator&) = delete;
    auto operator= (const random_generator&) -> random_generator& = delete;
  private:
    static const unsigned long seed = 19590921UL;

    std::mt19937 uint_gen;
    std::uniform_real_distribution<double> uniform_dist;
    std::normal_distribution<double> normal_dist;

    random_generator() :
    uint_gen(), uniform_dist(0.0, 1.0), normal_dist(0.0, 1.0)
    { this->uint_gen.seed(seed); }

    ~random_generator() = default;

  public:
    auto uniform() -> Scalar_T
    { return Scalar_T(this->uniform_dist(this->uint_gen)); }
    auto normal() -> Scalar_T
    { return Scalar_T(this->normal_dist(this->uint_gen)); }
  };
}

#endif // _GLUCAT_RANDOM_H
```

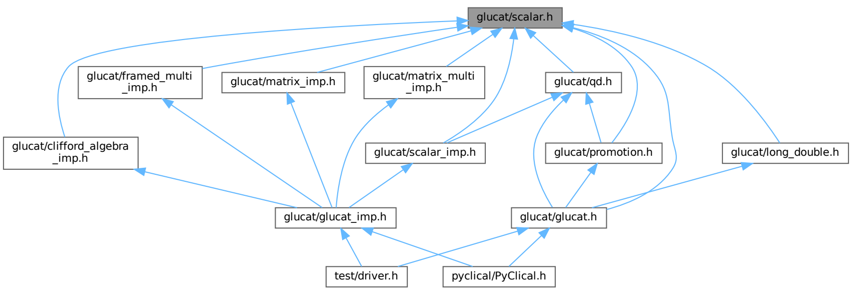
7.47 glucat/scalar.h File Reference

```
#include "glucat/portability.h"
#include "glucat/global.h"
#include <boost/numeric/ublas/traits.hpp>
#include <cmath>
#include <limits>
```

Include dependency graph for scalar.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::numeric_traits< Scalar_T >

    Extra traits which extend numeric limits.
- struct glucat::numeric_traits< Scalar_T >::promoted

    Extra traits which extend numeric limits.
- struct glucat::numeric_traits< Scalar_T >::demoted

    Demoted type for long double.

Namespaces

- namespace glucat

Functions

- template<typename Scalar_T >
  auto glucat::log2 (const Scalar_T &x) -> Scalar_T

    Log base 2 of scalar.

7.48 scalar.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_SCALAR_H
#define _GLUCAT_SCALAR_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    scalar.h : Define functions for scalar_t
                             -------------------
    begin                : 2001-12-20
    copyright            : (C) 2001-2016 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include "glucat/portability.h"
#include "glucat/global.h"

#include <boost/numeric/ublas/traits.hpp>

#include <cmath>
#include <limits>

namespace glucat
{
  // Reference: [AA], 2.4, p. 30-31
  template< typename Scalar_T >
  class numeric_traits
  {
  private:
    inline
    static
    auto
    isInf(const Scalar_T& val, bool_to_type<false>) -> bool
    { return false; }

    inline
    static
    auto
    isInf(const Scalar_T& val, bool_to_type<true>) -> bool
    { return _GLUCAT_ISINF(val); }

    inline
    static
    auto
    isNaN(const Scalar_T& val, bool_to_type<false>) -> bool
    { return false; }

    inline
    static
    auto
    isNaN(const Scalar_T& val, bool_to_type<true>) -> bool
```

```
    { return _GLUCAT_ISNAN(val); }

  public:
    inline
    static
    auto
    isInf(const Scalar_T& val) -> bool
    {
      return isInf(val,
             bool_to_type< std::numeric_limits<Scalar_T>::has_infinity >() );
    }

    inline
    static
    auto
    isNaN(const Scalar_T& val) -> bool
    {
      return isNaN(val,
             bool_to_type< std::numeric_limits<Scalar_T>::has_quiet_NaN >() );
    }

    inline
    static
    auto
    isNaN_or_isInf(const Scalar_T& val) -> bool
    {
      return isNaN(val,
             bool_to_type< std::numeric_limits<Scalar_T>::has_quiet_NaN >() )
          || isInf(val,
             bool_to_type< std::numeric_limits<Scalar_T>::has_infinity >() );
    }

    inline
    static
    auto
    NaN() -> Scalar_T
    {
      return std::numeric_limits<Scalar_T>::has_quiet_NaN
           ? std::numeric_limits<Scalar_T>::quiet_NaN()
           : Scalar_T(std::log(0.0));
    }

    inline
    static
    auto
    to_int(const Scalar_T& val) -> int
    { return static_cast<int>(val); }

    inline
    static
    auto
    to_double(const Scalar_T& val) -> double
    { return static_cast<double>(val); }

    template <typename Other_Scalar_T >
    inline
    static
    auto
    to_scalar_t(const Other_Scalar_T& val) -> Scalar_T
    { return static_cast<Scalar_T>(val); }

    struct promoted {using type = double;};

    struct demoted {using type = float;};

    inline
    static
    auto
    fmod(const Scalar_T& lhs, const Scalar_T& rhs) -> Scalar_T
    { return std::fmod(lhs, rhs); }

    inline
    static
    auto
    conj(const Scalar_T& val) -> Scalar_T
    { return val; }

    inline
    static
    auto
    real(const Scalar_T& val) -> Scalar_T
    { return val; }

    inline
    static
    auto
    imag(const Scalar_T& val) -> Scalar_T
```

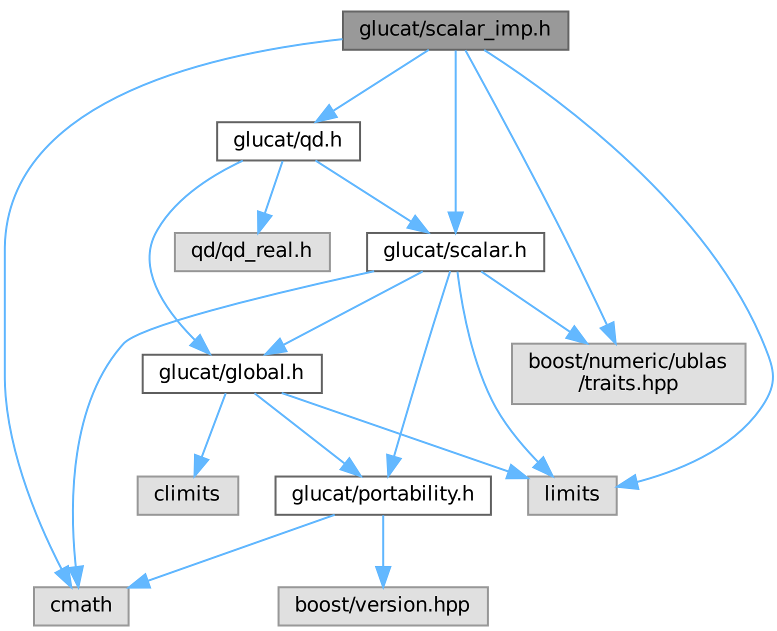
```
{ return Scalar_T(0); }

inline
static
auto
abs(const Scalar_T& val) -> Scalar_T
{ return boost::numeric::ublas::type_traits<Scalar_T>::UBLAS_ABS(val); }

inline
static
auto
pi() -> Scalar_T
{ return Scalar_T(3.14159265358979323); }

inline
static
auto
ln_2() -> Scalar_T
{ return Scalar_T(0.693147180559945309); }

inline
static
auto
pow(const Scalar_T& val, int n) -> Scalar_T
{ return std::pow(val, n); }

inline
static
auto
sqrt(const Scalar_T& val) -> Scalar_T
{ return boost::numeric::ublas::type_traits<Scalar_T>::UBLAS_SQRT(val); }

inline
static
auto
exp(const Scalar_T& val) -> Scalar_T
{ return std::exp(val); }

inline
static
auto
log(const Scalar_T& val) -> Scalar_T
{ return std::log(val); }

inline
static
auto
log2(const Scalar_T& val) -> Scalar_T
{ return log(val)/ln_2(); }

inline
static
auto
cos(const Scalar_T& val) -> Scalar_T
{ return std::cos(val); }

inline
static
auto
acos(const Scalar_T& val) -> Scalar_T
{ return std::acos(val); }

inline
static
auto
cosh(const Scalar_T& val) -> Scalar_T
{ return std::cosh(val); }

inline
static
auto
sin(const Scalar_T& val) -> Scalar_T
{ return std::sin(val); }

inline
static
auto
asin(const Scalar_T& val) -> Scalar_T
{ return std::asin(val); }

inline
static
auto
sinh(const Scalar_T& val) -> Scalar_T
{ return std::sinh(val); }

inline
```

```
    static
    auto
    tan(const Scalar_T& val) -> Scalar_T
    { return std::tan(val); }

    inline
    static
    auto
    atan(const Scalar_T& val) -> Scalar_T
    { return std::atan(val); }

    inline
    static
    auto
    tanh(const Scalar_T& val) -> Scalar_T
    { return std::tanh(val); }

  };

  template< typename Scalar_T >
  inline
  auto
  log2(const Scalar_T& x) -> Scalar_T
  { return numeric_traits<Scalar_T>::log2(x); }
}

#endif // _GLUCAT_SCALAR_H
```

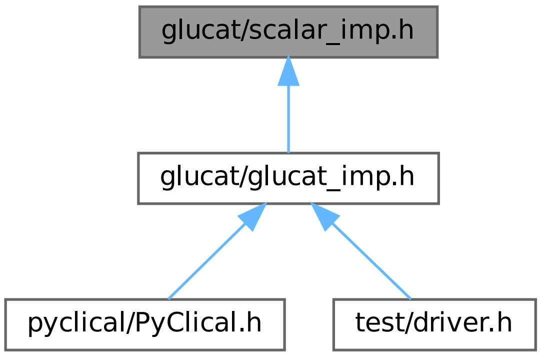
7.49 glucat/scalar_imp.h File Reference

```
#include "glucat/scalar.h"
#include "glucat/qd.h"
#include <boost/numeric/ublas/traits.hpp>
#include <cmath>
#include <limits>
```

Include dependency graph for scalar_imp.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Functions

- template<typename Scalar_T >
  auto glucat::to_promote (const Scalar_T &val) -> typename numeric_traits< Scalar_T >::promoted::type

  Cast to promote.
- template<typename Scalar_T >
  auto glucat::to_demote (const Scalar_T &val) -> typename numeric_traits< Scalar_T >::demoted::type

  Cast to demote.

7.50 scalar_imp.h

Go to the documentation of this file.

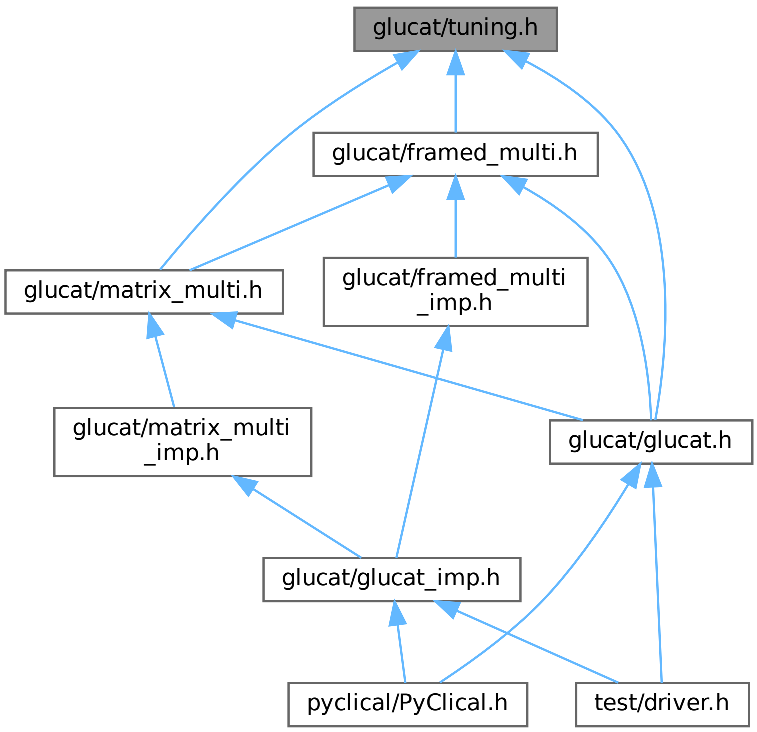
```
#ifndef _GLUCAT_SCALAR_IMP_H
#define _GLUCAT_SCALAR_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    scalar_imp.h : Define functions for scalar_t
                             -------------------
    begin                : 2001-12-20
    copyright            : (C) 2001-2014 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

```

```
 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 See also Arvind Raja's original header comments and references in glucat.h
 ***************************************************************************/

#include "glucat/scalar.h"
#include "glucat/qd.h"

#include <boost/numeric/ublas/traits.hpp>

#include <cmath>
#include <limits>

namespace glucat
{
  // Reference: [AA], 2.4, p. 30-31

  template< >
  template< typename Other_Scalar_T >
  inline
  auto
  numeric_traits<float>::
  to_scalar_t(const Other_Scalar_T& val) -> float
  { return static_cast<float>(numeric_traits<Other_Scalar_T>::to_double(val)); }

  template< >
  template< typename Other_Scalar_T >
  inline
  auto
  numeric_traits<double>::
  to_scalar_t(const Other_Scalar_T& val) -> double
  { return numeric_traits<Other_Scalar_T>::to_double(val); }

#if defined(_GLUCAT_USE_QD)
  template< >
  template< >
  inline
  auto
  numeric_traits<long double>::
  to_scalar_t(const dd_real& val) -> long double
  { return static_cast<long double>(val.x[0]) + static_cast<long double>(val.x[1]); }

  template< >
  template< >
  inline
  auto
  numeric_traits<long double>::
  to_scalar_t(const qd_real& val) -> long double
  { return static_cast<long double>(val.x[0]) + static_cast<long double>(val.x[1]); }

  template< >
  template< >
  inline
  auto
  numeric_traits<dd_real>::
  to_scalar_t(const long double& val) -> dd_real
  { return {double(val),double(val - static_cast<long double>(double(val)))}; }

  template< >
  template< >
  inline
  auto
  numeric_traits<dd_real>::
  to_scalar_t(const qd_real& val) -> dd_real
  { return {val.x[0],val.x[1]}; }

  template< >
  template< >
  inline
  auto
  numeric_traits<qd_real>::
  to_scalar_t(const long double& val) -> qd_real
  { return {double(val),double(val - static_cast<long double>(double(val))),0.0,0.0}; }

  template< >
  template< >
  inline
  auto
  numeric_traits<qd_real>::
  to_scalar_t(const dd_real& val) -> qd_real
  { return {val.x[0],val.x[1],0.0,0.0}; }
#endif
```

```

  template< typename Scalar_T >
  inline
  auto
  to_promote(const Scalar_T& val) -> typename numeric_traits<Scalar_T>::promoted::type
  {
    using promoted_scalar_t = typename numeric_traits<Scalar_T>::promoted::type;
    return numeric_traits<promoted_scalar_t>::to_scalar_t(val);
  }

  template< typename Scalar_T >
  inline
  auto
  to_demote(const Scalar_T& val) -> typename numeric_traits<Scalar_T>::demoted::type
  {
    using demoted_scalar_t = typename numeric_traits<Scalar_T>::demoted::type;
    return numeric_traits<demoted_scalar_t>::to_scalar_t(val);
  }
}

#endif // _GLUCAT_SCALAR_IMP_H
```

7.51 glucat/tuning.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- _GLUCAT_CTAssert (std::numeric_limits< unsigned int >::radix==2, CannotSetThresholds) namespace glucat

7.51.1 Function Documentation

7.51.1.1 _GLUCAT_CTAssert()

```
_GLUCAT_CTAssert (
            std::numeric_limits< unsigned int >::radix  = = 2,
            CannotSetThresholds )
```

Base class for policies

Precision policy

Tuning policy

Minimum index count needed to invoke matrix multiplication algorithm

Maximum steps of iterative refinement in division algorithm

Maximum number of steps in cyclic reduction square root iteration

Maximum number of steps in Denman-Beavers square root iteration

Maximum number of incomplete square roots in cascade log algorithm

Maximum number of steps in incomplete square root within cascade log algorithm

Maximum index count of folded frames in basis cache

Minimum map size needed to invoke generalized FFT

Minimum matrix dimension needed to invoke inverse generalized FFT

Minimum size needed for to invoke faster products algorithms

Denominator of proportion of different bits allowed in approximate equality

Extra number of different bits allowed in approximate equality

Precision used for exp, log and sqrt functions

Definition at line 35 of file tuning.h.

7.52 tuning.h

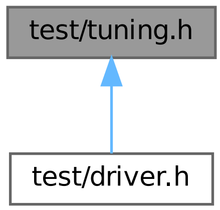
Go to the documentation of this file.

```
#ifndef GLUCAT_TUNING_H
#define GLUCAT_TUNING_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    tuning.h : Policy classes to control tuning
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja’s original header comments in glucat.h
 ***************************************************************************/

// If radix of int is not 2, we can’t easily set thresholds
_GLUCAT_CTAssert(std::numeric_limits<unsigned int>::radix == 2, CannotSetThresholds)

namespace glucat
{
  struct policy{};

  enum precision_t
  {
    precision_demoted,
    precision_same,
    precision_promoted
  };

  // Tuning policy default constants
  const unsigned int Tuning_Default_Mult_Matrix_Threshold   =       8;
  const unsigned int Tuning_Default_Div_Max_Steps           =       4;
  const unsigned int Tuning_Default_CR_Sqrt_Max_Steps       =     256;
  const unsigned int Tuning_Default_DB_Sqrt_Max_Steps       =     256;
  const unsigned int Tuning_Default_Log_Max_Outer_Steps     =     256;
  const unsigned int Tuning_Default_Log_Max_Inner_Steps     =      32;
  const unsigned int Tuning_Default_Basis_Max_Count         =      12;
  const unsigned int Tuning_Default_Fast_Size_Threshold     = 1 «  6;
  const unsigned int Tuning_Default_Inv_Fast_Dim_Threshold  = 1 «  3;
  const unsigned int Tuning_Default_Products_Size_Threshold = 1 « 22;
  const unsigned int Tuning_Default_Denom_Different_Bits    =       8;
  const unsigned int Tuning_Default_Extra_Different_Bits    =       8;
  const precision_t  Tuning_Default_Function_Precision      = precision_same;

  template
  <
    unsigned int Mult_Matrix_Threshold   = Tuning_Default_Mult_Matrix_Threshold,
    unsigned int Div_Max_Steps           = Tuning_Default_Div_Max_Steps,
    unsigned int CR_Sqrt_Max_Steps       = Tuning_Default_CR_Sqrt_Max_Steps,
    unsigned int DB_Sqrt_Max_Steps       = Tuning_Default_DB_Sqrt_Max_Steps,
    unsigned int Log_Max_Outer_Steps     = Tuning_Default_Log_Max_Outer_Steps,
    unsigned int Log_Max_Inner_Steps     = Tuning_Default_Log_Max_Inner_Steps,
    unsigned int Basis_Max_Count         = Tuning_Default_Basis_Max_Count,
    unsigned int Fast_Size_Threshold     = Tuning_Default_Fast_Size_Threshold,
    unsigned int Inv_Fast_Dim_Threshold  = Tuning_Default_Inv_Fast_Dim_Threshold,
    unsigned int Products_Size_Threshold = Tuning_Default_Products_Size_Threshold,
    unsigned int Denom_Different_Bits    = Tuning_Default_Denom_Different_Bits,
    unsigned int Extra_Different_Bits    = Tuning_Default_Extra_Different_Bits,
    precision_t  Function_Precision      = Tuning_Default_Function_Precision
  >
  struct tuning : policy
  {
    using tune_p = tuning
    <
```

```
      Mult_Matrix_Threshold,
      Div_Max_Steps,
      CR_Sqrt_Max_Steps,
      DB_Sqrt_Max_Steps,
      Log_Max_Outer_Steps,
      Log_Max_Inner_Steps,
      Basis_Max_Count,
      Fast_Size_Threshold,
      Inv_Fast_Dim_Threshold,
      Products_Size_Threshold,
      Denom_Different_Bits,
      Extra_Different_Bits,
      Function_Precision
    >;
  // Tuning for multiplication
    enum { mult_matrix_threshold = Mult_Matrix_Threshold };
  // Tuning for division
    enum { div_max_steps = Div_Max_Steps };
  // Tuning for sqrt
    enum { cr_sqrt_max_steps = CR_Sqrt_Max_Steps };
    enum { db_sqrt_max_steps = DB_Sqrt_Max_Steps };
  // Tuning for log
    enum { log_max_outer_steps = Log_Max_Outer_Steps };
    enum { log_max_inner_steps = Log_Max_Inner_Steps };
  // Tuning for basis cache
    enum { basis_max_count = Basis_Max_Count };
  // Tuning for FFT
    enum { fast_size_threshold = Fast_Size_Threshold };
    enum { inv_fast_dim_threshold = Inv_Fast_Dim_Threshold };
  // Tuning for products (other than geometric product)
    enum { products_size_threshold = Products_Size_Threshold };
  // Tuning for precision of exp, log and sqrt functions
    enum { denom_different_bits = Denom_Different_Bits };
    enum { extra_different_bits = Extra_Different_Bits };
    static const precision_t function_precision = Function_Precision;
  };

  using tuning_demoted = tuning
    <
      Tuning_Default_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Default_Basis_Max_Count,
      Tuning_Default_Fast_Size_Threshold,
      Tuning_Default_Inv_Fast_Dim_Threshold,
      Tuning_Default_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      precision_demoted
    >;

  using tuning_promoted = tuning
    <
      Tuning_Default_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Default_Basis_Max_Count,
      Tuning_Default_Fast_Size_Threshold,
      Tuning_Default_Inv_Fast_Dim_Threshold,
      Tuning_Default_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      precision_promoted
    >;
}

#endif // GLUCAT_TUNING_H
```

7.53 test/tuning.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Typedefs

- using glucat::tuning_slow
- using glucat::tuning_naive
- using glucat::tuning_fast

Variables

- const unsigned int glucat::Tuning_Int_Digits = std::numeric_limits<int>::digits
- const unsigned int glucat::Tuning_Max_Threshold = 1 << Tuning_Int_Digits
- const unsigned int glucat::Tuning_Slow_Mult_Matrix_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Slow_Basis_Max_Count = 0
- const unsigned int glucat::Tuning_Slow_Fast_Size_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Slow_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Naive_Mult_Matrix_Threshold = 0
- const unsigned int glucat::Tuning_Naive_Basis_Max_Count = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Naive_Fast_Size_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Naive_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
- const unsigned int glucat::Tuning_Fast_Mult_Matrix_Threshold = 0
- const unsigned int glucat::Tuning_Fast_Div_Max_Steps = 0
- const unsigned int glucat::Tuning_Fast_CR_Sqrt_Max_Steps = 256
- const unsigned int glucat::Tuning_Fast_DB_Sqrt_Max_Steps = 256
- const unsigned int glucat::Tuning_Fast_Log_Max_Outer_Steps = 16
- const unsigned int glucat::Tuning_Fast_Log_Max_Inner_Steps = 8
- const unsigned int glucat::Tuning_Fast_Basis_Max_Count = 1
- const unsigned int glucat::Tuning_Fast_Fast_Size_Threshold = 0
- const unsigned int glucat::Tuning_Fast_Inv_Fast_Dim_Threshold = 0
- const unsigned int glucat::Tuning_Fast_Products_Size_Threshold = 0

7.54 tuning.h

Go to the documentation of this file.

```
#ifndef GLUCAT_TEST_TUNING_H
#define GLUCAT_TEST_TUNING_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    tuning.h : Class definitions to control test tuning
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

namespace glucat
{
  const unsigned int Tuning_Int_Digits = std::numeric_limits<int>::digits;
  const unsigned int Tuning_Max_Threshold = 1 « Tuning_Int_Digits;

  // Specific tuning policy constants and tuning policies

  const unsigned int Tuning_Slow_Mult_Matrix_Threshold   = Tuning_Max_Threshold;
  const unsigned int Tuning_Slow_Basis_Max_Count         =       0;
  const unsigned int Tuning_Slow_Fast_Size_Threshold     = Tuning_Max_Threshold;
  const unsigned int Tuning_Slow_Inv_Fast_Dim_Threshold  = Tuning_Max_Threshold;
  const unsigned int Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold;

  using tuning_slow = tuning
    <
      Tuning_Slow_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Slow_Basis_Max_Count,
      Tuning_Slow_Fast_Size_Threshold,
      Tuning_Slow_Inv_Fast_Dim_Threshold,
      Tuning_Slow_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      Tuning_Default_Function_Precision
    >;

  const unsigned int Tuning_Naive_Mult_Matrix_Threshold   =       0;
  const unsigned int Tuning_Naive_Basis_Max_Count         = Tuning_Max_Threshold;
  const unsigned int Tuning_Naive_Fast_Size_Threshold     = Tuning_Max_Threshold;
  const unsigned int Tuning_Naive_Inv_Fast_Dim_Threshold  = Tuning_Max_Threshold;

  using tuning_naive = tuning
    <
      Tuning_Naive_Mult_Matrix_Threshold,
      Tuning_Default_Div_Max_Steps,
      Tuning_Default_CR_Sqrt_Max_Steps,
      Tuning_Default_DB_Sqrt_Max_Steps,
      Tuning_Default_Log_Max_Outer_Steps,
      Tuning_Default_Log_Max_Inner_Steps,
      Tuning_Naive_Basis_Max_Count,
      Tuning_Naive_Fast_Size_Threshold,
      Tuning_Naive_Inv_Fast_Dim_Threshold,
      Tuning_Default_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
```

```
      Tuning_Default_Function_Precision
    >;

  const unsigned int Tuning_Fast_Mult_Matrix_Threshold   =       0;
  const unsigned int Tuning_Fast_Div_Max_Steps           =       0;
  const unsigned int Tuning_Fast_CR_Sqrt_Max_Steps       =     256;
  const unsigned int Tuning_Fast_DB_Sqrt_Max_Steps       =     256;
  const unsigned int Tuning_Fast_Log_Max_Outer_Steps     =      16;
  const unsigned int Tuning_Fast_Log_Max_Inner_Steps     =       8;
  const unsigned int Tuning_Fast_Basis_Max_Count         =       1;
  const unsigned int Tuning_Fast_Fast_Size_Threshold     =       0;
  const unsigned int Tuning_Fast_Inv_Fast_Dim_Threshold  =       0;
  const unsigned int Tuning_Fast_Products_Size_Threshold =       0;

  using tuning_fast = tuning
    <
      Tuning_Fast_Mult_Matrix_Threshold,
      Tuning_Fast_Div_Max_Steps,
      Tuning_Fast_CR_Sqrt_Max_Steps,
      Tuning_Fast_DB_Sqrt_Max_Steps,
      Tuning_Fast_Log_Max_Outer_Steps,
      Tuning_Fast_Log_Max_Inner_Steps,
      Tuning_Fast_Basis_Max_Count,
      Tuning_Fast_Fast_Size_Threshold,
      Tuning_Fast_Inv_Fast_Dim_Threshold,
      Tuning_Fast_Products_Size_Threshold,
      Tuning_Default_Denom_Different_Bits,
      Tuning_Default_Extra_Different_Bits,
      Tuning_Default_Function_Precision
    >;
}
#endif // GLUCAT_TEST_TUNING_H
```

7.55 pyclical/glucat.pxd File Reference

Namespaces

- namespace glucat

7.56 glucat.pxd

Go to the documentation of this file.

```
# -*- coding: utf-8 -*-
# cython: language_level=3
#
# PyClical: Python interface to GluCat:
#           Generic library of universal Clifford algebra templates
#
# glucat.pxd: Basic Cython definitions
#             corresponding to C++ definitions from PyClical.h.
# Kept as a separate module from PyClical.pxd to avoid namespace clashes.
#
#    copyright            : (C) 2008-2012 by Paul C. Leopardi
#
#    This library is free software: you can redistribute it and/or modify
#    it under the terms of the GNU Lesser General Public License as published
#    by the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This library is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU Lesser General Public License
#    along with this library.  If not, see <http://www.gnu.org/licenses/>.

from libcpp.vector cimport vector

cdef extern from "PyClical.h":

    cdef cppclass String:
        char* c_str()

```

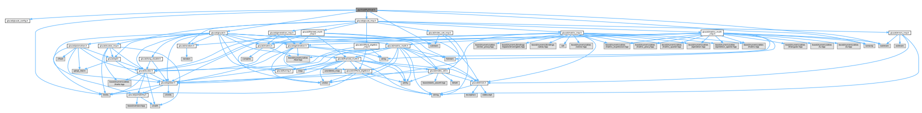
```
    cdef cppclass IndexSet:
        IndexSet ()
        IndexSet (IndexSet Ist) except+
        IndexSet (int idx) except+
        IndexSet (char* str) except+
        inline bint operator==(IndexSet Rhs)
        inline bint operator!=(IndexSet Rhs)
        inline bint operator<(IndexSet Rhs)
        inline IndexSet invert "operator~"()
        inline bint getitem "operator[]"(int idx)
        inline IndexSet set()
        inline IndexSet set(int idx) except+
        inline IndexSet set(int idx, int val) except+
        inline IndexSet reset()
        inline IndexSet reset(int idx) except+
        int count()
        int count_pos()
        int count_neg()
        int min()
        int max()
        int sign_of_mult(IndexSet Rhs)
        int sign_of_square()
        int hash_fn()

    int compare(IndexSet Lhs, IndexSet Rhs)
    int min_neg(IndexSet Ist)
    int max_pos(IndexSet Ist)

    ctypedef double scalar_t

    cdef cppclass Clifford:
        Clifford ()
        Clifford (Clifford Clf) except+
        Clifford (Clifford Clf, IndexSet ist) except+
        Clifford (scalar_t scr) except+
        Clifford (char* str) except+
        Clifford (IndexSet ist, scalar_t scr) except+
        Clifford (vector[scalar_t] vec, IndexSet ist) except+
        bint     operator==(Clifford Rhs)
        bint     operator!=(Clifford Rhs)
        Clifford neg "operator-"()
        scalar_t getitem "operator[]"(IndexSet Ist)
        Clifford call "operator()"(int grade)
        scalar_t scalar()
        Clifford pure()
        Clifford even()
        Clifford odd()
        vector[scalar_t] vector_part()
        vector[scalar_t] vector_part(IndexSet frm) except+
        Clifford involute()
        Clifford reverse()
        Clifford conj()
        Clifford random(IndexSet Ist, scalar_t fill)
        scalar_t norm()
        scalar_t quad()
        IndexSet frame()
        scalar_t max_abs()
        Clifford inv()
        Clifford pow(int m)
        Clifford outer_pow(int m)
        Clifford truncated(scalar_t limit)
        bint     isinf()
        bint     isnan()
        void     write(char* msg)

    scalar_t error_squared_tol(Clifford Clf)
    scalar_t error_squared(Clifford Lhs, Clifford Rhs, scalar_t threshold)
    bint     approx_equal(Clifford Lhs, Clifford Rhs, scalar_t threshold, scalar_t tol)
    scalar_t scalar(Clifford Clf)
    scalar_t real(Clifford Clf)
    scalar_t imag(Clifford Clf)
    Clifford pure(Clifford Clf)
    Clifford even(Clifford Clf)
    Clifford odd(Clifford Clf)
    Clifford involute(Clifford Clf)
    Clifford reverse(Clifford Clf)
    Clifford conj(Clifford Clf)
    scalar_t norm(Clifford Clf)
    scalar_t abs(Clifford Clf)
    scalar_t max_abs(Clifford Clf)
    scalar_t quad(Clifford Clf)
    Clifford inv(Clifford Clf)
    Clifford pow(Clifford Clf,int m)
    Clifford outer_pow(Clifford Clf,int m)

    Clifford complexifier(Clifford Clf)
    Clifford sqrt(Clifford Clf, Clifford I) except+
```

```
    Clifford sqrt(Clifford Clf)
    Clifford exp(Clifford Clf)
    Clifford log(Clifford Clf, Clifford I) except+
    Clifford log(Clifford Clf)
    Clifford cos(Clifford Clf, Clifford I) except+
    Clifford cos(Clifford Clf)
    Clifford acos(Clifford Clf, Clifford I) except+
    Clifford acos(Clifford Clf)
    Clifford cosh(Clifford Clf)
    Clifford acosh(Clifford Clf, Clifford I) except+
    Clifford acosh(Clifford Clf)
    Clifford sin(Clifford Clf, Clifford I) except+
    Clifford sin(Clifford Clf)
    Clifford asin(Clifford Clf, Clifford I) except+
    Clifford asin(Clifford Clf)
    Clifford sinh(Clifford Clf)
    Clifford asinh(Clifford Clf, Clifford I) except+
    Clifford asinh(Clifford Clf)
    Clifford tan(Clifford Clf, Clifford I) except+
    Clifford tan(Clifford Clf)
    Clifford atan(Clifford Clf, Clifford I) except+
    Clifford atan(Clifford Clf)
    Clifford tanh(Clifford Clf)
    Clifford atanh(Clifford Clf, Clifford I) except+
    Clifford atanh(Clifford Clf)

cdef extern from "PyClical.h" namespace "cga3":
    Clifford agc3(Clifford Clf)
    Clifford cga3(Clifford Clf)
    Clifford cga3std(Clifford Clf)
```

7.57 pyclical/PyClical.h File Reference

```
#include "glucat/glucat_config.h"
#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include <iostream>
#include <sstream>
#include <iomanip>
#include <limits>
```

Include dependency graph for PyClical.h:



Namespaces

- namespace cga3

    Definitions for 3D Conformal Geometric Algebra [DL].

Typedefs

- using String = std::string
- using IndexSet = index_set<lo_ndx, hi_ndx>
- using scalar_t = double
- using Clifford = matrix_multi<scalar_t,lo_ndx, hi_ndx,tuning_promoted>

Functions

- template<typename Scalar_T >
  PyObject ∗ PyFloat_FromDouble (Scalar_T v)
- template<typename Index_Set_T >
  String index_set_to_repr (const Index_Set_T &ist)

  The “official” string representation of Index_Set_T ist.
- template<typename Index_Set_T >
  String index_set_to_str (const Index_Set_T &ist)

  The "informal" string representation of Index_Set_T ist.
- template<typename Multivector_T >
  String clifford_to_repr (const Multivector_T &mv)

  The “official” string representation of Multivector_T mv.
- template<typename Multivector_T >
  String clifford_to_str (const Multivector_T &mv)

  The "informal" string representation of Multivector_T mv.
- template<typename Multivector_T >
  Multivector_T cga3::cga3 (const Multivector_T &x)

  Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].
- template<typename Multivector_T >
  Multivector_T cga3::cga3std (const Multivector_T &X)

  Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].
- template<typename Multivector_T >
  Multivector_T cga3::agc3 (const Multivector_T &X)

  Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].

Variables

- String glucat_package_version = GLUCAT_PACKAGE_VERSION
- const index_t lo_ndx = DEFAULT_LO
- const index_t hi_ndx = DEFAULT_HI
- const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon()

7.57.1 Typedef Documentation

7.57.1.1 Clifford

```
using Clifford = matrix_multi<scalar_t,lo_ndx, hi_ndx,tuning_promoted>
```

Definition at line 148 of file PyClical.h.

7.57.1.2 IndexSet

```
using IndexSet = index_set<lo_ndx, hi_ndx>
```

Definition at line 145 of file PyClical.h.

7.57.1.3 scalar_t

```
using scalar_t = double
```

Definition at line 147 of file PyClical.h.

7.57.1.4 String

```
using String = std::string
```

Definition at line 51 of file PyClical.h.

7.57.2 Function Documentation

7.57.2.1 clifford_to_repr()

```
template<typename Multivector_T >
String clifford_to_repr (
            const Multivector_T & mv)  [inline]
```

The “official” string representation of Multivector_T mv.

Definition at line 75 of file PyClical.h.

Referenced by PyClical.clifford::__repr__().

7.57.2.2 clifford_to_str()

```
template<typename Multivector_T >
String clifford_to_str (
            const Multivector_T & mv)  [inline]
```

The "informal" string representation of Multivector_T mv.

Definition at line 86 of file PyClical.h.

References glucat::abs().

Referenced by PyClical.clifford::__str__().

7.57.2.3 index_set_to_repr()

```
template<typename Index_Set_T >
String index_set_to_repr (
            const Index_Set_T & ist)  [inline]
```

The “official” string representation of Index_Set_T ist.

Definition at line 57 of file PyClical.h.

Referenced by PyClical.index_set::__repr__().

7.57.2.4 index_set_to_str()

```
template<typename Index_Set_T >
String index_set_to_str (
            const Index_Set_T & ist) [inline]
```

The "informal" string representation of Index_Set_T ist.

Definition at line 66 of file PyClical.h.

Referenced by PyClical.index_set::__str__().

7.57.2.5 PyFloat_FromDouble()

```
template<typename Scalar_T >
PyObject * PyFloat_FromDouble (
            Scalar_T v) [inline]
```

Create a PyFloatObject object from Scalar_T v. Needed because Scalar_T might not be the same as double.

Definition at line 45 of file PyClical.h.

7.57.3 Variable Documentation

7.57.3.1 epsilon

```
const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon()
```

Definition at line 150 of file PyClical.h.

Referenced by glucat::cascade_log(), and glucat::matrix::classify_eigenvalues().

7.57.3.2 glucat_package_version

```
String glucat_package_version = GLUCAT_PACKAGE_VERSION
```

Definition at line 53 of file PyClical.h.

7.57.3.3 hi_ndx

```
const index_t hi_ndx = DEFAULT_HI
```

Definition at line 144 of file PyClical.h.

7.57.3.4 lo_ndx

```
const index_t lo_ndx = DEFAULT_LO
```

Definition at line 143 of file PyClical.h.

7.58 PyClical.h

Go to the documentation of this file.

```
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    PyClical.h : C++ definitions needed by PyClical
                             -------------------
    copyright            : (C) 2008-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat/glucat.h
 ***************************************************************************/
// References for algorithms:
// [DL]:
// C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.

#include "glucat/glucat_config.h"
#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include <iostream>
#include <sstream>
#include <iomanip>
#include <limits>

template <typename Scalar_T>
inline PyObject* PyFloat_FromDouble(Scalar_T v)
{ return ::PyFloat_FromDouble(glucat::numeric_traits<Scalar_T>::to_double(v)); }


// String representations for use by PyClical Python classes.

using String = std::string;

String glucat_package_version = GLUCAT_PACKAGE_VERSION;

template<typename Index_Set_T>
inline String index_set_to_repr(const Index_Set_T& ist)
{
  std::ostringstream os;
  os « "index_set(" « ist « ")";
  return os.str();
}

template<typename Index_Set_T>
inline String index_set_to_str(const Index_Set_T& ist)
{
  std::ostringstream os;
  os « ist;
  return os.str();
}

template<typename Multivector_T>
inline String clifford_to_repr(const Multivector_T& mv)
{
  using scalar_t = typename Multivector_T::scalar_t;
  std::ostringstream os;
  os « std::setprecision(std::numeric_limits<scalar_t>::digits10 + 1);
  os « "clifford(\"" « mv « "\")";
  return os.str();
}

template<typename Multivector_T>
inline String clifford_to_str(const Multivector_T& mv)
{
  using scalar_t = typename Multivector_T::scalar_t;
```

```
  std::ostringstream os;
  if (abs(mv) < std::numeric_limits<scalar_t>::epsilon())
    os « 0.0;
  else
    os « std::setprecision(4) « mv.truncated(scalar_t(1.0e-4));
  return os.str();
}


namespace cga3
{
  template<typename Multivector_T>
  inline Multivector_T cga3(const Multivector_T& x)
  {
    using cl = Multivector_T;
    using ist = typename cl::index_set_t;
    static const cl ninf3 = cl(ist(4)) + cl(ist(-1));

    return (cl(ist(4)) - x) * ninf3 * (x - cl(ist(4)));
  }

  template<typename Multivector_T>
  inline Multivector_T cga3std(const Multivector_T& X)
  {
    using cl = Multivector_T;
    using ist = typename cl::index_set_t;
    using scalar_t = typename cl::scalar_t;
    static const cl ninf3 = cl(ist(4)) + cl(ist(-1));

    return scalar_t(-2.0) * X / (X & ninf3);
  }

  template<typename Multivector_T>
  inline Multivector_T agc3(const Multivector_T& X)
  {
    using cl = Multivector_T;
    using ist = typename cl::index_set_t;
    using scalar_t = typename cl::scalar_t;

    const cl& cga3stdX = cga3std(X);
    return (cl(ist(1))*cga3stdX[ist(1)] +
            cl(ist(2))*cga3stdX[ist(2)] +
            cl(ist(3))*cga3stdX[ist(3)]) / scalar_t(2.0);
  }
}


// Specifications of the IndexSet and Clifford C++ classes for use with PyClical.

using namespace glucat;
const index_t lo_ndx = DEFAULT_LO;
const index_t hi_ndx = DEFAULT_HI;
using IndexSet = index_set<lo_ndx, hi_ndx>;

using scalar_t = double;
using Clifford = matrix_multi<scalar_t,lo_ndx, hi_ndx,tuning_promoted>;

const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon();

// Do not warn about unused values. This affects clang++ as well as g++.

#pragma GCC diagnostic ignored "-Wunused-value"

#if defined(__clang__)
// Do not warn about unused functions. The affects clang++ only.

# pragma clang diagnostic ignored "-Wunused-function"

// Do not warn about unneeded internal declarations. The affects clang++ only.

# pragma clang diagnostic ignored "-Wunneeded-internal-declaration"
#endif
```

7.59 pyclical/PyClical.pxd File Reference

Namespaces

- namespace PyClical

7.60 PyClical.pxd

Go to the documentation of this file.

```
# -*- coding: utf-8 -*-
# cython: language_level=3
#
# PyClical: Python interface to GluCat:
#           Generic library of universal Clifford algebra templates
#
# PyClical.pxd: Basic Cython definitions for PyClical
#               corresponding to C++ definitions from PyClical.h.
#
#    copyright            : (C) 2008-2021 by Paul C. Leopardi
#
#    This library is free software: you can redistribute it and/or modify
#    it under the terms of the GNU Lesser General Public License as published
#    by the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This library is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU Lesser General Public License
#    along with this library.  If not, see <http://www.gnu.org/licenses/>.

cimport glucat
from glucat cimport IndexSet, String, Clifford, scalar_t, vector
from libcpp.string cimport string

cdef extern from "PyClical.h":
    string glucat_package_version

    IndexSet operator&(IndexSet Lhs, IndexSet Rhs)
    IndexSet operator|(IndexSet Lhs, IndexSet Rhs)
    IndexSet operator^(IndexSet Lhs, IndexSet Rhs)

    string index_set_to_repr(IndexSet& Ist)
    string index_set_to_str(IndexSet& Ist)

    Clifford operator+(Clifford Lhs, Clifford Rhs)
    Clifford operator-(Clifford Lhs, Clifford Rhs)
    Clifford operator*(Clifford Lhs, Clifford Rhs)
    Clifford operator&(Clifford Lhs, Clifford Rhs)
    Clifford operator%(Clifford Lhs, Clifford Rhs)
    Clifford operator^(Clifford Lhs, Clifford Rhs)
    Clifford operator/(Clifford Lhs, Clifford Rhs)
    Clifford operator|(Clifford Lhs, Clifford Rhs)

    string clifford_to_repr(Clifford& Clf)
    string clifford_to_str(Clifford& Clf)

    const scalar_t epsilon
```

7.61 pyclical/PyClical.pyx File Reference

Classes

- class PyClical.index_set
- class PyClical.clifford

Namespaces

- namespace PyClical

Functions

- PyClical.index_set_hidden_doctests ()
- PyClical.clifford_hidden_doctests ()
- PyClical.e (obj)
- PyClical.istpq (p, q)
- PyClical._test ()

Variables

- PyClical.__version__ = str(glucat_package_version,'utf-8')
- PyClical.lhs
- PyClical.rhs
- PyClical.threshold = error_squared_tol(rhs) if threshold is None else threshold
- PyClical.None
- PyClical.tol = error_squared_tol(rhs) if tol is None else tol
- PyClical.obj
- PyClical.i
- PyClical.ixt
- PyClical.fill
- PyClical.scalar_epsilon = epsilon
- float PyClical.pi = atan(clifford(1.0)) ∗ 4.0
- float PyClical.tau = atan(clifford(1.0)) ∗ 8.0
- PyClical.cl = clifford
- PyClical.ist = index_set
- PyClical.ninf3 = e(4) + e(-1)
- PyClical.nbar3 = e(4) - e(-1)

7.62 PyClical.pyx

Go to the documentation of this file.

```
# -*- coding: utf-8 -*-
# cython: language_level=3
# distutils: language = c++
#
# PyClical: Python interface to GluCat:
#           Generic library of universal Clifford algebra templates
#
# PyClical.pyx: Cython definitions visible from Python.
#
#    copyright            : (C) 2008-2021 by Paul C. Leopardi
#
#    This library is free software: you can redistribute it and/or modify
#    it under the terms of the GNU Lesser General Public License as published
#    by the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This library is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU Lesser General Public License
#    along with this library.  If not, see <http://www.gnu.org/licenses/>.

# References for definitions:
# [DL]:
# C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.

import math
import numbers
import collections

from PyClical cimport *

__version__ = str(glucat_package_version,'utf-8')

# Forward reference
cdef class index_set

cdef inline IndexSet toIndexSet(obj):
    """
    Return the C++ IndexSet instance wrapped by index_set(obj).
    """
    return index_set(obj).instance[0]

cdef class index_set:
    """
    Python class index_set wraps C++ class IndexSet.
```

```
    """
    cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.

    cdef inline wrap(index_set self, IndexSet other):
        """
        Wrap an instance of the C++ class IndexSet.
        """
        self.instanceinstance[0] = other
        return self

    cdef inline IndexSet unwrap(index_set self):
        """
        Return the wrapped C++ IndexSet instance.
        """
        return self.instanceinstance[0]

    cpdef copy(index_set self):
        """
        Copy this index_set object.

        >>> s=index_set(1); t=s.copy(); print(t)
        {1}
        """
        return index_set(self)

    def __cinit__(self, other = 0):
        """
        Construct an object of type index_set.

        >>> print(index_set(1))
        {1}
        >>> print(index_set({1,2}))
        {1,2}
        >>> print(index_set(index_set({1,2})))
        {1,2}
        >>> print(index_set({1,2}))
        {1,2}
        >>> print(index_set({1,2,1}))
        {1,2}
        >>> print(index_set("{1,2,1}"))
        {1,2}
        >>> print(index_set(""))
        {}
        """
        error_msg_prefix = "Cannot initialize index_set object from"
        if   isinstance(other, index_set):
            self.instanceinstance = new IndexSet((<index_set>other).unwrap())
        elif isinstance(other, numbers.Integral):
            self.instanceinstance = new IndexSet(<int>other)
        elif isinstance(other, (set, frozenset)):
            try:
                self.instanceinstance = new IndexSet()
                for idx in other:
                    self[idx] = True
            except IndexError:
                raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
            except (RuntimeError, TypeError):
                raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
        elif isinstance(other, str):
            try:
                bother = other.encode("UTF-8")
                self.instanceinstance = new IndexSet(<char *>bother)
            except RuntimeError:
                raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
        else:
            raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")

    def __dealloc__(self):
        """
        Clean up by deallocating the instance of C++ class IndexSet.
        """
        del self.instanceinstance

    def __richcmp__(lhs, rhs, int op):
        """
        Compare two objects of class index_set.

        >>> index_set(1) == index_set({1})
        True
        >>> index_set({1}) != index_set({1})
        False
        >>> index_set({1}) != index_set({2})
        True
        >>> index_set({1}) == index_set({2})
        False
        >>> index_set({1}) < index_set({2})
        True
```

```
        »> index_set({1}) <= index_set({2})
        True
        »> index_set({1}) > index_set({2})
        False
        »> index_set({1}) >= index_set({2})
        False
        """
        if (lhs is None) or (rhs is None):
            eq = bool(lhs is rhs)
            if op == 2: # ==
                return eq
            elif op == 3: # !=
                return not eq
            else:
                if op == 0: # <
                    return False
                elif op == 1: # <=
                    return eq
                elif op == 4: # >
                    return False
                elif op == 5: # >=
                    return eq
                else:
                    return NotImplemented
        else:
            eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )
            if op == 2: # ==
                return eq
            elif op == 3: # !=
                return not eq
            else:
                lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )
                if op == 0: # <
                    return lt
                elif op == 1: # <=
                    return lt or eq
                elif op == 4: # >
                    return not (lt or eq)
                elif op == 5: # >=
                    return not lt
                else:
                    return NotImplemented

    def __setitem__(self, idx, val):
        """
        Set the value of an index_set object at index idx to value val.

        »> s=index_set({1}); s[2] = True; print(s)
        {1,2}
        »> s=index_set({1,2}); s[1] = False; print(s)
        {2}
        """
        self.instanceinstance.set(idx, val)
        return

    def __getitem__(self, idx):
        """
        Get the value of an index_set object at an index.

        »> index_set({1})[1]
        True
        »> index_set({1})[2]
        False
        »> index_set({2})[-1]
        False
        »> index_set({2})[1]
        False
        »> index_set({2})[2]
        True
        »> index_set({2})[33]
        False
        """
        return self.instanceinstance.getitem(idx)

    def __contains__(self, idx):
        """
        Check that an index_set object contains the index idx: idx in self.

        »> 1 in index_set({1})
        True
        »> 2 in index_set({1})
        False
        »> -1 in index_set({2})
        False
        »> 1 in index_set({2})
        False
        »> 2 in index_set({2})
```

```
        True
        »> 33 in index_set({2})
        False
        """
        return self.instanceinstance.getitem(idx)

    def __iter__(self):
        """
        Iterate over the indices of an index_set.

        »> for i in index_set({-3,4,7}):print(i, end=",")
        -3,4,7,
        """
        for idx in range(self.min(), self.max()+1):
            if idx in self:
                yield idx

    def __invert__(self):
        """
        Set complement: not.

        »> print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))
        {-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}
        """
        return index_set().wrap( self.instanceinstance.invert() )

    def __xor__(lhs, rhs):
        """
        Symmetric set difference: exclusive or.

        »> print(index_set({1}) ^ index_set({2}))
        {1,2}
        »> print(index_set({1,2}) ^ index_set({2}))
        {1}
        """
        return index_set().wrap( toIndexSet(lhs) ^ toIndexSet(rhs) )

    def __ixor__(self, rhs):
        """
        Symmetric set difference: exclusive or.

        »> x = index_set({1}); x ^= index_set({2}); print(x)
        {1,2}
        »> x = index_set({1,2}); x ^= index_set({2}); print(x)
        {1}
        """
        return self.wrap( self.unwrap() ^ toIndexSet(rhs) )

    def __and__(lhs, rhs):
        """
        Set intersection: and.

        »> print(index_set({1}) & index_set({2}))
        {}
        »> print(index_set({1,2}) & index_set({2}))
        {2}
        """
        return index_set().wrap( toIndexSet(lhs) & toIndexSet(rhs) )

    def __iand__(self, rhs):
        """
        Set intersection: and.

        »> x = index_set({1}); x &= index_set({2}); print(x)
        {}
        »> x = index_set({1,2}); x &= index_set({2}); print(x)
        {2}
        """
        return self.wrap( self.unwrap() & toIndexSet(rhs) )

    def __or__(lhs, rhs):
        """
        Set union: or.

        »> print(index_set({1}) | index_set({2}))
        {1,2}
        »> print(index_set({1,2}) | index_set({2}))
        {1,2}
        """
        return index_set().wrap( toIndexSet(lhs) | toIndexSet(rhs) )

    def __ior__(self, rhs):
        """
        Set union: or.

```

```
        »> x = index_set({1}); x |= index_set({2}); print(x)
        {1,2}
        »> x = index_set({1,2}); x |= index_set({2}); print(x)
        {1,2}
        """
        return self.wrap( self.unwrap() | toIndexSet(rhs) )

    def count(self):
        """
        Cardinality: Number of indices included in set.

        »> index_set({-1,1,2}).count()
        3
        """
        return self.instanceinstance.count()

    def count_neg(self):
        """
        Number of negative indices included in set.

        »> index_set({-1,1,2}).count_neg()
        1
        """
        return self.instanceinstance.count_neg()

    def count_pos(self):
        """
        Number of positive indices included in set.

        »> index_set({-1,1,2}).count_pos()
        2
        """
        return self.instanceinstance.count_pos()

    def min(self):
        """
        Minimum member.

        »> index_set({-1,1,2}).min()
        -1
        """
        return self.instanceinstance.min()

    def max(self):
        """
        Maximum member.

        »> index_set({-1,1,2}).max()
        2
        """
        return self.instanceinstance.max()

    def hash_fn(self):
        """
        Hash function.
        """
        return self.instanceinstance.hash_fn()

    def sign_of_mult(self, rhs):
        """
        Sign of geometric product of two Clifford basis elements.

        »> s = index_set({1,2}); t=index_set({-1}); s.sign_of_mult(t)
        1
        """
        return self.instanceinstance.sign_of_mult(toIndexSet(rhs))

    def sign_of_square(self):
        """
        Sign of geometric square of a Clifford basis element.

        »> s = index_set({1,2}); s.sign_of_square()
        -1
        """
        return self.instanceinstance.sign_of_square()

    def __repr__(self):
        """
        The “official” string representation of self.

        »> index_set({1,2}).__repr__()
        'index_set({1,2})'
        »> repr(index_set({1,2}))
        'index_set({1,2})'
        """
        return index_set_to_repr( self.unwrap() ).decode()

```

```
    def __str__(self):
        """
        The “informal” string representation of self.

        »> index_set({1,2}).__str__()
        ’{1,2}’
        »> str(index_set({1,2}))
        ’{1,2}’
        """
        return index_set_to_str( self.unwrap() ).decode()

def index_set_hidden_doctests():
    """
    Tests for functions that Doctest cannot see.

    For index_set.__cinit__: Construct index_set.

    »> print(index_set(1))
    {1}
    »> print(index_set({1,2}))
    {1,2}
    »> print(index_set(index_set({1,2})))
    {1,2}
    »> print(index_set({1,2}))
    {1,2}
    »> print(index_set({1,2,1}))
    {1,2}
    »> print(index_set({1,2,1}))
    {1,2}
    »> print(index_set(""))
    {}
    »> print(index_set("{"))
    Traceback (most recent call last):
    ...
    ValueError: Cannot initialize index_set object from invalid string ’{’.
    »> print(index_set("{1"))
    Traceback (most recent call last):
    ...
    ValueError: Cannot initialize index_set object from invalid string ’{1’.
    »> print(index_set("{1,2,100}"))
    Traceback (most recent call last):
    ...
    ValueError: Cannot initialize index_set object from invalid string ’{1,2,100}’.
    »> print(index_set({1,2,100}))
    Traceback (most recent call last):
    ...
    IndexError: Cannot initialize index_set object from invalid {1, 2, 100}.
    »> print(index_set([1,2]))
    Traceback (most recent call last):
    ...
    TypeError: Cannot initialize index_set object from <class ’list’>.

    For index_set.__richcmp__: Compare two objects of class index_set.

    »> index_set(1) == index_set({1})
    True
    »> index_set({1}) != index_set({1})
    False
    »> index_set({1}) != index_set({2})
    True
    »> index_set({1}) == index_set({2})
    False
    »> index_set({1}) < index_set({2})
    True
    »> index_set({1}) <= index_set({2})
    True
    »> index_set({1}) > index_set({2})
    False
    »> index_set({1}) >= index_set({2})
    False
    »> None == index_set({1,2})
    False
    »> None != index_set({1,2})
    True
    »> None < index_set({1,2})
    False
    »> None <= index_set({1,2})
    False
    »> None > index_set({1,2})
    False
    »> None >= index_set({1,2})
    False
    »> index_set({1,2}) == None
    False
    »> index_set({1,2}) != None
    True
    »> index_set({1,2}) < None
```

```
    False
    »> index_set({1,2}) <= None
    False
    »> index_set({1,2}) > None
    False
    »> index_set({1,2}) >= None
    False
    """
    return

cpdef inline compare(lhs,rhs):
    """
    "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
    -1 if a<b, +1 if a>b, 0 if a==b.

    »> compare(index_set({1,2}),index_set({-1,3}))
    -1
    »> compare(index_set({-1,4}),index_set({-1,3}))
    1
    """
    return glucat.compare( toIndexSet(lhs), toIndexSet(rhs) )

cpdef inline min_neg(obj):
    """
    Minimum negative index, or 0 if none.

    »> min_neg(index_set({1,2}))
    0
    """
    return glucat.min_neg( toIndexSet(obj) )

cpdef inline max_pos(obj):
    """
    Maximum positive index, or 0 if none.

    »> max_pos(index_set({1,2}))
    2
    """
    return glucat.max_pos( toIndexSet(obj) )

cdef inline vector[scalar_t] list_to_vector(lst):
     """
     Create a C++ std:vector[scalar_t] from an iterable Python object.
     """
     cdef vector[scalar_t] v
     for s in lst:
         v.push_back(<scalar_t>s)
     return v

# Forward reference.
cdef class clifford

cdef inline Clifford toClifford(obj):
    return clifford(obj).instance[0]

cdef class clifford:
    """
    Python class clifford wraps C++ class Clifford.
    """
    cdef Clifford *instance # Wrapped instance of C++ class Clifford.

    cdef inline wrap(clifford self, Clifford other):
        """
        Wrap an instance of the C++ class Clifford.
        """
        self.instance[0] = other
        return self

    cdef inline Clifford unwrap(clifford self):
        """
        Return the wrapped C++ Clifford instance.
        """
        return self.instance[0]

    cpdef copy(clifford self):
        """
        Copy this clifford object.

        »> x=clifford("1{2}"); y=x.copy(); print(y)
        {2}
        """
        return clifford(self)

    def __cinit__(self, other = 0, ixt = None):
        """
        Construct an object of type clifford.

```

```
        >> print(clifford(2))
        2
        >> print(clifford(2.0))
        2
        >> print(clifford(1.0e-1))
        0.1
        >> print(clifford("2"))
        2
        >> print(clifford("2{1,2,3}"))
        2{1,2,3}
        >> print(clifford(clifford("2{1,2,3}")))
        2{1,2,3}
        >> print(clifford("-{1}"))
        -{1}
        >> print(clifford(2,index_set({1,2})))
        2{1,2}
        >> print(clifford([2,3],index_set({1,2})))
        2{1}+3{2}
        """
        error_msg_prefix = "Cannot initialize clifford object from"
        if ixt is None:
            try:
                if   isinstance(other, clifford):
                    self.instance = new Clifford((<clifford>other).unwrap())
                elif isinstance(other, index_set):
                    self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
                elif isinstance(other, numbers.Real):
                    self.instance = new Clifford(<scalar_t>other)
                elif isinstance(other, str):
                    try:
                        bother = other.encode("UTF-8")
                        self.instance = new Clifford(<char *>bother)
                    except RuntimeError:
                        raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
                else:
                    raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
            except RuntimeError as err:
                raise ValueError(error_msg_prefix + " " + str(type(other))
                                                  + " value " + repr(other) + ":"
                                                  + "\n\t" + str(err))
        elif isinstance(ixt, index_set):
            if   isinstance(other, numbers.Real):
                self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
            elif isinstance(other, collections.abc.Sequence):
                self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
            else:
                raise TypeError(error_msg_prefix + " (" + str(type(other))
                                                 + ", " + repr(ixt) + ").")
        else:
            raise TypeError(error_msg_prefix + " (" + str(type(other))
                                             + ", " + str(type(ixt)) + ").")

    def __dealloc__(self):
        """
        Clean up by deallocating the instance of C++ class Clifford.
        """
        del self.instance

    def __contains__(self, x):
        """
        Not applicable.

        >> x=clifford(index_set({-3,4,7})); -3 in x
        Traceback (most recent call last):
          ...
        TypeError: Not applicable.
        """
        raise TypeError("Not applicable.")

    def __iter__(self):
        """
        Not applicable.

        >> for a in clifford(index_set({-3,4,7})):print(a, end=",")
        Traceback (most recent call last):
          ...
        TypeError: Not applicable.
        """
        raise TypeError("Not applicable.")

    def reframe(self, ixt):
        """
        Put self into a larger frame, containing the union of self.frame() and index set ixt.
        This can be used to make multiplication faster, by multiplying within a common frame.

        >> clifford("2+3{1}").reframe(index_set({1,2,3}))
        clifford("2+3{1}")
```

```
        >>> s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() ==
(s|t);
        True
        """
        error_msg_prefix = "Cannot reframe"
        if isinstance(ixt, index_set):
            try:
                result = clifford()
                result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
            except RuntimeError as err:
                raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
                                                                      + str(ixt) + ":"
                                                                      + "\n\t" + str(err))
        else:
            raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
        return result

    def __richcmp__(lhs, rhs, int op):
        """
        Compare objects of type clifford.

        >>> clifford("{1}") == clifford("1{1}")
        True
        >>> clifford("{1}") != clifford("1.0{1}")
        False
        >>> clifford("{1}") != clifford("1.0")
        True
        >>> clifford("{1,2}") == None
        False
        >>> clifford("{1,2}") != None
        True
        >>> None == clifford("{1,2}")
        False
        >>> None != clifford("{1,2}")
        True
        """
        if op == 2: # ==
            if (lhs is None) or (rhs is None):
                return bool(lhs is rhs)
            else:
                return bool( toClifford(lhs) == toClifford(rhs) )
        elif op == 3: # !=
            if (lhs is None) or (rhs is None):
                return not bool(lhs is rhs)
            else:
                return bool( toClifford(lhs) != toClifford(rhs) )
        elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
            raise TypeError("This comparison operator is not implemented for "
                                + str(type(lhs)) + ", " + str(type(rhs)) + ".")
        else:
            return NotImplemented

    def __getitem__(self, ixt):
        """
        Subscripting: map from index set to scalar coordinate.

        >>> clifford("{1}")[index_set(1)]
        1.0
        >>> clifford("{1}")[index_set({1})]
        1.0
        >>> clifford("{1}")[index_set({1,2})]
        0.0
        >>> clifford("2{1,2}")[index_set({1,2})]
        2.0
        """
        return self.instance.getitem(toIndexSet(ixt))

    def __neg__(self):
        """
        Unary -.

        >>> print(-clifford("{1}"))
        -{1}
        """
        return clifford().wrap( self.instance.neg() )

    def __pos__(self):
        """
        Unary +.

        >>> print(+clifford("{1}"))
        {1}
        """
        return clifford(self)

    def __add__(lhs, rhs):
        """
```

```
        Geometric sum.

        >>> print(clifford(1) + clifford("{2}"))
        1+{2}
        >>> print(clifford("{1}") + clifford("{2}"))
        {1}+{2}
        """
        return clifford().wrap( toClifford(lhs) + toClifford(rhs) )

    def __iadd__(self, rhs):
        """
        Geometric sum.

        >>> x = clifford(1); x += clifford("{2}"); print(x)
        1+{2}
        """
        return self.wrap( self.unwrap() + toClifford(rhs) )

    def __sub__(lhs, rhs):
        """
        Geometric difference.

        >>> print(clifford(1) - clifford("{2}"))
        1-{2}
        >>> print(clifford("{1}") - clifford("{2}"))
        {1}-{2}
        """
        return clifford().wrap( toClifford(lhs) - toClifford(rhs) )

    def __isub__(self, rhs):
        """
        Geometric difference.

        >>> x = clifford(1); x -= clifford("{2}"); print(x)
        1-{2}
        """
        return self.wrap( self.unwrap() - toClifford(rhs) )

    def __mul__(lhs, rhs):
        """
        Geometric product.

        >>> print(clifford("{1}") * clifford("{2}"))
        {1,2}
        >>> print(clifford(2) * clifford("{2}"))
        2{2}
        >>> print(clifford("{1}") * clifford("{1,2}"))
        {2}
        """
        return clifford().wrap( toClifford(lhs) * toClifford(rhs) )

    def __imul__(self, rhs):
        """
        Geometric product.

        >>> x = clifford(2); x *= clifford("{2}"); print(x)
        2{2}
        >>> x = clifford("{1}"); x *= clifford("{2}"); print(x)
        {1,2}
        >>> x = clifford("{1}"); x *= clifford("{1,2}"); print(x)
        {2}
        """
        return self.wrap( self.unwrap() * toClifford(rhs) )

    def __mod__(lhs, rhs):
        """
        Contraction.

        >>> print(clifford("{1}") % clifford("{2}"))
        0
        >>> print(clifford(2) % clifford("{2}"))
        2{2}
        >>> print(clifford("{1}") % clifford("{1}"))
        1
        >>> print(clifford("{1}") % clifford("{1,2}"))
        {2}
        """
        return clifford().wrap( toClifford(lhs) % toClifford(rhs) )

    def __imod__(self, rhs):
        """
        Contraction.

        >>> x = clifford("{1}"); x %= clifford("{2}"); print(x)
        0
        >>> x = clifford(2); x %= clifford("{2}"); print(x)
        2{2}
```

```
        >> x = clifford("{1}"); x %= clifford("{1}"); print(x)
        1
        >> x = clifford("{1}"); x %= clifford("{1,2}"); print(x)
        {2}
        """
        return self.wrap( self.unwrap() % toClifford(rhs) )

    def __and__(lhs, rhs):
        """
        Inner product.

        >> print(clifford("{1}") & clifford("{2}"))
        0
        >> print(clifford(2) & clifford("{2}"))
        0
        >> print(clifford("{1}") & clifford("{1}"))
        1
        >> print(clifford("{1}") & clifford("{1,2}"))
        {2}
        """
        return clifford().wrap( toClifford(lhs) & toClifford(rhs) )

    def __iand__(self, rhs):
        """
        Inner product.

        >> x = clifford("{1}"); x &= clifford("{2}"); print(x)
        0
        >> x = clifford(2); x &= clifford("{2}"); print(x)
        0
        >> x = clifford("{1}"); x &= clifford("{1}"); print(x)
        1
        >> x = clifford("{1}"); x &= clifford("{1,2}"); print(x)
        {2}
        """
        return self.wrap( self.unwrap() & toClifford(rhs) )

    def __xor__(lhs, rhs):
        """
        Outer product.

        >> print(clifford("{1}") ^ clifford("{2}"))
        {1,2}
        >> print(clifford(2) ^ clifford("{2}"))
        2{2}
        >> print(clifford("{1}") ^ clifford("{1}"))
        0
        >> print(clifford("{1}") ^ clifford("{1,2}"))
        0
        """
        return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )

    def __ixor__(self, rhs):
        """
        Outer product.

        >> x = clifford("{1}"); x ^= clifford("{2}"); print(x)
        {1,2}
        >> x = clifford(2); x ^= clifford("{2}"); print(x)
        2{2}
        >> x = clifford("{1}"); x ^= clifford("{1}"); print(x)
        0
        >> x = clifford("{1}"); x ^= clifford("{1,2}"); print(x)
        0
        """
        return self.wrap( self.unwrap() ^ toClifford(rhs) )

    def __truediv__(lhs, rhs):
        """
        Geometric quotient.

        >> print(clifford("{1}") / clifford("{2}"))
        {1,2}
        >> print(clifford(2) / clifford("{2}"))
        2{2}
        >> print(clifford("{1}") / clifford("{1}"))
        1
        >> print(clifford("{1}") / clifford("{1,2}"))
        -{2}
        """
        return clifford().wrap( toClifford(lhs) / toClifford(rhs) )

    def __idiv__(self, rhs):
        """
        Geometric quotient.

        >> x = clifford("{1}"); x /= clifford("{2}"); print(x)
```

```
        {1,2}
        »> x = clifford(2); x /= clifford("{2}"); print(x)
        2{2}
        »> x = clifford("{1}"); x /= clifford("{1}"); print(x)
        1
        »> x = clifford("{1}"); x /= clifford("{1,2}"); print(x)
        -{2}
        """
        return self.wrap( self.unwrap() / toClifford(rhs) )

    def inv(self):
        """
        Geometric multiplicative inverse.

        »> x = clifford("{1}"); print(x.inv())
        {1}
        »> x = clifford(2); print(x.inv())
        0.5
        »> x = clifford("{1,2}"); print(x.inv())
        -{1,2}
        """
        return clifford().wrap( self.instance.inv() )

    def __or__(lhs, rhs):
        """
        Transform left hand side, using right hand side as a transformation.

        »> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|x)
        -{1}
        »> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|exp(x))
        -{1}
        """
        return clifford().wrap( toClifford(lhs) | toClifford(rhs) )

    def __ior__(self, rhs):
        """
        Transform left hand side, using right hand side as a transformation.

        »> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=x; print(y)
        -{1}
        »> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=exp(x); print(y)
        -{1}
        """
        return self.wrap( self.unwrap() | toClifford(rhs) )

    def __pow__(self, m, dummy):
        """
        Power: self to the m.

        »> x=clifford("{1}"); print(x ** 2)
        1
        »> x=clifford("2"); print(x ** 2)
        4
        »> x=clifford("2+{1}"); print(x ** 0)
        1
        »> x=clifford("2+{1}"); print(x ** 1)
        2+{1}
        »> x=clifford("2+{1}"); print(x ** 2)
        5+4{1}
        »> i=clifford("{1,2}"); print(exp(pi/2) * (i ** i))
        1
        """
        return pow(self, m)

    def pow(self, m):
        """
        Power: self to the m.

        »> x=clifford("{1}"); print(x.pow(2))
        1
        »> x=clifford("2"); print(x.pow(2))
        4
        »> x=clifford("2+{1}"); print(x.pow(0))
        1
        »> x=clifford("2+{1}"); print(x.pow(1))
        2+{1}
        »> x=clifford("2+{1}"); print(x.pow(2))
        5+4{1}
        »> print(clifford("1+{1}+{1,2}").pow(3))
        1+3{1}+3{1,2}
        »> i=clifford("{1,2}"); print(exp(pi/2) * i.pow(i))
        1
        """
        if isinstance(m, numbers.Integral):
            return clifford().wrap( self.instance.pow(m) )
        else:
            return exp(m * log(self))
```

```
    def outer_pow(self, m):
        """
        Outer product power.

        »> x=clifford("2+{1}"); print(x.outer_pow(0))
        1
        »> x=clifford("2+{1}"); print(x.outer_pow(1))
        2+{1}
        »> x=clifford("2+{1}"); print(x.outer_pow(2))
        4+4{1}
        »> print(clifford("1+{1}+{1,2}").outer_pow(3))
        1+3{1}+3{1,2}

        """
        return clifford().wrap( self.instance.outer_pow(m) )

    def __call__(self, grade):
        """
        Pure grade-vector part.

        »> print(clifford("{1}")(1))
        {1}
        »> print(clifford("{1}")(0))
        0
        »> print(clifford("1+{1}+{1,2}")(0))
        1
        »> print(clifford("1+{1}+{1,2}")(1))
        {1}
        »> print(clifford("1+{1}+{1,2}")(2))
        {1,2}
        »> print(clifford("1+{1}+{1,2}")(3))
        0
        """
        return clifford().wrap( self.instance.call(grade) )

    def scalar(self):
        """
        Scalar part.

        »> clifford("1+{1}+{1,2}").scalar()
        1.0
        »> clifford("{1,2}").scalar()
        0.0
        """
        return self.instance.scalar()

    def pure(self):
        """
        Pure part.

        »> print(clifford("1+{1}+{1,2}").pure())
        {1}+{1,2}
        »> print(clifford("{1,2}").pure())
        {1,2}
        """
        return clifford().wrap( self.instance.pure() )

    def even(self):
        """
        Even part of multivector, sum of even grade terms.

        »> print(clifford("1+{1}+{1,2}").even())
        1+{1,2}
        """
        return clifford().wrap( self.instance.even() )

    def odd(self):
        """
        Odd part of multivector, sum of odd grade terms.

        »> print(clifford("1+{1}+{1,2}").odd())
        {1}
        """
        return clifford().wrap( self.instance.odd() )

    def vector_part(self, frm = None):
        """
        Vector part of multivector, as a Python list, with respect to frm.

        »> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part())
        [2.0, 3.0]
        »> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part(index_set({-1,1,2})))
        [0.0, 2.0, 3.0]
        """
        error_msg_prefix = "Cannot take vector part of "
        cdef vector[scalar_t] vec
```

```
        cdef int n
        cdef int i
        try:
            if frm is None:
                vec = self.instance.vector_part()
            else:
                vec = self.instance.vector_part((<index_set>frm).unwrap())
            n = vec.size()
            lst = [0.0]*n
            for i in xrange(n):
                lst[i] = vec[i]
            return lst
        except RuntimeError as err:
            raise ValueError(error_msg_prefix + str(self) + " using invalid "
                                              + repr(frm) + " as frame:\n\t"
                                              + str(err))

    def involute(self):
        """
        Main involution, each {i} is replaced by -{i} in each term,
        eg. clifford("{1}") -> -clifford("{1}").

        >>> print(clifford("{1}").involute())
        -{1}
        >>> print((clifford("{2}") * clifford("{1}")).involute())
        -{1,2}
        >>> print((clifford("{1}") * clifford("{2}")).involute())
        {1,2}
        >>> print(clifford("1+{1}+{1,2}").involute())
        1-{1}+{1,2}
        """
        return clifford().wrap( self.instance.involute() )

    def reverse(self):
        """
        Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").

        >>> print(clifford("{1}").reverse())
        {1}
        >>> print((clifford("{2}") * clifford("{1}")).reverse())
        {1,2}
        >>> print((clifford("{1}") * clifford("{2}")).reverse())
        -{1,2}
        >>> print(clifford("1+{1}+{1,2}").reverse())
        1+{1}-{1,2}
        """
        return clifford().wrap( self.instance.reverse() )

    def conj(self):
        """
        Conjugation, reverse o involute == involute o reverse.

        >>> print((clifford("{1}")).conj())
        -{1}
        >>> print((clifford("{2}") * clifford("{1}")).conj())
        {1,2}
        >>> print((clifford("{1}") * clifford("{2}")).conj())
        -{1,2}
        >>> print(clifford("1+{1}+{1,2}").conj())
        1-{1}-{1,2}
        """
        return clifford().wrap( self.instance.conj() )

    def quad(self):
        """
        Quadratic form == (rev(x)*x)(0).

        >>> print(clifford("1+{1}+{1,2}").quad())
        3.0
        >>> print(clifford("1+{-1}+{1,2}+{1,2,3}").quad())
        2.0
        """
        return self.instance.quad()

    def norm(self):
        """
        Norm == sum of squares of coordinates.

        >>> clifford("1+{1}+{1,2}").norm()
        3.0
        >>> clifford("1+{-1}+{1,2}+{1,2,3}").norm()
        4.0
        """
        return self.instance.norm()

    def abs(self):
        """
```

```
        Absolute value: square root of norm.

        »> clifford("1+{-1}+{1,2}+{1,2,3}").abs()
        2.0
        """
        return glucat.abs( self.unwrap() )

    def max_abs(self):
        """
        Maximum of absolute values of components of multivector: multivector infinity norm.

        »> clifford("1+{-1}+{1,2}+{1,2,3}").max_abs()
        1.0
        »> clifford("3+2{1}+{1,2}").max_abs()
        3.0
        """
        return self.instance.max_abs()

    def truncated(self, limit):
        """
        Remove all terms of self with relative size smaller than limit.

        »> clifford("1e8+{1}+1e-8{1,2}").truncated(1.0e-6)
        clifford("100000000")
        »> clifford("1e4+{1}+1e-4{1,2}").truncated(1.0e-6)
        clifford("10000+{1}")
        """
        return clifford().wrap( self.instance.truncated(limit) )

    def isinf(self):
        """
        Check if a multivector contains any infinite values.

        »> clifford().isinf()
        False
        """
        return self.instance.isnan()

    def isnan(self):
        """
        Check if a multivector contains any IEEE NaN values.

        »> clifford().isnan()
        False
        """
        return self.instance.isnan()

    def frame(self):
        """
        Subalgebra generated by all generators of terms of given multivector.

        »> print(clifford("1+3{-1}+2{1,2}+4{-2,7}").frame())
        {-2,-1,1,2,7}
        »> s=clifford("1+3{-1}+2{1,2}+4{-2,7}").frame(); type(s)
        <class 'PyClical.index_set'>
        """
        return index_set().wrap( self.instance.frame() )

    def __repr__(self):
        """
        The “official” string representation of self.

        »> clifford("1+3{-1}+2{1,2}+4{-2,7}").__repr__()
        'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
        """
        return clifford_to_repr( self.unwrap() ).decode()

    def __str__(self):
        """
        The “informal” string representation of self.

        »> clifford("1+3{-1}+2{1,2}+4{-2,7}").__str__()
        '1+3{-1}+2{1,2}+4{-2,7}'
        """
        return clifford_to_str( self.unwrap() ).decode()

def clifford_hidden_doctests():
    """
    Tests for functions that Doctest cannot see.

    For clifford.__cinit__: Construct an object of type clifford.

    »> print(clifford(2))
    2
    »> print(clifford(2.0))
    2
    »> print(clifford(1.0e-1))
```

```
    0.1
    >>> print(clifford("2"))
    2
    >>> print(clifford("2{1,2,3}"))
    2{1,2,3}
    >>> print(clifford(clifford("2{1,2,3}")))
    2{1,2,3}
    >>> print(clifford("-{1}"))
    -{1}
    >>> print(clifford(2,index_set({1,2})))
    2{1,2}
    >>> print(clifford([2,3],index_set({1,2})))
    2{1}+3{2}
    >>> print(clifford([1,2]))
    Traceback (most recent call last):
      ...
    TypeError: Cannot initialize clifford object from <class 'list'>.
    >>> print(clifford(None))
    Traceback (most recent call last):
      ...
    TypeError: Cannot initialize clifford object from <class 'NoneType'>.
    >>> print(clifford(None,[1,2]))
    Traceback (most recent call last):
      ...
    TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).
    >>> print(clifford([1,2],[1,2]))
    Traceback (most recent call last):
      ...
    TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).
    >>> print(clifford(""))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string ''.
    >>> print(clifford("{"))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string '{'.
    >>> print(clifford("{1"))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string '{1'.
    >>> print(clifford("+"))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string '+'.
    >>> print(clifford("-"))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string '-'.
    >>> print(clifford("{1}+"))
    Traceback (most recent call last):
      ...
    ValueError: Cannot initialize clifford object from invalid string '{1}+'.

    For clifford.__richcmp__: Compare objects of type clifford.

    >>> clifford("{1}") == clifford("1{1}")
    True
    >>> clifford("{1}") != clifford("1.0{1}")
    False
    >>> clifford("{1}") != clifford("1.0")
    True
    >>> clifford("{1,2}") == None
    False
    >>> clifford("{1,2}") != None
    True
    >>> None == clifford("{1,2}")
    False
    >>> None != clifford("{1,2}")
    True
    """
    return

cpdef inline error_squared_tol(obj):
    """
    Quadratic norm error tolerance relative to a specific multivector.

    >>> print(error_squared_tol(clifford("{1}")) * 3.0 - error_squared_tol(clifford("1{1}-2{2}+3{3}")))
    0.0
    """
    return glucat.error_squared_tol(toClifford(obj))

cpdef inline error_squared(lhs, rhs, threshold):
    """
    Relative or absolute error using the quadratic norm.

    >>> err2=scalar_epsilon*scalar_epsilon
```

```

    >>> print(error_squared(clifford("{1}"), clifford("1{1}"), err2))
    0.0
    >>> print(error_squared(clifford("1{1}-3{2}+4{3}"), clifford("{1}"), err2))
    25.0
    """
    return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)

cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
    """
    Test for approximate equality of multivectors.

    >>> err2=scalar_epsilon*scalar_epsilon

    >>> print(approx_equal(clifford("{1}"), clifford("1{1}")))
    True
    >>> print(approx_equal(clifford("1{1}-3{2}+4{3}"), clifford("{1}")))
    False
    >>> print(approx_equal(clifford("1{1}-3{2}+4{3}+0.001"), clifford("1{1}-3{2}+4{3}"), err2, err2))
    False
    >>> print(approx_equal(clifford("1{1}-3{2}+4{3}+1.0e-30"), clifford("1{1}-3{2}+4{3}"), err2, err2))
    True
    """
    threshold = error_squared_tol(rhs) if threshold is None else threshold
    tol       = error_squared_tol(rhs) if tol       is None else tol
    return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)

cpdef inline inv(obj):
    """
    Geometric multiplicative inverse.

    >>> print(inv(clifford("{1}")))
    {1}
    >>> print(inv(clifford("{-1}")))
    -{-1}
    >>> print(inv(clifford("{-2,-1}")))
    -{-2,-1}
    >>> print(inv(clifford("{-1}+{1}")))
    nan
    """
    return clifford(obj).inv()

cpdef inline scalar(obj):
    """
    Scalar part.

    >>> scalar(clifford("1+{1}+{1,2}"))
    1.0
    >>> scalar(clifford("{1,2}"))
    0.0
    """
    return clifford(obj).scalar()

cpdef inline real(obj):
    """
    Real part: synonym for scalar part.

    >>> real(clifford("1+{1}+{1,2}"))
    1.0
    >>> real(clifford("{1,2}"))
    0.0
    """
    return clifford(obj).scalar()

cpdef inline imag(obj):
    """
    Imaginary part: deprecated (always 0).

    >>> imag(clifford("1+{1}+{1,2}"))
    0.0
    >>> imag(clifford("{1,2}"))
    0.0
    """
    return 0.0

cpdef inline pure(obj):
    """
    Pure part

    >>> print(pure(clifford("1+{1}+{1,2}")))
    {1}+{1,2}
    >>> print(pure(clifford("{1,2}")))
    {1,2}
    """
    return clifford(obj).pure()

cpdef inline even(obj):
```

```
    """
    Even part of multivector, sum of even grade terms.

    >>> print(even(clifford("1+{1}+{1,2}")))
    1+{1,2}
    """
    return clifford(obj).even()

cpdef inline odd(obj):
    """
    Odd part of multivector, sum of odd grade terms.

    >>> print(odd(clifford("1+{1}+{1,2}")))
    {1}
    """
    return clifford(obj).odd()

cpdef inline involute(obj):
    """
    Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})

    >>> print(involute(clifford("{1}")))
    -{1}
    >>> print(involute(clifford("{2}") * clifford("{1}")))
    -{1,2}
    >>> print(involute(clifford("{1}") * clifford("{2}")))
    {1,2}
    >>> print(involute(clifford("1+{1}+{1,2}")))
    1-{1}+{1,2}
    """
    return clifford(obj).involute()

cpdef inline reverse(obj):
    """
    Reversion, eg. {1}*{2} -> {2}*{1}

    >>> print(reverse(clifford("{1}")))
    {1}
    >>> print(reverse(clifford("{2}") * clifford("{1}")))
    {1,2}
    >>> print(reverse(clifford("{1}") * clifford("{2}")))
    -{1,2}
    >>> print(reverse(clifford("1+{1}+{1,2}")))
    1+{1}-{1,2}
    """
    return clifford(obj).reverse()

cpdef inline conj(obj):
    """
    Conjugation, reverse o involute == involute o reverse.

    >>> print(conj(clifford("{1}")))
    -{1}
    >>> print(conj(clifford("{2}") * clifford("{1}")))
    {1,2}
    >>> print(conj(clifford("{1}") * clifford("{2}")))
    -{1,2}
    >>> print(conj(clifford("1+{1}+{1,2}")))
    1-{1}-{1,2}
    """
    return clifford(obj).conj()

cpdef inline quad(obj):
    """
    Quadratic form == (rev(x)*x)(0).

    >>> print(quad(clifford("1+{1}+{1,2}")))
    3.0
    >>> print(quad(clifford("1+{-1}+{1,2}+{1,2,3}")))
    2.0
    """
    return clifford(obj).quad()

cpdef inline norm(obj):
    """
    norm == sum of squares of coordinates.

    >>> norm(clifford("1+{1}+{1,2}"))
    3.0
    >>> norm(clifford("1+{-1}+{1,2}+{1,2,3}"))
    4.0
    """
    return clifford(obj).norm()

cpdef inline abs(obj):
    """
    Absolute value of multivector: multivector 2-norm.
```

```

    »> abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
    2.0
    """
    return glucat.abs(toClifford(obj))

cpdef inline max_abs(obj):
    """
    Maximum absolute value of coordinates multivector: multivector infinity-norm.

    »> max_abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
    1.0
    »> max_abs(clifford("3+2{1}+{1,2}"))
    3.0

    """
    return glucat.max_abs(toClifford(obj))

cpdef inline pow(obj, m):
    """
    Integer power of multivector: obj to the m.

    »> x=clifford("{1}"); print(pow(x,2))
    1
    »> x=clifford("2"); print(pow(x,2))
    4
    »> x=clifford("2+{1}"); print(pow(x,0))
    1
    »> x=clifford("2+{1}"); print(pow(x,1))
    2+{1}
    »> x=clifford("2+{1}"); print(pow(x,2))
    5+4{1}
    »> print(pow(clifford("1+{1}+{1,2}"),3))
    1+3{1}+3{1,2}
    »> i=clifford("{1,2}"); print(exp(pi/2) * pow(i, i))
    1
    """
    try:
        math.pow(obj, m)
    except:
        return clifford(obj).pow(m)

cpdef inline outer_pow(obj, m):
    """
    Outer product power of multivector.

    »> print(outer_pow(clifford("1+{1}+{1,2}"),3))
    1+3{1}+3{1,2}
    """
    return clifford(obj).outer_pow(m)

cpdef inline complexifier(obj):
    """
    Square root of -1 which commutes with all members of the frame of the given multivector.

    »> print(complexifier(clifford(index_set({1}))))
    {1,2,3}
    »> print(complexifier(clifford(index_set({-1}))))
    {-1}
    »> print(complexifier(index_set({1})))
    {1,2,3}
    »> print(complexifier(index_set({-1})))
    {-1}
    """
    return clifford().wrap( glucat.complexifier(toClifford(obj)) )

cpdef inline sqrt(obj, i = None):
    """
    Square root of multivector with optional complexifier.

    »> print(sqrt(-1))
    {-1}
    »> print(sqrt(clifford("2{-1}")))
    1+{-1}
    »> j=sqrt(-1,complexifier(index_set({1}))); print(j); print(j*j)
    {1,2,3}
    -1
    »> j=sqrt(-1,"{1,2,3}"); print(j); print(j*j)
    {1,2,3}
    -1
    """
    if not (i is None):
        return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.sqrt(obj)
        except:
```

```
            return clifford().wrap( glucat.sqrt(toClifford(obj)) )

cpdef inline exp(obj):
    """
    Exponential of multivector.

    >>> x=clifford("{1,2}") * pi/4; print(exp(x))
    0.7071+0.7071{1,2}
    >>> x=clifford("{1,2}") * pi/2; print(exp(x))
    {1,2}
    """
    try:
        return math.exp(obj)
    except:
        return clifford().wrap( glucat.exp(toClifford(obj)) )

cpdef inline log(obj,i = None):
    """
    Natural logarithm of multivector with optional complexifier.

    >>> x=clifford("{-1}"); print((log(x,"{-1}") * 2/pi))
    {-1}
    >>> x=clifford("{1,2}"); print((log(x,"{1,2,3}") * 2/pi))
    {1,2}
    >>> x=clifford("{1,2}"); print((log(x) * 2/pi))
    {1,2}
    >>> x=clifford("{1,2}"); print((log(x,"{1,2}") * 2/pi))
    Traceback (most recent call last):
    ...
    RuntimeError: check_complex(val, i): i is not a valid complexifier for val
    """
    if not (i is None):
        return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.log(obj)
        except:
            return clifford().wrap( glucat.log(toClifford(obj)) )

cpdef inline cos(obj,i = None):
    """
    Cosine of multivector with optional complexifier.

    >>> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
    {1,2}
    >>> x=clifford("{1,2}"); print(cos(acos(x)))
    {1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.cos(obj)
        except:
            return clifford().wrap( glucat.cos(toClifford(obj)) )

cpdef inline acos(obj,i = None):
    """
    Inverse cosine of multivector with optional complexifier.

    >>> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
    {1,2}
    >>> x=clifford("{1,2}"); print(cos(acos(x),"{-1,1,2,3,4}"))
    {1,2}
    >>> print(acos(0) / pi)
    0.5
    >>> x=clifford("{1,2}"); print(cos(acos(x)))
    {1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.acos(obj)
        except:
            return clifford().wrap( glucat.acos(toClifford(obj)) )

cpdef inline cosh(obj):
    """
    Hyperbolic cosine of multivector.

    >>> x=clifford("{1,2}") * pi; print(cosh(x))
    -1
    >>> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
    {1,2,3}
    >>> x=clifford("{1,2}"); print(cosh(acosh(x)))
    {1,2}
```

```
    """
    try:
        return math.cosh(obj)
    except:
        return clifford().wrap( glucat.cosh(toClifford(obj)) )

cpdef inline acosh(obj,i = None):
    """
    Inverse hyperbolic cosine of multivector with optional complexifier.

    >>> print(acosh(0,"{-2,-1,1}"))
    1.571{-2,-1,1}
    >>> x=clifford("{1,2,3}"); print(cosh(acosh(x,"{-1,1,2,3,4}")))
    {1,2,3}
    >>> print(acosh(0))
    1.571{-1}
    >>> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
    {1,2,3}
    >>> x=clifford("{1,2}"); print(cosh(acosh(x)))
    {1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.acosh(obj)
        except:
            return clifford().wrap( glucat.acosh(toClifford(obj)) )

cpdef inline sin(obj,i = None):
    """
    Sine of multivector with optional complexifier.

    >>> s="{-1}"; x=clifford(s); print(asin(sin(x,s),s))
    {-1}
    >>> s="{-1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
    {-1}
    >>> x=clifford("{1,2,3}"); print(asin(sin(x)))
    {1,2,3}
    """
    if not (i is None):
        return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.sin(obj)
        except:
            return clifford().wrap( glucat.sin(toClifford(obj)) )

cpdef inline asin(obj,i = None):
    """
    Inverse sine of multivector with optional complexifier.

    >>> s="{-1}"; x=clifford(s); print(asin(sin(x,s),s))
    {-1}
    >>> s="{-1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
    {-1}
    >>> print(asin(1) / pi)
    0.5
    >>> x=clifford("{1,2,3}"); print(asin(sin(x)))
    {1,2,3}
    """
    if not (i is None):
        return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.asin(obj)
        except:
            return clifford().wrap( glucat.asin(toClifford(obj)) )

cpdef inline sinh(obj):
    """
    Hyperbolic sine of multivector.

    >>> x=clifford("{1,2}") * pi/2; print(sinh(x))
    {1,2}
    >>> x=clifford("{1,2}") * pi/6; print(sinh(x))
    0.5{1,2}
    """
    try:
        return math.sinh(obj)
    except:
        return clifford().wrap( glucat.sinh(toClifford(obj)) )

cpdef inline asinh(obj,i = None):
    """
    Inverse hyperbolic sine of multivector with optional complexifier.

```

```
    »> x=clifford("{1,2}"); print(asinh(x,"{1,2,3}") * 2/pi)
    {1,2}
    »> x=clifford("{1,2}"); print(asinh(x) * 2/pi)
    {1,2}
    »> x=clifford("{1,2}") / 2; print(asinh(x) * 6/pi)
    {1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.asinh(obj)
        except:
            return clifford().wrap( glucat.asinh(toClifford(obj)) )

cpdef inline tan(obj,i = None):
    """
    Tangent of multivector with optional complexifier.

    »> x=clifford("{1,2}"); print(tan(x,"{1,2,3}"))
    0.7616{1,2}
    »> x=clifford("{1,2}"); print(tan(x))
    0.7616{1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.tan(obj)
        except:
            return clifford().wrap( glucat.tan(toClifford(obj)) )

cpdef inline atan(obj,i = None):
    """
    Inverse tangent of multivector with optional complexifier.

    »> s=index_set({1,2,3}); x=clifford("{1}"); print(tan(atan(x,s),s))
    {1}
    »> x=clifford("{1}"); print(tan(atan(x)))
    {1}
    """
    if not (i is None):
        return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.atan(obj)
        except:
            return clifford().wrap( glucat.atan(toClifford(obj)) )

cpdef inline tanh(obj):
    """
    Hyperbolic tangent of multivector.

    »> x=clifford("{1,2}") * pi/4; print(tanh(x))
    {1,2}
    """
    try:
        return math.tanh(obj)
    except:
        return clifford().wrap( glucat.tanh(toClifford(obj)) )

cpdef inline atanh(obj,i = None):
    """
    Inverse hyperbolic tangent of multivector with optional complexifier.

    »> s=index_set({1,2,3}); x=clifford("{1,2}"); print(tanh(atanh(x,s)))
    {1,2}
    »> x=clifford("{1,2}"); print(tanh(atanh(x)))
    {1,2}
    """
    if not (i is None):
        return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
    else:
        try:
            return math.atanh(obj)
        except:
            return clifford().wrap( glucat.atanh(toClifford(obj)) )

cpdef inline random_clifford(index_set ixt, fill = 1.0):
    """
    Random multivector within a frame.

    »> print(random_clifford(index_set({-3,-1,2})).frame())
    {-3,-1,2}
    """
    return clifford().wrap( clifford().instance.random(ixt.unwrap(), <scalar_t>fill) )
```

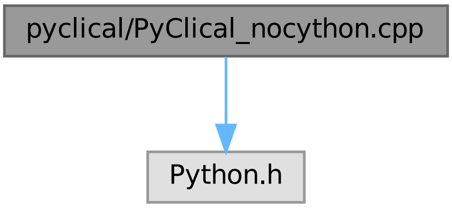
```
cpdef inline cga3(obj):
    """
    Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby definition.

    >>> x=clifford("2{1}+9{2}+{3}"); print(cga3(x))
    87{-1}+4{1}+18{2}+2{3}+85{4}
    """
    return clifford().wrap( glucat.cga3(toClifford(obj)) )

cpdef inline cga3std(obj):
    """
    Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.

    >>> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x)))
    87{-1}+4{1}+18{2}+2{3}+85{4}
    >>> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x))-cga3(x))
    0
    """
    return clifford().wrap( glucat.cga3std(toClifford(obj)) )

cpdef inline agc3(obj):
    """
    Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.

    >>> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x)))
    2{1}+9{2}+{3}
    >>> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x))-x)
    0
    """
    return clifford().wrap( glucat.agc3(toClifford(obj)) )

# Some abbreviations.
scalar_epsilon = epsilon

pi = atan(clifford(1.0)) * 4.0
tau = atan(clifford(1.0)) * 8.0

cl = clifford
"""
Abbreviation for clifford.

>>> print(cl(2))
2
>>> print(cl(2.0))
2
>>> print(cl(5.0e-1))
0.5
>>> print(cl("2"))
2
>>> print(cl("2{1,2,3}"))
2{1,2,3}
>>> print(cl(cl("2{1,2,3}")))
2{1,2,3}
"""

ist = index_set
"""
Abbreviation for index_set.

>>> print(ist("{1,2,3}"))
{1,2,3}
"""

def e(obj):
    """
    Abbreviation for clifford(index_set(obj)).

    >>> print(e(1))
    {1}
    >>> print(e(-1))
    {-1}
    >>> print(e(0))
    1
    """
    return clifford(index_set(obj))

def istpq(p, q):
    """
    Abbreviation for index_set({-q,...p}).

    >>> print(istpq(2,3))
    {-3,-2,-1,1,2}
    """
    return index_set(set(range(-q,p+1)))

ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
```

```
nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].

# Doctest interface.
def _test():
    import PyClical, doctest
    return doctest.testmod(PyClical)

if __name__ == "__main__":
    _test()
```

7.63 pyclical/PyClical_nocython.cpp File Reference

```
#include "Python.h"
```

Include dependency graph for PyClical_nocython.cpp:



Macros

- #define PY_SSIZE_T_CLEAN

7.63.1 Macro Definition Documentation

7.63.1.1 PY_SSIZE_T_CLEAN

```
#define PY_SSIZE_T_CLEAN
```

Definition at line 23 of file PyClical_nocython.cpp.

7.64 PyClical_nocython.cpp

Go to the documentation of this file.

```
/* Generated by Cython 0.29.28 */

/* BEGIN: Cython Metadata
{
    "distutils": {
        "depends": [
            "PyClical.h"
        ],
        "include_dirs": [
            "."
        ],
```

```
        "language": "c++",
        "name": "PyClical",
        "sources": [
            "PyClical.pyx"
        ]
    },
    "module_name": "PyClical"
}
END: Cython Metadata */

#ifndef PY_SSIZE_T_CLEAN
#define PY_SSIZE_T_CLEAN
#endif /* PY_SSIZE_T_CLEAN */
#include "Python.h"
#ifndef Py_PYTHON_H
    #error Python headers needed to compile C extensions, please install development version of Python.
#elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VERSION_HEX && PY_VERSION_HEX < 0x03030000)
    #error Cython requires Python 2.6+ or Python 3.3+.
#else
#define CYTHON_ABI "0_29_28"
#define CYTHON_HEX_VERSION 0x001D1CF0
#define CYTHON_FUTURE_DIVISION 1
#include <stddef.h>
#ifndef offsetof
  #define offsetof(type, member) ( (size_t) & ((type*)0) -> member )
#endif
#if !defined(WIN32) && !defined(MS_WINDOWS)
  #ifndef __stdcall
    #define __stdcall
  #endif
  #ifndef __cdecl
    #define __cdecl
  #endif
  #ifndef __fastcall
    #define __fastcall
  #endif
#endif
#ifndef DL_IMPORT
  #define DL_IMPORT(t) t
#endif
#ifndef DL_EXPORT
  #define DL_EXPORT(t) t
#endif
#define __PYX_COMMA ,
#ifndef HAVE_LONG_LONG
  #if PY_VERSION_HEX >= 0x02070000
    #define HAVE_LONG_LONG
  #endif
#endif
#ifndef PY_LONG_LONG
  #define PY_LONG_LONG LONG_LONG
#endif
#ifndef Py_HUGE_VAL
  #define Py_HUGE_VAL HUGE_VAL
#endif
#ifdef PYPY_VERSION
  #define CYTHON_COMPILING_IN_PYPY 1
  #define CYTHON_COMPILING_IN_PYSTON 0
  #define CYTHON_COMPILING_IN_CPYTHON 0
  #undef CYTHON_USE_TYPE_SLOTS
  #define CYTHON_USE_TYPE_SLOTS 0
  #undef CYTHON_USE_PYTYPE_LOOKUP
  #define CYTHON_USE_PYTYPE_LOOKUP 0
  #if PY_VERSION_HEX < 0x03050000
    #undef CYTHON_USE_ASYNC_SLOTS
    #define CYTHON_USE_ASYNC_SLOTS 0
  #elif !defined(CYTHON_USE_ASYNC_SLOTS)
    #define CYTHON_USE_ASYNC_SLOTS 1
  #endif
  #undef CYTHON_USE_PYLIST_INTERNALS
  #define CYTHON_USE_PYLIST_INTERNALS 0
  #undef CYTHON_USE_UNICODE_INTERNALS
  #define CYTHON_USE_UNICODE_INTERNALS 0
  #undef CYTHON_USE_UNICODE_WRITER
  #define CYTHON_USE_UNICODE_WRITER 0
  #undef CYTHON_USE_PYLONG_INTERNALS
  #define CYTHON_USE_PYLONG_INTERNALS 0
  #undef CYTHON_AVOID_BORROWED_REFS
  #define CYTHON_AVOID_BORROWED_REFS 1
  #undef CYTHON_ASSUME_SAFE_MACROS
  #define CYTHON_ASSUME_SAFE_MACROS 0
  #undef CYTHON_UNPACK_METHODS
  #define CYTHON_UNPACK_METHODS 0
  #undef CYTHON_FAST_THREAD_STATE
  #define CYTHON_FAST_THREAD_STATE 0
  #undef CYTHON_FAST_PYCALL
```

```
  #define CYTHON_FAST_PYCALL 0
  #undef CYTHON_PEP489_MULTI_PHASE_INIT
  #define CYTHON_PEP489_MULTI_PHASE_INIT 0
  #undef CYTHON_USE_TP_FINALIZE
  #define CYTHON_USE_TP_FINALIZE 0
  #undef CYTHON_USE_DICT_VERSIONS
  #define CYTHON_USE_DICT_VERSIONS 0
  #undef CYTHON_USE_EXC_INFO_STACK
  #define CYTHON_USE_EXC_INFO_STACK 0
#elif defined(PYSTON_VERSION)
  #define CYTHON_COMPILING_IN_PYPY 0
  #define CYTHON_COMPILING_IN_PYSTON 1
  #define CYTHON_COMPILING_IN_CPYTHON 0
  #ifndef CYTHON_USE_TYPE_SLOTS
    #define CYTHON_USE_TYPE_SLOTS 1
  #endif
  #undef CYTHON_USE_PYTYPE_LOOKUP
  #define CYTHON_USE_PYTYPE_LOOKUP 0
  #undef CYTHON_USE_ASYNC_SLOTS
  #define CYTHON_USE_ASYNC_SLOTS 0
  #undef CYTHON_USE_PYLIST_INTERNALS
  #define CYTHON_USE_PYLIST_INTERNALS 0
  #ifndef CYTHON_USE_UNICODE_INTERNALS
    #define CYTHON_USE_UNICODE_INTERNALS 1
  #endif
  #undef CYTHON_USE_UNICODE_WRITER
  #define CYTHON_USE_UNICODE_WRITER 0
  #undef CYTHON_USE_PYLONG_INTERNALS
  #define CYTHON_USE_PYLONG_INTERNALS 0
  #ifndef CYTHON_AVOID_BORROWED_REFS
    #define CYTHON_AVOID_BORROWED_REFS 0
  #endif
  #ifndef CYTHON_ASSUME_SAFE_MACROS
    #define CYTHON_ASSUME_SAFE_MACROS 1
  #endif
  #ifndef CYTHON_UNPACK_METHODS
    #define CYTHON_UNPACK_METHODS 1
  #endif
  #undef CYTHON_FAST_THREAD_STATE
  #define CYTHON_FAST_THREAD_STATE 0
  #undef CYTHON_FAST_PYCALL
  #define CYTHON_FAST_PYCALL 0
  #undef CYTHON_PEP489_MULTI_PHASE_INIT
  #define CYTHON_PEP489_MULTI_PHASE_INIT 0
  #undef CYTHON_USE_TP_FINALIZE
  #define CYTHON_USE_TP_FINALIZE 0
  #undef CYTHON_USE_DICT_VERSIONS
  #define CYTHON_USE_DICT_VERSIONS 0
  #undef CYTHON_USE_EXC_INFO_STACK
  #define CYTHON_USE_EXC_INFO_STACK 0
#else
  #define CYTHON_COMPILING_IN_PYPY 0
  #define CYTHON_COMPILING_IN_PYSTON 0
  #define CYTHON_COMPILING_IN_CPYTHON 1
  #ifndef CYTHON_USE_TYPE_SLOTS
    #define CYTHON_USE_TYPE_SLOTS 1
  #endif
  #if PY_VERSION_HEX < 0x02070000
    #undef CYTHON_USE_PYTYPE_LOOKUP
    #define CYTHON_USE_PYTYPE_LOOKUP 0
  #elif !defined(CYTHON_USE_PYTYPE_LOOKUP)
    #define CYTHON_USE_PYTYPE_LOOKUP 1
  #endif
  #if PY_MAJOR_VERSION < 3
    #undef CYTHON_USE_ASYNC_SLOTS
    #define CYTHON_USE_ASYNC_SLOTS 0
  #elif !defined(CYTHON_USE_ASYNC_SLOTS)
    #define CYTHON_USE_ASYNC_SLOTS 1
  #endif
  #if PY_VERSION_HEX < 0x02070000
    #undef CYTHON_USE_PYLONG_INTERNALS
    #define CYTHON_USE_PYLONG_INTERNALS 0
  #elif !defined(CYTHON_USE_PYLONG_INTERNALS)
    #define CYTHON_USE_PYLONG_INTERNALS 1
  #endif
  #ifndef CYTHON_USE_PYLIST_INTERNALS
    #define CYTHON_USE_PYLIST_INTERNALS 1
  #endif
  #ifndef CYTHON_USE_UNICODE_INTERNALS
    #define CYTHON_USE_UNICODE_INTERNALS 1
  #endif
  #if PY_VERSION_HEX < 0x030300F0 || PY_VERSION_HEX >= 0x030B00A2
    #undef CYTHON_USE_UNICODE_WRITER
    #define CYTHON_USE_UNICODE_WRITER 0
  #elif !defined(CYTHON_USE_UNICODE_WRITER)
    #define CYTHON_USE_UNICODE_WRITER 1
  #endif
```

```
  #ifndef CYTHON_AVOID_BORROWED_REFS
    #define CYTHON_AVOID_BORROWED_REFS 0
  #endif
  #ifndef CYTHON_ASSUME_SAFE_MACROS
    #define CYTHON_ASSUME_SAFE_MACROS 1
  #endif
  #ifndef CYTHON_UNPACK_METHODS
    #define CYTHON_UNPACK_METHODS 1
  #endif
  #if PY_VERSION_HEX >= 0x030B00A4
    #undef CYTHON_FAST_THREAD_STATE
    #define CYTHON_FAST_THREAD_STATE 0
  #elif !defined(CYTHON_FAST_THREAD_STATE)
    #define CYTHON_FAST_THREAD_STATE 1
  #endif
  #ifndef CYTHON_FAST_PYCALL
    #define CYTHON_FAST_PYCALL (PY_VERSION_HEX < 0x030B00A1)
  #endif
  #ifndef CYTHON_PEP489_MULTI_PHASE_INIT
    #define CYTHON_PEP489_MULTI_PHASE_INIT (PY_VERSION_HEX >= 0x03050000)
  #endif
  #ifndef CYTHON_USE_TP_FINALIZE
    #define CYTHON_USE_TP_FINALIZE (PY_VERSION_HEX >= 0x030400a1)
  #endif
  #ifndef CYTHON_USE_DICT_VERSIONS
    #define CYTHON_USE_DICT_VERSIONS (PY_VERSION_HEX >= 0x030600B1)
  #endif
  #if PY_VERSION_HEX >= 0x030B00A4
    #undef CYTHON_USE_EXC_INFO_STACK
    #define CYTHON_USE_EXC_INFO_STACK 0
  #elif !defined(CYTHON_USE_EXC_INFO_STACK)
    #define CYTHON_USE_EXC_INFO_STACK (PY_VERSION_HEX >= 0x030700A3)
  #endif
#endif
#if !defined(CYTHON_FAST_PYCCALL)
#define CYTHON_FAST_PYCCALL  (CYTHON_FAST_PYCALL && PY_VERSION_HEX >= 0x030600B1)
#endif
#if CYTHON_USE_PYLONG_INTERNALS
  #if PY_MAJOR_VERSION < 3
    #include "longintrepr.h"
  #endif
  #undef SHIFT
  #undef BASE
  #undef MASK
  #ifdef SIZEOF_VOID_P
    enum { __pyx_check_sizeof_voidp = 1 / (int)(SIZEOF_VOID_P == sizeof(void*)) };
  #endif
#endif
#ifndef __has_attribute
  #define __has_attribute(x) 0
#endif
#ifndef __has_cpp_attribute
  #define __has_cpp_attribute(x) 0
#endif
#ifndef CYTHON_RESTRICT
  #if defined(__GNUC__)
    #define CYTHON_RESTRICT __restrict__
  #elif defined(_MSC_VER) && _MSC_VER >= 1400
    #define CYTHON_RESTRICT __restrict
  #elif defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
    #define CYTHON_RESTRICT restrict
  #else
    #define CYTHON_RESTRICT
  #endif
#endif
#ifndef CYTHON_UNUSED
# if defined(__GNUC__)
#   if !(defined(__cplusplus)) || (__GNUC__ > 3 || (__GNUC__ == 3 && __GNUC_MINOR__ >= 4))
#     define CYTHON_UNUSED __attribute__ ((__unused__))
#   else
#     define CYTHON_UNUSED
#   endif
# elif defined(__ICC) || (defined(__INTEL_COMPILER) && !defined(_MSC_VER))
#   define CYTHON_UNUSED __attribute__ ((__unused__))
# else
#   define CYTHON_UNUSED
# endif
#endif
#ifndef CYTHON_MAYBE_UNUSED_VAR
#  if defined(__cplusplus)
     template<class T> void CYTHON_MAYBE_UNUSED_VAR( const T& ) { }
#  else
#    define CYTHON_MAYBE_UNUSED_VAR(x) (void)(x)
#  endif
#endif
#ifndef CYTHON_NCP_UNUSED
# if CYTHON_COMPILING_IN_CPYTHON
```

```
#  define CYTHON_NCP_UNUSED
# else
#  define CYTHON_NCP_UNUSED CYTHON_UNUSED
# endif
#endif
#define __Pyx_void_to_None(void_result) ((void)(void_result), Py_INCREF(Py_None), Py_None)
#ifdef _MSC_VER
    #ifndef _MSC_STDINT_H_
        #if _MSC_VER < 1300
           typedef unsigned char     uint8_t;
           typedef unsigned int      uint32_t;
        #else
           typedef unsigned __int8   uint8_t;
           typedef unsigned __int32  uint32_t;
        #endif
    #endif
#else
   #include <stdint.h>
#endif
#ifndef CYTHON_FALLTHROUGH
  #if defined(__cplusplus) && __cplusplus >= 201103L
    #if __has_cpp_attribute(fallthrough)
      #define CYTHON_FALLTHROUGH [[fallthrough]]
    #elif __has_cpp_attribute(clang::fallthrough)
      #define CYTHON_FALLTHROUGH [[clang::fallthrough]]
    #elif __has_cpp_attribute(gnu::fallthrough)
      #define CYTHON_FALLTHROUGH [[gnu::fallthrough]]
    #endif
  #endif
  #ifndef CYTHON_FALLTHROUGH
    #if __has_attribute(fallthrough)
      #define CYTHON_FALLTHROUGH __attribute__((fallthrough))
    #else
      #define CYTHON_FALLTHROUGH
    #endif
  #endif
  #if defined(__clang__ ) && defined(__apple_build_version__)
    #if __apple_build_version__ < 7000000
      #undef  CYTHON_FALLTHROUGH
      #define CYTHON_FALLTHROUGH
    #endif
  #endif
#endif

#ifndef __cplusplus
  #error "Cython files generated with the C++ option must be compiled with a C++ compiler."
#endif
#ifndef CYTHON_INLINE
  #if defined(__clang__)
    #define CYTHON_INLINE __inline__ __attribute__ ((__unused__))
  #else
    #define CYTHON_INLINE inline
  #endif
#endif
template<typename T>
void __Pyx_call_destructor(T& x) {
    x.~T();
}
template<typename T>
class __Pyx_FakeReference {
  public:
    __Pyx_FakeReference() : ptr(NULL) { }
    __Pyx_FakeReference(const T& ref) : ptr(const_cast<T*>(&ref)) { }
    T *operator->() { return ptr; }
    T *operator&() { return ptr; }
    operator T&() { return *ptr; }
    template<typename U> bool operator ==(U other) { return *ptr == other; }
    template<typename U> bool operator !=(U other) { return *ptr != other; }
  private:
    T *ptr;
};

#if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x02070600 && !defined(Py_OptimizeFlag)
  #define Py_OptimizeFlag 0
#endif
#define __PYX_BUILD_PY_SSIZE_T "n"
#define CYTHON_FORMAT_SSIZE_T "z"
#if PY_MAJOR_VERSION < 3
  #define __Pyx_BUILTIN_MODULE_NAME "__builtin__"
  #define __Pyx_PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)\
          PyCode_New(a+k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)
  #define __Pyx_DefaultClassType PyClass_Type
#else
  #define __Pyx_BUILTIN_MODULE_NAME "builtins"
  #define __Pyx_DefaultClassType PyType_Type
#if PY_VERSION_HEX >= 0x030B00A1
    static CYTHON_INLINE PyCodeObject* __Pyx_PyCode_New(int a, int k, int l, int s, int f,
```

```
                                                    PyObject *code, PyObject *c, PyObject* n, PyObject *v,
                                                    PyObject *fv, PyObject *cell, PyObject* fn,
                                                    PyObject *name, int fline, PyObject *lnos) {
        PyObject *kwds=NULL, *argcount=NULL, *posonlyargcount=NULL, *kwonlyargcount=NULL;
        PyObject *nlocals=NULL, *stacksize=NULL, *flags=NULL, *replace=NULL, *call_result=NULL, *empty=NULL;
        const char *fn_cstr=NULL;
        const char *name_cstr=NULL;
        PyCodeObject* co=NULL;
        PyObject *type, *value, *traceback;
        PyErr_Fetch(&type, &value, &traceback);
        if (!(kwds=PyDict_New())) goto end;
        if (!(argcount=PyLong_FromLong(a))) goto end;
        if (PyDict_SetItemString(kwds, "co_argcount", argcount) != 0) goto end;
        if (!(posonlyargcount=PyLong_FromLong(0))) goto end;
        if (PyDict_SetItemString(kwds, "co_posonlyargcount", posonlyargcount) != 0) goto end;
        if (!(kwonlyargcount=PyLong_FromLong(k))) goto end;
        if (PyDict_SetItemString(kwds, "co_kwonlyargcount", kwonlyargcount) != 0) goto end;
        if (!(nlocals=PyLong_FromLong(l))) goto end;
        if (PyDict_SetItemString(kwds, "co_nlocals", nlocals) != 0) goto end;
        if (!(stacksize=PyLong_FromLong(s))) goto end;
        if (PyDict_SetItemString(kwds, "co_stacksize", stacksize) != 0) goto end;
        if (!(flags=PyLong_FromLong(f))) goto end;
        if (PyDict_SetItemString(kwds, "co_flags", flags) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_code", code) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_consts", c) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_names", n) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_varnames", v) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_freevars", fv) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_cellvars", cell) != 0) goto end;
        if (PyDict_SetItemString(kwds, "co_linetable", lnos) != 0) goto end;
        if (!(fn_cstr=PyUnicode_AsUTF8AndSize(fn, NULL))) goto end;
        if (!(name_cstr=PyUnicode_AsUTF8AndSize(name, NULL))) goto end;
        if (!(co = PyCode_NewEmpty(fn_cstr, name_cstr, fline))) goto end;
        if (!(replace = PyObject_GetAttrString((PyObject*)co, "replace"))) goto cleanup_code_too;
        if (!(empty = PyTuple_New(0))) goto cleanup_code_too; // unfortunately __pyx_empty_tuple isn't available here
        if (!(call_result = PyObject_Call(replace, empty, kwds))) goto cleanup_code_too;
        Py_XDECREF((PyObject*)co);
        co = (PyCodeObject*)call_result;
        call_result = NULL;
        if (0) {
            cleanup_code_too:
            Py_XDECREF((PyObject*)co);
            co = NULL;
        }
        end:
        Py_XDECREF(kwds);
        Py_XDECREF(argcount);
        Py_XDECREF(posonlyargcount);
        Py_XDECREF(kwonlyargcount);
        Py_XDECREF(nlocals);
        Py_XDECREF(stacksize);
        Py_XDECREF(replace);
        Py_XDECREF(call_result);
        Py_XDECREF(empty);
        if (type) {
            PyErr_Restore(type, value, traceback);
        }
        return co;
    }
#else
  #define __Pyx_PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)\
          PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)
#endif
  #define __Pyx_DefaultClassType PyType_Type
#endif
#ifndef Py_TPFLAGS_CHECKTYPES
  #define Py_TPFLAGS_CHECKTYPES 0
#endif
#ifndef Py_TPFLAGS_HAVE_INDEX
  #define Py_TPFLAGS_HAVE_INDEX 0
#endif
#ifndef Py_TPFLAGS_HAVE_NEWBUFFER
  #define Py_TPFLAGS_HAVE_NEWBUFFER 0
#endif
#ifndef Py_TPFLAGS_HAVE_FINALIZE
  #define Py_TPFLAGS_HAVE_FINALIZE 0
#endif
#ifndef METH_STACKLESS
  #define METH_STACKLESS 0
#endif
#if PY_VERSION_HEX <= 0x030700A3 || !defined(METH_FASTCALL)
  #ifndef METH_FASTCALL
     #define METH_FASTCALL 0x80
  #endif
```

```
  typedef PyObject *(*__Pyx_PyCFunctionFast) (PyObject *self, PyObject *const *args, Py_ssize_t nargs);
  typedef PyObject *(*__Pyx_PyCFunctionFastWithKeywords) (PyObject *self, PyObject *const *args,
                                                          Py_ssize_t nargs, PyObject *kwnames);
#else
  #define __Pyx_PyCFunctionFast _PyCFunctionFast
  #define __Pyx_PyCFunctionFastWithKeywords _PyCFunctionFastWithKeywords
#endif
#if CYTHON_FAST_PYCCALL
#define __Pyx_PyFastCFunction_Check(func)\
    ((PyCFunction_Check(func) && (METH_FASTCALL == (PyCFunction_GET_FLAGS(func) & ~(METH_CLASS | METH_STATIC | METH_COEXIST | METH_KEYWORDS | METH_STACKLESS)))))
#else
#define __Pyx_PyFastCFunction_Check(func) 0
#endif
#if CYTHON_COMPILING_IN_PYPY && !defined(PyObject_Malloc)
  #define PyObject_Malloc(s)   PyMem_Malloc(s)
  #define PyObject_Free(p)     PyMem_Free(p)
  #define PyObject_Realloc(p)  PyMem_Realloc(p)
#endif
#if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX < 0x030400A1
  #define PyMem_RawMalloc(n)           PyMem_Malloc(n)
  #define PyMem_RawRealloc(p, n)       PyMem_Realloc(p, n)
  #define PyMem_RawFree(p)             PyMem_Free(p)
#endif
#if CYTHON_COMPILING_IN_PYSTON
  #define __Pyx_PyCode_HasFreeVars(co)  PyCode_HasFreeVars(co)
  #define __Pyx_PyFrame_SetLineNumber(frame, lineno) PyFrame_SetLineNumber(frame, lineno)
#else
  #define __Pyx_PyCode_HasFreeVars(co)  (PyCode_GetNumFree(co) > 0)
  #define __Pyx_PyFrame_SetLineNumber(frame, lineno)  (frame)->f_lineno = (lineno)
#endif
#if !CYTHON_FAST_THREAD_STATE || PY_VERSION_HEX < 0x02070000
  #define __Pyx_PyThreadState_Current PyThreadState_GET()
#elif PY_VERSION_HEX >= 0x03060000
  #define __Pyx_PyThreadState_Current _PyThreadState_UncheckedGet()
#elif PY_VERSION_HEX >= 0x03000000
  #define __Pyx_PyThreadState_Current PyThreadState_GET()
#else
  #define __Pyx_PyThreadState_Current _PyThreadState_Current
#endif
#if PY_VERSION_HEX < 0x030700A2 && !defined(PyThread_tss_create) && !defined(Py_tss_NEEDS_INIT)
#include "pythread.h"
#define Py_tss_NEEDS_INIT 0
typedef int Py_tss_t;
static CYTHON_INLINE int PyThread_tss_create(Py_tss_t *key) {
  *key = PyThread_create_key();
  return 0;
}
static CYTHON_INLINE Py_tss_t * PyThread_tss_alloc(void) {
  Py_tss_t *key = (Py_tss_t *)PyObject_Malloc(sizeof(Py_tss_t));
  *key = Py_tss_NEEDS_INIT;
  return key;
}
static CYTHON_INLINE void PyThread_tss_free(Py_tss_t *key) {
  PyObject_Free(key);
}
static CYTHON_INLINE int PyThread_tss_is_created(Py_tss_t *key) {
  return *key != Py_tss_NEEDS_INIT;
}
static CYTHON_INLINE void PyThread_tss_delete(Py_tss_t *key) {
  PyThread_delete_key(*key);
  *key = Py_tss_NEEDS_INIT;
}
static CYTHON_INLINE int PyThread_tss_set(Py_tss_t *key, void *value) {
  return PyThread_set_key_value(*key, value);
}
static CYTHON_INLINE void * PyThread_tss_get(Py_tss_t *key) {
  return PyThread_get_key_value(*key);
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON || defined(_PyDict_NewPresized)
#define __Pyx_PyDict_NewPresized(n)  ((n <= 8) ? PyDict_New() : _PyDict_NewPresized(n))
#else
#define __Pyx_PyDict_NewPresized(n)  PyDict_New()
#endif
#if PY_MAJOR_VERSION >= 3 || CYTHON_FUTURE_DIVISION
  #define __Pyx_PyNumber_Divide(x,y)         PyNumber_TrueDivide(x,y)
  #define __Pyx_PyNumber_InPlaceDivide(x,y)  PyNumber_InPlaceTrueDivide(x,y)
#else
  #define __Pyx_PyNumber_Divide(x,y)         PyNumber_Divide(x,y)
  #define __Pyx_PyNumber_InPlaceDivide(x,y)  PyNumber_InPlaceDivide(x,y)
#endif
#if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x030500A1 && CYTHON_USE_UNICODE_INTERNALS
#define __Pyx_PyDict_GetItemStr(dict, name)  _PyDict_GetItem_KnownHash(dict, name, ((PyASCIIObject *) name)->hash)
#else
```

```
#define __Pyx_PyDict_GetItemStr(dict, name)  PyDict_GetItem(dict, name)
#endif
#if PY_VERSION_HEX > 0x03030000 && defined(PyUnicode_KIND)
  #define CYTHON_PEP393_ENABLED 1
  #if defined(PyUnicode_IS_READY)
  #define __Pyx_PyUnicode_READY(op)       (likely(PyUnicode_IS_READY(op)) ?\
                                              0 : _PyUnicode_Ready((PyObject *)(op)))
  #else
  #define __Pyx_PyUnicode_READY(op)       (0)
  #endif
  #define __Pyx_PyUnicode_GET_LENGTH(u)   PyUnicode_GET_LENGTH(u)
  #define __Pyx_PyUnicode_READ_CHAR(u, i) PyUnicode_READ_CHAR(u, i)
  #define __Pyx_PyUnicode_MAX_CHAR_VALUE(u)   PyUnicode_MAX_CHAR_VALUE(u)
  #define __Pyx_PyUnicode_KIND(u)         PyUnicode_KIND(u)
  #define __Pyx_PyUnicode_DATA(u)         PyUnicode_DATA(u)
  #define __Pyx_PyUnicode_READ(k, d, i)   PyUnicode_READ(k, d, i)
  #define __Pyx_PyUnicode_WRITE(k, d, i, ch)  PyUnicode_WRITE(k, d, i, ch)
  #if defined(PyUnicode_IS_READY) && defined(PyUnicode_GET_SIZE)
  #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03090000
  #define __Pyx_PyUnicode_IS_TRUE(u)      (0 != (likely(PyUnicode_IS_READY(u)) ?
PyUnicode_GET_LENGTH(u) : ((PyCompactUnicodeObject *)(u))->wstr_length))
  #else
  #define __Pyx_PyUnicode_IS_TRUE(u)      (0 != (likely(PyUnicode_IS_READY(u)) ?
PyUnicode_GET_LENGTH(u) : PyUnicode_GET_SIZE(u)))
  #endif
  #else
  #define __Pyx_PyUnicode_IS_TRUE(u)      (0 != PyUnicode_GET_LENGTH(u))
  #endif
#else
  #define CYTHON_PEP393_ENABLED 0
  #define PyUnicode_1BYTE_KIND  1
  #define PyUnicode_2BYTE_KIND  2
  #define PyUnicode_4BYTE_KIND  4
  #define __Pyx_PyUnicode_READY(op)       (0)
  #define __Pyx_PyUnicode_GET_LENGTH(u)   PyUnicode_GET_SIZE(u)
  #define __Pyx_PyUnicode_READ_CHAR(u, i) ((Py_UCS4)(PyUnicode_AS_UNICODE(u)[i]))
  #define __Pyx_PyUnicode_MAX_CHAR_VALUE(u)   ((sizeof(Py_UNICODE) == 2) ? 65535 : 1114111)
  #define __Pyx_PyUnicode_KIND(u)         (sizeof(Py_UNICODE))
  #define __Pyx_PyUnicode_DATA(u)         ((void*)PyUnicode_AS_UNICODE(u))
  #define __Pyx_PyUnicode_READ(k, d, i)   ((void)(k), (Py_UCS4)(((Py_UNICODE*)d)[i]))
  #define __Pyx_PyUnicode_WRITE(k, d, i, ch)  (((void)(k)), ((Py_UNICODE*)d)[i] = ch)
  #define __Pyx_PyUnicode_IS_TRUE(u)      (0 != PyUnicode_GET_SIZE(u))
#endif
#if CYTHON_COMPILING_IN_PYPY
  #define __Pyx_PyUnicode_Concat(a, b)      PyNumber_Add(a, b)
  #define __Pyx_PyUnicode_ConcatSafe(a, b)  PyNumber_Add(a, b)
#else
  #define __Pyx_PyUnicode_Concat(a, b)      PyUnicode_Concat(a, b)
  #define __Pyx_PyUnicode_ConcatSafe(a, b)  ((unlikely((a) == Py_None) || unlikely((b) == Py_None)) ?\
      PyNumber_Add(a, b) : __Pyx_PyUnicode_Concat(a, b))
#endif
#if CYTHON_COMPILING_IN_PYPY && !defined(PyUnicode_Contains)
  #define PyUnicode_Contains(u, s)  PySequence_Contains(u, s)
#endif
#if CYTHON_COMPILING_IN_PYPY && !defined(PyByteArray_Check)
  #define PyByteArray_Check(obj)  PyObject_TypeCheck(obj, &PyByteArray_Type)
#endif
#if CYTHON_COMPILING_IN_PYPY && !defined(PyObject_Format)
  #define PyObject_Format(obj, fmt)  PyObject_CallMethod(obj, "__format__", "O", fmt)
#endif
#define __Pyx_PyString_FormatSafe(a, b)   ((unlikely((a) == Py_None || (PyString_Check(b) &&
!PyString_CheckExact(b)))) ? PyNumber_Remainder(a, b) : __Pyx_PyString_Format(a, b))
#define __Pyx_PyUnicode_FormatSafe(a, b)  ((unlikely((a) == Py_None || (PyUnicode_Check(b) &&
!PyUnicode_CheckExact(b)))) ? PyNumber_Remainder(a, b) : PyUnicode_Format(a, b))
#if PY_MAJOR_VERSION >= 3
  #define __Pyx_PyString_Format(a, b)  PyUnicode_Format(a, b)
#else
  #define __Pyx_PyString_Format(a, b)  PyString_Format(a, b)
#endif
#if PY_MAJOR_VERSION < 3 && !defined(PyObject_ASCII)
  #define PyObject_ASCII(o)            PyObject_Repr(o)
#endif
#if PY_MAJOR_VERSION >= 3
  #define PyBaseString_Type            PyUnicode_Type
  #define PyStringObject               PyUnicodeObject
  #define PyString_Type                PyUnicode_Type
  #define PyString_Check               PyUnicode_Check
  #define PyString_CheckExact          PyUnicode_CheckExact
#ifndef PyObject_Unicode
  #define PyObject_Unicode             PyObject_Str
#endif
#endif
#if PY_MAJOR_VERSION >= 3
  #define __Pyx_PyBaseString_Check(obj) PyUnicode_Check(obj)
  #define __Pyx_PyBaseString_CheckExact(obj) PyUnicode_CheckExact(obj)
#else
  #define __Pyx_PyBaseString_Check(obj) (PyString_Check(obj) || PyUnicode_Check(obj))
```

```
  #define __Pyx_PyBaseString_CheckExact(obj) (PyString_CheckExact(obj) || PyUnicode_CheckExact(obj))
#endif
#ifndef PySet_CheckExact
  #define PySet_CheckExact(obj)        (Py_TYPE(obj) == &PySet_Type)
#endif
#if PY_VERSION_HEX >= 0x030900A4
  #define __Pyx_SET_REFCNT(obj, refcnt) Py_SET_REFCNT(obj, refcnt)
  #define __Pyx_SET_SIZE(obj, size) Py_SET_SIZE(obj, size)
#else
  #define __Pyx_SET_REFCNT(obj, refcnt) Py_REFCNT(obj) = (refcnt)
  #define __Pyx_SET_SIZE(obj, size) Py_SIZE(obj) = (size)
#endif
#if CYTHON_ASSUME_SAFE_MACROS
  #define __Pyx_PySequence_SIZE(seq)  Py_SIZE(seq)
#else
  #define __Pyx_PySequence_SIZE(seq)  PySequence_Size(seq)
#endif
#if PY_MAJOR_VERSION >= 3
  #define PyIntObject                  PyLongObject
  #define PyInt_Type                   PyLong_Type
  #define PyInt_Check(op)              PyLong_Check(op)
  #define PyInt_CheckExact(op)         PyLong_CheckExact(op)
  #define PyInt_FromString             PyLong_FromString
  #define PyInt_FromUnicode            PyLong_FromUnicode
  #define PyInt_FromLong               PyLong_FromLong
  #define PyInt_FromSize_t             PyLong_FromSize_t
  #define PyInt_FromSsize_t            PyLong_FromSsize_t
  #define PyInt_AsLong                 PyLong_AsLong
  #define PyInt_AS_LONG                PyLong_AS_LONG
  #define PyInt_AsSsize_t              PyLong_AsSsize_t
  #define PyInt_AsUnsignedLongMask     PyLong_AsUnsignedLongMask
  #define PyInt_AsUnsignedLongLongMask PyLong_AsUnsignedLongLongMask
  #define PyNumber_Int                 PyNumber_Long
#endif
#if PY_MAJOR_VERSION >= 3
  #define PyBoolObject                 PyLongObject
#endif
#if PY_MAJOR_VERSION >= 3 && CYTHON_COMPILING_IN_PYPY
  #ifndef PyUnicode_InternFromString
    #define PyUnicode_InternFromString(s) PyUnicode_FromString(s)
  #endif
#endif
#if PY_VERSION_HEX < 0x030200A4
  typedef long Py_hash_t;
  #define __Pyx_PyInt_FromHash_t PyInt_FromLong
  #define __Pyx_PyInt_AsHash_t   __Pyx_PyIndex_AsHash_t
#else
  #define __Pyx_PyInt_FromHash_t PyInt_FromSsize_t
  #define __Pyx_PyInt_AsHash_t   __Pyx_PyIndex_AsSsize_t
#endif
#if PY_MAJOR_VERSION >= 3
  #define __Pyx_PyMethod_New(func, self, klass) ((self) ? ((void)(klass), PyMethod_New(func, self)) :
__Pyx_NewRef(func))
#else
  #define __Pyx_PyMethod_New(func, self, klass) PyMethod_New(func, self, klass)
#endif
#if CYTHON_USE_ASYNC_SLOTS
  #if PY_VERSION_HEX >= 0x030500B1
    #define __Pyx_PyAsyncMethodsStruct PyAsyncMethods
    #define __Pyx_PyType_AsAsync(obj) (Py_TYPE(obj)->tp_as_async)
  #else
    #define __Pyx_PyType_AsAsync(obj) ((__Pyx_PyAsyncMethodsStruct*) (Py_TYPE(obj)->tp_reserved))
  #endif
#else
  #define __Pyx_PyType_AsAsync(obj) NULL
#endif
#ifndef __Pyx_PyAsyncMethodsStruct
    typedef struct {
        unaryfunc am_await;
        unaryfunc am_aiter;
        unaryfunc am_anext;
    } __Pyx_PyAsyncMethodsStruct;
#endif

#if defined(WIN32) || defined(MS_WINDOWS)
  #define _USE_MATH_DEFINES
#endif
#include <math.h>
#ifdef NAN
#define __PYX_NAN() ((float) NAN)
#else
static CYTHON_INLINE float __PYX_NAN() {
  float value;
  memset(&value, 0xFF, sizeof(value));
  return value;
}
#endif
```

```
#if defined(__CYGWIN__) && defined(_LDBL_EQ_DBL)
#define __Pyx_truncl trunc
#else
#define __Pyx_truncl truncl
#endif

#define __PYX_MARK_ERR_POS(f_index, lineno) \
    { __pyx_filename = __pyx_f[f_index]; (void)__pyx_filename; __pyx_lineno = lineno; (void)__pyx_lineno; __pyx_clineno = __LINE__; (void)__pyx_clineno; }
#define __PYX_ERR(f_index, lineno, Ln_error) \
    { __PYX_MARK_ERR_POS(f_index, lineno) goto Ln_error; }

#ifndef __PYX_EXTERN_C
  #ifdef __cplusplus
    #define __PYX_EXTERN_C extern "C"
  #else
    #define __PYX_EXTERN_C extern
  #endif
#endif

#define __PYX_HAVE__PyClical
#define __PYX_HAVE_API__PyClical
/* Early includes */
#include "ios"
#include "new"
#include "stdexcept"
#include "typeinfo"
#include <vector>
#include "PyClical.h"
#include <string.h>
#include <string>
#ifdef _OPENMP
#include <omp.h>
#endif /* _OPENMP */

#if defined(PYREX_WITHOUT_ASSERTIONS) && !defined(CYTHON_WITHOUT_ASSERTIONS)
#define CYTHON_WITHOUT_ASSERTIONS
#endif

typedef struct {PyObject **p; const char *s; const Py_ssize_t n; const char* encoding;
                const char is_unicode; const char is_str; const char intern; } __Pyx_StringTabEntry;

#define __PYX_DEFAULT_STRING_ENCODING_IS_ASCII 0
#define __PYX_DEFAULT_STRING_ENCODING_IS_UTF8 0
#define __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT (PY_MAJOR_VERSION >= 3 && __PYX_DEFAULT_STRING_ENCODING_IS_UTF8)
#define __PYX_DEFAULT_STRING_ENCODING ""
#define __Pyx_PyObject_FromString __Pyx_PyBytes_FromString
#define __Pyx_PyObject_FromStringAndSize __Pyx_PyBytes_FromStringAndSize
#define __Pyx_uchar_cast(c) ((unsigned char)c)
#define __Pyx_long_cast(x) ((long)x)
#define __Pyx_fits_Py_ssize_t(v, type, is_signed)  (\
    (sizeof(type) < sizeof(Py_ssize_t))  ||\
    (sizeof(type) > sizeof(Py_ssize_t) &&\
          likely(v < (type)PY_SSIZE_T_MAX ||\
                 v == (type)PY_SSIZE_T_MAX)  &&\
          (!is_signed || likely(v > (type)PY_SSIZE_T_MIN ||\
                                v == (type)PY_SSIZE_T_MIN)))  ||\
    (sizeof(type) == sizeof(Py_ssize_t) &&\
          (is_signed || likely(v < (type)PY_SSIZE_T_MAX ||\
                               v == (type)PY_SSIZE_T_MAX)))  )
static CYTHON_INLINE int __Pyx_is_valid_index(Py_ssize_t i, Py_ssize_t limit) {
    return (size_t) i < (size_t) limit;
}
#if defined (__cplusplus) && __cplusplus >= 201103L
    #include <cstdlib>
    #define __Pyx_sst_abs(value) std::abs(value)
#elif SIZEOF_INT >= SIZEOF_SIZE_T
    #define __Pyx_sst_abs(value) abs(value)
#elif SIZEOF_LONG >= SIZEOF_SIZE_T
    #define __Pyx_sst_abs(value) labs(value)
#elif defined (_MSC_VER)
    #define __Pyx_sst_abs(value) ((Py_ssize_t)_abs64(value))
#elif defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
    #define __Pyx_sst_abs(value) llabs(value)
#elif defined (__GNUC__)
    #define __Pyx_sst_abs(value) __builtin_llabs(value)
#else
    #define __Pyx_sst_abs(value) ((value<0) ? -value : value)
#endif
static CYTHON_INLINE const char* __Pyx_PyObject_AsString(PyObject*);
static CYTHON_INLINE const char* __Pyx_PyObject_AsStringAndSize(PyObject*, Py_ssize_t* length);
#define __Pyx_PyByteArray_FromString(s) PyByteArray_FromStringAndSize((const char*)s, strlen((const char*)s))
#define __Pyx_PyByteArray_FromStringAndSize(s, l) PyByteArray_FromStringAndSize((const char*)s, l)
#define __Pyx_PyBytes_FromString        PyBytes_FromString
#define __Pyx_PyBytes_FromStringAndSize PyBytes_FromStringAndSize
```

```
static CYTHON_INLINE PyObject* __Pyx_PyUnicode_FromString(const char*);
#if PY_MAJOR_VERSION < 3
    #define __Pyx_PyStr_FromString        __Pyx_PyBytes_FromString
    #define __Pyx_PyStr_FromStringAndSize __Pyx_PyBytes_FromStringAndSize
#else
    #define __Pyx_PyStr_FromString        __Pyx_PyUnicode_FromString
    #define __Pyx_PyStr_FromStringAndSize __Pyx_PyUnicode_FromStringAndSize
#endif
#define __Pyx_PyBytes_AsWritableString(s)     ((char*) PyBytes_AS_STRING(s))
#define __Pyx_PyBytes_AsWritableSString(s)    ((signed char*) PyBytes_AS_STRING(s))
#define __Pyx_PyBytes_AsWritableUString(s)    ((unsigned char*) PyBytes_AS_STRING(s))
#define __Pyx_PyBytes_AsString(s)     ((const char*) PyBytes_AS_STRING(s))
#define __Pyx_PyBytes_AsSString(s)    ((const signed char*) PyBytes_AS_STRING(s))
#define __Pyx_PyBytes_AsUString(s)    ((const unsigned char*) PyBytes_AS_STRING(s))
#define __Pyx_PyObject_AsWritableString(s)    ((char*) __Pyx_PyObject_AsString(s))
#define __Pyx_PyObject_AsWritableSString(s)    ((signed char*) __Pyx_PyObject_AsString(s))
#define __Pyx_PyObject_AsWritableUString(s)    ((unsigned char*) __Pyx_PyObject_AsString(s))
#define __Pyx_PyObject_AsSString(s)    ((const signed char*) __Pyx_PyObject_AsString(s))
#define __Pyx_PyObject_AsUString(s)    ((const unsigned char*) __Pyx_PyObject_AsString(s))
#define __Pyx_PyObject_FromCString(s)  __Pyx_PyObject_FromString((const char*)s)
#define __Pyx_PyBytes_FromCString(s)   __Pyx_PyBytes_FromString((const char*)s)
#define __Pyx_PyByteArray_FromCString(s)   __Pyx_PyByteArray_FromString((const char*)s)
#define __Pyx_PyStr_FromCString(s)     __Pyx_PyStr_FromString((const char*)s)
#define __Pyx_PyUnicode_FromCString(s) __Pyx_PyUnicode_FromString((const char*)s)
static CYTHON_INLINE size_t __Pyx_Py_UNICODE_strlen(const Py_UNICODE *u) {
    const Py_UNICODE *u_end = u;
    while (*u_end++) ;
    return (size_t)(u_end - u - 1);
}
#define __Pyx_PyUnicode_FromUnicode(u)       PyUnicode_FromUnicode(u, __Pyx_Py_UNICODE_strlen(u))
#define __Pyx_PyUnicode_FromUnicodeAndLength PyUnicode_FromUnicode
#define __Pyx_PyUnicode_AsUnicode            PyUnicode_AsUnicode
#define __Pyx_NewRef(obj) (Py_INCREF(obj), obj)
#define __Pyx_Owned_Py_None(b) __Pyx_NewRef(Py_None)
static CYTHON_INLINE PyObject * __Pyx_PyBool_FromLong(long b);
static CYTHON_INLINE int __Pyx_PyObject_IsTrue(PyObject*);
static CYTHON_INLINE int __Pyx_PyObject_IsTrueAndDecref(PyObject*);
static CYTHON_INLINE PyObject* __Pyx_PyNumber_IntOrLong(PyObject* x);
#define __Pyx_PySequence_Tuple(obj)\
    (likely(PyTuple_CheckExact(obj)) ? __Pyx_NewRef(obj) : PySequence_Tuple(obj))
static CYTHON_INLINE Py_ssize_t __Pyx_PyIndex_AsSsize_t(PyObject*);
static CYTHON_INLINE PyObject * __Pyx_PyInt_FromSize_t(size_t);
static CYTHON_INLINE Py_hash_t __Pyx_PyIndex_AsHash_t(PyObject*);
#if CYTHON_ASSUME_SAFE_MACROS
#define __pyx_PyFloat_AsDouble(x) (PyFloat_CheckExact(x) ? PyFloat_AS_DOUBLE(x) : PyFloat_AsDouble(x))
#else
#define __pyx_PyFloat_AsDouble(x) PyFloat_AsDouble(x)
#endif
#define __pyx_PyFloat_AsFloat(x) ((float) __pyx_PyFloat_AsDouble(x))
#if PY_MAJOR_VERSION >= 3
#define __Pyx_PyNumber_Int(x) (PyLong_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Long(x))
#else
#define __Pyx_PyNumber_Int(x) (PyInt_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Int(x))
#endif
#define __Pyx_PyNumber_Float(x) (PyFloat_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Float(x))
#if PY_MAJOR_VERSION < 3 && __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
static int __Pyx_sys_getdefaultencoding_not_ascii;
static int __Pyx_init_sys_getdefaultencoding_params(void) {
    PyObject* sys;
    PyObject* default_encoding = NULL;
    PyObject* ascii_chars_u = NULL;
    PyObject* ascii_chars_b = NULL;
    const char* default_encoding_c;
    sys = PyImport_ImportModule("sys");
    if (!sys) goto bad;
    default_encoding = PyObject_CallMethod(sys, (char*) "getdefaultencoding", NULL);
    Py_DECREF(sys);
    if (!default_encoding) goto bad;
    default_encoding_c = PyBytes_AsString(default_encoding);
    if (!default_encoding_c) goto bad;
    if (strcmp(default_encoding_c, "ascii") == 0) {
        __Pyx_sys_getdefaultencoding_not_ascii = 0;
    } else {
        char ascii_chars[128];
        int c;
        for (c = 0; c < 128; c++) {
            ascii_chars[c] = c;
        }
        __Pyx_sys_getdefaultencoding_not_ascii = 1;
        ascii_chars_u = PyUnicode_DecodeASCII(ascii_chars, 128, NULL);
        if (!ascii_chars_u) goto bad;
        ascii_chars_b = PyUnicode_AsEncodedString(ascii_chars_u, default_encoding_c, NULL);
        if (!ascii_chars_b || !PyBytes_Check(ascii_chars_b) || memcmp(ascii_chars,
PyBytes_AS_STRING(ascii_chars_b), 128) != 0) {
            PyErr_Format(
                PyExc_ValueError,
                "This module compiled with c_string_encoding=ascii, but default encoding '%.200s' is
```

```
not a superset of ascii.",
                    default_encoding_c);
                goto bad;
            }
            Py_DECREF(ascii_chars_u);
            Py_DECREF(ascii_chars_b);
        }
        Py_DECREF(default_encoding);
        return 0;
bad:
        Py_XDECREF(default_encoding);
        Py_XDECREF(ascii_chars_u);
        Py_XDECREF(ascii_chars_b);
        return -1;
}
#endif
#if __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT && PY_MAJOR_VERSION >= 3
#define __Pyx_PyUnicode_FromStringAndSize(c_str, size) PyUnicode_DecodeUTF8(c_str, size, NULL)
#else
#define __Pyx_PyUnicode_FromStringAndSize(c_str, size) PyUnicode_Decode(c_str, size,
__PYX_DEFAULT_STRING_ENCODING, NULL)
#if __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
static char* __PYX_DEFAULT_STRING_ENCODING;
static int __Pyx_init_sys_getdefaultencoding_params(void) {
    PyObject* sys;
    PyObject* default_encoding = NULL;
    char* default_encoding_c;
    sys = PyImport_ImportModule("sys");
    if (!sys) goto bad;
    default_encoding = PyObject_CallMethod(sys, (char*) (const char*) "getdefaultencoding", NULL);
    Py_DECREF(sys);
    if (!default_encoding) goto bad;
    default_encoding_c = PyBytes_AsString(default_encoding);
    if (!default_encoding_c) goto bad;
    __PYX_DEFAULT_STRING_ENCODING = (char*) malloc(strlen(default_encoding_c) + 1);
    if (!__PYX_DEFAULT_STRING_ENCODING) goto bad;
    strcpy(__PYX_DEFAULT_STRING_ENCODING, default_encoding_c);
    Py_DECREF(default_encoding);
    return 0;
bad:
    Py_XDECREF(default_encoding);
    return -1;
}
#endif
#endif


/* Test for GCC > 2.95 */
#if defined(__GNUC__)     && (__GNUC__ > 2 || (__GNUC__ == 2 && (__GNUC_MINOR__ > 95)))
  #define likely(x)   __builtin_expect(!!(x), 1)
  #define unlikely(x) __builtin_expect(!!(x), 0)
#else /* !__GNUC__ or GCC < 2.95 */
  #define likely(x)   (x)
  #define unlikely(x) (x)
#endif /* __GNUC__ */
static CYTHON_INLINE void __Pyx_pretend_to_initialize(void* ptr) { (void)ptr; }

static PyObject *__pyx_m = NULL;
static PyObject *__pyx_d;
static PyObject *__pyx_b;
static PyObject *__pyx_cython_runtime = NULL;
static PyObject *__pyx_empty_tuple;
static PyObject *__pyx_empty_bytes;
static PyObject *__pyx_empty_unicode;
static int __pyx_lineno;
static int __pyx_clineno = 0;
static const char * __pyx_cfilenm= __FILE__;
static const char *__pyx_filename;


static const char *__pyx_f[] = {
  "PyClical.pyx",
  "stringsource",
};

/*--- Type declarations ---*/
struct __pyx_obj_8PyClical_index_set;
struct __pyx_obj_8PyClical_clifford;
struct __pyx_obj_8PyClical___pyx_scope_struct____iter__;
struct __pyx_opt_args_8PyClical_approx_equal;
struct __pyx_opt_args_8PyClical_sqrt;
struct __pyx_opt_args_8PyClical_log;
struct __pyx_opt_args_8PyClical_cos;
struct __pyx_opt_args_8PyClical_acos;
struct __pyx_opt_args_8PyClical_acosh;
struct __pyx_opt_args_8PyClical_sin;
struct __pyx_opt_args_8PyClical_asin;
```

```
struct __pyx_opt_args_8PyClical_asinh;
struct __pyx_opt_args_8PyClical_tan;
struct __pyx_opt_args_8PyClical_atan;
struct __pyx_opt_args_8PyClical_atanh;
struct __pyx_opt_args_8PyClical_random_clifford;

/* "PyClical.pyx":1359
 *     return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
 *
 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):             # «««««««
 *     """
 *     Test for approximate equality of multivectors.
 */
struct __pyx_opt_args_8PyClical_approx_equal {
  int __pyx_n;
  PyObject *threshold;
  PyObject *tol;
};

/* "PyClical.pyx":1591
 *     return clifford().wrap( glucat.complexifier(toClifford(obj)) )
 *
 * cpdef inline sqrt(obj, i = None):              # «««««««
 *     """
 *     Square root of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_sqrt {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1628
 *         return clifford().wrap( glucat.exp(toClifford(obj)) )
 *
 * cpdef inline log(obj,i = None):              # «««««««
 *     """
 *     Natural logarithm of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_log {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1651
 *             return clifford().wrap( glucat.log(toClifford(obj)) )
 *
 * cpdef inline cos(obj,i = None):              # «««««««
 *     """
 *     Cosine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_cos {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1668
 *             return clifford().wrap( glucat.cos(toClifford(obj)) )
 *
 * cpdef inline acos(obj,i = None):              # «««««««
 *     """
 *     Inverse cosine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_acos {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1705
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )
 *
 * cpdef inline acosh(obj,i = None):              # «««««««
 *     """
 *     Inverse hyperbolic cosine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_acosh {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1728
 *             return clifford().wrap( glucat.acosh(toClifford(obj)) )
 *
 * cpdef inline sin(obj,i = None):              # «««««««
 *     """
 *     Sine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_sin {
```

```
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1747
 *             return clifford().wrap( glucat.sin(toClifford(obj)) )
 *
 * cpdef inline asin(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse sine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_asin {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1782
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )
 *
 * cpdef inline asinh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic sine of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_asinh {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1801
 *             return clifford().wrap( glucat.asinh(toClifford(obj)) )
 *
 * cpdef inline tan(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Tangent of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_tan {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1818
 *             return clifford().wrap( glucat.tan(toClifford(obj)) )
 *
 * cpdef inline atan(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse tangent of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_atan {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1847
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )
 *
 * cpdef inline atanh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic tangent of multivector with optional complexifier.
 */
struct __pyx_opt_args_8PyClical_atanh {
  int __pyx_n;
  PyObject *i;
};

/* "PyClical.pyx":1864
 *             return clifford().wrap( glucat.atanh(toClifford(obj)) )
 *
 * cpdef inline random_clifford(index_set ixt, fill = 1.0):             # <<<<<<<<<<<<<<
 *     """
 *     Random multivector within a frame.
 */
struct __pyx_opt_args_8PyClical_random_clifford {
  int __pyx_n;
  PyObject *fill;
};

/* "PyClical.pyx":38
 *
 * # Forward reference
 * cdef class index_set             # <<<<<<<<<<<<<<
 *
 * cdef inline IndexSet toIndexSet(obj):
 */
struct __pyx_obj_8PyClical_index_set {
  PyObject_HEAD
  struct __pyx_vtabstruct_8PyClical_index_set *__pyx_vtab;
  IndexSet *instance;
```

```
};


/* "PyClical.pyx":532
 *
 * # Forward reference.
 * cdef class clifford             # «««««««
 *
 * cdef inline Clifford toClifford(obj):
 */
struct __pyx_obj_8PyClical_clifford {
  PyObject_HEAD
  struct __pyx_vtabstruct_8PyClical_clifford *__pyx_vtab;
  Clifford *instance;
};


/* "PyClical.pyx":229
 *         return self.instance.getitem(idx)
 *
 *     def __iter__(self):             # «««««««
 *         """
 *         Iterate over the indices of an index_set.
 */
struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ {
  PyObject_HEAD
  PyObject *__pyx_v_idx;
  struct __pyx_obj_8PyClical_index_set *__pyx_v_self;
  PyObject *__pyx_t_0;
  Py_ssize_t __pyx_t_1;
  PyObject *(*__pyx_t_2)(PyObject *);
};



/* "PyClical.pyx":46
 *     return index_set(obj).instance[0]
 *
 * cdef class index_set:             # «««««««
 *     """
 *     Python class index_set wraps C++ class IndexSet.
 */

struct __pyx_vtabstruct_8PyClical_index_set {
  PyObject *(*wrap)(struct __pyx_obj_8PyClical_index_set *, IndexSet);
  IndexSet (*unwrap)(struct __pyx_obj_8PyClical_index_set *);
  PyObject *(*copy)(struct __pyx_obj_8PyClical_index_set *, int __pyx_skip_dispatch);
};
static struct __pyx_vtabstruct_8PyClical_index_set *__pyx_vtabptr_8PyClical_index_set;
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_9index_set_wrap(struct __pyx_obj_8PyClical_index_set *, IndexSet);
static CYTHON_INLINE IndexSet __pyx_f_8PyClical_9index_set_unwrap(struct __pyx_obj_8PyClical_index_set *);


/* "PyClical.pyx":537
 *     return clifford(obj).instance[0]
 *
 * cdef class clifford:             # «««««««
 *     """
 *     Python class clifford wraps C++ class Clifford.
 */

struct __pyx_vtabstruct_8PyClical_clifford {
  PyObject *(*wrap)(struct __pyx_obj_8PyClical_clifford *, Clifford);
  Clifford (*unwrap)(struct __pyx_obj_8PyClical_clifford *);
  PyObject *(*copy)(struct __pyx_obj_8PyClical_clifford *, int __pyx_skip_dispatch);
};
static struct __pyx_vtabstruct_8PyClical_clifford *__pyx_vtabptr_8PyClical_clifford;
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_8clifford_wrap(struct __pyx_obj_8PyClical_clifford *, Clifford);
static CYTHON_INLINE Clifford __pyx_f_8PyClical_8clifford_unwrap(struct __pyx_obj_8PyClical_clifford *);

/* --- Runtime support code (head) --- */
/* Refnanny.proto */
#ifndef CYTHON_REFNANNY
  #define CYTHON_REFNANNY 0
#endif
#if CYTHON_REFNANNY
  typedef struct {
    void (*INCREF)(void*, PyObject*, int);
    void (*DECREF)(void*, PyObject*, int);
    void (*GOTREF)(void*, PyObject*, int);
    void (*GIVEREF)(void*, PyObject*, int);
    void* (*SetupContext)(const char*, int, const char*);
    void (*FinishContext)(void**);
```

```
  } __Pyx_RefNannyAPIStruct;
  static __Pyx_RefNannyAPIStruct *__Pyx_RefNanny = NULL;
  static __Pyx_RefNannyAPIStruct *__Pyx_RefNannyImportAPI(const char *modname);
  #define __Pyx_RefNannyDeclarations void *__pyx_refnanny = NULL;
#ifdef WITH_THREAD
  #define __Pyx_RefNannySetupContext(name, acquire_gil)\
          if (acquire_gil) {\
              PyGILState_STATE __pyx_gilstate_save = PyGILState_Ensure();\
              __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__);\
              PyGILState_Release(__pyx_gilstate_save);\
          } else {\
              __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__);\
          }
#else
  #define __Pyx_RefNannySetupContext(name, acquire_gil)\
          __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__)
#endif
  #define __Pyx_RefNannyFinishContext()\
          __Pyx_RefNanny->FinishContext(&__pyx_refnanny)
  #define __Pyx_INCREF(r)  __Pyx_RefNanny->INCREF(__pyx_refnanny, (PyObject *)(r), __LINE__)
  #define __Pyx_DECREF(r)  __Pyx_RefNanny->DECREF(__pyx_refnanny, (PyObject *)(r), __LINE__)
  #define __Pyx_GOTREF(r)  __Pyx_RefNanny->GOTREF(__pyx_refnanny, (PyObject *)(r), __LINE__)
  #define __Pyx_GIVEREF(r) __Pyx_RefNanny->GIVEREF(__pyx_refnanny, (PyObject *)(r), __LINE__)
  #define __Pyx_XINCREF(r)  do { if((r) != NULL) {__Pyx_INCREF(r); }} while(0)
  #define __Pyx_XDECREF(r)  do { if((r) != NULL) {__Pyx_DECREF(r); }} while(0)
  #define __Pyx_XGOTREF(r)  do { if((r) != NULL) {__Pyx_GOTREF(r); }} while(0)
  #define __Pyx_XGIVEREF(r) do { if((r) != NULL) {__Pyx_GIVEREF(r);}} while(0)
#else
  #define __Pyx_RefNannyDeclarations
  #define __Pyx_RefNannySetupContext(name, acquire_gil)
  #define __Pyx_RefNannyFinishContext()
  #define __Pyx_INCREF(r) Py_INCREF(r)
  #define __Pyx_DECREF(r) Py_DECREF(r)
  #define __Pyx_GOTREF(r)
  #define __Pyx_GIVEREF(r)
  #define __Pyx_XINCREF(r) Py_XINCREF(r)
  #define __Pyx_XDECREF(r) Py_XDECREF(r)
  #define __Pyx_XGOTREF(r)
  #define __Pyx_XGIVEREF(r)
#endif
#define __Pyx_XDECREF_SET(r, v) do {\
        PyObject *tmp = (PyObject *) r;\
        r = v; __Pyx_XDECREF(tmp);\
    } while (0)
#define __Pyx_DECREF_SET(r, v) do {\
        PyObject *tmp = (PyObject *) r;\
        r = v; __Pyx_DECREF(tmp);\
    } while (0)
#define __Pyx_CLEAR(r)    do { PyObject* tmp = ((PyObject*)(r)); r = NULL; __Pyx_DECREF(tmp);} while(0)
#define __Pyx_XCLEAR(r)   do { if((r) != NULL) {PyObject* tmp = ((PyObject*)(r)); r = NULL; __Pyx_DECREF(tmp);}} while(0)

/* PyObjectGetAttrStr.proto */
#if CYTHON_USE_TYPE_SLOTS
static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStr(PyObject* obj, PyObject* attr_name);
#else
#define __Pyx_PyObject_GetAttrStr(o,n) PyObject_GetAttr(o,n)
#endif

/* GetBuiltinName.proto */
static PyObject *__Pyx_GetBuiltinName(PyObject *name);

/* PyCFunctionFastCall.proto */
#if CYTHON_FAST_PYCCALL
static CYTHON_INLINE PyObject *__Pyx_PyCFunction_FastCall(PyObject *func, PyObject **args, Py_ssize_t nargs);
#else
#define __Pyx_PyCFunction_FastCall(func, args, nargs)  (assert(0), NULL)
#endif

/* PyFunctionFastCall.proto */
#if CYTHON_FAST_PYCALL
#define __Pyx_PyFunction_FastCall(func, args, nargs)\
    __Pyx_PyFunction_FastCallDict((func), (args), (nargs), NULL)
#if 1 || PY_VERSION_HEX < 0x030600B1
static PyObject *__Pyx_PyFunction_FastCallDict(PyObject *func, PyObject **args, Py_ssize_t nargs, PyObject *kwargs);
#else
#define __Pyx_PyFunction_FastCallDict(func, args, nargs, kwargs) _PyFunction_FastCallDict(func, args, nargs, kwargs)
#endif
#define __Pyx_BUILD_ASSERT_EXPR(cond)\
    (sizeof(char [1 - 2*!(cond)]) - 1)
#ifndef Py_MEMBER_SIZE
#define Py_MEMBER_SIZE(type, member) sizeof(((type *)0)->member)
#endif
```

```
#if CYTHON_FAST_PYCALL
  static size_t __pyx_pyframe_localsplus_offset = 0;
  #include "frameobject.h"
  #define __Pxy_PyFrame_Initialize_Offsets()\
    ((void)__Pyx_BUILD_ASSERT_EXPR(sizeof(PyFrameObject) == offsetof(PyFrameObject, f_localsplus) + Py_MEMBER_SIZE(PyFrameObject, f_localsplus)),\
     (void)(__pyx_pyframe_localsplus_offset = ((size_t)PyFrame_Type.tp_basicsize) - Py_MEMBER_SIZE(PyFrameObject, f_localsplus)))
  #define __Pyx_PyFrame_GetLocalsplus(frame)\
    (assert(__pyx_pyframe_localsplus_offset), (PyObject **)(((char *)(frame)) + __pyx_pyframe_localsplus_offset))
#endif // CYTHON_FAST_PYCALL
#endif

/* PyObjectCall.proto */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_Call(PyObject *func, PyObject *arg, PyObject *kw);
#else
#define __Pyx_PyObject_Call(func, arg, kw) PyObject_Call(func, arg, kw)
#endif

/* PyObjectCallMethO.proto */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallMethO(PyObject *func, PyObject *arg);
#endif

/* PyObjectCallOneArg.proto */
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg);

/* PyThreadStateGet.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_PyThreadState_declare  PyThreadState *__pyx_tstate;
#define __Pyx_PyThreadState_assign  __pyx_tstate = __Pyx_PyThreadState_Current;
#define __Pyx_PyErr_Occurred()  __pyx_tstate->curexc_type
#else
#define __Pyx_PyThreadState_declare
#define __Pyx_PyThreadState_assign
#define __Pyx_PyErr_Occurred()  PyErr_Occurred()
#endif

/* PyErrFetchRestore.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_PyErr_Clear() __Pyx_ErrRestore(NULL, NULL, NULL)
#define __Pyx_ErrRestoreWithState(type, value, tb)  __Pyx_ErrRestoreInState(PyThreadState_GET(), type, value, tb)
#define __Pyx_ErrFetchWithState(type, value, tb)    __Pyx_ErrFetchInState(PyThreadState_GET(), type, value, tb)
#define __Pyx_ErrRestore(type, value, tb)  __Pyx_ErrRestoreInState(__pyx_tstate, type, value, tb)
#define __Pyx_ErrFetch(type, value, tb)    __Pyx_ErrFetchInState(__pyx_tstate, type, value, tb)
static CYTHON_INLINE void __Pyx_ErrRestoreInState(PyThreadState *tstate, PyObject *type, PyObject *value, PyObject *tb);
static CYTHON_INLINE void __Pyx_ErrFetchInState(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb);
#if CYTHON_COMPILING_IN_CPYTHON
#define __Pyx_PyErr_SetNone(exc) (Py_INCREF(exc), __Pyx_ErrRestore((exc), NULL, NULL))
#else
#define __Pyx_PyErr_SetNone(exc) PyErr_SetNone(exc)
#endif
#else
#define __Pyx_PyErr_Clear() PyErr_Clear()
#define __Pyx_PyErr_SetNone(exc) PyErr_SetNone(exc)
#define __Pyx_ErrRestoreWithState(type, value, tb)  PyErr_Restore(type, value, tb)
#define __Pyx_ErrFetchWithState(type, value, tb)  PyErr_Fetch(type, value, tb)
#define __Pyx_ErrRestoreInState(tstate, type, value, tb)  PyErr_Restore(type, value, tb)
#define __Pyx_ErrFetchInState(tstate, type, value, tb)  PyErr_Fetch(type, value, tb)
#define __Pyx_ErrRestore(type, value, tb)  PyErr_Restore(type, value, tb)
#define __Pyx_ErrFetch(type, value, tb)  PyErr_Fetch(type, value, tb)
#endif

/* WriteUnraisableException.proto */
static void __Pyx_WriteUnraisable(const char *name, int clineno,
                                  int lineno, const char *filename,
                                  int full_traceback, int nogil);

/* PyDictVersioning.proto */
#if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_TYPE_SLOTS
#define __PYX_DICT_VERSION_INIT  ((PY_UINT64_T) -1)
#define __PYX_GET_DICT_VERSION(dict)  (((PyDictObject*)(dict))->ma_version_tag)
#define __PYX_UPDATE_DICT_CACHE(dict, value, cache_var, version_var)\
    (version_var) = __PYX_GET_DICT_VERSION(dict);\
    (cache_var) = (value);
#define __PYX_PY_DICT_LOOKUP_IF_MODIFIED(VAR, DICT, LOOKUP) {\
    static PY_UINT64_T __pyx_dict_version = 0;\
    static PyObject *__pyx_dict_cached_value = NULL;\
    if (likely(__PYX_GET_DICT_VERSION(DICT) == __pyx_dict_version)) {\
        (VAR) = __pyx_dict_cached_value;\
    } else {\
```

```
        (VAR) = __pyx_dict_cached_value = (LOOKUP);\
        __pyx_dict_version = __PYX_GET_DICT_VERSION(DICT);\
    }\
}
static CYTHON_INLINE PY_UINT64_T __Pyx_get_tp_dict_version(PyObject *obj);
static CYTHON_INLINE PY_UINT64_T __Pyx_get_object_dict_version(PyObject *obj);
static CYTHON_INLINE int __Pyx_object_dict_version_matches(PyObject* obj, PY_UINT64_T tp_dict_version, PY_UINT64_T obj_dict_version);
#else
#define __PYX_GET_DICT_VERSION(dict)  (0)
#define __PYX_UPDATE_DICT_CACHE(dict, value, cache_var, version_var)
#define __PYX_PY_DICT_LOOKUP_IF_MODIFIED(VAR, DICT, LOOKUP)  (VAR) = (LOOKUP);
#endif

/* PyObjectCallNoArg.proto */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallNoArg(PyObject *func);
#else
#define __Pyx_PyObject_CallNoArg(func) __Pyx_PyObject_Call(func, __pyx_empty_tuple, NULL)
#endif

/* RaiseDoubleKeywords.proto */
static void __Pyx_RaiseDoubleKeywordsError(const char* func_name, PyObject* kw_name);

/* ParseKeywords.proto */
static int __Pyx_ParseOptionalKeywords(PyObject *kwds, PyObject **argnames[],\
    PyObject *kwds2, PyObject *values[], Py_ssize_t num_pos_args,\
    const char* function_name);

/* RaiseArgTupleInvalid.proto */
static void __Pyx_RaiseArgtupleInvalid(const char* func_name, int exact,
    Py_ssize_t num_min, Py_ssize_t num_max, Py_ssize_t num_found);

/* GetModuleGlobalName.proto */
#if CYTHON_USE_DICT_VERSIONS
#define __Pyx_GetModuleGlobalName(var, name)  {\
    static PY_UINT64_T __pyx_dict_version = 0;\
    static PyObject *__pyx_dict_cached_value = NULL;\
    (var) = (likely(__pyx_dict_version == __PYX_GET_DICT_VERSION(__pyx_d))) ?\
        (likely(__pyx_dict_cached_value) ? __Pyx_NewRef(__pyx_dict_cached_value) : __Pyx_GetBuiltinName(name)) :\
        __Pyx__GetModuleGlobalName(name, &__pyx_dict_version, &__pyx_dict_cached_value);\
}
#define __Pyx_GetModuleGlobalNameUncached(var, name)  {\
    PY_UINT64_T __pyx_dict_version;\
    PyObject *__pyx_dict_cached_value;\
    (var) = __Pyx__GetModuleGlobalName(name, &__pyx_dict_version, &__pyx_dict_cached_value);\
}
static PyObject *__Pyx__GetModuleGlobalName(PyObject *name, PY_UINT64_T *dict_version, PyObject **dict_cached_value);
#else
#define __Pyx_GetModuleGlobalName(var, name)  (var) = __Pyx__GetModuleGlobalName(name)
#define __Pyx_GetModuleGlobalNameUncached(var, name)  (var) = __Pyx__GetModuleGlobalName(name)
static CYTHON_INLINE PyObject *__Pyx__GetModuleGlobalName(PyObject *name);
#endif

/* GetTopmostException.proto */
#if CYTHON_USE_EXC_INFO_STACK
static _PyErr_StackItem * __Pyx_PyErr_GetTopmostException(PyThreadState *tstate);
#endif

/* SaveResetException.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_ExceptionSave(type, value, tb)  __Pyx__ExceptionSave(__pyx_tstate, type, value, tb)
static CYTHON_INLINE void __Pyx__ExceptionSave(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb);
#define __Pyx_ExceptionReset(type, value, tb)  __Pyx__ExceptionReset(__pyx_tstate, type, value, tb)
static CYTHON_INLINE void __Pyx__ExceptionReset(PyThreadState *tstate, PyObject *type, PyObject *value, PyObject *tb);
#else
#define __Pyx_ExceptionSave(type, value, tb)   PyErr_GetExcInfo(type, value, tb)
#define __Pyx_ExceptionReset(type, value, tb)  PyErr_SetExcInfo(type, value, tb)
#endif

/* PyErrExceptionMatches.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_PyErr_ExceptionMatches(err) __Pyx_PyErr_ExceptionMatchesInState(__pyx_tstate, err)
static CYTHON_INLINE int __Pyx_PyErr_ExceptionMatchesInState(PyThreadState* tstate, PyObject* err);
#else
#define __Pyx_PyErr_ExceptionMatches(err)  PyErr_ExceptionMatches(err)
#endif

/* GetException.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_GetException(type, value, tb)  __Pyx__GetException(__pyx_tstate, type, value, tb)
static int __Pyx__GetException(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb);
```

```
#else
static int __Pyx_GetException(PyObject **type, PyObject **value, PyObject **tb);
#endif

/* RaiseException.proto */
static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb, PyObject *cause);

/* PyObjectCall2Args.proto */
static CYTHON_UNUSED PyObject* __Pyx_PyObject_Call2Args(PyObject* function, PyObject* arg1, PyObject* arg2);

/* PyIntBinop.proto */
#if !CYTHON_COMPILING_IN_PYPY
static PyObject* __Pyx_PyInt_AddObjC(PyObject *op1, PyObject *op2, long intval, int inplace, int zerodivision_check);
#else
#define __Pyx_PyInt_AddObjC(op1, op2, intval, inplace, zerodivision_check)\
    (inplace ? PyNumber_InPlaceAdd(op1, op2) : PyNumber_Add(op1, op2))
#endif

/* PySequenceContains.proto */
static CYTHON_INLINE int __Pyx_PySequence_ContainsTF(PyObject* item, PyObject* seq, int eq) {
    int result = PySequence_Contains(seq, item);
    return unlikely(result < 0) ? result : (result == (eq == Py_EQ));
}

/* IncludeCppStringH.proto */
#include <string>

/* decode_c_string_utf16.proto */
static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16(const char *s, Py_ssize_t size, const char *errors) {
    int byteorder = 0;
    return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
}
static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16LE(const char *s, Py_ssize_t size, const char *errors) {
    int byteorder = -1;
    return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
}
static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16BE(const char *s, Py_ssize_t size, const char *errors) {
    int byteorder = 1;
    return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
}

/* decode_c_bytes.proto */
static CYTHON_INLINE PyObject* __Pyx_decode_c_bytes(
         const char* cstring, Py_ssize_t length, Py_ssize_t start, Py_ssize_t stop,
         const char* encoding, const char* errors,
         PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors));

/* decode_cpp_string.proto */
static CYTHON_INLINE PyObject* __Pyx_decode_cpp_string(
         std::string cppstring, Py_ssize_t start, Py_ssize_t stop,
         const char* encoding, const char* errors,
         PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors)) {
    return __Pyx_decode_c_bytes(
        cppstring.data(), cppstring.size(), start, stop, encoding, errors, decode_func);
}

/* SwapException.proto */
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_ExceptionSwap(type, value, tb)  __Pyx__ExceptionSwap(__pyx_tstate, type, value, tb)
static CYTHON_INLINE void __Pyx__ExceptionSwap(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb);
#else
static CYTHON_INLINE void __Pyx_ExceptionSwap(PyObject **type, PyObject **value, PyObject **tb);
#endif

/* SetItemInt.proto */
#define __Pyx_SetItemInt(o, i, v, type, is_signed, to_py_func, is_list, wraparound, boundscheck)\
    (__Pyx_fits_Py_ssize_t(i, type, is_signed) ?\
    __Pyx_SetItemInt_Fast(o, (Py_ssize_t)i, v, is_list, wraparound, boundscheck) :\
    (is_list ? (PyErr_SetString(PyExc_IndexError, "list assignment index out of range"), -1) :\
               __Pyx_SetItemInt_Generic(o, to_py_func(i), v)))
static int __Pyx_SetItemInt_Generic(PyObject *o, PyObject *j, PyObject *v);
static CYTHON_INLINE int __Pyx_SetItemInt_Fast(PyObject *o, Py_ssize_t i, PyObject *v,
                                               int is_list, int wraparound, int boundscheck);

/* ArgTypeTest.proto */
#define __Pyx_ArgTypeTest(obj, type, none_allowed, name, exact)\
    ((likely((Py_TYPE(obj) == type) | (none_allowed && (obj == Py_None)))) ? 1 :\
        __Pyx__ArgTypeTest(obj, type, name, exact))
static int __Pyx__ArgTypeTest(PyObject *obj, PyTypeObject *type, const char *name, int exact);

/* Import.proto */
```

```
static PyObject *__Pyx_Import(PyObject *name, PyObject *from_list, int level);

/* IncludeStringH.proto */
#include <string.h>

/* PyObject_GenericGetAttrNoDict.proto */
#if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
static CYTHON_INLINE PyObject* __Pyx_PyObject_GenericGetAttrNoDict(PyObject* obj, PyObject* attr_name);
#else
#define __Pyx_PyObject_GenericGetAttrNoDict PyObject_GenericGetAttr
#endif

/* PyObject_GenericGetAttr.proto */
#if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
static PyObject* __Pyx_PyObject_GenericGetAttr(PyObject* obj, PyObject* attr_name);
#else
#define __Pyx_PyObject_GenericGetAttr PyObject_GenericGetAttr
#endif

/* SetVTable.proto */
static int __Pyx_SetVtable(PyObject *dict, void *vtable);

/* PyObjectGetAttrStrNoError.proto */
static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStrNoError(PyObject* obj, PyObject* attr_name);

/* SetupReduce.proto */
static int __Pyx_setup_reduce(PyObject* type_obj);

/* BytesEquals.proto */
static CYTHON_INLINE int __Pyx_PyBytes_Equals(PyObject* s1, PyObject* s2, int equals);

/* UnicodeEquals.proto */
static CYTHON_INLINE int __Pyx_PyUnicode_Equals(PyObject* s1, PyObject* s2, int equals);

/* CLineInTraceback.proto */
#ifdef CYTHON_CLINE_IN_TRACEBACK
#define __Pyx_CLineForTraceback(tstate, c_line)  (((CYTHON_CLINE_IN_TRACEBACK)) ? c_line : 0)
#else
static int __Pyx_CLineForTraceback(PyThreadState *tstate, int c_line);
#endif

/* CodeObjectCache.proto */
typedef struct {
    PyCodeObject* code_object;
    int code_line;
} __Pyx_CodeObjectCacheEntry;
struct __Pyx_CodeObjectCache {
    int count;
    int max_count;
    __Pyx_CodeObjectCacheEntry* entries;
};
static struct __Pyx_CodeObjectCache __pyx_code_cache = {0,0,NULL};
static int __pyx_bisect_code_objects(__Pyx_CodeObjectCacheEntry* entries, int count, int code_line);
static PyCodeObject *__pyx_find_code_object(int code_line);
static void __pyx_insert_code_object(int code_line, PyCodeObject* code_object);

/* AddTraceback.proto */
static void __Pyx_AddTraceback(const char *funcname, int c_line,
                               int py_line, const char *filename);

/* GCCDiagnostics.proto */
#if defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ >= 6))
#define __Pyx_HAS_GCC_DIAGNOSTIC
#endif

/* CppExceptionConversion.proto */
#ifndef __Pyx_CppExn2PyErr
#include <new>
#include <typeinfo>
#include <stdexcept>
#include <ios>
static void __Pyx_CppExn2PyErr() {
  try {
    if (PyErr_Occurred())
      ; // let the latest Python exn pass through and ignore the current one
    else
      throw;
  } catch (const std::bad_alloc& exn) {
    PyErr_SetString(PyExc_MemoryError, exn.what());
  } catch (const std::bad_cast& exn) {
    PyErr_SetString(PyExc_TypeError, exn.what());
  } catch (const std::bad_typeid& exn) {
    PyErr_SetString(PyExc_TypeError, exn.what());
  } catch (const std::domain_error& exn) {
    PyErr_SetString(PyExc_ValueError, exn.what());
  } catch (const std::invalid_argument& exn) {
```

```
    PyErr_SetString(PyExc_ValueError, exn.what());
  } catch (const std::ios_base::failure& exn) {
    PyErr_SetString(PyExc_IOError, exn.what());
  } catch (const std::out_of_range& exn) {
    PyErr_SetString(PyExc_IndexError, exn.what());
  } catch (const std::overflow_error& exn) {
    PyErr_SetString(PyExc_OverflowError, exn.what());
  } catch (const std::range_error& exn) {
    PyErr_SetString(PyExc_ArithmeticError, exn.what());
  } catch (const std::underflow_error& exn) {
    PyErr_SetString(PyExc_ArithmeticError, exn.what());
  } catch (const std::exception& exn) {
    PyErr_SetString(PyExc_RuntimeError, exn.what());
  }
  catch (...)
  {
    PyErr_SetString(PyExc_RuntimeError, "Unknown exception");
  }
}
#endif

/* CIntFromPy.proto */
static CYTHON_INLINE int __Pyx_PyInt_As_int(PyObject *);

/* CIntToPy.proto */
static CYTHON_INLINE PyObject* __Pyx_PyInt_From_int(int value);

/* CIntToPy.proto */
static CYTHON_INLINE PyObject* __Pyx_PyInt_From_long(long value);

/* CIntFromPy.proto */
static CYTHON_INLINE long __Pyx_PyInt_As_long(PyObject *);

/* FastTypeChecks.proto */
#if CYTHON_COMPILING_IN_CPYTHON
#define __Pyx_TypeCheck(obj, type) __Pyx_IsSubtype(Py_TYPE(obj), (PyTypeObject *)type)
static CYTHON_INLINE int __Pyx_IsSubtype(PyTypeObject *a, PyTypeObject *b);
static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches(PyObject *err, PyObject *type);
static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches2(PyObject *err, PyObject *type1, PyObject *type2);
#else
#define __Pyx_TypeCheck(obj, type) PyObject_TypeCheck(obj, (PyTypeObject *)type)
#define __Pyx_PyErr_GivenExceptionMatches(err, type) PyErr_GivenExceptionMatches(err, type)
#define __Pyx_PyErr_GivenExceptionMatches2(err, type1, type2) (PyErr_GivenExceptionMatches(err, type1) || PyErr_GivenExceptionMatches(err, type2))
#endif
#define __Pyx_PyException_Check(obj) __Pyx_TypeCheck(obj, PyExc_Exception)

/* FetchCommonType.proto */
static PyTypeObject* __Pyx_FetchCommonType(PyTypeObject* type);

/* PyObjectGetMethod.proto */
static int __Pyx_PyObject_GetMethod(PyObject *obj, PyObject *name, PyObject **method);

/* PyObjectCallMethod1.proto */
static PyObject* __Pyx_PyObject_CallMethod1(PyObject* obj, PyObject* method_name, PyObject* arg);

/* CoroutineBase.proto */
typedef PyObject *(*__pyx_coroutine_body_t)(PyObject *, PyThreadState *, PyObject *);
#if CYTHON_USE_EXC_INFO_STACK
#define __Pyx_ExcInfoStruct  _PyErr_StackItem
#else
typedef struct {
    PyObject *exc_type;
    PyObject *exc_value;
    PyObject *exc_traceback;
} __Pyx_ExcInfoStruct;
#endif
typedef struct {
    PyObject_HEAD
    __pyx_coroutine_body_t body;
    PyObject *closure;
    __Pyx_ExcInfoStruct gi_exc_state;
    PyObject *gi_weakreflist;
    PyObject *classobj;
    PyObject *yieldfrom;
    PyObject *gi_name;
    PyObject *gi_qualname;
    PyObject *gi_modulename;
    PyObject *gi_code;
    PyObject *gi_frame;
    int resume_label;
    char is_running;
} __pyx_CoroutineObject;
static __pyx_CoroutineObject *__Pyx__Coroutine_New(
    PyTypeObject *type, __pyx_coroutine_body_t body, PyObject *code, PyObject *closure,
    PyObject *name, PyObject *qualname, PyObject *module_name);
```

```
static __pyx_CoroutineObject *__Pyx__Coroutine_NewInit(
            __pyx_CoroutineObject *gen, __pyx_coroutine_body_t body, PyObject *code, PyObject *closure,
            PyObject *name, PyObject *qualname, PyObject *module_name);
static CYTHON_INLINE void __Pyx_Coroutine_ExceptionClear(__Pyx_ExcInfoStruct *self);
static int __Pyx_Coroutine_clear(PyObject *self);
static PyObject *__Pyx_Coroutine_Send(PyObject *self, PyObject *value);
static PyObject *__Pyx_Coroutine_Close(PyObject *self);
static PyObject *__Pyx_Coroutine_Throw(PyObject *gen, PyObject *args);
#if CYTHON_USE_EXC_INFO_STACK
#define __Pyx_Coroutine_SwapException(self)
#define __Pyx_Coroutine_ResetAndClearException(self) __Pyx_Coroutine_ExceptionClear(&(self)->gi_exc_state)
#else
#define __Pyx_Coroutine_SwapException(self) {\
    __Pyx_ExceptionSwap(&(self)->gi_exc_state.exc_type, &(self)->gi_exc_state.exc_value, &(self)->gi_exc_state.exc_traceback);\
    __Pyx_Coroutine_ResetFrameBackpointer(&(self)->gi_exc_state);\
    }
#define __Pyx_Coroutine_ResetAndClearException(self) {\
    __Pyx_ExceptionReset((self)->gi_exc_state.exc_type, (self)->gi_exc_state.exc_value, (self)->gi_exc_state.exc_traceback);\
    (self)->gi_exc_state.exc_type = (self)->gi_exc_state.exc_value = (self)->gi_exc_state.exc_traceback = NULL;\
    }
#endif
#if CYTHON_FAST_THREAD_STATE
#define __Pyx_PyGen_FetchStopIterationValue(pvalue)\
    __Pyx_PyGen__FetchStopIterationValue(__pyx_tstate, pvalue)
#else
#define __Pyx_PyGen_FetchStopIterationValue(pvalue)\
    __Pyx_PyGen__FetchStopIterationValue(__Pyx_PyThreadState_Current, pvalue)
#endif
static int __Pyx_PyGen__FetchStopIterationValue(PyThreadState *tstate, PyObject **pvalue);
static CYTHON_INLINE void __Pyx_Coroutine_ResetFrameBackpointer(__Pyx_ExcInfoStruct *exc_state);

/* PatchModuleWithCoroutine.proto */
static PyObject* __Pyx_Coroutine_patch_module(PyObject* module, const char* py_code);

/* PatchGeneratorABC.proto */
static int __Pyx_patch_abc(void);

/* Generator.proto */
#define __Pyx_Generator_USED
static PyTypeObject *__pyx_GeneratorType = 0;
#define __Pyx_Generator_CheckExact(obj) (Py_TYPE(obj) == __pyx_GeneratorType)
#define __Pyx_Generator_New(body, code, closure, name, qualname, module_name)\
    __Pyx__Coroutine_New(__pyx_GeneratorType, body, code, closure, name, qualname, module_name)
static PyObject *__Pyx_Generator_Next(PyObject *self);
static int __pyx_Generator_init(void);

/* CheckBinaryVersion.proto */
static int __Pyx_check_binary_version(void);

/* InitStrings.proto */
static int __Pyx_InitStrings(__Pyx_StringTabEntry *t);

static CYTHON_INLINE PyObject *__pyx_f_8PyClical_9index_set_wrap(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, IndexSet __pyx_v_other); /* proto*/
static CYTHON_INLINE IndexSet __pyx_f_8PyClical_9index_set_unwrap(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto*/
static PyObject *__pyx_f_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, int __pyx_skip_dispatch); /* proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_8clifford_wrap(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, Clifford __pyx_v_other); /* proto*/
static CYTHON_INLINE Clifford __pyx_f_8PyClical_8clifford_unwrap(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto*/
static PyObject *__pyx_f_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, int __pyx_skip_dispatch); /* proto*/

/* Module declarations from 'libcpp.vector' */

/* Module declarations from 'glucat' */

/* Module declarations from 'libc.string' */

/* Module declarations from 'libcpp.string' */

/* Module declarations from 'PyClical' */
static PyTypeObject *__pyx_ptype_8PyClical_index_set = 0;
static PyTypeObject *__pyx_ptype_8PyClical_clifford = 0;
static PyTypeObject *__pyx_ptype_8PyClical___pyx_scope_struct____iter__ = 0;
static CYTHON_INLINE IndexSet __pyx_f_8PyClical_toIndexSet(PyObject *); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_compare(PyObject *, PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_min_neg(PyObject *, int __pyx_skip_dispatch); /*proto*/
```

```
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_pos(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE std::vector<scalar_t>  __pyx_f_8PyClical_list_to_vector(PyObject *); /*proto*/
static CYTHON_INLINE Clifford __pyx_f_8PyClical_toClifford(PyObject *); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_error_squared_tol(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_error_squared(PyObject *, PyObject *, PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_approx_equal(PyObject *, PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_approx_equal *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_inv(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_scalar(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_real(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_imag(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_pure(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_even(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_odd(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_involute(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_reverse(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_conj(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_quad(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_norm(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_abs(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_abs(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_pow(PyObject *, PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_outer_pow(PyObject *, PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_complexifier(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sqrt(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_sqrt *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_exp(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_log(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_log *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cos(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_cos *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_acos(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_acos *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cosh(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_acosh(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_acosh *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sin(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_sin *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asin(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asin *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sinh(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asinh(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asinh *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_tan(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_tan *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_atan(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atan *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_tanh(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_atanh(PyObject *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atanh *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_random_clifford(struct __pyx_obj_8PyClical_index_set *, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_random_clifford *__pyx_optional_args); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cga3(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cga3std(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_agc3(PyObject *, int __pyx_skip_dispatch); /*proto*/
static CYTHON_INLINE PyObject *__pyx_convert_PyObject_string_to_py_std__in_string(std::string const &); /*proto*/
static CYTHON_INLINE PyObject *__pyx_convert_PyUnicode_string_to_py_std__in_string(std::string const &); /*proto*/
static CYTHON_INLINE PyObject *__pyx_convert_PyStr_string_to_py_std__in_string(std::string const &); /*proto*/
static CYTHON_INLINE PyObject *__pyx_convert_PyBytes_string_to_py_std__in_string(std::string const &); /*proto*/
static CYTHON_INLINE PyObject *__pyx_convert_PyByteArray_string_to_py_std__in_string(std::string const &); /*proto*/
#define __Pyx_MODULE_NAME "PyClical"
extern int __pyx_module_is_main_PyClical;
int __pyx_module_is_main_PyClical = 0;

/* Implementation of 'PyClical' */
static PyObject *__pyx_builtin_IndexError;
static PyObject *__pyx_builtin_RuntimeError;
static PyObject *__pyx_builtin_TypeError;
static PyObject *__pyx_builtin_ValueError;
static PyObject *__pyx_builtin_NotImplemented;
```

```
static PyObject *__pyx_builtin_range;
static PyObject *__pyx_builtin_xrange;
static const char __pyx_k_[] = ".";
static const char __pyx_k_e[] = "e";
static const char __pyx_k_i[] = "i";
static const char __pyx_k_m[] = "m";
static const char __pyx_k_p[] = "p";
static const char __pyx_k_q[] = "q";
static const char __pyx_k__2[] = " ";
static const char __pyx_k__5[] = ":";
static const char __pyx_k__6[] = "\n\t";
static const char __pyx_k__7[] = " (";
static const char __pyx_k__8[] = ", ";
static const char __pyx_k__9[] = ").";
static const char __pyx_k_cl[] = "cl";
static const char __pyx_k_pi[] = "pi";
static const char __pyx_k_abc[] = "abc";
static const char __pyx_k_cos[] = "cos";
static const char __pyx_k_exp[] = "exp";
static const char __pyx_k_frm[] = "frm";
static const char __pyx_k_inv[] = "inv";
static const char __pyx_k_ist[] = "ist";
static const char __pyx_k_ixt[] = "ixt";
static const char __pyx_k_lhs[] = "lhs";
static const char __pyx_k_log[] = "log";
static const char __pyx_k_max[] = "max";
static const char __pyx_k_min[] = "min";
static const char __pyx_k_obj[] = "obj";
static const char __pyx_k_odd[] = "odd";
static const char __pyx_k_pow[] = "pow";
static const char __pyx_k_rhs[] = "rhs";
static const char __pyx_k_sin[] = "sin";
static const char __pyx_k_tan[] = "tan";
static const char __pyx_k_tau[] = "tau";
static const char __pyx_k_tol[] = "tol";
static const char __pyx_k_Real[] = "Real";
static const char __pyx_k_acos[] = "acos";
static const char __pyx_k_args[] = "args";
static const char __pyx_k_asin[] = "asin";
static const char __pyx_k_atan[] = "atan";
static const char __pyx_k_conj[] = "conj";
static const char __pyx_k_copy[] = "copy";
static const char __pyx_k_cosh[] = "cosh";
static const char __pyx_k_even[] = "even";
static const char __pyx_k_fill[] = "fill";
static const char __pyx_k_from[] = " from ";
static const char __pyx_k_iter[] = "__iter__";
static const char __pyx_k_main[] = "__main__";
static const char __pyx_k_math[] = "math";
static const char __pyx_k_name[] = "__name__";
static const char __pyx_k_norm[] = "norm";
static const char __pyx_k_pure[] = "pure";
static const char __pyx_k_quad[] = "quad";
static const char __pyx_k_send[] = "send";
static const char __pyx_k_sinh[] = "sinh";
static const char __pyx_k_sqrt[] = "sqrt";
static const char __pyx_k_tanh[] = "tanh";
static const char __pyx_k_test[] = "_test";
static const char __pyx_k_UTF_8[] = "UTF-8";
static const char __pyx_k_acosh[] = "acosh";
static const char __pyx_k_asinh[] = "asinh";
static const char __pyx_k_atanh[] = "atanh";
static const char __pyx_k_close[] = "close";
static const char __pyx_k_grade[] = "grade";
static const char __pyx_k_istpq[] = "istpq";
static const char __pyx_k_nbar3[] = "nbar3";
static const char __pyx_k_ninf3[] = "ninf3";
static const char __pyx_k_other[] = "other";
static const char __pyx_k_range[] = "range";
static const char __pyx_k_throw[] = "throw";
static const char __pyx_k_using[] = " using (";
static const char __pyx_k_utf_8[] = "utf-8";
static const char __pyx_k_value[] = " value ";
static const char __pyx_k_encode[] = "encode";
static const char __pyx_k_import[] = "__import__";
static const char __pyx_k_reduce[] = "__reduce__";
static const char __pyx_k_scalar[] = "scalar";
static const char __pyx_k_test_2[] = "__test__";
static const char __pyx_k_xrange[] = "xrange";
static const char __pyx_k_doctest[] = "doctest";
static const char __pyx_k_invalid[] = " invalid ";
static const char __pyx_k_numbers[] = "numbers";
static const char __pyx_k_reverse[] = "reverse";
static const char __pyx_k_testmod[] = "testmod";
static const char __pyx_k_version[] = "__version__";
static const char __pyx_k_Integral[] = "Integral";
static const char __pyx_k_PyClical[] = "PyClical";
```

```
static const char __pyx_k_Sequence[] = "Sequence";
static const char __pyx_k_as_frame[] = " as frame:\n\t";
static const char __pyx_k_clifford[] = "clifford";
static const char __pyx_k_getstate[] = "__getstate__";
static const char __pyx_k_involute[] = "involute";
static const char __pyx_k_setstate[] = "__setstate__";
static const char __pyx_k_to_frame[] = " to frame ";
static const char __pyx_k_TypeError[] = "TypeError";
static const char __pyx_k_index_set[] = "index_set";
static const char __pyx_k_outer_pow[] = "outer_pow";
static const char __pyx_k_reduce_ex[] = "__reduce_ex__";
static const char __pyx_k_threshold[] = "threshold";
static const char __pyx_k_IndexError[] = "IndexError";
static const char __pyx_k_ValueError[] = "ValueError";
static const char __pyx_k_pyx_vtable[] = "__pyx_vtable__";
static const char __pyx_k_collections[] = "collections";
static const char __pyx_k_e_line_1936[] = "e (line 1936)";
static const char __pyx_k_PyClical_pyx[] = "PyClical.pyx";
static const char __pyx_k_RuntimeError[] = "RuntimeError";
static const char __pyx_k_abs_line_1522[] = "abs (line 1522)";
static const char __pyx_k_cos_line_1651[] = "cos (line 1651)";
static const char __pyx_k_exp_line_1614[] = "exp (line 1614)";
static const char __pyx_k_inv_line_1378[] = "inv (line 1378)";
static const char __pyx_k_log_line_1628[] = "log (line 1628)";
static const char __pyx_k_odd_line_1446[] = "odd (line 1446)";
static const char __pyx_k_pow_line_1543[] = "pow (line 1543)";
static const char __pyx_k_reduce_cython[] = "__reduce_cython__";
static const char __pyx_k_sin_line_1728[] = "sin (line 1728)";
static const char __pyx_k_tan_line_1801[] = "tan (line 1801)";
static const char __pyx_k_using_invalid[] = " using invalid ";
static const char __pyx_k_Cannot_reframe[] = "Cannot reframe";
static const char __pyx_k_NotImplemented[] = "NotImplemented";
static const char __pyx_k_Not_applicable[] = "Not applicable.";
static const char __pyx_k_acos_line_1668[] = "acos (line 1668)";
static const char __pyx_k_agc3_line_1893[] = "agc3 (line 1893)";
static const char __pyx_k_asin_line_1747[] = "asin (line 1747)";
static const char __pyx_k_atan_line_1818[] = "atan (line 1818)";
static const char __pyx_k_cga3_line_1873[] = "cga3 (line 1873)";
static const char __pyx_k_conj_line_1485[] = "conj (line 1485)";
static const char __pyx_k_cosh_line_1689[] = "cosh (line 1689)";
static const char __pyx_k_even_line_1437[] = "even (line 1437)";
static const char __pyx_k_imag_line_1415[] = "imag (line 1415)";
static const char __pyx_k_invalid_string[] = " invalid string ";
static const char __pyx_k_norm_line_1511[] = "norm (line 1511)";
static const char __pyx_k_pure_line_1426[] = "pure (line 1426)";
static const char __pyx_k_quad_line_1500[] = "quad (line 1500)";
static const char __pyx_k_real_line_1404[] = "real (line 1404)";
static const char __pyx_k_scalar_epsilon[] = "scalar_epsilon";
static const char __pyx_k_sinh_line_1768[] = "sinh (line 1768)";
static const char __pyx_k_sqrt_line_1591[] = "sqrt (line 1591)";
static const char __pyx_k_tanh_line_1835[] = "tanh (line 1835)";
static const char __pyx_k_acosh_line_1705[] = "acosh (line 1705)";
static const char __pyx_k_asinh_line_1782[] = "asinh (line 1782)";
static const char __pyx_k_atanh_line_1847[] = "atanh (line 1847)";
static const char __pyx_k_istpq_line_1949[] = "istpq (line 1949)";
static const char __pyx_k_setstate_cython[] = "__setstate_cython__";
static const char __pyx_k_compare_line_492[] = "compare (line 492)";
static const char __pyx_k_index_set___iter[] = "index_set.__iter__";
static const char __pyx_k_max_pos_line_513[] = "max_pos (line 513)";
static const char __pyx_k_min_neg_line_504[] = "min_neg (line 504)";
static const char __pyx_k_scalar_line_1393[] = "scalar (line 1393)";
static const char __pyx_k_cga3std_line_1882[] = "cga3std (line 1882)";
static const char __pyx_k_max_abs_line_1531[] = "max_abs (line 1531)";
static const char __pyx_k_reverse_line_1470[] = "reverse (line 1470)";
static const char __pyx_k_cline_in_traceback[] = "cline_in_traceback";
static const char __pyx_k_involute_line_1455[] = "involute (line 1455)";
static const char __pyx_k_outer_pow_line_1567[] = "outer_pow (line 1567)";
static const char __pyx_k_clifford_inv_line_926[] = "clifford.inv (line 926)";
static const char __pyx_k_clifford_pow_line_980[] = "clifford.pow (line 980)";
static const char __pyx_k_approx_equal_line_1359[] = "approx_equal (line 1359)";
static const char __pyx_k_clifford_abs_line_1175[] = "clifford.abs (line 1175)";
static const char __pyx_k_clifford_copy_line_556[] = "clifford.copy (line 556)";
static const char __pyx_k_clifford_odd_line_1070[] = "clifford.odd (line 1070)";
static const char __pyx_k_complexifier_line_1576[] = "complexifier (line 1576)";
static const char __pyx_k_index_set_copy_line_65[] = "index_set.copy (line 65)";
static const char __pyx_k_index_set_max_line_351[] = "index_set.max (line 351)";
static const char __pyx_k_index_set_min_line_342[] = "index_set.min (line 342)";
static const char __pyx_k_clifford_conj_line_1138[] = "clifford.conj (line 1138)";
static const char __pyx_k_clifford_even_line_1061[] = "clifford.even (line 1061)";
static const char __pyx_k_clifford_norm_line_1164[] = "clifford.norm (line 1164)";
static const char __pyx_k_clifford_pure_line_1050[] = "clifford.pure (line 1050)";
static const char __pyx_k_clifford_quad_line_1153[] = "clifford.quad (line 1153)";
static const char __pyx_k_error_squared_line_1346[] = "error_squared (line 1346)";
static const char __pyx_k_Unary_print_clifford_1_1[] = "\n        Unary -.\n\n        »>
      print(-clifford(\"{1}\"))\n        -{1}\n        ";
static const char __pyx_k_clifford___or___line_939[] = "clifford.__or__ (line 939)";
static const char __pyx_k_clifford_frame_line_1224[] = "clifford.frame (line 1224)";
```

```
static const char __pyx_k_clifford_hidden_doctests[] = "clifford_hidden_doctests";
static const char __pyx_k_clifford_isinf_line_1206[] = "clifford.isinf (line 1206)";
static const char __pyx_k_clifford_isnan_line_1215[] = "clifford.isnan (line 1215)";
static const char __pyx_k_index_set_count_line_315[] = "index_set.count (line 315)";
static const char __pyx_k_clifford___add___line_740[] = "clifford.__add__ (line 740)";
static const char __pyx_k_clifford___and___line_836[] = "clifford.__and__ (line 836)";
static const char __pyx_k_clifford___ior___line_950[] = "clifford.__ior__ (line 950)";
static const char __pyx_k_clifford___mod___line_806[] = "clifford.__mod__ (line 806)";
static const char __pyx_k_clifford___mul___line_780[] = "clifford.__mul__ (line 780)";
static const char __pyx_k_clifford___neg___line_722[] = "clifford.__neg__ (line 722)";
static const char __pyx_k_clifford___pos___line_731[] = "clifford.__pos__ (line 731)";
static const char __pyx_k_clifford___pow___line_961[] = "clifford.__pow__ (line 961)";
static const char __pyx_k_clifford___sub___line_760[] = "clifford.__sub__ (line 760)";
static const char __pyx_k_clifford___xor___line_866[] = "clifford.__xor__ (line 866)";
static const char __pyx_k_clifford_reframe_line_649[] = "clifford.reframe (line 649)";
static const char __pyx_k_clifford_scalar_line_1039[] = "clifford.scalar (line 1039)";
static const char __pyx_k_index_set___or___line_293[] = "index_set.__or__ (line 293)";
static const char __pyx_k_index_set_hidden_doctests[] = "index_set_hidden_doctests";
static const char __pyx_k_random_clifford_line_1864[] = "random_clifford (line 1864)";
static const char __pyx_k_Cannot_take_vector_part_of[] = "Cannot take vector part of ";
static const char __pyx_k_Unary_print_clifford_1_1_2[] = "\n        Unary +.\n\n        >>>
    print(+clifford(\"{1}\"))\n        {1}\n        ";
static const char __pyx_k_clifford___iadd___line_751[] = "clifford.__iadd__ (line 751)";
static const char __pyx_k_clifford___iand___line_851[] = "clifford.__iand__ (line 851)";
static const char __pyx_k_clifford___idiv___line_911[] = "clifford.__idiv__ (line 911)";
static const char __pyx_k_clifford___imod___line_821[] = "clifford.__imod__ (line 821)";
static const char __pyx_k_clifford___imul___line_793[] = "clifford.__imul__ (line 793)";
static const char __pyx_k_clifford___isub___line_771[] = "clifford.__isub__ (line 771)";
static const char __pyx_k_clifford___iter___line_638[] = "clifford.__iter__ (line 638)";
static const char __pyx_k_clifford___ixor___line_881[] = "clifford.__ixor__ (line 881)";
static const char __pyx_k_clifford___str___line_1244[] = "clifford.__str__ (line 1244)";
static const char __pyx_k_clifford_max_abs_line_1184[] = "clifford.max_abs (line 1184)";
static const char __pyx_k_clifford_reverse_line_1123[] = "clifford.reverse (line 1123)";
static const char __pyx_k_index_set___and___line_271[] = "index_set.__and__ (line 271)";
static const char __pyx_k_index_set___ior___line_304[] = "index_set.__ior__ (line 304)";
static const char __pyx_k_index_set___str___line_395[] = "index_set.__str__ (line 395)";
static const char __pyx_k_index_set___xor___line_249[] = "index_set.__xor__ (line 249)";
static const char __pyx_k_clifford___call___line_1020[] = "clifford.__call__ (line 1020)";
static const char __pyx_k_clifford___repr___line_1235[] = "clifford.__repr__ (line 1235)";
static const char __pyx_k_clifford_involute_line_1107[] = "clifford.involute (line 1107)";
static const char __pyx_k_error_squared_tol_line_1337[] = "error_squared_tol (line 1337)";
static const char __pyx_k_index_set___iand___line_282[] = "index_set.__iand__ (line 282)";
static const char __pyx_k_index_set___iter___line_229[] = "index_set.__iter__ (line 229)";
static const char __pyx_k_index_set___ixor___line_260[] = "index_set.__ixor__ (line 260)";
static const char __pyx_k_index_set___repr___line_384[] = "index_set.__repr__ (line 384)";
static const char __pyx_k_clifford_outer_pow_line_1004[] = "clifford.outer_pow (line 1004)";
static const char __pyx_k_clifford_truncated_line_1195[] = "clifford.truncated (line 1195)";
static const char __pyx_k_index_set_count_neg_line_324[] = "index_set.count_neg (line 324)";
static const char __pyx_k_index_set_count_pos_line_333[] = "index_set.count_pos (line 333)";
static const char __pyx_k_clifford___getitem___line_707[] = "clifford.__getitem__ (line 707)";
static const char __pyx_k_clifford___truediv___line_896[] = "clifford.__truediv__ (line 896)";
static const char __pyx_k_index_set___invert___line_240[] = "index_set.__invert__ (line 240)";
static const char __pyx_k_Abbreviation_for_index_set_q_p[] = "\n    Abbreviation for
    index_set({-q,...p}).\n\n    >>> print(istpq(2,3))\n    {-3,-2,-1,1,2}\n    ";
static const char __pyx_k_Conjugation_reverse_o_involute[] = "\n        Conjugation, reverse o
    involute == involute o reverse.\n\n        >>> print((clifford(\"{1}\")).conj())\n        -{1}\n
    >>> print((clifford(\"{2}\") * clifford(\"{1}\")).conj())\n        {1,2}\n        >>>
    print((clifford(\"{1}\") * clifford(\"{2}\")).conj())\n        -{1,2}\n        >>>
    print(clifford(\"1+{1}+{1,2}\").conj())\n        1-{1}-{1,2}\n        ";
static const char __pyx_k_Geometric_product_x_clifford_2[] = "\n        Geometric product.\n\n
    >>> x = clifford(2); x *= clifford(\"{2}\"); print(x)\n        2{2}\n        >>> x = clifford(\"{1}\");
    x *= clifford(\"{2}\"); print(x)\n        {1,2}\n        >>> x = clifford(\"{1}\"); x *=
    clifford(\"{1,2}\"); print(x)\n        {2}\n        ";
static const char __pyx_k_Geometric_sum_print_clifford_1[] = "\n        Geometric sum.\n\n        >>>
    print(clifford(1) + clifford(\"{2}\"))\n        1+{2}\n        >>> print(clifford(\"{1}\") +
    clifford(\"{2}\"))\n        {1}+{2}\n        ";
static const char __pyx_k_Hyperbolic_sine_of_multivector[] = "\n    Hyperbolic sine of
    multivector.\n\n    >>> x=clifford(\"{1,2}\") * pi/2; print(sinh(x))\n    {1,2}\n    >>>
    x=clifford(\"{1,2}\") * pi/6; print(sinh(x))\n    0.5{1,2}\n    ";
static const char __pyx_k_Inner_product_print_clifford_1[] = "\n        Inner product.\n\n        >>>
    print(clifford(\"{1}\") & clifford(\"{2}\"))\n        0\n        >>> print(clifford(2) &
    clifford(\"{2}\"))\n        0\n        >>> print(clifford(\"{1}\") & clifford(\"{1}\"))\n        1\n
    >>> print(clifford(\"{1}\") & clifford(\"{1,2}\"))\n        {2}\n        ";
static const char __pyx_k_Inverse_tangent_of_multivector[] = "\n    Inverse tangent of multivector
    with optional complexifier.\n\n    >>> s=index_set({1,2,3}); x=clifford(\"{1}\");
    print(tan(atan(x,s),s))\n    {1}\n    >>> x=clifford(\"{1}\"); print(tan(atan(x)))\n    {1}\n    ";
static const char __pyx_k_Iterate_over_the_indices_of_an[] = "\n        Iterate over the indices of an
    index_set.\n\n        >>> for i in index_set({-3,4,7}):print(i, end=\",\")\n        -3,4,7,\n        ";
static const char __pyx_k_Maximum_member_index_set_1_1_2[] = "\n        Maximum member.\n\n        >>>
    index_set({-1,1,2}).max()\n        2\n        ";
static const char __pyx_k_Maximum_positive_index_or_0_if[] = "\n    Maximum positive index, or 0 if
    none.\n\n    >>> max_pos(index_set({1,2}))\n    2\n    ";
static const char __pyx_k_Minimum_member_index_set_1_1_2[] = "\n        Minimum member.\n\n        >>>
    index_set({-1,1,2}).min()\n        -1\n        ";
static const char __pyx_k_Minimum_negative_index_or_0_if[] = "\n    Minimum negative index, or 0 if
    none.\n\n    >>> min_neg(index_set({1,2}))\n    0\n    ";
static const char __pyx_k_Odd_part_of_multivector_sum_of[] = "\n        Odd part of multivector, sum
```

```
of odd grade terms.\n\n        >>> print(clifford(\"1+{1}+{1,2}\").odd())\n        {1}\n        ";
static const char __pyx_k_Outer_product_power_x_clifford[] = "\n        Outer product power.\n\n
>>> x=clifford(\"2+{1}\"); print(x.outer_pow(0))\n        1\n        >>> x=clifford(\"2+{1}\");
print(x.outer_pow(1))\n        2+{1}\n        >>> x=clifford(\"2+{1}\"); print(x.outer_pow(2))\n
4+4{1}\n        >>> print(clifford(\"1+{1}+{1,2}\").outer_pow(3))\n        1+3{1}+3{1,2}\n\n        ";
static const char __pyx_k_Outer_product_print_clifford_1[] = "\n        Outer product.\n\n        >>>
print(clifford(\"{1}\") ^ clifford(\"{2}\"))\n        {1,2}\n        >>> print(clifford(2) ^
clifford(\"{2}\"))\n        2{2}\n        >>> print(clifford(\"{1}\") ^ clifford(\"{1}\"))\n        0\n
>>> print(clifford(\"{1}\") ^ clifford(\"{1,2}\"))\n        0\n        ";
static const char __pyx_k_Power_self_to_the_m_x_clifford[] = "\n        Power: self to the m.\n\n
>>> x=clifford(\"{1}\"); print(x ** 2)\n        1\n        >>> x=clifford(\"2\"); print(x ** 2)\n
4\n        >>> x=clifford(\"2+{1}\"); print(x ** 0)\n        1\n        >>> x=clifford(\"2+{1}\");
print(x ** 1)\n        2+{1}\n        >>> x=clifford(\"2+{1}\"); print(x ** 2)\n        5+4{1}\n
>>> i=clifford(\"{1,2}\"); print(exp(pi/2) * (i ** i))\n        1\n        ";
static const char __pyx_k_Pure_part_print_clifford_1_1_1[] = "\n        Pure part.\n\n        >>>
print(clifford(\"1+{1}+{1,2}\").pure())\n        {1}+{1,2}\n        >>>
print(clifford(\"{1,2}\").pure())\n        {1,2}\n        ";
static const char __pyx_k_Quadratic_form_rev_x_x_0_print[] = "\n        Quadratic form ==
(rev(x)*x)(0).\n\n        >>> print(clifford(\"1+{1}+{1,2}\").quad())\n        3.0\n        >>>
print(clifford(\"1+{-1}+{1,2}+{1,2,3}\").quad())\n        2.0\n        ";
static const char __pyx_k_Quadratic_norm_error_tolerance[] = "\n    Quadratic norm error tolerance
relative to a specific multivector.\n\n    >>> print(error_squared_tol(clifford(\"{1}\")) * 3.0 -
error_squared_tol(clifford(\"1{1}-2{2}+3{3}\")))\n    0.0\n    ";
static const char __pyx_k_Set_complement_not_print_index[] = "\n        Set complement: not.\n\n
>>>
print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))\n
{-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}\n
";
static const char __pyx_k_Set_union_or_print_index_set_1[] = "\n        Set union: or.\n\n        >>>
print(index_set({1}) | index_set({2}))\n        {1,2}\n        >>> print(index_set({1,2}) |
index_set({2}))\n        {1,2}\n        ";
static const char __pyx_k_Transform_left_hand_side_using[] = "\n        Transform left hand side,
using right hand side as a transformation.\n\n        >>> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); print(y|x)\n        -{1}\n        >>> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); print(y|exp(x))\n        -{1}\n        ";
static const char __pyx_k_clifford_vector_part_line_1079[] = "clifford.vector_part (line 1079)";
static const char __pyx_k_index_set___getitem___line_191[] = "index_set.__getitem__ (line 191)";
static const char __pyx_k_index_set___setitem___line_179[] = "index_set.__setitem__ (line 179)";
static const char __pyx_k_lexicographic_compare_eg_3_4_5[] = "\n    \"lexicographic compare\" eg.
{3,4,5} is less than {3,7,8};\n    -1 if a<b, +1 if a>b, 0 if a==b.\n\n    >>>
compare(index_set({1,2}),index_set({-1,3}))\n    -1\n    >>>
compare(index_set({-1,4}),index_set({-1,3}))\n    1\n    ";
static const char __pyx_k_Abbreviation_for_clifford_index[] = "\n    Abbreviation for
clifford(index_set(obj)).\n\n    >>> print(e(1))\n    {1}\n    >>> print(e(-1))\n    {-1}\n    >>>
print(e(0))\n    1\n    ";
static const char __pyx_k_Absolute_value_of_multivector_m[] = "\n    Absolute value of multivector:
multivector 2-norm.\n\n    >>> abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    2.0\n    ";
static const char __pyx_k_Absolute_value_square_root_of_n[] = "\n        Absolute value: square root
of norm.\n\n        >>> clifford(\"1+{-1}+{1,2}+{1,2,3}\").abs()\n        2.0\n        ";
static const char __pyx_k_Cardinality_Number_of_indices_i[] = "\n        Cardinality: Number of
indices included in set.\n\n        >>> index_set({-1,1,2}).count()\n        3\n        ";
static const char __pyx_k_Check_if_a_multivector_contains[] = "\n        Check if a multivector
contains any infinite values.\n\n        >>> clifford().isinf()\n        False\n        ";
static const char __pyx_k_Contraction_print_clifford_1_cl[] = "\n        Contraction.\n\n        >>>
print(clifford(\"{1}\") % clifford(\"{2}\"))\n        0\n        >>> print(clifford(2) %
clifford(\"{2}\"))\n        2{2}\n        >>> print(clifford(\"{1}\") % clifford(\"{1}\"))\n        1\n
>>> print(clifford(\"{1}\") % clifford(\"{1,2}\"))\n        {2}\n        ";
static const char __pyx_k_Contraction_x_clifford_1_x_clif[] = "\n        Contraction.\n\n        >>> x
= clifford(\"{1}\"); x %= clifford(\"{2}\"); print(x)\n        0\n        >>> x = clifford(2); x %=
clifford(\"{2}\"); print(x)\n        2{2}\n        >>> x = clifford(\"{1}\"); x %= clifford(\"{1}\");
print(x)\n        1\n        >>> x = clifford(\"{1}\"); x %= clifford(\"{1,2}\"); print(x)\n
{2}\n        ";
static const char __pyx_k_Convert_CGA3_null_vector_to_Euc[] = "\n    Convert CGA3 null vector to
Euclidean 3D vector using Doran and Lasenby definition.\n\n    >>> x=clifford(\"2{1}+9{2}+{3}\");
print(agc3(cga3(x)))\n    2{1}+9{2}+{3}\n    >>> x=clifford(\"2{1}+9{2}+{3}\");
print(agc3(cga3(x))-x)\n    0\n    ";
static const char __pyx_k_Convert_CGA3_null_vector_to_sta[] = "\n    Convert CGA3 null vector to
standard conformal null vector using Doran and Lasenby definition.\n\n    >>>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x)))\n    87{-1}+4{1}+18{2}+2{3}+85{4}\n    >>>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x))-cga3(x))\n    0\n    ";
static const char __pyx_k_Convert_Euclidean_3D_multivecto[] = "\n    Convert Euclidean 3D multivector
to Conformal Geometric Algebra using Doran and Lasenby definition.\n\n    >>>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3(x))\n    87{-1}+4{1}+18{2}+2{3}+85{4}\n    ";
static const char __pyx_k_Copy_this_clifford_object_x_cli[] = "\n        Copy this clifford
object.\n\n        >>> x=clifford(\"1{2}\"); y=x.copy(); print(y)\n        {2}\n        ";
static const char __pyx_k_Copy_this_index_set_object_s_in[] = "\n        Copy this index_set
object.\n\n        >>> s=index_set(1); t=s.copy(); print(t)\n        {1}\n        ";
static const char __pyx_k_Cosine_of_multivector_with_opti[] = "\n    Cosine of multivector with
optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{1,2,3}\"))\n    {1,2}\n
>>> x=clifford(\"{1,2}\"); print(cos(acos(x)))\n    {1,2}\n    ";
static const char __pyx_k_Even_part_of_multivector_sum_of[] = "\n        Even part of multivector, sum
of even grade terms.\n\n        >>> print(clifford(\"1+{1}+{1,2}\").even())\n        1+{1,2}\n
";
static const char __pyx_k_Exponential_of_multivector_x_cl[] = "\n    Exponential of multivector.\n\n
>>> x=clifford(\"{1,2}\") * pi/4; print(exp(x))\n    0.7071+0.7071{1,2}\n    >>> x=clifford(\"{1,2}\") *
pi/2; print(exp(x))\n    {1,2}\n    ";
static const char __pyx_k_Geometric_difference_print_clif[] = "\n        Geometric difference.\n\n
```

```
    >> print(clifford(1) - clifford(\"{2}\"))\n        1-{2}\n        >> print(clifford(\"{1}\") -
    clifford(\"{2}\"))\n        {1}-{2}\n        ";
static const char __pyx_k_Geometric_difference_x_clifford[] = "\n        Geometric difference.\n\n
    >> x = clifford(1); x -= clifford(\"{2}\"); print(x)\n        1-{2}\n        ";
static const char __pyx_k_Geometric_multiplicative_invers[] = "\n        Geometric multiplicative
    inverse.\n\n        >> x = clifford(\"{1}\"); print(x.inv())\n        {1}\n        >> x = clifford(2);
    print(x.inv())\n        0.5\n        >> x = clifford(\"{1,2}\"); print(x.inv())\n        -{1,2}\n
    ";
static const char __pyx_k_Geometric_product_print_cliffor[] = "\n        Geometric product.\n\n
    >> print(clifford(\"{1}\") * clifford(\"{2}\"))\n        {1,2}\n        >> print(clifford(2) *
    clifford(\"{2}\"))\n        2{2}\n        >> print(clifford(\"{1}\") * clifford(\"{1,2}\"))\n
    {2}\n        ";
static const char __pyx_k_Geometric_quotient_print_cliffo[] = "\n        Geometric quotient.\n\n
    >> print(clifford(\"{1}\") / clifford(\"{2}\"))\n        {1,2}\n        >> print(clifford(2) /
    clifford(\"{2}\"))\n        2{2}\n        >> print(clifford(\"{1}\") / clifford(\"{1}\"))\n        1\n
    >> print(clifford(\"{1}\") / clifford(\"{1,2}\"))\n        -{2}\n        ";
static const char __pyx_k_Geometric_quotient_x_clifford_1[] = "\n        Geometric quotient.\n\n
    >> x = clifford(\"{1}\"); x /= clifford(\"{2}\"); print(x)\n        {1,2}\n        >> x = clifford(2);
    x /= clifford(\"{2}\"); print(x)\n        2{2}\n        >> x = clifford(\"{1}\"); x /=
    clifford(\"{1}\"); print(x)\n        1\n        >> x = clifford(\"{1}\"); x /= clifford(\"{1,2}\");
    print(x)\n        -{2}\n        ";
static const char __pyx_k_Geometric_sum_x_clifford_1_x_cl[] = "\n        Geometric sum.\n\n        >>
    x = clifford(1); x += clifford(\"{2}\"); print(x)\n        1+{2}\n        ";
static const char __pyx_k_Get_the_value_of_an_index_set_o[] = "\n        Get the value of an index_set
    object at an index.\n\n        >> index_set({1})[1]\n        True\n        >> index_set({1})[2]\n
    False\n        >> index_set({2})[-1]\n        False\n        >> index_set({2})[1]\n        False\n
    >> index_set({2})[2]\n        True\n        >> index_set({2})[33]\n        False\n        ";
static const char __pyx_k_Hyperbolic_cosine_of_multivecto[] = "\n    Hyperbolic cosine of
    multivector.\n\n    >> x=clifford(\"{1,2}\") * pi; print(cosh(x))\n    -1\n    >>
    x=clifford(\"{1,2,3}\"); print(cosh(acosh(x)))\n    {1,2,3}\n    >> x=clifford(\"{1,2}\");
    print(cosh(acosh(x)))\n    {1,2}\n    ";
static const char __pyx_k_Hyperbolic_tangent_of_multivect[] = "\n    Hyperbolic tangent of
    multivector.\n\n    >> x=clifford(\"{1,2}\") * pi/4; print(tanh(x))\n    {1,2}\n    ";
static const char __pyx_k_Imaginary_part_deprecated_alway[] = "\n    Imaginary part: deprecated
    (always 0).\n\n    >> imag(clifford(\"1+{1}+{1,2}\"))\n    0.0\n    >> imag(clifford(\"{1,2}\"))\n
    0.0\n    ";
static const char __pyx_k_Inner_product_x_clifford_1_x_cl[] = "\n        Inner product.\n\n        >>
    x = clifford(\"{1}\"); x &= clifford(\"{2}\"); print(x)\n        0\n        >> x = clifford(2); x &=
    clifford(\"{2}\"); print(x)\n        0\n        >> x = clifford(\"{1}\"); x &= clifford(\"{1}\");
    print(x)\n        1\n        >> x = clifford(\"{1}\"); x &= clifford(\"{1,2}\"); print(x)\n
    {2}\n        ";
static const char __pyx_k_Integer_power_of_multivector_ob[] = "\n    Integer power of multivector: obj
    to the m.\n\n    >> x=clifford(\"{1}\"); print(pow(x,2))\n    1\n    >> x=clifford(\"2\");
    print(pow(x,2))\n    4\n    >> x=clifford(\"2+{1}\"); print(pow(x,0))\n    1\n    >>
    x=clifford(\"2+{1}\"); print(pow(x,1))\n    2+{1}\n    >> x=clifford(\"2+{1}\"); print(pow(x,2))\n
    5+4{1}\n    >> print(pow(clifford(\"1+{1}+{1,2}\"),3))\n    1+3{1}+3{1,2}\n    >>
    i=clifford(\"{1,2}\"); print(exp(pi/2) * pow(i, i))\n    1\n    ";
static const char __pyx_k_Inverse_cosine_of_multivector_w[] = "\n    Inverse cosine of multivector
    with optional complexifier.\n\n    >> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{1,2,3}\"))\n
    {1,2}\n    >> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{-1,1,2,3,4}\"))\n    {1,2}\n    >>
    print(acos(0) / pi)\n    0.5\n    >> x=clifford(\"{1,2}\"); print(cos(acos(x)))\n    {1,2}\n    ";
static const char __pyx_k_Inverse_hyperbolic_cosine_of_mu[] = "\n    Inverse hyperbolic cosine of
    multivector with optional complexifier.\n\n    >> print(acosh(0,\"{-2,-1,1}\"))\n    1.571{-2,-1,1}\n
    >> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x,\"{-1,1,2,3,4}\")))\n    {1,2,3}\n    >>
    print(acosh(0))\n    1.571{-1}\n    >> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x)))\n    {1,2,3}\n
    >> x=clifford(\"{1,2}\"); print(cosh(acosh(x)))\n    {1,2}\n    ";
static const char __pyx_k_Inverse_hyperbolic_sine_of_mult[] = "\n    Inverse hyperbolic sine of
    multivector with optional complexifier.\n\n    >> x=clifford(\"{1,2}\"); print(asinh(x,\"{1,2,3}\") *
    2/pi)\n    {1,2}\n    >> x=clifford(\"{1,2}\"); print(asinh(x) * 2/pi)\n    {1,2}\n    >>
    x=clifford(\"{1,2}\") / 2; print(asinh(x) * 6/pi)\n    {1,2}\n    ";
static const char __pyx_k_Inverse_hyperbolic_tangent_of_m[] = "\n    Inverse hyperbolic tangent of
    multivector with optional complexifier.\n\n    >> s=index_set({1,2,3}); x=clifford(\"{1,2}\");
    print(tanh(atanh(x,s)))\n    {1,2}\n    >> x=clifford(\"{1,2}\"); print(tanh(atanh(x)))\n    {1,2}\n
    ";
static const char __pyx_k_Inverse_sine_of_multivector_wit[] = "\n    Inverse sine of multivector with
    optional complexifier.\n\n    >> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n    {-1}\n    >>
    s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n    {-1}\n    >> print(asin(1) / pi)\n
    0.5\n    >> x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n    {1,2,3}\n    ";
static const char __pyx_k_Main_involution_each_i_is_repla[] = "\n        Main involution, each {i} is
    replaced by -{i} in each term,\n        eg. clifford(\"{1}\") -> -clifford(\"{1}\").\n\n        >>
    print(clifford(\"{1}\").involute())\n        -{1}\n        >> print((clifford(\"{2}\") *
    clifford(\"{1}\")).involute())\n        -{1,2}\n        >> print((clifford(\"{1}\") *
    clifford(\"{2}\")).involute())\n        {1,2}\n        >>
    print(clifford(\"1+{1}+{1,2}\").involute())\n        1-{1}+{1,2}\n        ";
static const char __pyx_k_Maximum_absolute_value_of_coord[] = "\n    Maximum absolute value of
    coordinates multivector: multivector infinity-norm.\n\n    >>
    max_abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    1.0\n    >> max_abs(clifford(\"3+2{1}+{1,2}\"))\n
    3.0\n\n    ";
static const char __pyx_k_Maximum_of_absolute_values_of_c[] = "\n        Maximum of absolute values of
    components of multivector: multivector infinity norm.\n\n        >>
    clifford(\"1+{-1}+{1,2}+{1,2,3}\").max_abs()\n        1.0\n        >>
    clifford(\"3+2{1}+{1,2}\").max_abs()\n        3.0\n        ";
static const char __pyx_k_Natural_logarithm_of_multivecto[] = "\n    Natural logarithm of multivector
    with optional complexifier.\n\n    >> x=clifford(\"{-1}\"); print((log(x,\"{-1}\") * 2/pi))\n
    {-1}\n    >> x=clifford(\"{1,2}\"); print((log(x,\"{1,2,3}\") * 2/pi))\n    {1,2}\n    >>
    x=clifford(\"{1,2}\"); print((log(x) * 2/pi))\n    {1,2}\n    >> x=clifford(\"{1,2}\");
    print((log(x,\"{1,2}\") * 2/pi))\n    Traceback (most recent call last):\n    ...\n    RuntimeError:
```

```
check_complex(val, i): i is not a valid complexifier for val\n    ";
static const char __pyx_k_Norm_sum_of_squares_of_coordina[] = "\n        Norm == sum of squares of
coordinates.\n\n        >>> clifford(\"1+{1}+{1,2}\").norm()\n        3.0\n        >>>
clifford(\"1+{-1}+{1,2}+{1,2,3}\").norm()\n        4.0\n        ";
static const char __pyx_k_Not_applicable_for_a_in_cliffor[] = "\n        Not applicable.\n\n        >>>
for a in clifford(index_set({-3,4,7})):print(a, end=\",\")\n        Traceback (most recent call
last):\n          ...\n        TypeError: Not applicable.\n        ";
static const char __pyx_k_Number_of_negative_indices_incl[] = "\n        Number of negative indices
included in set.\n\n        >>> index_set({-1,1,2}).count_neg()\n        1\n        ";
static const char __pyx_k_Number_of_positive_indices_incl[] = "\n        Number of positive indices
included in set.\n\n        >>> index_set({-1,1,2}).count_pos()\n        2\n        ";
static const char __pyx_k_Outer_product_power_of_multivec[] = "\n    Outer product power of
multivector.\n\n    >>> print(outer_pow(clifford(\"1+{1}+{1,2}\"),3))\n    1+3{1}+3{1,2}\n    ";
static const char __pyx_k_Outer_product_x_clifford_1_x_cl[] = "\n        Outer product.\n\n        >>>
x = clifford(\"{1}\"); x ^= clifford(\"{2}\"); print(x)\n        {1,2}\n        >>> x = clifford(2); x
^= clifford(\"{2}\"); print(x)\n        2{2}\n        >>> x = clifford(\"{1}\"); x ^=
clifford(\"{1}\"); print(x)\n        0\n        >>> x = clifford(\"{1}\"); x ^= clifford(\"{1,2}\");
print(x)\n        0\n        ";
static const char __pyx_k_Pure_grade_vector_part_print_cl[] = "\n        Pure grade-vector part.\n\n
>>> print(clifford(\"{1}\")(1))\n        {1}\n        >>> print(clifford(\"{1}\")(0))\n        0\n
>>> print(clifford(\"1+{1}+{1,2}\")(0))\n        1\n        >>> print(clifford(\"1+{1}+{1,2}\")(1))\n
{1}\n        >>> print(clifford(\"1+{1}+{1,2}\")(2))\n        {1,2}\n        >>>
print(clifford(\"1+{1}+{1,2}\")(3))\n        0\n        ";
static const char __pyx_k_Pure_part_print_pure_clifford_1[] = "\n    Pure part\n\n    >>>
print(pure(clifford(\"1+{1}+{1,2}\")))\n    {1}+{1,2}\n    >>> print(pure(clifford(\"{1,2}\")))\n
{1,2}\n    ";
static const char __pyx_k_Put_self_into_a_larger_frame_co[] = "\n        Put self into a larger frame,
containing the union of self.frame() and index set ixt.\n        This can be used to make
multiplication faster, by multiplying within a common frame.\n\n        >>>
clifford(\"2+3{1}\").reframe(index_set({1,2,3}))\n        clifford(\"2+3{1}\")\n        >>>
s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() == (s|t);\n
True\n        ";
static const char __pyx_k_Random_multivector_within_a_fra[] = "\n    Random multivector within a
frame.\n\n    >>> print(random_clifford(index_set({-3,-1,2})).frame())\n    {-3,-1,2}\n    ";
static const char __pyx_k_Real_part_synonym_for_scalar_pa[] = "\n    Real part: synonym for scalar
part.\n\n    >>> real(clifford(\"1+{1}+{1,2}\"))\n    1.0\n    >>> real(clifford(\"{1,2}\"))\n    0.0\n
";
static const char __pyx_k_Relative_or_absolute_error_usin[] = "\n    Relative or absolute error using
the quadratic norm.\n\n    >>> err2=scalar_epsilon*scalar_epsilon\n\n    >>>
print(error_squared(clifford(\"{1}\"), clifford(\"1{1}\"), err2))\n    0.0\n    >>>
print(error_squared(clifford(\"1{1}-3{2}+4{3}\"), clifford(\"{1}\"), err2))\n    25.0\n    ";
static const char __pyx_k_Remove_all_terms_of_self_with_r[] = "\n        Remove all terms of self with
relative size smaller than limit.\n\n        >>> clifford(\"1e8+{1}+1e-8{1,2}\").truncated(1.0e-6)\n
clifford(\"100000000\")\n        >>> clifford(\"1e4+{1}+1e-4{1,2}\").truncated(1.0e-6)\n
clifford(\"10000+{1}\")\n        ";
static const char __pyx_k_Reversion_eg_1_2_2_1_print_reve[] = "\n    Reversion, eg. {1}*{2} ->
{2}*{1}\n\n    >>> print(reverse(clifford(\"{1}\")))\n    {1}\n    >>> print(reverse(clifford(\"{2}\") *
clifford(\"{1}\")))\n    {1,2}\n    >>> print(reverse(clifford(\"{1}\") * clifford(\"{2}\")))\n
-{1,2}\n    >>> print(reverse(clifford(\"1+{1}+{1,2}\")))\n    1+{1}-{1,2}\n    ";
static const char __pyx_k_Reversion_eg_clifford_1_cliffor[] = "\n        Reversion, eg.
clifford(\"{1}\")*clifford(\"{2}\") -> clifford(\"{2}\")*clifford(\"{1}\").\n\n        >>>
print(clifford(\"{1}\").reverse())\n        {1}\n        >>> print((clifford(\"{2}\") *
clifford(\"{1}\")).reverse())\n        {1,2}\n        >>> print((clifford(\"{1}\") *
clifford(\"{2}\")).reverse())\n        -{1,2}\n        >>> print(clifford(\"1+{1}+{1,2}\").reverse())\n
1+{1}-{1,2}\n        ";
static const char __pyx_k_Scalar_part_clifford_1_1_1_2_sc[] = "\n        Scalar part.\n\n        >>>
clifford(\"1+{1}+{1,2}\").scalar()\n        1.0\n        >>> clifford(\"{1,2}\").scalar()\n
0.0\n        ";
static const char __pyx_k_Scalar_part_scalar_clifford_1_1[] = "\n    Scalar part.\n\n    >>>
scalar(clifford(\"1+{1}+{1,2}\"))\n    1.0\n    >>> scalar(clifford(\"{1,2}\"))\n    0.0\n    ";
static const char __pyx_k_Set_intersection_and_print_inde[] = "\n        Set intersection: and.\n\n
>>> print(index_set({1}) & index_set({2}))\n        {}\n        >>> print(index_set({1,2}) &
index_set({2}))\n        {2}\n        ";
static const char __pyx_k_Set_intersection_and_x_index_se[] = "\n        Set intersection: and.\n\n
>>> x = index_set({1}); x &= index_set({2}); print(x)\n        {}\n        >>> x = index_set({1,2}); x
&= index_set({2}); print(x)\n        {2}\n        ";
static const char __pyx_k_Set_the_value_of_an_index_set_o[] = "\n        Set the value of an index_set
object at index idx to value val.\n\n        >>> s=index_set({1}); s[2] = True; print(s)\n
{1,2}\n        >>> s=index_set({1,2}); s[1] = False; print(s)\n        {2}\n        ";
static const char __pyx_k_Set_union_or_x_index_set_1_x_in[] = "\n        Set union: or.\n\n        >>>
x = index_set({1}); x |= index_set({2}); print(x)\n        {1,2}\n        >>> x = index_set({1,2}); x
|= index_set({2}); print(x)\n        {1,2}\n        ";
static const char __pyx_k_Sign_of_geometric_product_of_tw[] = "\n        Sign of geometric product of
two Clifford basis elements.\n\n        >>> s = index_set({1,2}); t=index_set({-1});
s.sign_of_mult(t)\n        1\n        ";
static const char __pyx_k_Sign_of_geometric_square_of_a_C[] = "\n        Sign of geometric square of a
Clifford basis element.\n\n        >>> s = index_set({1,2}); s.sign_of_square()\n        -1\n        ";
static const char __pyx_k_Sine_of_multivector_with_option[] = "\n    Sine of multivector with optional
complexifier.\n\n    >>> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n    {-1}\n    >>>
s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n    {-1}\n    >>>
x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n    {1,2,3}\n    ";
static const char __pyx_k_Square_root_of_1_which_commutes[] = "\n    Square root of -1 which commutes
with all members of the frame of the given multivector.\n\n    >>>
print(complexifier(clifford(index_set({1}))))\n    {1,2,3}\n    >>>
print(complexifier(clifford(index_set({-1}))))\n    {-1}\n    >>> print(complexifier(index_set({1})))\n
{1,2,3}\n    >>> print(complexifier(index_set({-1})))\n    {-1}\n    ";
static const char __pyx_k_Square_root_of_multivector_with[] = "\n    Square root of multivector with
```

```
optional complexifier.\n\n    >>> print(sqrt(-1))\n    {-1}\n    >>> print(sqrt(clifford(\"2{-1}\")))\n
1+{-1}\n    >>> j=sqrt(-1,complexifier(index_set({1}))); print(j); print(j*j)\n    {1,2,3}\n    -1\n
>>> j=sqrt(-1,\"{1,2,3}\"); print(j); print(j*j)\n    {1,2,3}\n    -1\n    ";
static const char __pyx_k_Subalgebra_generated_by_all_gen[] = "\n        Subalgebra generated by all
generators of terms of given multivector.\n\n        >>>
print(clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame())\n        {-2,-1,1,2,7}\n        >>>
s=clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame(); type(s)\n        <class 'PyClical.index_set'>\n
";
static const char __pyx_k_Subscripting_map_from_index_set[] = "\n        Subscripting: map from index
set to scalar coordinate.\n\n        >>> clifford(\"{1}\")[index_set(1)]\n        1.0\n        >>>
clifford(\"{1}\")[index_set({1})]\n        1.0\n        >>> clifford(\"{1}\")[index_set({1,2})]\n
0.0\n        >>> clifford(\"2{1,2}\")[index_set({1,2})]\n        2.0\n        ";
static const char __pyx_k_Symmetric_set_difference_exclus[] = "\n        Symmetric set difference:
exclusive or.\n\n        >>> print(index_set({1}) ^ index_set({2}))\n        {1,2}\n        >>>
print(index_set({1,2}) ^ index_set({2}))\n        {1}\n        ";
static const char __pyx_k_Tangent_of_multivector_with_opt[] = "\n    Tangent of multivector with
optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(tan(x,\"{1,2,3}\"))\n    0.7616{1,2}\n
>>> x=clifford(\"{1,2}\"); print(tan(x))\n    0.7616{1,2}\n    ";
static const char __pyx_k_Test_for_approximate_equality_o[] = "\n    Test for approximate equality of
multivectors.\n\n    >>> err2=scalar_epsilon*scalar_epsilon\n\n    >>>
print(approx_equal(clifford(\"{1}\"), clifford(\"1{1}\")))\n    True\n    >>>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}\"), clifford(\"{1}\")))\n    False\n    >>>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}+0.001\"), clifford(\"1{1}-3{2}+4{3}\"), err2, err2))\n
False\n    >>> print(approx_equal(clifford(\"1{1}-3{2}+4{3}+1.0e-30\"), clifford(\"1{1}-3{2}+4{3}\"),
err2, err2))\n    True\n    ";
static const char __pyx_k_Tests_for_functions_that_Doctes[] = "\n    Tests for functions that Doctest
cannot see.\n\n    For index_set.__cinit__: Construct index_set.\n\n    >>> print(index_set(1))\n
{1}\n    >>> print(index_set({1,2}))\n    {1,2}\n    >>> print(index_set(index_set({1,2})))\n    {1,2}\n
>>> print(index_set({1,2}))\n    {1,2}\n    >>> print(index_set({1,2,1}))\n    {1,2}\n    >>>
print(index_set({1,2,1}))\n    {1,2}\n    >>> print(index_set(\"\"))\n    {}\n    >>>
print(index_set(\"{\"))\n    Traceback (most recent call last):\n    ...\n    ValueError: Cannot
initialize index_set object from invalid string '{'.\n    >>> print(index_set(\"{1\"))\n    Traceback
(most recent call last):\n    ...\n    ValueError: Cannot initialize index_set object from invalid
string '{1'.\n    >>> print(index_set(\"{1,2,100\"))\n    Traceback (most recent call last):\n
...\n    ValueError: Cannot initialize index_set object from invalid string '{1,2,100'.\n    >>>
print(index_set({1,2,100}))\n    Traceback (most recent call last):\n    ...\n    IndexError: Cannot
initialize index_set object from invalid {1, 2, 100}.\n    >>> print(index_set([1,2]))\n    Traceback
(most recent call last):\n    ...\n    TypeError: Cannot initialize index_set object from <class
'list'>.\n\n    For index_set.__richcmp__: Compare two objects of class index_set.\n\n    >>>
index_set(1) == index_set({1})\n    True\n    >>> index_set({1}) != index_set({1})\n    False\n    >>>
index_set({1}) != index_set({2})\n    True\n    >>> index_set({1}) == index_set({2})\n    False\n    >>>
index_set({1}) < index_set({2})\n    True\n    >>> index_set({1}) <= index_set({2})\n    True\n    >>>
index_set({1}) > index_set({2})\n    False\n    >>> index_set({1}) >= index_set({2})\n    False\n    >>>
None == index_set({1,2})\n    False\n    >>> None != index_set({1,2})\n    True\n    >>> None <
index_set({1,2})\n    False\n    >>> None <= index_set({1,2})\n    False\n    >>> None >
index_set({1,2})\n    False\n    >>> None >= index_set({1,2})\n    False\n    >>> ""index_set({1,2}) ==
None\n    False\n    >>> index_set({1,2}) != None\n    True\n    >>> index_set({1,2}) < None\n
False\n    >>> index_set({1,2}) <= None\n    False\n    >>> index_set({1,2}) > None\n    False\n    >>>
index_set({1,2}) >= None\n    False\n    ";
        static const char __pyx_k_The_informal_string_representat[] = "\n        The
\342\200\234informal\342\200\235 string representation of self.\n\n        >>>
index_set({1,2}).__str__()\n        '{1,2}'\n        >>> str(index_set({1,2}))\n        '{1,2}'\n
";
        static const char __pyx_k_The_official_string_representat[] = "\n        The
\342\200\234official\342\200\235 string representation of self.\n\n        >>>
index_set({1,2}).__repr__()\n        'index_set({1,2})'\n        >>> repr(index_set({1,2}))\n
'index_set({1,2})'\n        ";
        static const char __pyx_k_This_comparison_operator_is_not[] = "This comparison operator is not
implemented for ";
        static const char __pyx_k_Vector_part_of_multivector_as_a[] = "\n        Vector part of
multivector, as a Python list, with respect to frm.\n\n        >>>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part())\n        [2.0, 3.0]\n        >>>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part(index_set({-1,1,2})))\n        [0.0, 2.0, 3.0]\n
";
        static const char __pyx_k_index_set_sign_of_mult_line_366[] = "index_set.sign_of_mult (line
366)";
        static const char __pyx_k_norm_sum_of_squares_of_coordina[] = "\n    norm == sum of squares of
coordinates.\n\n    >>> norm(clifford(\"1+{1}+{1,2}\"))\n    3.0\n    >>>
norm(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    4.0\n    ";
        static const char __pyx_k_Cannot_initialize_clifford_objec[] = "Cannot initialize clifford
object from";
        static const char __pyx_k_Cannot_initialize_index_set_obje[] = "Cannot initialize index_set
object from";
        static const char __pyx_k_Conjugation_reverse_o_involute_2[] = "\n    Conjugation, reverse o
involute == involute o reverse.\n\n    >>> print(conj(clifford(\"{1}\")))\n    -{1}\n    >>>
print(conj(clifford(\"{2}\") * clifford(\"{1}\")))\n    {1,2}\n    >>> print(conj(clifford(\"{1}\") *
clifford(\"{2}\")))\n    -{1,2}\n    >>> print(conj(clifford(\"1+{1}+{1,2}\")))\n    1-{1}-{1,2}\n
";
        static const char __pyx_k_Odd_part_of_multivector_sum_of_2[] = "\n    Odd part of multivector,
sum of odd grade terms.\n\n    >>> print(odd(clifford(\"1+{1}+{1,2}\")))\n    {1}\n    ";
        static const char __pyx_k_Power_self_to_the_m_x_clifford_2[] = "\n        Power: self to the
m.\n\n        >>> x=clifford(\"{1}\"); print(x.pow(2))\n        1\n        >>> x=clifford(\"2\");
print(x.pow(2))\n        4\n        >>> x=clifford(\"2+{1}\"); print(x.pow(0))\n        1\n        >>>
x=clifford(\"2+{1}\"); print(x.pow(1))\n        2+{1}\n        >>> x=clifford(\"2+{1}\");
print(x.pow(2))\n        5+4{1}\n        >>> print(clifford(\"1+{1}+{1,2}\").pow(3))\n
1+3{1}+3{1,2}\n        >>> i=clifford(\"{1,2}\"); print(exp(pi/2) * i.pow(i))\n        1\n        ";
        static const char __pyx_k_Quadratic_form_rev_x_x_0_print_2[] = "\n    Quadratic form ==
```

```
(rev(x)*x)(0).\n\n    >>> print(quad(clifford(\"1+{1}+{1,2}\")))\n    3.0\n    >>>
print(quad(clifford(\"1+{-1}+{1,2}+{1,2,3}\")))\n    2.0\n    ";
        static const char __pyx_k_Transform_left_hand_side_using_2[] = "\n        Transform left hand
side, using right hand side as a transformation.\n\n        >>> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); y|=x; print(y)\n        -{1}\n        >>> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); y|=exp(x); print(y)\n        -{1}\n        ";
        static const char __pyx_k_clifford_hidden_doctests_line_12[] = "clifford_hidden_doctests (line
1253)";
        static const char __pyx_k_index_set_hidden_doctests_line_4[] = "index_set_hidden_doctests
(line 406)";
        static const char __pyx_k_index_set_sign_of_square_line_37[] = "index_set.sign_of_square (line
375)";
        static const char __pyx_k_no_default___reduce___due_to_non[] = "no default __reduce__ due to
non-trivial __cinit__";
        static const char __pyx_k_Check_if_a_multivector_contains_2[] = "\n        Check if a
multivector contains any IEEE NaN values.\n\n        >>> clifford().isnan()\n        False\n        ";
        static const char __pyx_k_Even_part_of_multivector_sum_of_2[] = "\n    Even part of
multivector, sum of even grade terms.\n\n    >>> print(even(clifford(\"1+{1}+{1,2}\")))\n    1+{1,2}\n
";
        static const char __pyx_k_Geometric_multiplicative_invers_2[] = "\n    Geometric
multiplicative inverse.\n\n    >>> print(inv(clifford(\"{1}\")))\n    {1}\n    >>>
print(inv(clifford(\"{-1}\")))\n    -{-1}\n    >>> print(inv(clifford(\"{-2,-1}\")))\n    -{-2,-1}\n
>>> print(inv(clifford(\"{-1}+{1}\")))\n    nan\n    ";
        static const char __pyx_k_Main_involution_each_i_is_repla_2[] = "\n    Main involution, each
{i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})\n\n    >>>
print(involute(clifford(\"{1}\")))\n    -{1}\n    >>> print(involute(clifford(\"{2}\") *
clifford(\"{1}\")))\n    -{1,2}\n    >>> print(involute(clifford(\"{1}\") * clifford(\"{2}\")))\n
{1,2}\n    >>> print(involute(clifford(\"1+{1}+{1,2}\")))\n    1-{1}+{1,2}\n    ";
        static const char __pyx_k_Symmetric_set_difference_exclus_2[] = "\n        Symmetric set
difference: exclusive or.\n\n        >>> x = index_set({1}); x ^= index_set({2}); print(x)\n
{1,2}\n        >>> x = index_set({1,2}); x ^= index_set({2}); print(x)\n        {1}\n        ";
        static const char __pyx_k_Tests_for_functions_that_Doctes_2[] = "\n    Tests for functions
that Doctest cannot see.\n\n    For clifford.__cinit__: Construct an object of type clifford.\n\n
>>> print(clifford(2))\n    2\n    >>> print(clifford(2.0))\n    2\n    >>> print(clifford(1.0e-1))\n
0.1\n    >>> print(clifford(\"2\"))\n    2\n    >>> print(clifford(\"2{1,2,3}\"))\n    2{1,2,3}\n    >>>
print(clifford(clifford(\"2{1,2,3}\")))\n    2{1,2,3}\n    >>> print(clifford(\"-{1}\"))\n    -{1}\n
>>> print(clifford(2,index_set({1,2})))\n    2{1,2}\n    >>> print(clifford([2,3],index_set({1,2})))\n
2{1}+3{2}\n    >>> print(clifford([1,2]))\n    Traceback (most recent call last):\n      ...\n
TypeError: Cannot initialize clifford object from <class 'list'>.\n    >>> print(clifford(None))\n
Traceback (most recent call last):\n      ...\n    TypeError: Cannot initialize clifford object from
<class 'NoneType'>.\n    >>> print(clifford(None,[1,2]))\n    Traceback (most recent call last):\n
...\n    TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).\n
>>> print(clifford([1,2],[1,2]))\n    Traceback (most recent call last):\n      ...\n    TypeError:
Cannot initialize clifford object from (<class 'list'>, <class 'list'>).\n    >>>
print(clifford(\"\"))\n    Traceback (most recent call last):\n      ...\n    ValueError: Cannot
initialize clifford object from invalid string ''.\n    >>> print(clifford(\"{\"))\n    Traceback (most
recent call last):\n      ...\n    ValueError: Cannot initialize clifford object from invalid string
'{'.\n    >>> print(clifford(\"{1\"))\n    Traceback (most recent call last):\n      ...\n
ValueError: Cannot initialize clifford object from invalid string '{1'.\n    >>>
print(clifford(\"+\"))\n    Traceback (most recent call last):\n      ...\n    ValueError: Cannot
initialize clifford object from invalid string '+'.\n    >>> print(clifford(\"-\"))\n    Traceback
(most recent call last):\n      ...\n    ValueError: Cannot initialize clifford object fro""m invalid
string '-'.\n    >>> print(clifford(\"{1}+\"))\n    Traceback (most recent call last):\n      ...\n
ValueError: Cannot initialize clifford object from invalid string '{1}+'.\n\n    For
clifford.__richcmp__: Compare objects of type clifford.\n\n    >>> clifford(\"{1}\") ==
clifford(\"1{1}\")\n    True\n    >>> clifford(\"{1}\") != clifford(\"1.0{1}\")\n    False\n    >>>
clifford(\"{1}\") != clifford(\"1.0\")\n    True\n    >>> clifford(\"{1,2}\") == None\n    False\n
>>> clifford(\"{1,2}\") != None\n    True\n    >>> None == clifford(\"{1,2}\")\n    False\n    >>> None
!= clifford(\"{1,2}\")\n    True\n    ";
                static const char __pyx_k_The_informal_string_representat_2[] = "\n        The
\342\200\234informal\342\200\235 string representation of self.\n\n        >>>
clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").__str__()\n        '1+3{-1}+2{1,2}+4{-2,7}'\n        ";
                static const char __pyx_k_The_official_string_representat_2[] = "\n        The
\342\200\234official\342\200\235 string representation of self.\n\n        >>>
clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").__repr__()\n        'clifford(\"1+3{-1}+2{1,2}+4{-2,7}\")'\n
";
                static PyObject *__pyx_kp_u_;
                static PyObject *__pyx_kp_u_Abbreviation_for_clifford_index;
                static PyObject *__pyx_kp_u_Abbreviation_for_index_set_q_p;
                static PyObject *__pyx_kp_u_Absolute_value_of_multivector_m;
                static PyObject *__pyx_kp_u_Absolute_value_square_root_of_n;
                static PyObject *__pyx_kp_u_Cannot_initialize_clifford_objec;
                static PyObject *__pyx_kp_u_Cannot_initialize_index_set_obje;
                static PyObject *__pyx_kp_u_Cannot_reframe;
                static PyObject *__pyx_kp_u_Cannot_take_vector_part_of;
                static PyObject *__pyx_kp_u_Cardinality_Number_of_indices_i;
                static PyObject *__pyx_kp_u_Check_if_a_multivector_contains;
                static PyObject *__pyx_kp_u_Check_if_a_multivector_contains_2;
                static PyObject *__pyx_kp_u_Conjugation_reverse_o_involute;
                static PyObject *__pyx_kp_u_Conjugation_reverse_o_involute_2;
                static PyObject *__pyx_kp_u_Contraction_print_clifford_1_cl;
                static PyObject *__pyx_kp_u_Contraction_x_clifford_1_x_clif;
                static PyObject *__pyx_kp_u_Convert_CGA3_null_vector_to_Euc;
                static PyObject *__pyx_kp_u_Convert_CGA3_null_vector_to_sta;
                static PyObject *__pyx_kp_u_Convert_Euclidean_3D_multivecto;
                static PyObject *__pyx_kp_u_Copy_this_clifford_object_x_cli;
                static PyObject *__pyx_kp_u_Copy_this_index_set_object_s_in;
```

```
static PyObject *__pyx_kp_u_Cosine_of_multivector_with_opti;
static PyObject *__pyx_kp_u_Even_part_of_multivector_sum_of;
static PyObject *__pyx_kp_u_Even_part_of_multivector_sum_of_2;
static PyObject *__pyx_kp_u_Exponential_of_multivector_x_cl;
static PyObject *__pyx_kp_u_Geometric_difference_print_clif;
static PyObject *__pyx_kp_u_Geometric_difference_x_clifford;
static PyObject *__pyx_kp_u_Geometric_multiplicative_invers;
static PyObject *__pyx_kp_u_Geometric_multiplicative_invers_2;
static PyObject *__pyx_kp_u_Geometric_product_print_cliffor;
static PyObject *__pyx_kp_u_Geometric_product_x_clifford_2;
static PyObject *__pyx_kp_u_Geometric_quotient_print_cliffo;
static PyObject *__pyx_kp_u_Geometric_quotient_x_clifford_1;
static PyObject *__pyx_kp_u_Geometric_sum_print_clifford_1;
static PyObject *__pyx_kp_u_Geometric_sum_x_clifford_1_x_cl;
static PyObject *__pyx_kp_u_Get_the_value_of_an_index_set_o;
static PyObject *__pyx_kp_u_Hyperbolic_cosine_of_multivecto;
static PyObject *__pyx_kp_u_Hyperbolic_sine_of_multivector;
static PyObject *__pyx_kp_u_Hyperbolic_tangent_of_multivect;
static PyObject *__pyx_kp_u_Imaginary_part_deprecated_alway;
static PyObject *__pyx_n_s_IndexError;
static PyObject *__pyx_kp_u_Inner_product_print_clifford_1;
static PyObject *__pyx_kp_u_Inner_product_x_clifford_1_x_cl;
static PyObject *__pyx_kp_u_Integer_power_of_multivector_ob;
static PyObject *__pyx_n_s_Integral;
static PyObject *__pyx_kp_u_Inverse_cosine_of_multivector_w;
static PyObject *__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu;
static PyObject *__pyx_kp_u_Inverse_hyperbolic_sine_of_mult;
static PyObject *__pyx_kp_u_Inverse_hyperbolic_tangent_of_m;
static PyObject *__pyx_kp_u_Inverse_sine_of_multivector_wit;
static PyObject *__pyx_kp_u_Inverse_tangent_of_multivector;
static PyObject *__pyx_kp_u_Iterate_over_the_indices_of_an;
static PyObject *__pyx_kp_u_Main_involution_each_i_is_repla;
static PyObject *__pyx_kp_u_Main_involution_each_i_is_repla_2;
static PyObject *__pyx_kp_u_Maximum_absolute_value_of_coord;
static PyObject *__pyx_kp_u_Maximum_member_index_set_1_1_2;
static PyObject *__pyx_kp_u_Maximum_of_absolute_values_of_c;
static PyObject *__pyx_kp_u_Maximum_positive_index_or_0_if;
static PyObject *__pyx_kp_u_Minimum_member_index_set_1_1_2;
static PyObject *__pyx_kp_u_Minimum_negative_index_or_0_if;
static PyObject *__pyx_kp_u_Natural_logarithm_of_multivecto;
static PyObject *__pyx_kp_u_Norm_sum_of_squares_of_coordina;
static PyObject *__pyx_n_s_NotImplemented;
static PyObject *__pyx_kp_u_Not_applicable;
static PyObject *__pyx_kp_u_Not_applicable_for_a_in_cliffor;
static PyObject *__pyx_kp_u_Number_of_negative_indices_incl;
static PyObject *__pyx_kp_u_Number_of_positive_indices_incl;
static PyObject *__pyx_kp_u_Odd_part_of_multivector_sum_of;
static PyObject *__pyx_kp_u_Odd_part_of_multivector_sum_of_2;
static PyObject *__pyx_kp_u_Outer_product_power_of_multivec;
static PyObject *__pyx_kp_u_Outer_product_power_x_clifford;
static PyObject *__pyx_kp_u_Outer_product_print_clifford_1;
static PyObject *__pyx_kp_u_Outer_product_x_clifford_1_x_cl;
static PyObject *__pyx_kp_u_Power_self_to_the_m_x_clifford;
static PyObject *__pyx_kp_u_Power_self_to_the_m_x_clifford_2;
static PyObject *__pyx_kp_u_Pure_grade_vector_part_print_cl;
static PyObject *__pyx_kp_u_Pure_part_print_clifford_1_1_1;
static PyObject *__pyx_kp_u_Pure_part_print_pure_clifford_1;
static PyObject *__pyx_kp_u_Put_self_into_a_larger_frame_co;
static PyObject *__pyx_n_s_PyClical;
static PyObject *__pyx_kp_s_PyClical_pyx;
static PyObject *__pyx_kp_u_Quadratic_form_rev_x_x_0_print;
static PyObject *__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2;
static PyObject *__pyx_kp_u_Quadratic_norm_error_tolerance;
static PyObject *__pyx_kp_u_Random_multivector_within_a_fra;
static PyObject *__pyx_n_s_Real;
static PyObject *__pyx_kp_u_Real_part_synonym_for_scalar_pa;
static PyObject *__pyx_kp_u_Relative_or_absolute_error_usin;
static PyObject *__pyx_kp_u_Remove_all_terms_of_self_with_r;
static PyObject *__pyx_kp_u_Reversion_eg_1_2_2_1_print_reve;
static PyObject *__pyx_kp_u_Reversion_eg_clifford_1_cliffor;
static PyObject *__pyx_n_s_RuntimeError;
static PyObject *__pyx_kp_u_Scalar_part_clifford_1_1_1_2_sc;
static PyObject *__pyx_kp_u_Scalar_part_scalar_clifford_1_1;
static PyObject *__pyx_n_s_Sequence;
static PyObject *__pyx_kp_u_Set_complement_not_print_index;
static PyObject *__pyx_kp_u_Set_intersection_and_print_inde;
static PyObject *__pyx_kp_u_Set_intersection_and_x_index_se;
static PyObject *__pyx_kp_u_Set_the_value_of_an_index_set_o;
static PyObject *__pyx_kp_u_Set_union_or_print_index_set_1;
static PyObject *__pyx_kp_u_Set_union_or_x_index_set_1_x_in;
static PyObject *__pyx_kp_u_Sign_of_geometric_product_of_tw;
static PyObject *__pyx_kp_u_Sign_of_geometric_square_of_a_C;
static PyObject *__pyx_kp_u_Sine_of_multivector_with_option;
static PyObject *__pyx_kp_u_Square_root_of_1_which_commutes;
static PyObject *__pyx_kp_u_Square_root_of_multivector_with;
static PyObject *__pyx_kp_u_Subalgebra_generated_by_all_gen;
static PyObject *__pyx_kp_u_Subscripting_map_from_index_set;
```

```
static PyObject *__pyx_kp_u_Symmetric_set_difference_exclus;
static PyObject *__pyx_kp_u_Symmetric_set_difference_exclus_2;
static PyObject *__pyx_kp_u_Tangent_of_multivector_with_opt;
static PyObject *__pyx_kp_u_Test_for_approximate_equality_o;
static PyObject *__pyx_kp_u_Tests_for_functions_that_Doctes;
static PyObject *__pyx_kp_u_Tests_for_functions_that_Doctes_2;
static PyObject *__pyx_kp_u_The_informal_string_representat;
static PyObject *__pyx_kp_u_The_informal_string_representat_2;
static PyObject *__pyx_kp_u_The_official_string_representat;
static PyObject *__pyx_kp_u_The_official_string_representat_2;
static PyObject *__pyx_kp_u_This_comparison_operator_is_not;
static PyObject *__pyx_kp_u_Transform_left_hand_side_using;
static PyObject *__pyx_kp_u_Transform_left_hand_side_using_2;
static PyObject *__pyx_n_s_TypeError;
static PyObject *__pyx_kp_u_UTF_8;
static PyObject *__pyx_kp_u_Unary_print_clifford_1_1;
static PyObject *__pyx_kp_u_Unary_print_clifford_1_1_2;
static PyObject *__pyx_n_s_ValueError;
static PyObject *__pyx_kp_u_Vector_part_of_multivector_as_a;
static PyObject *__pyx_kp_u__2;
static PyObject *__pyx_kp_u__5;
static PyObject *__pyx_kp_u__6;
static PyObject *__pyx_kp_u__7;
static PyObject *__pyx_kp_u__8;
static PyObject *__pyx_kp_u__9;
static PyObject *__pyx_n_s_abc;
static PyObject *__pyx_kp_u_abs_line_1522;
static PyObject *__pyx_n_s_acos;
static PyObject *__pyx_kp_u_acos_line_1668;
static PyObject *__pyx_n_s_acosh;
static PyObject *__pyx_kp_u_acosh_line_1705;
static PyObject *__pyx_kp_u_agc3_line_1893;
static PyObject *__pyx_kp_u_approx_equal_line_1359;
static PyObject *__pyx_n_s_args;
static PyObject *__pyx_kp_u_as_frame;
static PyObject *__pyx_n_s_asin;
static PyObject *__pyx_kp_u_asin_line_1747;
static PyObject *__pyx_n_s_asinh;
static PyObject *__pyx_kp_u_asinh_line_1782;
static PyObject *__pyx_n_s_atan;
static PyObject *__pyx_kp_u_atan_line_1818;
static PyObject *__pyx_n_s_atanh;
static PyObject *__pyx_kp_u_atanh_line_1847;
static PyObject *__pyx_kp_u_cga3_line_1873;
static PyObject *__pyx_kp_u_cga3std_line_1882;
static PyObject *__pyx_n_s_cl;
static PyObject *__pyx_n_s_clifford;
static PyObject *__pyx_kp_u_clifford___add___line_740;
static PyObject *__pyx_kp_u_clifford___and___line_836;
static PyObject *__pyx_kp_u_clifford___call___line_1020;
static PyObject *__pyx_kp_u_clifford___getitem___line_707;
static PyObject *__pyx_kp_u_clifford___iadd___line_751;
static PyObject *__pyx_kp_u_clifford___iand___line_851;
static PyObject *__pyx_kp_u_clifford___idiv___line_911;
static PyObject *__pyx_kp_u_clifford___imod___line_821;
static PyObject *__pyx_kp_u_clifford___imul___line_793;
static PyObject *__pyx_kp_u_clifford___ior___line_950;
static PyObject *__pyx_kp_u_clifford___isub___line_771;
static PyObject *__pyx_kp_u_clifford___iter___line_638;
static PyObject *__pyx_kp_u_clifford___ixor___line_881;
static PyObject *__pyx_kp_u_clifford___mod___line_806;
static PyObject *__pyx_kp_u_clifford___mul___line_780;
static PyObject *__pyx_kp_u_clifford___neg___line_722;
static PyObject *__pyx_kp_u_clifford___or___line_939;
static PyObject *__pyx_kp_u_clifford___pos___line_731;
static PyObject *__pyx_kp_u_clifford___pow___line_961;
static PyObject *__pyx_kp_u_clifford___repr___line_1235;
static PyObject *__pyx_kp_u_clifford___str___line_1244;
static PyObject *__pyx_kp_u_clifford___sub___line_760;
static PyObject *__pyx_kp_u_clifford___truediv___line_896;
static PyObject *__pyx_kp_u_clifford___xor___line_866;
static PyObject *__pyx_kp_u_clifford_abs_line_1175;
static PyObject *__pyx_kp_u_clifford_conj_line_1138;
static PyObject *__pyx_kp_u_clifford_copy_line_556;
static PyObject *__pyx_kp_u_clifford_even_line_1061;
static PyObject *__pyx_kp_u_clifford_frame_line_1224;
static PyObject *__pyx_n_s_clifford_hidden_doctests;
static PyObject *__pyx_kp_u_clifford_hidden_doctests_line_12;
static PyObject *__pyx_kp_u_clifford_inv_line_926;
static PyObject *__pyx_kp_u_clifford_involute_line_1107;
static PyObject *__pyx_kp_u_clifford_isinf_line_1206;
static PyObject *__pyx_kp_u_clifford_isnan_line_1215;
static PyObject *__pyx_kp_u_clifford_max_abs_line_1184;
static PyObject *__pyx_kp_u_clifford_norm_line_1164;
static PyObject *__pyx_kp_u_clifford_odd_line_1070;
static PyObject *__pyx_kp_u_clifford_outer_pow_line_1004;
static PyObject *__pyx_kp_u_clifford_pow_line_980;
```

```
                  static PyObject *__pyx_kp_u_clifford_pure_line_1050;
                  static PyObject *__pyx_kp_u_clifford_quad_line_1153;
                  static PyObject *__pyx_kp_u_clifford_reframe_line_649;
                  static PyObject *__pyx_kp_u_clifford_reverse_line_1123;
                  static PyObject *__pyx_kp_u_clifford_scalar_line_1039;
                  static PyObject *__pyx_kp_u_clifford_truncated_line_1195;
                  static PyObject *__pyx_kp_u_clifford_vector_part_line_1079;
                  static PyObject *__pyx_n_s_cline_in_traceback;
                  static PyObject *__pyx_n_s_close;
                  static PyObject *__pyx_n_s_collections;
                  static PyObject *__pyx_kp_u_compare_line_492;
                  static PyObject *__pyx_kp_u_complexifier_line_1576;
                  static PyObject *__pyx_n_s_conj;
                  static PyObject *__pyx_kp_u_conj_line_1485;
                  static PyObject *__pyx_n_s_copy;
                  static PyObject *__pyx_n_s_cos;
                  static PyObject *__pyx_kp_u_cos_line_1651;
                  static PyObject *__pyx_n_s_cosh;
                  static PyObject *__pyx_kp_u_cosh_line_1689;
                  static PyObject *__pyx_n_s_doctest;
                  static PyObject *__pyx_n_s_e;
                  static PyObject *__pyx_kp_u_e_line_1936;
                  static PyObject *__pyx_n_s_encode;
                  static PyObject *__pyx_kp_u_error_squared_line_1346;
                  static PyObject *__pyx_kp_u_error_squared_tol_line_1337;
                  static PyObject *__pyx_n_s_even;
                  static PyObject *__pyx_kp_u_even_line_1437;
                  static PyObject *__pyx_n_s_exp;
                  static PyObject *__pyx_kp_u_exp_line_1614;
                  static PyObject *__pyx_n_s_fill;
                  static PyObject *__pyx_n_s_frm;
                  static PyObject *__pyx_kp_u_from;
                  static PyObject *__pyx_n_s_getstate;
                  static PyObject *__pyx_n_s_grade;
                  static PyObject *__pyx_n_s_i;
                  static PyObject *__pyx_kp_u_imag_line_1415;
                  static PyObject *__pyx_n_s_import;
                  static PyObject *__pyx_n_s_index_set;
                  static PyObject *__pyx_kp_u_index_set___and___line_271;
                  static PyObject *__pyx_kp_u_index_set___getitem___line_191;
                  static PyObject *__pyx_kp_u_index_set___iand___line_282;
                  static PyObject *__pyx_kp_u_index_set___invert___line_240;
                  static PyObject *__pyx_kp_u_index_set___ior___line_304;
                  static PyObject *__pyx_n_s_index_set___iter;
                  static PyObject *__pyx_kp_u_index_set___iter___line_229;
                  static PyObject *__pyx_kp_u_index_set___ixor___line_260;
                  static PyObject *__pyx_kp_u_index_set___or___line_293;
                  static PyObject *__pyx_kp_u_index_set___repr___line_384;
                  static PyObject *__pyx_kp_u_index_set___setitem___line_179;
                  static PyObject *__pyx_kp_u_index_set___str___line_395;
                  static PyObject *__pyx_kp_u_index_set___xor___line_249;
                  static PyObject *__pyx_kp_u_index_set_copy_line_65;
                  static PyObject *__pyx_kp_u_index_set_count_line_315;
                  static PyObject *__pyx_kp_u_index_set_count_neg_line_324;
                  static PyObject *__pyx_kp_u_index_set_count_pos_line_333;
                  static PyObject *__pyx_n_s_index_set_hidden_doctests;
                  static PyObject *__pyx_kp_u_index_set_hidden_doctests_line_4;
                  static PyObject *__pyx_kp_u_index_set_max_line_351;
                  static PyObject *__pyx_kp_u_index_set_min_line_342;
                  static PyObject *__pyx_kp_u_index_set_sign_of_mult_line_366;
                  static PyObject *__pyx_kp_u_index_set_sign_of_square_line_37;
                  static PyObject *__pyx_n_s_inv;
                  static PyObject *__pyx_kp_u_inv_line_1378;
                  static PyObject *__pyx_kp_u_invalid;
                  static PyObject *__pyx_kp_u_invalid_string;
                  static PyObject *__pyx_n_s_involute;
                  static PyObject *__pyx_kp_u_involute_line_1455;
                  static PyObject *__pyx_n_s_ist;
                  static PyObject *__pyx_n_s_istpq;
                  static PyObject *__pyx_kp_u_istpq_line_1949;
                  static PyObject *__pyx_n_s_iter;
                  static PyObject *__pyx_n_s_ixt;
                  static PyObject *__pyx_kp_u_lexicographic_compare_eg_3_4_5;
                  static PyObject *__pyx_n_s_lhs;
                  static PyObject *__pyx_n_s_log;
                  static PyObject *__pyx_kp_u_log_line_1628;
                  static PyObject *__pyx_n_s_m;
                  static PyObject *__pyx_n_s_main;
                  static PyObject *__pyx_n_u_main;
                  static PyObject *__pyx_n_s_math;
                  static PyObject *__pyx_n_s_max;
                  static PyObject *__pyx_kp_u_max_abs_line_1531;
                  static PyObject *__pyx_kp_u_max_pos_line_513;
                  static PyObject *__pyx_n_s_min;
                  static PyObject *__pyx_kp_u_min_neg_line_504;
                  static PyObject *__pyx_n_s_name;
                  static PyObject *__pyx_n_s_nbar3;
```

```
static PyObject *__pyx_n_s_ninf3;
static PyObject *__pyx_kp_s_no_default___reduce___due_to_non;
static PyObject *__pyx_n_s_norm;
static PyObject *__pyx_kp_u_norm_line_1511;
static PyObject *__pyx_kp_u_norm_sum_of_squares_of_coordina;
static PyObject *__pyx_n_s_numbers;
static PyObject *__pyx_n_s_obj;
static PyObject *__pyx_n_s_odd;
static PyObject *__pyx_kp_u_odd_line_1446;
static PyObject *__pyx_n_s_other;
static PyObject *__pyx_n_s_outer_pow;
static PyObject *__pyx_kp_u_outer_pow_line_1567;
static PyObject *__pyx_n_s_p;
static PyObject *__pyx_n_s_pi;
static PyObject *__pyx_n_s_pow;
static PyObject *__pyx_kp_u_pow_line_1543;
static PyObject *__pyx_n_s_pure;
static PyObject *__pyx_kp_u_pure_line_1426;
static PyObject *__pyx_n_s_pyx_vtable;
static PyObject *__pyx_n_s_q;
static PyObject *__pyx_n_s_quad;
static PyObject *__pyx_kp_u_quad_line_1500;
static PyObject *__pyx_kp_u_random_clifford_line_1864;
static PyObject *__pyx_n_s_range;
static PyObject *__pyx_kp_u_real_line_1404;
static PyObject *__pyx_n_s_reduce;
static PyObject *__pyx_n_s_reduce_cython;
static PyObject *__pyx_n_s_reduce_ex;
static PyObject *__pyx_n_s_reverse;
static PyObject *__pyx_kp_u_reverse_line_1470;
static PyObject *__pyx_n_s_rhs;
static PyObject *__pyx_n_s_scalar;
static PyObject *__pyx_n_s_scalar_epsilon;
static PyObject *__pyx_kp_u_scalar_line_1393;
static PyObject *__pyx_n_s_send;
static PyObject *__pyx_n_s_setstate;
static PyObject *__pyx_n_s_setstate_cython;
static PyObject *__pyx_n_s_sin;
static PyObject *__pyx_kp_u_sin_line_1728;
static PyObject *__pyx_n_s_sinh;
static PyObject *__pyx_kp_u_sinh_line_1768;
static PyObject *__pyx_n_s_sqrt;
static PyObject *__pyx_kp_u_sqrt_line_1591;
static PyObject *__pyx_n_s_tan;
static PyObject *__pyx_kp_u_tan_line_1801;
static PyObject *__pyx_n_s_tanh;
static PyObject *__pyx_kp_u_tanh_line_1835;
static PyObject *__pyx_n_s_tau;
static PyObject *__pyx_n_s_test;
static PyObject *__pyx_n_s_test_2;
static PyObject *__pyx_n_s_testmod;
static PyObject *__pyx_n_s_threshold;
static PyObject *__pyx_n_s_throw;
static PyObject *__pyx_kp_u_to_frame;
static PyObject *__pyx_n_s_tol;
static PyObject *__pyx_kp_u_using;
static PyObject *__pyx_kp_u_using_invalid;
static PyObject *__pyx_kp_u_utf_8;
static PyObject *__pyx_kp_u_value;
static PyObject *__pyx_n_s_version;
static PyObject *__pyx_n_s_xrange;
static PyObject *__pyx_pf_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static int __pyx_pf_8PyClical_9index_set_2__cinit__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_other); /* proto */
static void __pyx_pf_8PyClical_9index_set_4__dealloc__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_6__richcmp__(struct __pyx_obj_8PyClical_index_set *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op); /* proto */
static int __pyx_pf_8PyClical_9index_set_8__setitem__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_10__getitem__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx); /* proto */
static int __pyx_pf_8PyClical_9index_set_12__contains__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_14__iter__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_17__invert__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_19__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_21__ixor__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_23__and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_25__iand__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
```

```
static PyObject *__pyx_pf_8PyClical_9index_set_27__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_29__ior__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_31count(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_33count_neg(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_35count_pos(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_37min(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_39max(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_41hash_fn(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_43sign_of_mult(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_45sign_of_square(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_47__repr__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_49__str__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_51__reduce_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_9index_set_53__setstate_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_index_set *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v___pyx_state); /* proto */
static PyObject *__pyx_pf_8PyClical_index_set_hidden_doctests(CYTHON_UNUSED PyObject *__pyx_self); /* proto */
static PyObject *__pyx_pf_8PyClical_2compare(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_4min_neg(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_6max_pos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static int __pyx_pf_8PyClical_8clifford_2__cinit__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_other, PyObject *__pyx_v_ixt); /* proto */
static void __pyx_pf_8PyClical_8clifford_4__dealloc__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static int __pyx_pf_8PyClical_8clifford_6__contains__(CYTHON_UNUSED struct __pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v_x); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_8__iter__(CYTHON_UNUSED struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_10reframe(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_ixt); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_12__richcmp__(struct __pyx_obj_8PyClical_clifford *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_14__getitem__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_ixt); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_16__neg__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_18__pos__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_20__add__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_22__iadd__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_24__sub__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_26__isub__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_28__mul__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_30__imul__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_32__mod__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_34__imod__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_36__and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_38__iand__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_40__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_42__ixor__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_44__truediv__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
#if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
static PyObject *__pyx_pf_8PyClical_8clifford_46__idiv__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
#endif
static PyObject *__pyx_pf_8PyClical_8clifford_48inv(struct __pyx_obj_8PyClical_clifford
```

```
*__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_50__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_52__ior__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_54__pow__(PyObject *__pyx_v_self, PyObject *__pyx_v_m, CYTHON_UNUSED PyObject *__pyx_v_dummy); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_56pow(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_m); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_58outer_pow(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_m); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_60__call__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_grade); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_62scalar(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_64pure(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_66even(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_68odd(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_70vector_part(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_frm); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_72involute(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_74reverse(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_76conj(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_78quad(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_80norm(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_82abs(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_84max_abs(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_86truncated(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_limit); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_88isinf(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_90isnan(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_92frame(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_94__repr__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_96__str__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_98__reduce_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_100__setstate_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v___pyx_state); /* proto */
static PyObject *__pyx_pf_8PyClical_8clifford_hidden_doctests(CYTHON_UNUSED PyObject *__pyx_self); /* proto */
static PyObject *__pyx_pf_8PyClical_10error_squared_tol(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_12error_squared(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold); /* proto */
static PyObject *__pyx_pf_8PyClical_14approx_equal(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, PyObject *__pyx_v_tol); /* proto */
static PyObject *__pyx_pf_8PyClical_16inv(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_18scalar(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_20real(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_22imag(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_24pure(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_26even(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_28odd(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_30involute(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_32reverse(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_34conj(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_36quad(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_38norm(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_40abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
```

```
static PyObject *__pyx_pf_8PyClical_42max_abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_44pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_m); /* proto */
static PyObject *__pyx_pf_8PyClical_46outer_pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_m); /* proto */
static PyObject *__pyx_pf_8PyClical_48complexifier(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_50sqrt(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_52exp(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_54log(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_56cos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_58acos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_60cosh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_62acosh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_64sin(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_66asin(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_68sinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_70asinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_72tan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_74atan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_76tanh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_78atanh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i); /* proto */
static PyObject *__pyx_pf_8PyClical_80random_clifford(CYTHON_UNUSED PyObject *__pyx_self, struct __pyx_obj_8PyClical_index_set *__pyx_v_ixt, PyObject *__pyx_v_fill); /* proto */
static PyObject *__pyx_pf_8PyClical_82cga3(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_84cga3std(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_86agc3(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_88e(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /* proto */
static PyObject *__pyx_pf_8PyClical_90istpq(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_p, PyObject *__pyx_v_q); /* proto */
static PyObject *__pyx_pf_8PyClical_92_test(CYTHON_UNUSED PyObject *__pyx_self); /* proto */
static PyObject *__pyx_tp_new_8PyClical_index_set(PyTypeObject *t, PyObject *a, PyObject *k); /*proto*/
static PyObject *__pyx_tp_new_8PyClical_clifford(PyTypeObject *t, PyObject *a, PyObject *k); /*proto*/
static PyObject *__pyx_tp_new_8PyClical___pyx_scope_struct____iter__(PyTypeObject *t, PyObject *a, PyObject *k); /*proto*/
static PyObject *__pyx_float_0_0;
static PyObject *__pyx_float_1_0;
static PyObject *__pyx_float_4_0;
static PyObject *__pyx_float_8_0;
static PyObject *__pyx_int_0;
static PyObject *__pyx_int_1;
static PyObject *__pyx_int_4;
static PyObject *__pyx_int_neg_1;
static PyObject *__pyx_tuple__3;
static PyObject *__pyx_tuple__4;
static PyObject *__pyx_tuple__10;
static PyObject *__pyx_tuple__11;
static PyObject *__pyx_tuple__12;
static PyObject *__pyx_tuple__15;
static PyObject *__pyx_tuple__16;
static PyObject *__pyx_tuple__18;
static PyObject *__pyx_tuple__20;
static PyObject *__pyx_tuple__21;
static PyObject *__pyx_tuple__22;
static PyObject *__pyx_codeobj__13;
static PyObject *__pyx_codeobj__14;
static PyObject *__pyx_codeobj__17;
static PyObject *__pyx_codeobj__19;
static PyObject *__pyx_codeobj__23;
/* Late includes */

/* "PyClical.pyx":40
 * cdef class index_set
 *
 * cdef inline IndexSet toIndexSet(obj):             # «««««««
 *     """
```

```
 *     Return the C++ IndexSet instance wrapped by index_set(obj).
 */

static CYTHON_INLINE IndexSet __pyx_f_8PyClical_toIndexSet(PyObject *__pyx_v_obj) {
  IndexSet __pyx_r;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("toIndexSet", 0);

  /* "PyClical.pyx":44
 *     Return the C++ IndexSet instance wrapped by index_set(obj).
 *     """
 *     return index_set(obj).instance[0]             # «««««««
 *
 * cdef class index_set:
 */
  __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set), __pyx_v_obj); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 44, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = (((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1)->instance[0]);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":40
 * cdef class index_set
 *
 * cdef inline IndexSet toIndexSet(obj):             # «««««««
 *     """
 *     Return the C++ IndexSet instance wrapped by index_set(obj).
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_WriteUnraisable("PyClical.toIndexSet", __pyx_clineno, __pyx_lineno, __pyx_filename, 1, 0);
  __Pyx_pretend_to_initialize(&__pyx_r);
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":52
 *     cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.
 *
 *     cdef inline wrap(index_set self, IndexSet other):             # «««««««
 *         """
 *         Wrap an instance of the C++ class IndexSet.
 */

static CYTHON_INLINE PyObject *__pyx_f_8PyClical_9index_set_wrap(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, IndexSet __pyx_v_other) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("wrap", 0);

  /* "PyClical.pyx":56
 *         Wrap an instance of the C++ class IndexSet.
 *         """
 *         self.instance[0] = other             # «««««««
 *         return self
 *
 */
  (__pyx_v_self->instance[0]) = __pyx_v_other;

  /* "PyClical.pyx":57
 *         """
 *         self.instance[0] = other
 *         return self             # «««««««
 *
 *     cdef inline IndexSet unwrap(index_set self):
 */
  __Pyx_XDECREF(__pyx_r);
  __Pyx_INCREF(((PyObject *)__pyx_v_self));
  __pyx_r = ((PyObject *)__pyx_v_self);
  goto __pyx_L0;

  /* "PyClical.pyx":52
 *     cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.
 *
 *     cdef inline wrap(index_set self, IndexSet other):             # «««««««
 *         """
 *         Wrap an instance of the C++ class IndexSet.
 */

```

```
  /* function exit code */
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":59
 *         return self
 *
 *     cdef inline IndexSet unwrap(index_set self):             # <<<<<<<<<<<<<<
 *         """
 *         Return the wrapped C++ IndexSet instance.
 */

static CYTHON_INLINE IndexSet __pyx_f_8PyClical_9index_set_unwrap(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
  IndexSet __pyx_r;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("unwrap", 0);

  /* "PyClical.pyx":63
 *         Return the wrapped C++ IndexSet instance.
 *         """
 *         return self.instance[0]             # <<<<<<<<<<<<<<
 *
 *     cpdef copy(index_set self):
 */
  __pyx_r = (__pyx_v_self->instance[0]);
  goto __pyx_L0;

  /* "PyClical.pyx":59
 *         return self
 *
 *     cdef inline IndexSet unwrap(index_set self):             # <<<<<<<<<<<<<<
 *         """
 *         Return the wrapped C++ IndexSet instance.
 */

  /* function exit code */
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":65
 *         return self.instance[0]
 *
 *     cpdef copy(index_set self):             # <<<<<<<<<<<<<<
 *         """
 *         Copy this index_set object.
 */

static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused); /*proto*/
static PyObject *__pyx_f_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set *__pyx_v_self,
int __pyx_skip_dispatch) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  PyObject *__pyx_t_4 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("copy", 0);
  /* Check if called by wrapper */
  if (unlikely(__pyx_skip_dispatch)) ;
  /* Check if overridden in Python */
  else if (unlikely((Py_TYPE(((PyObject *)__pyx_v_self))->tp_dictoffset != 0) || (Py_TYPE(((PyObject
*)__pyx_v_self))->tp_flags & (Py_TPFLAGS_IS_ABSTRACT | Py_TPFLAGS_HEAPTYPE)))) {
    #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
    static PY_UINT64_T __pyx_tp_dict_version = __PYX_DICT_VERSION_INIT, __pyx_obj_dict_version =
__PYX_DICT_VERSION_INIT;
    if (unlikely(!__Pyx_object_dict_version_matches(((PyObject *)__pyx_v_self), __pyx_tp_dict_version,
__pyx_obj_dict_version))) {
      PY_UINT64_T __pyx_type_dict_guard = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
      #endif
      __pyx_t_1 = __Pyx_PyObject_GetAttrStr(((PyObject *)__pyx_v_self), __pyx_n_s_copy); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 65, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      if (!PyCFunction_Check(__pyx_t_1) || (PyCFunction_GET_FUNCTION(__pyx_t_1) !=
(PyCFunction)(void*)__pyx_pw_8PyClical_9index_set_1copy)) {
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_t_1);
        __pyx_t_3 = __pyx_t_1; __pyx_t_4 = NULL;
```

```
        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_3))) {
          __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
          if (likely(__pyx_t_4)) {
            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
            __Pyx_INCREF(__pyx_t_4);
            __Pyx_INCREF(function);
            __Pyx_DECREF_SET(__pyx_t_3, function);
          }
        }
        __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
        __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
        if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 65, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        goto __pyx_L0;
      }
      #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
      __pyx_tp_dict_version = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
      __pyx_obj_dict_version = __Pyx_get_object_dict_version(((PyObject *)__pyx_v_self));
      if (unlikely(__pyx_type_dict_guard != __pyx_tp_dict_version)) {
        __pyx_tp_dict_version = __pyx_obj_dict_version = __PYX_DICT_VERSION_INIT;
      }
      #endif
      __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
      #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
    }
    #endif
  }

  /* "PyClical.pyx":72
 *         {1}
 *         """
 *         return index_set(self)             # <<<<<<<<<<<<<<
 *
 *     def __cinit__(self, other = 0):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set), ((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 72, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":65
 *         return self.instance[0]
 *
 *     cpdef copy(index_set self):             # <<<<<<<<<<<<<<
 *         """
 *         Copy this index_set object.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_XDECREF(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_4);
  __Pyx_AddTraceback("PyClical.index_set.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = 0;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_copy[] = "\n        Copy this index_set object.\n\n        >>> s=index_set(1); t=s.copy(); print(t)\n        {1}\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("copy (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_copy(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
```

```
static PyObject *__pyx_pf_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("copy", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_9index_set_copy(__pyx_v_self, 1); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 65, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":74
 *         return index_set(self)
 *
 *     def __cinit__(self, other = 0):             # <<<<<<<<<<<<<<
 *         """
 *         Construct an object of type index_set.
 */

/* Python wrapper */
static int __pyx_pw_8PyClical_9index_set_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
static int __pyx_pw_8PyClical_9index_set_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
  PyObject *__pyx_v_other = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__cinit__ (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_other,0};
    PyObject* values[1] = {0};
    values[0] = ((PyObject *)__pyx_int_0);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_other);
          if (value) { values[0] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args,
"__cinit__") < 0)) __PYX_ERR(0, 74, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_other = values[0];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("__cinit__", 0, 0, 1, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 74,
```

```
__pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return -1;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_9index_set_2__cinit__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), __pyx_v_other);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static int __pyx_pf_8PyClical_9index_set_2__cinit__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_other) {
  PyObject *__pyx_v_error_msg_prefix = NULL;
  PyObject *__pyx_v_idx = NULL;
  PyObject *__pyx_v_bother = NULL;
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_t_2;
  IndexSet *__pyx_t_3;
  PyObject *__pyx_t_4 = NULL;
  PyObject *__pyx_t_5 = NULL;
  int __pyx_t_6;
  int __pyx_t_7;
  PyObject *__pyx_t_8 = NULL;
  PyObject *__pyx_t_9 = NULL;
  PyObject *__pyx_t_10 = NULL;
  Py_ssize_t __pyx_t_11;
  PyObject *(*__pyx_t_12)(PyObject *);
  PyObject *__pyx_t_13 = NULL;
  PyObject *__pyx_t_14 = NULL;
  PyObject *__pyx_t_15 = NULL;
  PyObject *__pyx_t_16 = NULL;
  char *__pyx_t_17;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__cinit__", 0);

  /* "PyClical.pyx":93
 *         {}
 *         """
 *         error_msg_prefix = "Cannot initialize index_set object from"             # <<<<<<<<<<<<<<
 *         if   isinstance(other, index_set):
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 */
  __Pyx_INCREF(__pyx_kp_u_Cannot_initialize_index_set_obje);
  __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_initialize_index_set_obje;

  /* "PyClical.pyx":94
 *         """
 *         error_msg_prefix = "Cannot initialize index_set object from"
 *         if   isinstance(other, index_set):             # <<<<<<<<<<<<<<
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 *         elif isinstance(other, numbers.Integral):
 */
  __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyClical_index_set);
  __pyx_t_2 = (__pyx_t_1 != 0);
  if (__pyx_t_2) {

    /* "PyClical.pyx":95
 *         error_msg_prefix = "Cannot initialize index_set object from"
 *         if   isinstance(other, index_set):
 *             self.instance = new IndexSet((<index_set>other).unwrap())             # <<<<<<<<<<<<<<
 *         elif isinstance(other, numbers.Integral):
 *             self.instance = new IndexSet(<int>other)
 */
    try {
      __pyx_t_3 = new IndexSet(__pyx_f_8PyClical_9index_set_unwrap(((struct
__pyx_obj_8PyClical_index_set *)__pyx_v_other)));
    } catch(...) {
      __Pyx_CppExn2PyErr();
      __PYX_ERR(0, 95, __pyx_L1_error)
    }
    __pyx_v_self->instance = __pyx_t_3;

    /* "PyClical.pyx":94
 *         """
 *         error_msg_prefix = "Cannot initialize index_set object from"
 *         if   isinstance(other, index_set):             # <<<<<<<<<<<<<<
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 *         elif isinstance(other, numbers.Integral):
 */
```

```
      goto __pyx_L3;
    }

    /* "PyClical.pyx":96
 *         if   isinstance(other, index_set):
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 *         elif isinstance(other, numbers.Integral):             # «««««««
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):
 */
    __Pyx_GetModuleGlobalName(__pyx_t_4, __pyx_n_s_numbers); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 96, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_4);
    __pyx_t_5 = __Pyx_PyObject_GetAttrStr(__pyx_t_4, __pyx_n_s_Integral); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 96, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_5);
    __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
    __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_5); if (unlikely(__pyx_t_2 == ((int)-1))) __PYX_ERR(0, 96, __pyx_L1_error)
    __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
    __pyx_t_1 = (__pyx_t_2 != 0);
    if (__pyx_t_1) {

      /* "PyClical.pyx":97
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 *         elif isinstance(other, numbers.Integral):
 *             self.instance = new IndexSet(<int>other)             # «««««««
 *         elif isinstance(other, (set, frozenset)):
 *             try:
 */
      __pyx_t_6 = __Pyx_PyInt_As_int(__pyx_v_other); if (unlikely((__pyx_t_6 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 97, __pyx_L1_error)
      try {
        __pyx_t_3 = new IndexSet(((int)__pyx_t_6));
      } catch(...) {
        __Pyx_CppExn2PyErr();
        __PYX_ERR(0, 97, __pyx_L1_error)
      }
      __pyx_v_self->instance = __pyx_t_3;

      /* "PyClical.pyx":96
 *         if   isinstance(other, index_set):
 *             self.instance = new IndexSet((<index_set>other).unwrap())
 *         elif isinstance(other, numbers.Integral):             # «««««««
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):
 */
      goto __pyx_L3;
    }

    /* "PyClical.pyx":98
 *         elif isinstance(other, numbers.Integral):
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):             # «««««««
 *             try:
 *                 self.instance = new IndexSet()
 */
    __pyx_t_2 = PySet_Check(__pyx_v_other);
    __pyx_t_7 = (__pyx_t_2 != 0);
    if (!__pyx_t_7) {
    } else {
      __pyx_t_1 = __pyx_t_7;
      goto __pyx_L4_bool_binop_done;
    }
    __pyx_t_7 = PyFrozenSet_Check(__pyx_v_other);
    __pyx_t_2 = (__pyx_t_7 != 0);
    __pyx_t_1 = __pyx_t_2;
    __pyx_L4_bool_binop_done:;
    __pyx_t_2 = (__pyx_t_1 != 0);
    if (__pyx_t_2) {

      /* "PyClical.pyx":99
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):
 *             try:             # «««««««
 *                 self.instance = new IndexSet()
 *                 for idx in other:
 */
      {
        __Pyx_PyThreadState_declare
        __Pyx_PyThreadState_assign
        __Pyx_ExceptionSave(&__pyx_t_8, &__pyx_t_9, &__pyx_t_10);
        __Pyx_XGOTREF(__pyx_t_8);
        __Pyx_XGOTREF(__pyx_t_9);
        __Pyx_XGOTREF(__pyx_t_10);
        /*try:*/ {

```

```
      /* "PyClical.pyx":100
 *         elif isinstance(other, (set, frozenset)):
 *             try:
 *                 self.instance = new IndexSet()             # <<<<<<<<<<<<<<
 *                 for idx in other:
 *                     self[idx] = True
 */
      __pyx_t_3 = new IndexSet();
      __pyx_v_self->instance = __pyx_t_3;

      /* "PyClical.pyx":101
 *             try:
 *                 self.instance = new IndexSet()
 *                 for idx in other:             # <<<<<<<<<<<<<<
 *                     self[idx] = True
 *             except IndexError:
 */
      if (likely(PyList_CheckExact(__pyx_v_other)) || PyTuple_CheckExact(__pyx_v_other)) {
        __pyx_t_5 = __pyx_v_other; __Pyx_INCREF(__pyx_t_5); __pyx_t_11 = 0;
        __pyx_t_12 = NULL;
      } else {
        __pyx_t_11 = -1; __pyx_t_5 = PyObject_GetIter(__pyx_v_other); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 101, __pyx_L6_error)
        __Pyx_GOTREF(__pyx_t_5);
        __pyx_t_12 = Py_TYPE(__pyx_t_5)->tp_iternext; if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 101, __pyx_L6_error)
      }
      for (;;) {
        if (likely(!__pyx_t_12)) {
          if (likely(PyList_CheckExact(__pyx_t_5))) {
            if (__pyx_t_11 >= PyList_GET_SIZE(__pyx_t_5)) break;
            #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
            __pyx_t_4 = PyList_GET_ITEM(__pyx_t_5, __pyx_t_11); __Pyx_INCREF(__pyx_t_4); __pyx_t_11++; if (unlikely(0 < 0)) __PYX_ERR(0, 101, __pyx_L6_error)
            #else
            __pyx_t_4 = PySequence_ITEM(__pyx_t_5, __pyx_t_11); __pyx_t_11++; if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 101, __pyx_L6_error)
            __Pyx_GOTREF(__pyx_t_4);
            #endif
          } else {
            if (__pyx_t_11 >= PyTuple_GET_SIZE(__pyx_t_5)) break;
            #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
            __pyx_t_4 = PyTuple_GET_ITEM(__pyx_t_5, __pyx_t_11); __Pyx_INCREF(__pyx_t_4); __pyx_t_11++; if (unlikely(0 < 0)) __PYX_ERR(0, 101, __pyx_L6_error)
            #else
            __pyx_t_4 = PySequence_ITEM(__pyx_t_5, __pyx_t_11); __pyx_t_11++; if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 101, __pyx_L6_error)
            __Pyx_GOTREF(__pyx_t_4);
            #endif
          }
        } else {
          __pyx_t_4 = __pyx_t_12(__pyx_t_5);
          if (unlikely(!__pyx_t_4)) {
            PyObject* exc_type = PyErr_Occurred();
            if (exc_type) {
              if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
              else __PYX_ERR(0, 101, __pyx_L6_error)
            }
            break;
          }
          __Pyx_GOTREF(__pyx_t_4);
        }
        __Pyx_XDECREF_SET(__pyx_v_idx, __pyx_t_4);
        __pyx_t_4 = 0;

        /* "PyClical.pyx":102
 *                 self.instance = new IndexSet()
 *                 for idx in other:
 *                     self[idx] = True             # <<<<<<<<<<<<<<
 *             except IndexError:
 *                 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
 */
        if (unlikely(PyObject_SetItem(((PyObject *)__pyx_v_self), __pyx_v_idx, Py_True) < 0)) __PYX_ERR(0, 102, __pyx_L6_error)

        /* "PyClical.pyx":101
 *             try:
 *                 self.instance = new IndexSet()
 *                 for idx in other:             # <<<<<<<<<<<<<<
 *                     self[idx] = True
 *             except IndexError:
 */
      }
      __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;

      /* "PyClical.pyx":99
```

```
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):
 *             try:              # «««««««
 *                 self.instance = new IndexSet()
 *                 for idx in other:
 */
      }
      __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
      __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
      goto __pyx_L11_try_end;
      __pyx_L6_error:;
      __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;

      /* "PyClical.pyx":103
 *                 for idx in other:
 *                     self[idx] = True
 *             except IndexError:              # «««««««
 *                 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
 *             except (RuntimeError, TypeError):
 */
      __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_IndexError);
      if (__pyx_t_6) {
        __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_4, &__pyx_t_13) < 0) __PYX_ERR(0, 103, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_5);
        __Pyx_GOTREF(__pyx_t_4);
        __Pyx_GOTREF(__pyx_t_13);

        /* "PyClical.pyx":104
 *                     self[idx] = True
 *             except IndexError:
 *                 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")              # «««««««
 *             except (RuntimeError, TypeError):
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 */
        __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 104, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_14);
        __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 104, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __pyx_t_16 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 104, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_16);
        __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __pyx_t_15 = PyNumber_Add(__pyx_t_16, __pyx_kp_u_); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 104, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
        __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_IndexError, __pyx_t_15); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 104, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_16);
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __Pyx_Raise(__pyx_t_16, 0, 0, 0);
        __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
        __PYX_ERR(0, 104, __pyx_L8_except_error)
      }

      /* "PyClical.pyx":105
 *             except IndexError:
 *                 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
 *             except (RuntimeError, TypeError):              # «««««««
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):
 */
      __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError) || __Pyx_PyErr_ExceptionMatches(__pyx_builtin_TypeError);
      if (__pyx_t_6) {
        __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        if (__Pyx_GetException(&__pyx_t_13, &__pyx_t_4, &__pyx_t_5) < 0) __PYX_ERR(0, 105, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_13);
        __Pyx_GOTREF(__pyx_t_4);
        __Pyx_GOTREF(__pyx_t_5);

        /* "PyClical.pyx":106
 *                 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
 *             except (RuntimeError, TypeError):
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")              # «««««««
```

```
 *         elif isinstance(other, str):
 *             try:
 */
        __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 106, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_16);
        __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 106, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __pyx_t_14 = PyNumber_Add(__pyx_t_16, __pyx_t_15); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 106, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_14);
        __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __pyx_t_15 = PyNumber_Add(__pyx_t_14, __pyx_kp_u_); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 106, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
        __pyx_t_14 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_15); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 106, __pyx_L8_except_error)
        __Pyx_GOTREF(__pyx_t_14);
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __Pyx_Raise(__pyx_t_14, 0, 0, 0);
        __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
        __PYX_ERR(0, 106, __pyx_L8_except_error)
      }
      goto __pyx_L8_except_error;
      __pyx_L8_except_error:;

      /* "PyClical.pyx":99
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):
 *             try:             # <<<<<<<<<<<<<<
 *                 self.instance = new IndexSet()
 *                 for idx in other:
 */
      __Pyx_XGIVEREF(__pyx_t_8);
      __Pyx_XGIVEREF(__pyx_t_9);
      __Pyx_XGIVEREF(__pyx_t_10);
      __Pyx_ExceptionReset(__pyx_t_8, __pyx_t_9, __pyx_t_10);
      goto __pyx_L1_error;
      __pyx_L11_try_end:;
    }

    /* "PyClical.pyx":98
 *         elif isinstance(other, numbers.Integral):
 *             self.instance = new IndexSet(<int>other)
 *         elif isinstance(other, (set, frozenset)):             # <<<<<<<<<<<<<<
 *             try:
 *                 self.instance = new IndexSet()
 */
    goto __pyx_L3;
  }

  /* "PyClical.pyx":107
 *             except (RuntimeError, TypeError):
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):             # <<<<<<<<<<<<<<
 *             try:
 *                 bother = other.encode("UTF-8")
 */
  __pyx_t_2 = PyUnicode_Check(__pyx_v_other);
  __pyx_t_1 = (__pyx_t_2 != 0);
  if (likely(__pyx_t_1)) {

    /* "PyClical.pyx":108
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):
 *             try:             # <<<<<<<<<<<<<<
 *                 bother = other.encode("UTF-8")
 *                 self.instance = new IndexSet(<char *>bother)
 */
    {
      __Pyx_PyThreadState_declare
      __Pyx_PyThreadState_assign
      __Pyx_ExceptionSave(&__pyx_t_10, &__pyx_t_9, &__pyx_t_8);
      __Pyx_XGOTREF(__pyx_t_10);
      __Pyx_XGOTREF(__pyx_t_9);
      __Pyx_XGOTREF(__pyx_t_8);
      /*try:*/ {

        /* "PyClical.pyx":109
 *         elif isinstance(other, str):
 *             try:
 *                 bother = other.encode("UTF-8")             # <<<<<<<<<<<<<<
 *                 self.instance = new IndexSet(<char *>bother)
 *             except RuntimeError:
```

```
 */
        __pyx_t_4 = __Pyx_PyObject_GetAttrStr(__pyx_v_other, __pyx_n_s_encode); if
(unlikely(!__pyx_t_4)) __PYX_ERR(0, 109, __pyx_L18_error)
        __Pyx_GOTREF(__pyx_t_4);
        __pyx_t_13 = NULL;
        if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_4))) {
          __pyx_t_13 = PyMethod_GET_SELF(__pyx_t_4);
          if (likely(__pyx_t_13)) {
            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_4);
            __Pyx_INCREF(__pyx_t_13);
            __Pyx_INCREF(function);
            __Pyx_DECREF_SET(__pyx_t_4, function);
          }
        }
        __pyx_t_5 = (__pyx_t_13) ? __Pyx_PyObject_Call2Args(__pyx_t_4, __pyx_t_13, __pyx_kp_u_UTF_8) :
__Pyx_PyObject_CallOneArg(__pyx_t_4, __pyx_kp_u_UTF_8);
        __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 109, __pyx_L18_error)
        __Pyx_GOTREF(__pyx_t_5);
        __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
        __pyx_v_bother = __pyx_t_5;
        __pyx_t_5 = 0;

        /* "PyClical.pyx":110
 *             try:
 *                 bother = other.encode("UTF-8")
 *                 self.instance = new IndexSet(<char *>bother)             # <<<<<<<<
 *             except RuntimeError:
 *                 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 */
        __pyx_t_17 = __Pyx_PyObject_AsWritableString(__pyx_v_bother); if (unlikely((!__pyx_t_17) &&
PyErr_Occurred())) __PYX_ERR(0, 110, __pyx_L18_error)
        try {
          __pyx_t_3 = new IndexSet(((char *)__pyx_t_17));
        } catch(...) {
          __Pyx_CppExn2PyErr();
          __PYX_ERR(0, 110, __pyx_L18_error)
        }
        __pyx_v_self->instance = __pyx_t_3;

        /* "PyClical.pyx":108
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):
 *             try:             # <<<<<<<<
 *                 bother = other.encode("UTF-8")
 *                 self.instance = new IndexSet(<char *>bother)
 */
      }
      __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
      __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
      goto __pyx_L23_try_end;
      __pyx_L18_error:;
      __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
      __Pyx_XDECREF(__pyx_t_14); __pyx_t_14 = 0;
      __Pyx_XDECREF(__pyx_t_15); __pyx_t_15 = 0;
      __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
      __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;

      /* "PyClical.pyx":111
 *                 bother = other.encode("UTF-8")
 *                 self.instance = new IndexSet(<char *>bother)
 *             except RuntimeError:             # <<<<<<<<
 *                 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 *         else:
 */
      __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
      if (__pyx_t_6) {
        __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_4, &__pyx_t_13) < 0) __PYX_ERR(0, 111,
__pyx_L20_except_error)
        __Pyx_GOTREF(__pyx_t_5);
        __Pyx_GOTREF(__pyx_t_4);
        __Pyx_GOTREF(__pyx_t_13);

        /* "PyClical.pyx":112
 *                 self.instance = new IndexSet(<char *>bother)
 *             except RuntimeError:
 *                 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
# <<<<<<<<
 *         else:
 *             raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 */
        __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid_string); if
(unlikely(!__pyx_t_14)) __PYX_ERR(0, 112, __pyx_L20_except_error)
```

```
        __Pyx_GOTREF(__pyx_t_14);
        __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 112,
__pyx_L20_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __pyx_t_16 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_16)) __PYX_ERR(0,
112, __pyx_L20_except_error)
        __Pyx_GOTREF(__pyx_t_16);
        __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __pyx_t_15 = PyNumber_Add(__pyx_t_16, __pyx_kp_u_); if (unlikely(!__pyx_t_15)) __PYX_ERR(0,
112, __pyx_L20_except_error)
        __Pyx_GOTREF(__pyx_t_15);
        __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
        __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_15); if
(unlikely(!__pyx_t_16)) __PYX_ERR(0, 112, __pyx_L20_except_error)
        __Pyx_GOTREF(__pyx_t_16);
        __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
        __Pyx_Raise(__pyx_t_16, 0, 0, 0);
        __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
        __PYX_ERR(0, 112, __pyx_L20_except_error)
      }
      goto __pyx_L20_except_error;
      __pyx_L20_except_error:;

      /* "PyClical.pyx":108
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):
 *             try:             # <<<<<<<<<<<<<<
 *                 bother = other.encode("UTF-8")
 *                 self.instance = new IndexSet(<char *>bother)
 */
      __Pyx_XGIVEREF(__pyx_t_10);
      __Pyx_XGIVEREF(__pyx_t_9);
      __Pyx_XGIVEREF(__pyx_t_8);
      __Pyx_ExceptionReset(__pyx_t_10, __pyx_t_9, __pyx_t_8);
      goto __pyx_L1_error;
      __pyx_L23_try_end:;
    }

    /* "PyClical.pyx":107
 *             except (RuntimeError, TypeError):
 *                 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
 *         elif isinstance(other, str):             # <<<<<<<<<<<<<<
 *             try:
 *                 bother = other.encode("UTF-8")
 */
    goto __pyx_L3;
  }

  /* "PyClical.pyx":114
 *                 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 *         else:
 *             raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")             # <<<<<<<<<<<<<<
 *
 *     def __dealloc__(self):
 */
  /*else*/ {
    __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u__2); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 114, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_13);
    __pyx_t_4 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 114, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_4);
    __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0,
114, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_5);
    __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
    __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
    __pyx_t_4 = __Pyx_PyUnicode_Concat(__pyx_t_5, __pyx_kp_u_); if (unlikely(!__pyx_t_4)) __PYX_ERR(0,
114, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_4);
    __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
    __pyx_t_5 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_4); if
(unlikely(!__pyx_t_5)) __PYX_ERR(0, 114, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_5);
    __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
    __Pyx_Raise(__pyx_t_5, 0, 0, 0);
    __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
    __PYX_ERR(0, 114, __pyx_L1_error)
  }
  __pyx_L3:;

  /* "PyClical.pyx":74
 *         return index_set(self)
 *
 *     def __cinit__(self, other = 0):             # <<<<<<<<<<<<<<
 *         """
```

```
 *         Construct an object of type index_set.
 */

  /* function exit code */
  __pyx_r = 0;
  goto __pyx_L0;
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_4);
  __Pyx_XDECREF(__pyx_t_5);
  __Pyx_XDECREF(__pyx_t_13);
  __Pyx_XDECREF(__pyx_t_14);
  __Pyx_XDECREF(__pyx_t_15);
  __Pyx_XDECREF(__pyx_t_16);
  __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = -1;
  __pyx_L0:;
  __Pyx_XDECREF(__pyx_v_error_msg_prefix);
  __Pyx_XDECREF(__pyx_v_idx);
  __Pyx_XDECREF(__pyx_v_bother);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":116
 *             raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *
 *     def __dealloc__(self):             # «««««««
 *         """
 *         Clean up by deallocating the instance of C++ class IndexSet.
 */

/* Python wrapper */
static void __pyx_pw_8PyClical_9index_set_5__dealloc__(PyObject *__pyx_v_self); /*proto*/
static void __pyx_pw_8PyClical_9index_set_5__dealloc__(PyObject *__pyx_v_self) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__dealloc__ (wrapper)", 0);
  __pyx_pf_8PyClical_9index_set_4__dealloc__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
}

static void __pyx_pf_8PyClical_9index_set_4__dealloc__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__dealloc__", 0);

  /* "PyClical.pyx":120
 *         Clean up by deallocating the instance of C++ class IndexSet.
 *         """
 *         del self.instance             # «««««««
 *
 *     def __richcmp__(lhs, rhs, int op):
 */
  delete __pyx_v_self->instance;

  /* "PyClical.pyx":116
 *             raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *
 *     def __dealloc__(self):             # «««««««
 *         """
 *         Clean up by deallocating the instance of C++ class IndexSet.
 */

  /* function exit code */
  __Pyx_RefNannyFinishContext();
}

/* "PyClical.pyx":122
 *         del self.instance
 *
 *     def __richcmp__(lhs, rhs, int op):             # «««««««
 *         """
 *         Compare two objects of class index_set.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_7__richcmp__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op); /*proto*/
static PyObject *__pyx_pw_8PyClical_9index_set_7__richcmp__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__richcmp__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_6__richcmp__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs), ((int)__pyx_v_op));

```

```
  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_6__richcmp__(struct __pyx_obj_8PyClical_index_set *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op) {
  PyObject *__pyx_v_eq = NULL;
  PyObject *__pyx_v_lt = NULL;
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_t_2;
  int __pyx_t_3;
  PyObject *__pyx_t_4 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__richcmp__", 0);

  /* "PyClical.pyx":143
 *         False
 *         """
 *         if (lhs is None) or (rhs is None):             # «««««««
 *             eq = bool(lhs is rhs)
 *             if op == 2: # ==
 */
  __pyx_t_2 = (((PyObject *)__pyx_v_lhs) == Py_None);
  __pyx_t_3 = (__pyx_t_2 != 0);
  if (!__pyx_t_3) {
  } else {
    __pyx_t_1 = __pyx_t_3;
    goto __pyx_L4_bool_binop_done;
  }
  __pyx_t_3 = (__pyx_v_rhs == Py_None);
  __pyx_t_2 = (__pyx_t_3 != 0);
  __pyx_t_1 = __pyx_t_2;
  __pyx_L4_bool_binop_done:;
  if (__pyx_t_1) {

    /* "PyClical.pyx":144
 *         """
 *         if (lhs is None) or (rhs is None):
 *             eq = bool(lhs is rhs)             # «««««««
 *             if op == 2: # ==
 *                 return eq
 */
    __pyx_t_1 = (((PyObject *)__pyx_v_lhs) == __pyx_v_rhs);
    __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 144, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_4);
    __pyx_v_eq = __pyx_t_4;
    __pyx_t_4 = 0;

    /* "PyClical.pyx":145
 *         if (lhs is None) or (rhs is None):
 *             eq = bool(lhs is rhs)
 *             if op == 2: # ==             # «««««««
 *                 return eq
 *             elif op == 3: # !=
 */
    switch (__pyx_v_op) {
      case 2:

      /* "PyClical.pyx":146
 *             eq = bool(lhs is rhs)
 *             if op == 2: # ==
 *                 return eq             # «««««««
 *             elif op == 3: # !=
 *                 return not eq
 */
      __Pyx_XDECREF(__pyx_r);
      __Pyx_INCREF(__pyx_v_eq);
      __pyx_r = __pyx_v_eq;
      goto __pyx_L0;

      /* "PyClical.pyx":145
 *         if (lhs is None) or (rhs is None):
 *             eq = bool(lhs is rhs)
 *             if op == 2: # ==             # «««««««
 *                 return eq
 *             elif op == 3: # !=
 */
      break;
      case 3:

      /* "PyClical.pyx":148
```

```
 *                 return eq
 *             elif op == 3: # !=
 *                 return not eq             # «««««««
 *             else:
 *                 if op == 0: # <
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 148, __pyx_L1_error)
      __pyx_t_4 = __Pyx_PyBool_FromLong((!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 148, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_4);
      __pyx_r = __pyx_t_4;
      __pyx_t_4 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":147
 *             if op == 2: # ==
 *                 return eq
 *             elif op == 3: # !=             # «««««««
 *                 return not eq
 *             else:
 */
      break;
      default:

      /* "PyClical.pyx":150
 *                 return not eq
 *             else:
 *                 if op == 0: # <             # «««««««
 *                     return False
 *                 elif op == 1: # <=
 */
      switch (__pyx_v_op) {
        case 0:

        /* "PyClical.pyx":151
 *             else:
 *                 if op == 0: # <
 *                     return False             # «««««««
 *                 elif op == 1: # <=
 *                     return eq
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(Py_False);
        __pyx_r = Py_False;
        goto __pyx_L0;

        /* "PyClical.pyx":150
 *                 return not eq
 *             else:
 *                 if op == 0: # <             # «««««««
 *                     return False
 *                 elif op == 1: # <=
 */
        break;
        case 1:

        /* "PyClical.pyx":153
 *                     return False
 *                 elif op == 1: # <=
 *                     return eq             # «««««««
 *                 elif op == 4: # >
 *                     return False
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_v_eq);
        __pyx_r = __pyx_v_eq;
        goto __pyx_L0;

        /* "PyClical.pyx":152
 *                 if op == 0: # <
 *                     return False
 *                 elif op == 1: # <=             # «««««««
 *                     return eq
 *                 elif op == 4: # >
 */
        break;
        case 4:

        /* "PyClical.pyx":155
 *                     return eq
 *                 elif op == 4: # >
 *                     return False             # «««««««
 *                 elif op == 5: # >=
 *                     return eq
 */
```

```
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(Py_False);
        __pyx_r = Py_False;
        goto __pyx_L0;

        /* "PyClical.pyx":154
 *                 elif op == 1: # <=
 *                     return eq
 *                 elif op == 4: # >              # «««««««
 *                     return False
 *                 elif op == 5: # >=
 */
        break;
        case 5:

        /* "PyClical.pyx":157
 *                     return False
 *                 elif op == 5: # >=
 *                     return eq              # «««««««
 *                 else:
 *                     return NotImplemented
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_v_eq);
        __pyx_r = __pyx_v_eq;
        goto __pyx_L0;

        /* "PyClical.pyx":156
 *                 elif op == 4: # >
 *                     return False
 *                 elif op == 5: # >=              # «««««««
 *                     return eq
 *                 else:
 */
        break;
        default:

        /* "PyClical.pyx":159
 *                     return eq
 *                 else:
 *                     return NotImplemented              # «««««««
 *         else:
 *             eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_builtin_NotImplemented);
        __pyx_r = __pyx_builtin_NotImplemented;
        goto __pyx_L0;
        break;
      }
      break;
    }

    /* "PyClical.pyx":143
 *         False
 *         """
 *         if (lhs is None) or (rhs is None):              # «««««««
 *             eq = bool(lhs is rhs)
 *             if op == 2: # ==
 */
  }

  /* "PyClical.pyx":161
 *                     return NotImplemented
 *         else:
 *             eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )              # «««««««
 *             if op == 2: # ==
 *                 return eq
 */
  /*else*/ {
    __pyx_t_1 = (__pyx_f_8PyClical_toIndexSet(((PyObject *)__pyx_v_lhs)) ==
__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs));
    __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 161,
__pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_4);
    __pyx_v_eq = __pyx_t_4;
    __pyx_t_4 = 0;

    /* "PyClical.pyx":162
 *         else:
 *             eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )
 *             if op == 2: # ==              # «««««««
 *                 return eq
 *             elif op == 3: # !=
 */
    switch (__pyx_v_op) {
      case 2:
```

```
      /* "PyClical.pyx":163
 *             eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )
 *             if op == 2: # ==
 *                 return eq             # «««««««
 *             elif op == 3: # !=
 *                 return not eq
 */
      __Pyx_XDECREF(__pyx_r);
      __Pyx_INCREF(__pyx_v_eq);
      __pyx_r = __pyx_v_eq;
      goto __pyx_L0;

      /* "PyClical.pyx":162
 *         else:
 *             eq = bool( toIndexSet(lhs) == toIndexSet(rhs) )
 *             if op == 2: # ==             # «««««««
 *                 return eq
 *             elif op == 3: # !=
 */
      break;
      case 3:

      /* "PyClical.pyx":165
 *                 return eq
 *             elif op == 3: # !=
 *                 return not eq             # «««««««
 *             else:
 *                 lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 165, __pyx_L1_error)
      __pyx_t_4 = __Pyx_PyBool_FromLong((!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 165, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_4);
      __pyx_r = __pyx_t_4;
      __pyx_t_4 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":164
 *             if op == 2: # ==
 *                 return eq
 *             elif op == 3: # !=             # «««««««
 *                 return not eq
 *             else:
 */
      break;
      default:

      /* "PyClical.pyx":167
 *                 return not eq
 *             else:
 *                 lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )             # «««««««
 *                 if op == 0: # <
 *                     return lt
 */
      __pyx_t_1 = (__pyx_f_8PyClical_toIndexSet(((PyObject *)__pyx_v_lhs)) < __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs));
      __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 167, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_4);
      __pyx_v_lt = __pyx_t_4;
      __pyx_t_4 = 0;

      /* "PyClical.pyx":168
 *             else:
 *                 lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )
 *                 if op == 0: # <             # «««««««
 *                     return lt
 *                 elif op == 1: # <=
 */
      switch (__pyx_v_op) {
        case 0:

        /* "PyClical.pyx":169
 *                 lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )
 *                 if op == 0: # <
 *                     return lt             # «««««««
 *                 elif op == 1: # <=
 *                     return lt or eq
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_v_lt);
        __pyx_r = __pyx_v_lt;
        goto __pyx_L0;

```

```
        /* "PyClical.pyx":168
 *         else:
 *             lt = bool( toIndexSet(lhs) < toIndexSet(rhs) )
 *             if op == 0: # <             # «««««««
 *                 return lt
 *             elif op == 1: # <=
 */
        break;
        case 1:

        /* "PyClical.pyx":171
 *                 return lt
 *             elif op == 1: # <=
 *                 return lt or eq             # «««««««
 *             elif op == 4: # >
 *                 return not (lt or eq)
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 171, __pyx_L1_error)
        if (!__pyx_t_1) {
        } else {
          __Pyx_INCREF(__pyx_v_lt);
          __pyx_t_4 = __pyx_v_lt;
          goto __pyx_L6_bool_binop_done;
        }
        __Pyx_INCREF(__pyx_v_eq);
        __pyx_t_4 = __pyx_v_eq;
        __pyx_L6_bool_binop_done:;
        __pyx_r = __pyx_t_4;
        __pyx_t_4 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":170
 *             if op == 0: # <
 *                 return lt
 *             elif op == 1: # <=             # «««««««
 *                 return lt or eq
 *             elif op == 4: # >
 */
        break;
        case 4:

        /* "PyClical.pyx":173
 *                 return lt or eq
 *             elif op == 4: # >
 *                 return not (lt or eq)             # «««««««
 *             elif op == 5: # >=
 *                 return not lt
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_2 < 0)) __PYX_ERR(0, 173, __pyx_L1_error)
        if (!__pyx_t_2) {
        } else {
          __pyx_t_1 = __pyx_t_2;
          goto __pyx_L8_bool_binop_done;
        }
        __pyx_t_2 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_2 < 0)) __PYX_ERR(0, 173, __pyx_L1_error)
        __pyx_t_1 = __pyx_t_2;
        __pyx_L8_bool_binop_done:;
        __pyx_t_4 = __Pyx_PyBool_FromLong((!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 173, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_4);
        __pyx_r = __pyx_t_4;
        __pyx_t_4 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":172
 *             elif op == 1: # <=
 *                 return lt or eq
 *             elif op == 4: # >             # «««««««
 *                 return not (lt or eq)
 *             elif op == 5: # >=
 */
        break;
        case 5:

        /* "PyClical.pyx":175
 *                 return not (lt or eq)
 *             elif op == 5: # >=
 *                 return not lt             # «««««««
 *             else:
 *                 return NotImplemented
 */
        __Pyx_XDECREF(__pyx_r);
```

```
        __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 175, __pyx_L1_error)
        __pyx_t_4 = __Pyx_PyBool_FromLong((!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 175, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_4);
        __pyx_r = __pyx_t_4;
        __pyx_t_4 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":174
 *                 elif op == 4: # >
 *                     return not (lt or eq)
 *                 elif op == 5: # >=             # «««««««
 *                     return not lt
 *                 else:
 */
        break;
        default:

        /* "PyClical.pyx":177
 *                     return not lt
 *                 else:
 *                     return NotImplemented             # «««««««
 *
 *     def __setitem__(self, idx, val):
 */
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_builtin_NotImplemented);
        __pyx_r = __pyx_builtin_NotImplemented;
        goto __pyx_L0;
        break;
      }
      break;
    }
  }

  /* "PyClical.pyx":122
 *         del self.instance
 *
 *     def __richcmp__(lhs, rhs, int op):             # «««««««
 *         """
 *         Compare two objects of class index_set.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_4);
  __Pyx_AddTraceback("PyClical.index_set.__richcmp__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XDECREF(__pyx_v_eq);
  __Pyx_XDECREF(__pyx_v_lt);
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":179
 *                     return NotImplemented
 *
 *     def __setitem__(self, idx, val):             # «««««««
 *         """
 *         Set the value of an index_set object at index idx to value val.
 */

/* Python wrapper */
static int __pyx_pw_8PyClical_9index_set_9__setitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val); /*proto*/
static char __pyx_doc_8PyClical_9index_set_8__setitem__[] = "\n        Set the value of an index_set object at index idx to value val.\n\n        »> s=index_set({1}); s[2] = True; print(s)\n        {1,2}\n        »> s=index_set({1,2}); s[1] = False; print(s)\n        {2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_8__setitem__;
#endif
static int __pyx_pw_8PyClical_9index_set_9__setitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val) {
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__setitem__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_8__setitem__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_idx), ((PyObject *)__pyx_v_val));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

```

```
static int __pyx_pf_8PyClical_9index_set_8__setitem__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val) {
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_t_2;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__setitem__", 0);

  /* "PyClical.pyx":188
 *         {2}
 *         """
 *         self.instance.set(idx, val)             # «««««««
 *         return
 *
 */
  __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 188, __pyx_L1_error)
  __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_val); if (unlikely((__pyx_t_2 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 188, __pyx_L1_error)
  try {
    __pyx_v_self->instance->set(__pyx_t_1, __pyx_t_2);
  } catch(...) {
    __Pyx_CppExn2PyErr();
    __PYX_ERR(0, 188, __pyx_L1_error)
  }

  /* "PyClical.pyx":189
 *         """
 *         self.instance.set(idx, val)
 *         return             # «««««««
 *
 *     def __getitem__(self, idx):
 */
  __pyx_r = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":179
 *                     return NotImplemented
 *
 *     def __setitem__(self, idx, val):             # «««««««
 *         """
 *         Set the value of an index_set object at index idx to value val.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_AddTraceback("PyClical.index_set.__setitem__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = -1;
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":191
 *         return
 *
 *     def __getitem__(self, idx):             # «««««««
 *         """
 *         Get the value of an index_set object at an index.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_11__getitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx); /*proto*/
static char __pyx_doc_8PyClical_9index_set_10__getitem__[] = "\n        Get the value of an index_set object at an index.\n\n        >>> index_set({1})[1]\n        True\n        >>> index_set({1})[2]\n        False\n        >>> index_set({2})[-1]\n        False\n        >>> index_set({2})[1]\n        False\n        >>> index_set({2})[2]\n        True\n        >>> index_set({2})[33]\n        False\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_10__getitem__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_11__getitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__getitem__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_10__getitem__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_idx));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
```

```
static PyObject *__pyx_pf_8PyClical_9index_set_10__getitem__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__getitem__", 0);

  /* "PyClical.pyx":208
 *         False
 *         """
 *         return self.instance.getitem(idx)             # <<<<<<<<<<<<<<
 *
 *     def __contains__(self, idx):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 208, __pyx_L1_error)
  __pyx_t_2 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->operator[](__pyx_t_1)); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 208, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":191
 *         return
 *
 *     def __getitem__(self, idx):             # <<<<<<<<<<<<<<
 *         """
 *         Get the value of an index_set object at an index.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.index_set.__getitem__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":210
 *         return self.instance.getitem(idx)
 *
 *     def __contains__(self, idx):             # <<<<<<<<<<<<<<
 *         """
 *         Check that an index_set object contains the index idx: idx in self.
 */

/* Python wrapper */
static int __pyx_pw_8PyClical_9index_set_13__contains__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx); /*proto*/
static char __pyx_doc_8PyClical_9index_set_12__contains__[] = "\n        Check that an index_set object contains the index idx: idx in self.\n\n        >>> 1 in index_set({1})\n        True\n        >>> 2 in index_set({1})\n        False\n        >>> -1 in index_set({2})\n        False\n        >>> 1 in index_set({2})\n        False\n        >>> 2 in index_set({2})\n        True\n        >>> 33 in index_set({2})\n        False\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_12__contains__;
#endif
static int __pyx_pw_8PyClical_9index_set_13__contains__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx) {
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__contains__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_12__contains__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_idx));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static int __pyx_pf_8PyClical_9index_set_12__contains__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_idx) {
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
```

```
  __Pyx_RefNannySetupContext("__contains__", 0);

  /* "PyClical.pyx":227
 *         False
 *         """
 *         return self.instance.getitem(idx)             # <<<<<<<<
 *
 *     def __iter__(self):
 */
  __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 227, __pyx_L1_error)
  __pyx_r = __pyx_v_self->instance->operator[](__pyx_t_1);
  goto __pyx_L0;

  /* "PyClical.pyx":210
 *         return self.instance.getitem(idx)
 *
 *     def __contains__(self, idx):             # <<<<<<<<
 *         """
 *         Check that an index_set object contains the index idx: idx in self.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_AddTraceback("PyClical.index_set.__contains__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = -1;
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
static PyObject *__pyx_gb_8PyClical_9index_set_16generator(__pyx_CoroutineObject *__pyx_generator,
CYTHON_UNUSED PyThreadState *__pyx_tstate, PyObject *__pyx_sent_value); /* proto */

/* "PyClical.pyx":229
 *         return self.instance.getitem(idx)
 *
 *     def __iter__(self):             # <<<<<<<<
 *         """
 *         Iterate over the indices of an index_set.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_15__iter__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_9index_set_14__iter__[] = "\n        Iterate over the indices of an
index_set.\n\n        >>> for i in index_set({-3,4,7}):print(i, end=\",\")\n        -3,4,7,\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_14__iter__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_15__iter__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__iter__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_14__iter__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_14__iter__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
  struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *__pyx_cur_scope;
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__iter__", 0);
  __pyx_cur_scope = (struct __pyx_obj_8PyClical___pyx_scope_struct____iter__
*)__pyx_tp_new_8PyClical___pyx_scope_struct____iter__(__pyx_ptype_8PyClical___pyx_scope_struct____iter__,
__pyx_empty_tuple, NULL);
  if (unlikely(!__pyx_cur_scope)) {
    __pyx_cur_scope = ((struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *)Py_None);
    __Pyx_INCREF(Py_None);
    __PYX_ERR(0, 229, __pyx_L1_error)
  } else {
    __Pyx_GOTREF(__pyx_cur_scope);
  }
  __pyx_cur_scope->__pyx_v_self = __pyx_v_self;
  __Pyx_INCREF((PyObject *)__pyx_cur_scope->__pyx_v_self);
  __Pyx_GIVEREF((PyObject *)__pyx_cur_scope->__pyx_v_self);
  {
    __pyx_CoroutineObject *gen = __Pyx_Generator_New((__pyx_coroutine_body_t)
__pyx_gb_8PyClical_9index_set_16generator, NULL, (PyObject *) __pyx_cur_scope, __pyx_n_s_iter,
__pyx_n_s_index_set___iter, __pyx_n_s_PyClical); if (unlikely(!gen)) __PYX_ERR(0, 229, __pyx_L1_error)
    __Pyx_DECREF(__pyx_cur_scope);
```

```
    __Pyx_RefNannyFinishContext();
    return (PyObject *) gen;
  }

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_AddTraceback("PyClical.index_set.__iter__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __Pyx_DECREF(((PyObject *)__pyx_cur_scope));
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_gb_8PyClical_9index_set_16generator(__pyx_CoroutineObject *__pyx_generator, CYTHON_UNUSED PyThreadState *__pyx_tstate, PyObject *__pyx_sent_value) /* generator body */
{
  struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *__pyx_cur_scope = ((struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *)__pyx_generator->closure);
  PyObject *__pyx_r = NULL;
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  PyObject *__pyx_t_4 = NULL;
  Py_ssize_t __pyx_t_5;
  PyObject *(*__pyx_t_6)(PyObject *);
  int __pyx_t_7;
  int __pyx_t_8;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__iter__", 0);
  switch (__pyx_generator->resume_label) {
    case 0: goto __pyx_L3_first_run;
    case 1: goto __pyx_L7_resume_from_yield;
    default: /* CPython raises the right error here */
    __Pyx_RefNannyFinishContext();
    return NULL;
  }
  __pyx_L3_first_run:;
  if (unlikely(!__pyx_sent_value)) __PYX_ERR(0, 229, __pyx_L1_error)

  /* "PyClical.pyx":236
 *          -3,4,7,
 *          """
 *          for idx in range(self.min(), self.max()+1):             # «««««««
 *              if idx in self:
 *                  yield idx
 */
  __pyx_t_2 = __Pyx_PyObject_GetAttrStr(((PyObject *)__pyx_cur_scope->__pyx_v_self), __pyx_n_s_min); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_t_3 = NULL;
  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_2))) {
    __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_2);
    if (likely(__pyx_t_3)) {
      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_2);
      __Pyx_INCREF(__pyx_t_3);
      __Pyx_INCREF(function);
      __Pyx_DECREF_SET(__pyx_t_2, function);
    }
  }
  __pyx_t_1 = (__pyx_t_3) ? __Pyx_PyObject_CallOneArg(__pyx_t_2, __pyx_t_3) : __Pyx_PyObject_CallNoArg(__pyx_t_2);
  __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(((PyObject *)__pyx_cur_scope->__pyx_v_self), __pyx_n_s_max); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __pyx_t_4 = NULL;
  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
    __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
    if (likely(__pyx_t_4)) {
      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
      __Pyx_INCREF(__pyx_t_4);
      __Pyx_INCREF(function);
      __Pyx_DECREF_SET(__pyx_t_3, function);
    }
  }
  __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
  if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
```

```
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  __pyx_t_3 = __Pyx_PyInt_AddObjC(__pyx_t_2, __pyx_int_1, 1, 0, 0); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __pyx_t_2 = PyTuple_New(2); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_GIVEREF(__pyx_t_1);
  PyTuple_SET_ITEM(__pyx_t_2, 0, __pyx_t_1);
  __Pyx_GIVEREF(__pyx_t_3);
  PyTuple_SET_ITEM(__pyx_t_2, 1, __pyx_t_3);
  __pyx_t_1 = 0;
  __pyx_t_3 = 0;
  __pyx_t_3 = __Pyx_PyObject_Call(__pyx_builtin_range, __pyx_t_2, NULL); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (likely(PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(__pyx_t_3)) {
    __pyx_t_2 = __pyx_t_3; __Pyx_INCREF(__pyx_t_2); __pyx_t_5 = 0;
    __pyx_t_6 = NULL;
  } else {
    __pyx_t_5 = -1; __pyx_t_2 = PyObject_GetIter(__pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_2);
    __pyx_t_6 = Py_TYPE(__pyx_t_2)->tp_iternext; if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 236, __pyx_L1_error)
  }
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  for (;;) {
    if (likely(!__pyx_t_6)) {
      if (likely(PyList_CheckExact(__pyx_t_2))) {
        if (__pyx_t_5 >= PyList_GET_SIZE(__pyx_t_2)) break;
        #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
        __pyx_t_3 = PyList_GET_ITEM(__pyx_t_2, __pyx_t_5); __Pyx_INCREF(__pyx_t_3); __pyx_t_5++; if (unlikely(0 < 0)) __PYX_ERR(0, 236, __pyx_L1_error)
        #else
        __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_5); __pyx_t_5++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_3);
        #endif
      } else {
        if (__pyx_t_5 >= PyTuple_GET_SIZE(__pyx_t_2)) break;
        #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
        __pyx_t_3 = PyTuple_GET_ITEM(__pyx_t_2, __pyx_t_5); __Pyx_INCREF(__pyx_t_3); __pyx_t_5++; if (unlikely(0 < 0)) __PYX_ERR(0, 236, __pyx_L1_error)
        #else
        __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_5); __pyx_t_5++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_3);
        #endif
      }
    } else {
      __pyx_t_3 = __pyx_t_6(__pyx_t_2);
      if (unlikely(!__pyx_t_3)) {
        PyObject* exc_type = PyErr_Occurred();
        if (exc_type) {
          if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
          else __PYX_ERR(0, 236, __pyx_L1_error)
        }
        break;
      }
      __Pyx_GOTREF(__pyx_t_3);
    }
    __Pyx_XGOTREF(__pyx_cur_scope->__pyx_v_idx);
    __Pyx_XDECREF_SET(__pyx_cur_scope->__pyx_v_idx, __pyx_t_3);
    __Pyx_GIVEREF(__pyx_t_3);
    __pyx_t_3 = 0;

    /* "PyClical.pyx":237
 *         """
 *         for idx in range(self.min(), self.max()+1):
 *             if idx in self:             # <<<<<<<<
 *                 yield idx
 *
 */
    __pyx_t_7 = (__Pyx_PySequence_ContainsTF(__pyx_cur_scope->__pyx_v_idx, ((PyObject *)__pyx_cur_scope->__pyx_v_self), Py_EQ)); if (unlikely(__pyx_t_7 < 0)) __PYX_ERR(0, 237, __pyx_L1_error)
    __pyx_t_8 = (__pyx_t_7 != 0);
    if (__pyx_t_8) {

      /* "PyClical.pyx":238
 *         for idx in range(self.min(), self.max()+1):
 *             if idx in self:
 *                 yield idx             # <<<<<<<<
 *
 *     def __invert__(self):
```

```
*/
        __Pyx_INCREF(__pyx_cur_scope->__pyx_v_idx);
        __pyx_r = __pyx_cur_scope->__pyx_v_idx;
        __Pyx_XGIVEREF(__pyx_t_2);
        __pyx_cur_scope->__pyx_t_0 = __pyx_t_2;
        __pyx_cur_scope->__pyx_t_1 = __pyx_t_5;
        __pyx_cur_scope->__pyx_t_2 = __pyx_t_6;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        __Pyx_Coroutine_ResetAndClearException(__pyx_generator);
        /* return from generator, yielding value */
        __pyx_generator->resume_label = 1;
        return __pyx_r;
        __pyx_L7_resume_from_yield:;
        __pyx_t_2 = __pyx_cur_scope->__pyx_t_0;
        __pyx_cur_scope->__pyx_t_0 = 0;
        __Pyx_XGOTREF(__pyx_t_2);
        __pyx_t_5 = __pyx_cur_scope->__pyx_t_1;
        __pyx_t_6 = __pyx_cur_scope->__pyx_t_2;
        if (unlikely(!__pyx_sent_value)) __PYX_ERR(0, 238, __pyx_L1_error)

        /* "PyClical.pyx":237
 *             """
 *             for idx in range(self.min(), self.max()+1):
 *                 if idx in self:             # <<<<<<<<<<
 *                     yield idx
 *
 */
      }

    /* "PyClical.pyx":236
 *             -3,4,7,
 *             """
 *             for idx in range(self.min(), self.max()+1):             # <<<<<<<<<<
 *                 if idx in self:
 *                     yield idx
 */
    }
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    CYTHON_MAYBE_UNUSED_VAR(__pyx_cur_scope);

    /* "PyClical.pyx":229
 *             return self.instance.getitem(idx)
 *
 *         def __iter__(self):             # <<<<<<<<<<
 *             """
 *             Iterate over the indices of an index_set.
 */

    /* function exit code */
    PyErr_SetNone(PyExc_StopIteration);
    goto __pyx_L0;
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_XDECREF(__pyx_t_2);
    __Pyx_XDECREF(__pyx_t_3);
    __Pyx_XDECREF(__pyx_t_4);
    __Pyx_AddTraceback("__iter__", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_L0:;
    __Pyx_XDECREF(__pyx_r); __pyx_r = 0;
    #if !CYTHON_USE_EXC_INFO_STACK
    __Pyx_Coroutine_ResetAndClearException(__pyx_generator);
    #endif
    __pyx_generator->resume_label = -1;
    __Pyx_Coroutine_clear((PyObject*)__pyx_generator);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
}

/* "PyClical.pyx":240
 *                     yield idx
 *
 *         def __invert__(self):             # <<<<<<<<<<
 *             """
 *             Set complement: not.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_18__invert__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_9index_set_17__invert__[] = "\n        Set complement: not.\n\n
>>>
print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))\n
{-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}\n
";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_17__invert__;
#endif
```

```
static PyObject *__pyx_pw_8PyClical_9index_set_18__invert__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__invert__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_17__invert__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_17__invert__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__invert__", 0);

  /* "PyClical.pyx":247
 * {-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}
 *         """
 *         return index_set().wrap( self.instance.invert() )             # <<<<<<<<<<<<<<
 *
 *     def __xor__(lhs, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 247, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1), __pyx_v_self->instance->operator~()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 247, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":240
 *                 yield idx
 *
 *     def __invert__(self):             # <<<<<<<<<<<<<<
 *         """
 *         Set complement: not.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.index_set.__invert__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":249
 *         return index_set().wrap( self.instance.invert() )
 *
 *     def __xor__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Symmetric set difference: exclusive or.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_20__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_19__xor__[] = "\n        Symmetric set difference: exclusive or.\n\n        >>> print(index_set({1}) ^ index_set({2}))\n        {1,2}\n        >>> print(index_set({1,2}) ^ index_set({2}))\n        {1}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_19__xor__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_20__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__xor__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_19__xor__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));
```

```

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_19__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__xor__", 0);

  /* "PyClical.pyx":258
 *         {1}
 *         """
 *         return index_set().wrap( toIndexSet(lhs) ^ toIndexSet(rhs) )             # «««««««
 *
 *     def __ixor__(self, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 258, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1), (__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) ^ __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 258, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":249
 *         return index_set().wrap( self.instance.invert() )
 *
 *     def __xor__(lhs, rhs):             # «««««««
 *         """
 *         Symmetric set difference: exclusive or.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.index_set.__xor__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":260
 *         return index_set().wrap( toIndexSet(lhs) ^ toIndexSet(rhs) )
 *
 *     def __ixor__(self, rhs):             # «««««««
 *         """
 *         Symmetric set difference: exclusive or.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_22__ixor__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_21__ixor__[] = "\n        Symmetric set difference: exclusive or.\n\n        »> x = index_set({1}); x ^= index_set({2}); print(x)\n        {1,2}\n        »> x = index_set({1,2}); x ^= index_set({2}); print(x)\n        {1}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_21__ixor__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_22__ixor__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__ixor__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_21__ixor__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

```

```
static PyObject *__pyx_pf_8PyClical_9index_set_21__ixor__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__ixor__", 0);

  /* "PyClical.pyx":269
 *         {1}
 *         """
 *         return self.wrap( self.unwrap() ^ toIndexSet(rhs) )             # «««««««
 *
 *     def __and__(lhs, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self, (__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) ^ __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 269, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":260
 *         return index_set().wrap( toIndexSet(lhs) ^ toIndexSet(rhs) )
 *
 *     def __ixor__(self, rhs):              # «««««««
 *         """
 *         Symmetric set difference: exclusive or.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__ixor__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":271
 *         return self.wrap( self.unwrap() ^ toIndexSet(rhs) )
 *
 *     def __and__(lhs, rhs):              # «««««««
 *         """
 *         Set intersection: and.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_24__and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_23__and__[] = "\n        Set intersection: and.\n\n        >>> print(index_set({1}) & index_set({2}))\n        {}\n        >>> print(index_set({1,2}) & index_set({2}))\n        {2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_23__and__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_24__and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__and__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_23__and__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_23__and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__and__", 0);

  /* "PyClical.pyx":280
```

```
 *         {2}
 *         """
 *         return index_set().wrap( toIndexSet(lhs) & toIndexSet(rhs) )             # <<<<<<<<
 *
 *     def __iand__(self, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 280, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1),
(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) & __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 280, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":271
 *         return self.wrap( self.unwrap() ^ toIndexSet(rhs) )
 *
 *     def __and__(lhs, rhs):             # <<<<<<<<
 *         """
 *         Set intersection: and.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.index_set.__and__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":282
 *         return index_set().wrap( toIndexSet(lhs) & toIndexSet(rhs) )
 *
 *     def __iand__(self, rhs):             # <<<<<<<<
 *         """
 *         Set intersection: and.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_26__iand__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_25__iand__[] = "\n        Set intersection: and.\n\n
>>> x = index_set({1}); x &= index_set({2}); print(x)\n        {}\n        >>> x = index_set({1,2}); x
&= index_set({2}); print(x)\n        {2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_25__iand__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_26__iand__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__iand__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_25__iand__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_25__iand__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__iand__", 0);

  /* "PyClical.pyx":291
 *         {2}
 *         """
 *         return self.wrap( self.unwrap() & toIndexSet(rhs) )             # <<<<<<<<
 *
 *     def __or__(lhs, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
```

```
  __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self, (__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) & __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 291, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":282
 *         return index_set().wrap( toIndexSet(lhs) & toIndexSet(rhs) )
 *
 *     def __iand__(self, rhs):             # «««««««
 *         """
 *         Set intersection: and.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__iand__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":293
 *         return self.wrap( self.unwrap() & toIndexSet(rhs) )
 *
 *     def __or__(lhs, rhs):             # «««««««
 *         """
 *         Set union: or.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_28__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_27__or__[] = "\n        Set union: or.\n\n        >>> print(index_set({1}) | index_set({2}))\n        {1,2}\n        >>> print(index_set({1,2}) | index_set({2}))\n        {1,2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_27__or__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_28__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__or__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_27__or__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_27__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__or__", 0);

  /* "PyClical.pyx":302
 *         {1,2}
 *         """
 *         return index_set().wrap( toIndexSet(lhs) | toIndexSet(rhs) )             # «««««««
 *
 *     def __ior__(self, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 302, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1), (__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) | __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 302, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;
```

```

  /* "PyClical.pyx":293
 *         return self.wrap( self.unwrap() & toIndexSet(rhs) )
 *
 *     def __or__(lhs, rhs):             # <<<<<<<<
 *         """
 *         Set union: or.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.index_set.__or__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":304
 *         return index_set().wrap( toIndexSet(lhs) | toIndexSet(rhs) )
 *
 *     def __ior__(self, rhs):             # <<<<<<<<
 *         """
 *         Set union: or.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_30__ior__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_29__ior__[] = "\n        Set union: or.\n\n        >>> x = index_set({1}); x |= index_set({2}); print(x)\n        {1,2}\n        >>> x = index_set({1,2}); x |= index_set({2}); print(x)\n        {1,2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_29__ior__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_30__ior__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__ior__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_29__ior__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_29__ior__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__ior__", 0);

  /* "PyClical.pyx":313
 *         {1,2}
 *         """
 *         return self.wrap( self.unwrap() | toIndexSet(rhs) )             # <<<<<<<<
 *
 *     def count(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self, (__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) | __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 313, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":304
 *         return index_set().wrap( toIndexSet(lhs) | toIndexSet(rhs) )
 *
 *     def __ior__(self, rhs):             # <<<<<<<<
 *         """
 *         Set union: or.
 */

  /* function exit code */
  __pyx_L1_error:;
```

```
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__ior__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":315
 *         return self.wrap( self.unwrap() | toIndexSet(rhs) )
 *
 *     def count(self):              # «««««««
 *         """
 *         Cardinality: Number of indices included in set.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_32count(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_31count[] = "\n        Cardinality: Number of indices included in set.\n\n        >>> index_set({-1,1,2}).count()\n        3\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_32count(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("count (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_31count(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_31count(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("count", 0);

  /* "PyClical.pyx":322
 *         3
 *         """
 *         return self.instance.count()              # «««««««
 *
 *     def count_neg(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 322, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":315
 *         return self.wrap( self.unwrap() | toIndexSet(rhs) )
 *
 *     def count(self):              # «««««««
 *         """
 *         Cardinality: Number of indices included in set.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.count", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":324
 *         return self.instance.count()
 *
 *     def count_neg(self):              # «««««««
 *         """
 *         Number of negative indices included in set.
 */

```

```
/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_34count_neg(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_33count_neg[] = "\n        Number of negative indices
included in set.\n\n        >>> index_set({-1,1,2}).count_neg()\n        1\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_34count_neg(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("count_neg (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_33count_neg(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_33count_neg(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("count_neg", 0);

  /* "PyClical.pyx":331
 *         1
 *         """
 *         return self.instance.count_neg()             # <<<<<<<<<<<<<<
 *
 *     def count_pos(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count_neg()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 331, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":324
 *         return self.instance.count()
 *
 *     def count_neg(self):             # <<<<<<<<<<<<<<
 *         """
 *         Number of negative indices included in set.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.count_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":333
 *         return self.instance.count_neg()
 *
 *     def count_pos(self):             # <<<<<<<<<<<<<<
 *         """
 *         Number of positive indices included in set.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_36count_pos(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_35count_pos[] = "\n        Number of positive indices
included in set.\n\n        >>> index_set({-1,1,2}).count_pos()\n        2\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_36count_pos(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("count_pos (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_35count_pos(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
```

```

static PyObject *__pyx_pf_8PyClical_9index_set_35count_pos(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("count_pos", 0);

  /* "PyClical.pyx":340
 *         2
 *         """
 *         return self.instance.count_pos()             # «««««««
 *
 *     def min(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count_pos()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 340, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":333
 *         return self.instance.count_neg()
 *
 *     def count_pos(self):             # «««««««
 *         """
 *         Number of positive indices included in set.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.count_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":342
 *         return self.instance.count_pos()
 *
 *     def min(self):             # «««««««
 *         """
 *         Minimum member.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_38min(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_37min[] = "\n        Minimum member.\n\n        »> index_set({-1,1,2}).min()\n        -1\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_38min(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("min (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_37min(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_37min(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("min", 0);

  /* "PyClical.pyx":349
 *         -1
 *         """
 *         return self.instance.min()             # «««««««
 *
 *     def max(self):
```

```
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->min()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 349, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":342
 *         return self.instance.count_pos()
 *
 *     def min(self):              # «««««««
 *         """
 *         Minimum member.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.min", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":351
 *         return self.instance.min()
 *
 *     def max(self):              # «««««««
 *         """
 *         Maximum member.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_40max(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_39max[] = "\n        Maximum member.\n\n        >>>
index_set({-1,1,2}).max()\n        2\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_40max(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("max (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_39max(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_39max(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("max", 0);

  /* "PyClical.pyx":358
 *         2
 *         """
 *         return self.instance.max()              # «««««««
 *
 *     def hash_fn(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->max()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 358, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":351
 *         return self.instance.min()
 *
 *     def max(self):              # «««««««
 *         """
 *         Maximum member.
 */

```

```
  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.max", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":360
 *         return self.instance.max()
 *
 *     def hash_fn(self):             # <<<<<<<<
 *         """
 *         Hash function.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_42hash_fn(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_41hash_fn[] = "\n        Hash function.\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_42hash_fn(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("hash_fn (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_41hash_fn(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_41hash_fn(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("hash_fn", 0);

  /* "PyClical.pyx":364
 *         Hash function.
 *         """
 *         return self.instance.hash_fn()             # <<<<<<<<
 *
 *     def sign_of_mult(self, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->hash_fn()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 364, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":360
 *         return self.instance.max()
 *
 *     def hash_fn(self):             # <<<<<<<<
 *         """
 *         Hash function.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.hash_fn", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":366
 *         return self.instance.hash_fn()
 *
 *     def sign_of_mult(self, rhs):             # <<<<<<<<
 *         """
 *         Sign of geometric product of two Clifford basis elements.
 */
```

```
/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_44sign_of_mult(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_9index_set_43sign_of_mult[] = "\n        Sign of geometric product of two Clifford basis elements.\n\n        >>> s = index_set({1,2}); t=index_set({-1}); s.sign_of_mult(t)\n        1\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_44sign_of_mult(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("sign_of_mult (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_43sign_of_mult(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_43sign_of_mult(struct __pyx_obj_8PyClical_index_set *__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("sign_of_mult", 0);

  /* "PyClical.pyx":373
 *         1
 *         """
 *         return self.instance.sign_of_mult(toIndexSet(rhs))             # <<<<<<<<<<<<<<
 *
 *     def sign_of_square(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->sign_of_mult(__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 373, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":366
 *         return self.instance.hash_fn()
 *
 *     def sign_of_mult(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Sign of geometric product of two Clifford basis elements.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.sign_of_mult", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":375
 *         return self.instance.sign_of_mult(toIndexSet(rhs))
 *
 *     def sign_of_square(self):             # <<<<<<<<<<<<<<
 *         """
 *         Sign of geometric square of a Clifford basis element.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_46sign_of_square(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_9index_set_45sign_of_square[] = "\n        Sign of geometric square of a Clifford basis element.\n\n        >>> s = index_set({1,2}); s.sign_of_square()\n        -1\n        ";
static PyObject *__pyx_pw_8PyClical_9index_set_46sign_of_square(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("sign_of_square (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_45sign_of_square(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

```

```
  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_45sign_of_square(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("sign_of_square", 0);

  /* "PyClical.pyx":382
 *         -1
 *         """
 *         return self.instance.sign_of_square()             # «««««««
 *
 *     def __repr__(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->sign_of_square()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 382, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":375
 *         return self.instance.sign_of_mult(toIndexSet(rhs))
 *
 *     def sign_of_square(self):             # «««««««
 *         """
 *         Sign of geometric square of a Clifford basis element.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.sign_of_square", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":384
 *         return self.instance.sign_of_square()
 *
 *     def __repr__(self):             # «««««««
 *         """
 *         The official string representation of self.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_48__repr__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_9index_set_47__repr__[] = "\n        The \342\200\234official\342\200\235 string representation of self.\n\n        »> index_set({1,2}).__repr__()\n        'index_set({1,2})'\n        »> repr(index_set({1,2}))\n        'index_set({1,2})'\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_47__repr__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_48__repr__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__repr__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_47__repr__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_47__repr__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
```

```
  __Pyx_RefNannySetupContext("__repr__", 0);

  /* "PyClical.pyx":393
 *         'index_set({1,2})'
 *         """
 *         return index_set_to_repr( self.unwrap() ).decode()             # «««««««
 *
 *     def __str__(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_decode_cpp_string(index_set_to_repr(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self)), 0, PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 393, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":384
 *         return self.instance.sign_of_square()
 *
 *     def __repr__(self):             # «««««««
 *         """
 *         The official string representation of self.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__repr__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":395
 *         return index_set_to_repr( self.unwrap() ).decode()
 *
 *     def __str__(self):             # «««««««
 *         """
 *         The informal string representation of self.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_50__str__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_9index_set_49__str__[] = "\n        The \342\200\234informal\342\200\235 string representation of self.\n\n        >> index_set({1,2}).__str__()\n        '{1,2}'\n        >> str(index_set({1,2}))\n        '{1,2}'\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_49__str__;
#endif
static PyObject *__pyx_pw_8PyClical_9index_set_50__str__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__str__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_49__str__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_49__str__(struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__str__", 0);

  /* "PyClical.pyx":404
 *         '{1,2}'
 *         """
 *         return index_set_to_str( self.unwrap() ).decode()             # «««««««
 *
 * def index_set_hidden_doctests():
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_decode_cpp_string(index_set_to_str(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self)), 0, PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 404, __pyx_L1_error)
```

```
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":395
 *         return index_set_to_repr( self.unwrap() ).decode()
 *
 *     def __str__(self):             # «««««««
 *         """
 *         The informal string representation of self.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__str__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "(tree fragment)":1
 * def __reduce_cython__(self):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_52__reduce_cython__(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static PyObject *__pyx_pw_8PyClical_9index_set_52__reduce_cython__(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__reduce_cython__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_51__reduce_cython__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_51__reduce_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_index_set *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__reduce_cython__", 0);

  /* "(tree fragment)":2
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # «««««««
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */
  __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__3, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 2, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_Raise(__pyx_t_1, 0, 0, 0);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __PYX_ERR(1, 2, __pyx_L1_error)

  /* "(tree fragment)":1
 * def __reduce_cython__(self):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__reduce_cython__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "(tree fragment)":3
```

```
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_9index_set_54__setstate_cython__(PyObject *__pyx_v_self, PyObject *__pyx_v___pyx_state); /*proto*/
static PyObject *__pyx_pw_8PyClical_9index_set_54__setstate_cython__(PyObject *__pyx_v_self, PyObject *__pyx_v___pyx_state) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__setstate_cython__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_9index_set_53__setstate_cython__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self), ((PyObject *)__pyx_v___pyx_state));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_9index_set_53__setstate_cython__(CYTHON_UNUSED struct __pyx_obj_8PyClical_index_set *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v___pyx_state) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__setstate_cython__", 0);

  /* "(tree fragment)":4
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # «««««««
 */
  __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__4, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 4, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_Raise(__pyx_t_1, 0, 0, 0);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __PYX_ERR(1, 4, __pyx_L1_error)

  /* "(tree fragment)":3
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.index_set.__setstate_cython__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":406
 *         return index_set_to_str( self.unwrap() ).decode()
 *
 * def index_set_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_1index_set_hidden_doctests(PyObject *__pyx_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_index_set_hidden_doctests[] = "\n    Tests for functions that Doctest cannot see.\n\n    For index_set.__cinit__: Construct index_set.\n\n    >>> print(index_set(1))\n    {1}\n    >>> print(index_set({1,2}))\n    {1,2}\n    >>> print(index_set(index_set({1,2})))\n    {1,2}\n    >>> print(index_set({1,2}))\n    {1,2}\n    >>> print(index_set({1,2,1}))\n    {1,2}\n    >>> print(index_set({1,2,1}))\n    {1,2}\n    >>> print(index_set(\"\"))\n    {}\n    >>> print(index_set(\"{\"))\n    Traceback (most recent call last):\n    ...\n    ValueError: Cannot initialize index_set object from invalid string '{'.\n    >>> print(index_set(\"{1\"))\n    Traceback (most recent call last):\n    ...\n    ValueError: Cannot initialize index_set object from invalid string '{1'.\n    >>> print(index_set(\"{1,2,100}\"))\n    Traceback (most recent call last):\n    ...\n    ValueError: Cannot initialize index_set object from invalid string '{1,2,100}'.\n    >>> print(index_set({1,2,100}))\n    Traceback (most recent call last):\n    ...\n    IndexError: Cannot initialize index_set object from invalid {1, 2, 100}.\n    >>> print(index_set([1,2]))\n    Traceback (most recent call last):\n    ...\n    TypeError: Cannot initialize index_set object from <class 'list'>.\n\n    For index_set.__richcmp__: Compare two objects of class index_set.\n\n    >>
```

```
index_set(1) == index_set({1})\n    True\n    »> index_set({1}) != index_set({1})\n    False\n    »>
index_set({1}) != index_set({2})\n    True\n    »> index_set({1}) == index_set({2})\n    False\n    »>
index_set({1}) < index_set({2})\n    True\n    »> index_set({1}) <= index_set({2})\n    True\n    »>
index_set({1}) > index_set({2})\n    False\n    »> index_set({1}) >= index_set({2})\n    False\n    »>
None == index_set({1,2})\n    False\n    »> None != index_set({1,2})\n    True\n    »> None <
index_set({1,2})\n    False\n    »> None <= index_set({1,2})\n    False\n    »> None >
index_set({1,2})\n    False\n    »> None >= index_set({1,2})\n    False\n    »> ""index_set({1,2}) ==
None\n    False\n    »> index_set({1,2}) != None\n    True\n    »> index_set({1,2}) < None\n
False\n    »> index_set({1,2}) <= None\n    False\n    »> index_set({1,2}) > None\n    False\n    »>
index_set({1,2}) >= None\n    False\n    ";
        static PyMethodDef __pyx_mdef_8PyClical_1index_set_hidden_doctests =
{"index_set_hidden_doctests", (PyCFunction)__pyx_pw_8PyClical_1index_set_hidden_doctests, METH_NOARGS,
__pyx_doc_8PyClical_index_set_hidden_doctests};
        static PyObject *__pyx_pw_8PyClical_1index_set_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("index_set_hidden_doctests (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_index_set_hidden_doctests(__pyx_self);

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_index_set_hidden_doctests(CYTHON_UNUSED PyObject
*__pyx_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("index_set_hidden_doctests", 0);

          /* "PyClical.pyx":490
 *     False
 *     """
 *     return             # «««««««
 *
 * cpdef inline compare(lhs,rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_r = Py_None; __Pyx_INCREF(Py_None);
          goto __pyx_L0;

          /* "PyClical.pyx":406
 *         return index_set_to_str( self.unwrap() ).decode()
 *
 * def index_set_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */

          /* function exit code */
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":492
 *     return
 *
 * cpdef inline compare(lhs,rhs):             # «««««««
 *     """
 *     "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
 */

        static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
        static CYTHON_INLINE PyObject *__pyx_f_8PyClical_compare(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, CYTHON_UNUSED int __pyx_skip_dispatch) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("compare", 0);

          /* "PyClical.pyx":502
 *     1
 *     """
 *     return glucat.compare( toIndexSet(lhs), toIndexSet(rhs) )             # «««««««
 *
 * cpdef inline min_neg(obj):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyInt_From_int(compare(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs),
__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 502,
```

```
__pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":492
 *     return
 *
 * cpdef inline compare(lhs,rhs):             # «««««««
 *     """
 *     "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = 0;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
        static char __pyx_doc_8PyClical_2compare[] = "\n    \"lexicographic compare\" eg. {3,4,5} is less than {3,7,8};\n    -1 if a<b, +1 if a>b, 0 if a==b.\n\n    »> compare(index_set({1,2}),index_set({-1,3}))\n    -1\n    »> compare(index_set({-1,4}),index_set({-1,3}))\n    1\n    ";
        static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
          PyObject *__pyx_v_lhs = 0;
          PyObject *__pyx_v_rhs = 0;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("compare (wrapper)", 0);
          {
            static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_lhs,&__pyx_n_s_rhs,0};
            PyObject* values[2] = {0,0};
            if (unlikely(__pyx_kwds)) {
              Py_ssize_t kw_args;
              const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
              switch (pos_args) {
                case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                CYTHON_FALLTHROUGH;
                case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                CYTHON_FALLTHROUGH;
                case  0: break;
                default: goto __pyx_L5_argtuple_error;
              }
              kw_args = PyDict_Size(__pyx_kwds);
              switch (pos_args) {
                case  0:
                if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) != 0)) kw_args--;
                else goto __pyx_L5_argtuple_error;
                CYTHON_FALLTHROUGH;
                case  1:
                if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) != 0)) kw_args--;
                else {
                  __Pyx_RaiseArgtupleInvalid("compare", 1, 2, 2, 1); __PYX_ERR(0, 492, __pyx_L3_error)
                }
              }
              if (unlikely(kw_args > 0)) {
                if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "compare") < 0)) __PYX_ERR(0, 492, __pyx_L3_error)
              }
            } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
              goto __pyx_L5_argtuple_error;
            } else {
              values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
              values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
            }
            __pyx_v_lhs = values[0];
            __pyx_v_rhs = values[1];
          }
          goto __pyx_L4_argument_unpacking_done;
          __pyx_L5_argtuple_error:;
          __Pyx_RaiseArgtupleInvalid("compare", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 492, __pyx_L3_error)
```

```
        __pyx_L3_error:;
        __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __Pyx_RefNannyFinishContext();
        return NULL;
        __pyx_L4_argument_unpacking_done:;
        __pyx_r = __pyx_pf_8PyClical_2compare(__pyx_self, __pyx_v_lhs, __pyx_v_rhs);

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_2compare(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("compare", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_compare(__pyx_v_lhs, __pyx_v_rhs, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 492, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":504
 *     return glucat.compare( toIndexSet(lhs), toIndexSet(rhs) )
 *
 * cpdef inline min_neg(obj):             # «««««««
 *     """
 *     Minimum negative index, or 0 if none.
 */

      static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_min_neg(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("min_neg", 0);

        /* "PyClical.pyx":511
 *     0
 *     """
 *     return glucat.min_neg( toIndexSet(obj) )             # «««««««
 *
 * cpdef inline max_pos(obj):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyInt_From_int(min_neg(__pyx_f_8PyClical_toIndexSet(__pyx_v_obj))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 511, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":504
 *     return glucat.compare( toIndexSet(lhs), toIndexSet(rhs) )
 *
 * cpdef inline min_neg(obj):             # «««««««
 *     """
 *     Minimum negative index, or 0 if none.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.min_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
      __pyx_r = 0;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
    static char __pyx_doc_8PyClical_4min_neg[] = "\n    Minimum negative index, or 0 if none.\n\n»> min_neg(index_set({1,2}))\n    0\n    ";
    static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
      __Pyx_RefNannySetupContext("min_neg (wrapper)", 0);
      __pyx_r = __pyx_pf_8PyClical_4min_neg(__pyx_self, ((PyObject *)__pyx_v_obj));

      /* function exit code */
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    static PyObject *__pyx_pf_8PyClical_4min_neg(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("min_neg", 0);
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __pyx_f_8PyClical_min_neg(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 504, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_r = __pyx_t_1;
      __pyx_t_1 = 0;
      goto __pyx_L0;

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_AddTraceback("PyClical.min_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":513
 *     return glucat.min_neg( toIndexSet(obj) )
 *
 * cpdef inline max_pos(obj):             # «««««««
 *     """
 *     Maximum positive index, or 0 if none.
 */

    static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
    static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_pos(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("max_pos", 0);

      /* "PyClical.pyx":520
 *     2
 *     """
 *     return glucat.max_pos( toIndexSet(obj) )             # «««««««
 *
 * cdef inline vector[scalar_t] list_to_vector(lst):
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyInt_From_int(max_pos(__pyx_f_8PyClical_toIndexSet(__pyx_v_obj))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 520, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_r = __pyx_t_1;
      __pyx_t_1 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":513
```

```
 *     return glucat.min_neg( toIndexSet(obj) )
 *
 * cpdef inline max_pos(obj):             # «««««««
 *     """
 *     Maximum positive index, or 0 if none.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.max_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = 0;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
        static char __pyx_doc_8PyClical_6max_pos[] = "\n    Maximum positive index, or 0 if none.\n\n »> max_pos(index_set({1,2}))\n    2\n    ";
        static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("max_pos (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_6max_pos(__pyx_self, ((PyObject *)__pyx_v_obj));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_6max_pos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("max_pos", 0);
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __pyx_f_8PyClical_max_pos(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 513, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.max_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":522
 *     return glucat.max_pos( toIndexSet(obj) )
 *
 * cdef inline vector[scalar_t] list_to_vector(lst):             # «««««««
 *      """
 *      Create a C++ std:vector[scalar_t] from an iterable Python object.
 */

        static CYTHON_INLINE std::vector<scalar_t>  __pyx_f_8PyClical_list_to_vector(PyObject *__pyx_v_lst) {
          std::vector<scalar_t>  __pyx_v_v;
          PyObject *__pyx_v_s = NULL;
          std::vector<scalar_t>  __pyx_r;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          Py_ssize_t __pyx_t_2;
          PyObject *(*__pyx_t_3)(PyObject *);
          PyObject *__pyx_t_4 = NULL;
          scalar_t __pyx_t_5;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("list_to_vector", 0);

          /* "PyClical.pyx":527
```

```
*       """
*       cdef vector[scalar_t] v
*       for s in lst:             # «««««««
*           v.push_back(<scalar_t>s)
*       return v
*/
          if (likely(PyList_CheckExact(__pyx_v_lst)) || PyTuple_CheckExact(__pyx_v_lst)) {
            __pyx_t_1 = __pyx_v_lst; __Pyx_INCREF(__pyx_t_1); __pyx_t_2 = 0;
            __pyx_t_3 = NULL;
          } else {
            __pyx_t_2 = -1; __pyx_t_1 = PyObject_GetIter(__pyx_v_lst); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 527, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_t_3 = Py_TYPE(__pyx_t_1)->tp_iternext; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 527, __pyx_L1_error)
          }
          for (;;) {
            if (likely(!__pyx_t_3)) {
              if (likely(PyList_CheckExact(__pyx_t_1))) {
                if (__pyx_t_2 >= PyList_GET_SIZE(__pyx_t_1)) break;
                #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
                __pyx_t_4 = PyList_GET_ITEM(__pyx_t_1, __pyx_t_2); __Pyx_INCREF(__pyx_t_4); __pyx_t_2++; if (unlikely(0 < 0)) __PYX_ERR(0, 527, __pyx_L1_error)
                #else
                __pyx_t_4 = PySequence_ITEM(__pyx_t_1, __pyx_t_2); __pyx_t_2++; if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 527, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_4);
                #endif
              } else {
                if (__pyx_t_2 >= PyTuple_GET_SIZE(__pyx_t_1)) break;
                #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
                __pyx_t_4 = PyTuple_GET_ITEM(__pyx_t_1, __pyx_t_2); __Pyx_INCREF(__pyx_t_4); __pyx_t_2++; if (unlikely(0 < 0)) __PYX_ERR(0, 527, __pyx_L1_error)
                #else
                __pyx_t_4 = PySequence_ITEM(__pyx_t_1, __pyx_t_2); __pyx_t_2++; if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 527, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_4);
                #endif
              }
            } else {
              __pyx_t_4 = __pyx_t_3(__pyx_t_1);
              if (unlikely(!__pyx_t_4)) {
                PyObject* exc_type = PyErr_Occurred();
                if (exc_type) {
                  if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
                  else __PYX_ERR(0, 527, __pyx_L1_error)
                }
                break;
              }
              __Pyx_GOTREF(__pyx_t_4);
            }
            __Pyx_XDECREF_SET(__pyx_v_s, __pyx_t_4);
            __pyx_t_4 = 0;

            /* "PyClical.pyx":528
*       cdef vector[scalar_t] v
*       for s in lst:
*           v.push_back(<scalar_t>s)            # «««««««
*       return v
*
*/
            __pyx_t_5 = __pyx_PyFloat_AsDouble(__pyx_v_s); if (unlikely((__pyx_t_5 == ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 528, __pyx_L1_error)
            try {
              __pyx_v_v.push_back(((scalar_t)__pyx_t_5));
            } catch(...) {
              __Pyx_CppExn2PyErr();
              __PYX_ERR(0, 528, __pyx_L1_error)
            }

            /* "PyClical.pyx":527
*       """
*       cdef vector[scalar_t] v
*       for s in lst:             # «««««««
*           v.push_back(<scalar_t>s)
*       return v
*/
          }
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

          /* "PyClical.pyx":529
*       for s in lst:
*           v.push_back(<scalar_t>s)
*       return v              # «««««««
*
* # Forward reference.
```

```
*/
        __pyx_r = __pyx_v_v;
        goto __pyx_L0;

        /* "PyClical.pyx":522
 *     return glucat.max_pos( toIndexSet(obj) )
 *
 * cdef inline vector[scalar_t] list_to_vector(lst):             # «««««««
 *     """
 *     Create a C++ std:vector[scalar_t] from an iterable Python object.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_4);
        __Pyx_WriteUnraisable("PyClical.list_to_vector", __pyx_clineno, __pyx_lineno, __pyx_filename, 1, 0);
        __Pyx_pretend_to_initialize(&__pyx_r);
        __pyx_L0:;
        __Pyx_XDECREF(__pyx_v_s);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":534
 * cdef class clifford
 *
 * cdef inline Clifford toClifford(obj):             # «««««««
 *     return clifford(obj).instance[0]
 *
 */

      static CYTHON_INLINE Clifford __pyx_f_8PyClical_toClifford(PyObject *__pyx_v_obj) {
        Clifford __pyx_r;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("toClifford", 0);

        /* "PyClical.pyx":535
 *
 * cdef inline Clifford toClifford(obj):
 *     return clifford(obj).instance[0]             # «««««««
 *
 * cdef class clifford:
 */
        __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 535, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = (((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1)->instance[0]);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":534
 * cdef class clifford
 *
 * cdef inline Clifford toClifford(obj):             # «««««««
 *     return clifford(obj).instance[0]
 *
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_WriteUnraisable("PyClical.toClifford", __pyx_clineno, __pyx_lineno, __pyx_filename, 1, 0);
        __Pyx_pretend_to_initialize(&__pyx_r);
        __pyx_L0:;
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":543
 *     cdef Clifford *instance # Wrapped instance of C++ class Clifford.
 *
 *     cdef inline wrap(clifford self, Clifford other):             # «««««««
 *         """
 *         Wrap an instance of the C++ class Clifford.
 */

      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_8clifford_wrap(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, Clifford __pyx_v_other) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
```

```
    __Pyx_RefNannySetupContext("wrap", 0);

    /* "PyClical.pyx":547
 *         Wrap an instance of the C++ class Clifford.
 *         """
 *         self.instance[0] = other             # <<<<<<<<
 *         return self
 *
 */
    (__pyx_v_self->instance[0]) = __pyx_v_other;

    /* "PyClical.pyx":548
 *         """
 *         self.instance[0] = other
 *         return self             # <<<<<<<<
 *
 *     cdef inline Clifford unwrap(clifford self):
 */
    __Pyx_XDECREF(__pyx_r);
    __Pyx_INCREF(((PyObject *)__pyx_v_self));
    __pyx_r = ((PyObject *)__pyx_v_self);
    goto __pyx_L0;

    /* "PyClical.pyx":543
 *     cdef Clifford *instance # Wrapped instance of C++ class Clifford.
 *
 *     cdef inline wrap(clifford self, Clifford other):             # <<<<<<<<
 *         """
 *         Wrap an instance of the C++ class Clifford.
 */

    /* function exit code */
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":550
 *         return self
 *
 *     cdef inline Clifford unwrap(clifford self):             # <<<<<<<<
 *         """
 *         Return the wrapped C++ Clifford instance.
 */

  static CYTHON_INLINE Clifford __pyx_f_8PyClical_8clifford_unwrap(struct
__pyx_obj_8PyClical_clifford *__pyx_v_self) {
    Clifford __pyx_r;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("unwrap", 0);

    /* "PyClical.pyx":554
 *         Return the wrapped C++ Clifford instance.
 *         """
 *         return self.instance[0]             # <<<<<<<<
 *
 *     cpdef copy(clifford self):
 */
    __pyx_r = (__pyx_v_self->instance[0]);
    goto __pyx_L0;

    /* "PyClical.pyx":550
 *         return self
 *
 *     cdef inline Clifford unwrap(clifford self):             # <<<<<<<<
 *         """
 *         Return the wrapped C++ Clifford instance.
 */

    /* function exit code */
    __pyx_L0:;
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":556
 *         return self.instance[0]
 *
 *     cpdef copy(clifford self):             # <<<<<<<<
 *         """
 *         Copy this clifford object.
 */

  static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
  static PyObject *__pyx_f_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford
```

```
*__pyx_v_self, int __pyx_skip_dispatch) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  PyObject *__pyx_t_4 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("copy", 0);
  /* Check if called by wrapper */
  if (unlikely(__pyx_skip_dispatch)) ;
  /* Check if overridden in Python */
  else if (unlikely((Py_TYPE(((PyObject *)__pyx_v_self))->tp_dictoffset != 0) || (Py_TYPE(((PyObject *)__pyx_v_self))->tp_flags & (Py_TPFLAGS_IS_ABSTRACT | Py_TPFLAGS_HEAPTYPE)))) {
    #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
    static PY_UINT64_T __pyx_tp_dict_version = __PYX_DICT_VERSION_INIT, __pyx_obj_dict_version = __PYX_DICT_VERSION_INIT;
    if (unlikely(!__Pyx_object_dict_version_matches(((PyObject *)__pyx_v_self), __pyx_tp_dict_version, __pyx_obj_dict_version))) {
      PY_UINT64_T __pyx_type_dict_guard = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
      #endif
      __pyx_t_1 = __Pyx_PyObject_GetAttrStr(((PyObject *)__pyx_v_self), __pyx_n_s_copy); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 556, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      if (!PyCFunction_Check(__pyx_t_1) || (PyCFunction_GET_FUNCTION(__pyx_t_1) != (PyCFunction)(void*)__pyx_pw_8PyClical_8clifford_1copy)) {
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(__pyx_t_1);
        __pyx_t_3 = __pyx_t_1; __pyx_t_4 = NULL;
        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_3))) {
          __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
          if (likely(__pyx_t_4)) {
            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
            __Pyx_INCREF(__pyx_t_4);
            __Pyx_INCREF(function);
            __Pyx_DECREF_SET(__pyx_t_3, function);
          }
        }
        __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
        __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
        if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 556, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        goto __pyx_L0;
      }
      #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
      __pyx_tp_dict_version = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
      __pyx_obj_dict_version = __Pyx_get_object_dict_version(((PyObject *)__pyx_v_self));
      if (unlikely(__pyx_type_dict_guard != __pyx_tp_dict_version)) {
        __pyx_tp_dict_version = __pyx_obj_dict_version = __PYX_DICT_VERSION_INIT;
      }
      #endif
      __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
      #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
    }
    #endif
  }

  /* "PyClical.pyx":563
 *         {2}
 *         """
 *         return clifford(self)             # <<<<<<<<<<<<<<
 *
 *     def __cinit__(self, other = 0, ixt = None):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), ((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 563, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":556
 *         return self.instance[0]
 *
 *     cpdef copy(clifford self):             # <<<<<<<<<<<<<<
 *         """
 *         Copy this clifford object.
 */
```

```

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_XDECREF(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_4);
  __Pyx_AddTraceback("PyClical.clifford.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = 0;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_8clifford_copy[] = "\n        Copy this clifford object.\n\n»> x=clifford(\"1{2}\"); y=x.copy(); print(y)\n        {2}\n        ";
static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("copy (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_copy(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("copy", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_8clifford_copy(__pyx_v_self, 1); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 556, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.clifford.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":565
 *         return clifford(self)
 *
 *     def __cinit__(self, other = 0, ixt = None):             # «««««««
 *         """
 *         Construct an object of type clifford.
 */

/* Python wrapper */
static int __pyx_pw_8PyClical_8clifford_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static int __pyx_pw_8PyClical_8clifford_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_other = 0;
  PyObject *__pyx_v_ixt = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  int __pyx_r;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__cinit__ (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_other,&__pyx_n_s_ixt,0};
    PyObject* values[2] = {0,0};
    values[0] = ((PyObject *)__pyx_int_0);
    values[1] = ((PyObject *)Py_None);
```

```
            if (unlikely(__pyx_kwds)) {
              Py_ssize_t kw_args;
              const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
              switch (pos_args) {
                case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                CYTHON_FALLTHROUGH;
                case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                CYTHON_FALLTHROUGH;
                case  0: break;
                default: goto __pyx_L5_argtuple_error;
              }
              kw_args = PyDict_Size(__pyx_kwds);
              switch (pos_args) {
                case  0:
                if (kw_args > 0) {
                  PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_other);
                  if (value) { values[0] = value; kw_args--; }
                }
                CYTHON_FALLTHROUGH;
                case  1:
                if (kw_args > 0) {
                  PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_ixt);
                  if (value) { values[1] = value; kw_args--; }
                }
              }
              if (unlikely(kw_args > 0)) {
                if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "__cinit__") < 0)) __PYX_ERR(0, 565, __pyx_L3_error)
              }
            } else {
              switch (PyTuple_GET_SIZE(__pyx_args)) {
                case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                CYTHON_FALLTHROUGH;
                case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                CYTHON_FALLTHROUGH;
                case  0: break;
                default: goto __pyx_L5_argtuple_error;
              }
            }
            __pyx_v_other = values[0];
            __pyx_v_ixt = values[1];
          }
          goto __pyx_L4_argument_unpacking_done;
          __pyx_L5_argtuple_error:;
          __Pyx_RaiseArgtupleInvalid("__cinit__", 0, 0, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 565, __pyx_L3_error)
          __pyx_L3_error:;
          __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __Pyx_RefNannyFinishContext();
          return -1;
          __pyx_L4_argument_unpacking_done:;
          __pyx_r = __pyx_pf_8PyClical_8clifford_2__cinit__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), __pyx_v_other, __pyx_v_ixt);

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static int __pyx_pf_8PyClical_8clifford_2__cinit__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_other, PyObject *__pyx_v_ixt) {
          PyObject *__pyx_v_error_msg_prefix = NULL;
          PyObject *__pyx_v_bother = NULL;
          PyObject *__pyx_v_err = NULL;
          int __pyx_r;
          __Pyx_RefNannyDeclarations
          int __pyx_t_1;
          int __pyx_t_2;
          PyObject *__pyx_t_3 = NULL;
          PyObject *__pyx_t_4 = NULL;
          PyObject *__pyx_t_5 = NULL;
          Clifford *__pyx_t_6;
          PyObject *__pyx_t_7 = NULL;
          PyObject *__pyx_t_8 = NULL;
          scalar_t __pyx_t_9;
          PyObject *__pyx_t_10 = NULL;
          PyObject *__pyx_t_11 = NULL;
          PyObject *__pyx_t_12 = NULL;
          PyObject *__pyx_t_13 = NULL;
          char *__pyx_t_14;
          int __pyx_t_15;
          PyObject *__pyx_t_16 = NULL;
          PyObject *__pyx_t_17 = NULL;
          PyObject *__pyx_t_18 = NULL;
          int __pyx_t_19;
          char const *__pyx_t_20;
```

```
        PyObject *__pyx_t_21 = NULL;
        PyObject *__pyx_t_22 = NULL;
        PyObject *__pyx_t_23 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__cinit__", 0);

        /* "PyClical.pyx":588
*        2{1}+3{2}
*        """
*        error_msg_prefix = "Cannot initialize clifford object from"             # <<<<<<<<
*        if ixt is None:
*            try:
*/
        __Pyx_INCREF(__pyx_kp_u_Cannot_initialize_clifford_objec);
        __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_initialize_clifford_objec;

        /* "PyClical.pyx":589
*        """
*        error_msg_prefix = "Cannot initialize clifford object from"
*        if ixt is None:             # <<<<<<<<
*            try:
*                if   isinstance(other, clifford):
*/
        __pyx_t_1 = (__pyx_v_ixt == Py_None);
        __pyx_t_2 = (__pyx_t_1 != 0);
        if (__pyx_t_2) {

          /* "PyClical.pyx":590
*        error_msg_prefix = "Cannot initialize clifford object from"
*        if ixt is None:
*            try:             # <<<<<<<<
*                if   isinstance(other, clifford):
*                    self.instance = new Clifford((<clifford>other).unwrap())
*/
          {
            __Pyx_PyThreadState_declare
            __Pyx_PyThreadState_assign
            __Pyx_ExceptionSave(&__pyx_t_3, &__pyx_t_4, &__pyx_t_5);
            __Pyx_XGOTREF(__pyx_t_3);
            __Pyx_XGOTREF(__pyx_t_4);
            __Pyx_XGOTREF(__pyx_t_5);
            /*try:*/ {

              /* "PyClical.pyx":591
*        if ixt is None:
*            try:
*                if   isinstance(other, clifford):             # <<<<<<<<
*                    self.instance = new Clifford((<clifford>other).unwrap())
*                elif isinstance(other, index_set):
*/
              __pyx_t_2 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyClical_clifford);
              __pyx_t_1 = (__pyx_t_2 != 0);
              if (__pyx_t_1) {

                /* "PyClical.pyx":592
*            try:
*                if   isinstance(other, clifford):
*                    self.instance = new Clifford((<clifford>other).unwrap())             # <<<<<<<<
*                elif isinstance(other, index_set):
*                    self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
*/
                try {
                  __pyx_t_6 = new Clifford(__pyx_f_8PyClical_8clifford_unwrap(((struct
      __pyx_obj_8PyClical_clifford *)__pyx_v_other)));
                } catch(...) {
                  __Pyx_CppExn2PyErr();
                  __PYX_ERR(0, 592, __pyx_L4_error)
                }
                __pyx_v_self->instance = __pyx_t_6;

                /* "PyClical.pyx":591
*        if ixt is None:
*            try:
*                if   isinstance(other, clifford):             # <<<<<<<<
*                    self.instance = new Clifford((<clifford>other).unwrap())
*                elif isinstance(other, index_set):
*/
                goto __pyx_L10;
              }

              /* "PyClical.pyx":593
*                if   isinstance(other, clifford):
*                    self.instance = new Clifford((<clifford>other).unwrap())
*                elif isinstance(other, index_set):             # <<<<<<<<
*                    self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
```

```
 *                  elif isinstance(other, numbers.Real):
 */
                __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyClical_index_set);
                __pyx_t_2 = (__pyx_t_1 != 0);
                if (__pyx_t_2) {

                  /* "PyClical.pyx":594
 *                      self.instance = new Clifford((<clifford>other).unwrap())
 *                  elif isinstance(other, index_set):
 *                      self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
# «««««««
 *                  elif isinstance(other, numbers.Real):
 *                      self.instance = new Clifford(<scalar_t>other)
 */
                  try {
                    __pyx_t_6 = new Clifford(__pyx_f_8PyClical_9index_set_unwrap(((struct
__pyx_obj_8PyClical_index_set *)__pyx_v_other)), ((scalar_t)1.0));
                  } catch(...) {
                    __Pyx_CppExn2PyErr();
                    __PYX_ERR(0, 594, __pyx_L4_error)
                  }
                  __pyx_v_self->instance = __pyx_t_6;

                  /* "PyClical.pyx":593
 *                  if   isinstance(other, clifford):
 *                      self.instance = new Clifford((<clifford>other).unwrap())
 *                  elif isinstance(other, index_set):              # «««««««
 *                      self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
 *                  elif isinstance(other, numbers.Real):
 */
                  goto __pyx_L10;
                }

                /* "PyClical.pyx":595
 *                  elif isinstance(other, index_set):
 *                      self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
 *                  elif isinstance(other, numbers.Real):              # «««««««
 *                      self.instance = new Clifford(<scalar_t>other)
 *                  elif isinstance(other, str):
 */
                __Pyx_GetModuleGlobalName(__pyx_t_7, __pyx_n_s_numbers); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 595, __pyx_L4_error)
                __Pyx_GOTREF(__pyx_t_7);
                __pyx_t_8 = __Pyx_PyObject_GetAttrStr(__pyx_t_7, __pyx_n_s_Real); if
 (unlikely(!__pyx_t_8)) __PYX_ERR(0, 595, __pyx_L4_error)
                __Pyx_GOTREF(__pyx_t_8);
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_8); if (unlikely(__pyx_t_2 ==
 ((int)-1))) __PYX_ERR(0, 595, __pyx_L4_error)
                __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                __pyx_t_1 = (__pyx_t_2 != 0);
                if (__pyx_t_1) {

                  /* "PyClical.pyx":596
 *                      self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
 *                  elif isinstance(other, numbers.Real):
 *                      self.instance = new Clifford(<scalar_t>other)              # «««««««
 *                  elif isinstance(other, str):
 *                      try:
 */
                  __pyx_t_9 = __pyx_PyFloat_AsDouble(__pyx_v_other); if (unlikely((__pyx_t_9 ==
 ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 596, __pyx_L4_error)
                  try {
                    __pyx_t_6 = new Clifford(((scalar_t)__pyx_t_9));
                  } catch(...) {
                    __Pyx_CppExn2PyErr();
                    __PYX_ERR(0, 596, __pyx_L4_error)
                  }
                  __pyx_v_self->instance = __pyx_t_6;

                  /* "PyClical.pyx":595
 *                  elif isinstance(other, index_set):
 *                      self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
 *                  elif isinstance(other, numbers.Real):              # «««««««
 *                      self.instance = new Clifford(<scalar_t>other)
 *                  elif isinstance(other, str):
 */
                  goto __pyx_L10;
                }

                /* "PyClical.pyx":597
 *                  elif isinstance(other, numbers.Real):
 *                      self.instance = new Clifford(<scalar_t>other)
 *                  elif isinstance(other, str):              # «««««««
 *                      try:
 *                          bother = other.encode("UTF-8")
 */
```

```
                __pyx_t_1 = PyUnicode_Check(__pyx_v_other);
                __pyx_t_2 = (__pyx_t_1 != 0);
                if (likely(__pyx_t_2)) {

                  /* "PyClical.pyx":598
 *                     self.instance = new Clifford(<scalar_t>other)
 *                 elif isinstance(other, str):
 *                     try:             # «««««««
 *                         bother = other.encode("UTF-8")
 *                         self.instance = new Clifford(<char *>bother)
 */
                  {
                    __Pyx_PyThreadState_declare
                    __Pyx_PyThreadState_assign
                    __Pyx_ExceptionSave(&__pyx_t_10, &__pyx_t_11, &__pyx_t_12);
                    __Pyx_XGOTREF(__pyx_t_10);
                    __Pyx_XGOTREF(__pyx_t_11);
                    __Pyx_XGOTREF(__pyx_t_12);
                    /*try:*/ {

                      /* "PyClical.pyx":599
 *                 elif isinstance(other, str):
 *                     try:
 *                         bother = other.encode("UTF-8")             # «««««««
 *                         self.instance = new Clifford(<char *>bother)
 *                     except RuntimeError:
 */
                      __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_v_other, __pyx_n_s_encode); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 599, __pyx_L11_error)
                      __Pyx_GOTREF(__pyx_t_7);
                      __pyx_t_13 = NULL;
                      if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_7))) {
                        __pyx_t_13 = PyMethod_GET_SELF(__pyx_t_7);
                        if (likely(__pyx_t_13)) {
                          PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_7);
                          __Pyx_INCREF(__pyx_t_13);
                          __Pyx_INCREF(function);
                          __Pyx_DECREF_SET(__pyx_t_7, function);
                        }
                      }
                      __pyx_t_8 = (__pyx_t_13) ? __Pyx_PyObject_Call2Args(__pyx_t_7, __pyx_t_13,
__pyx_kp_u_UTF_8) : __Pyx_PyObject_CallOneArg(__pyx_t_7, __pyx_kp_u_UTF_8);
                      __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
                      if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 599, __pyx_L11_error)
                      __Pyx_GOTREF(__pyx_t_8);
                      __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                      __pyx_v_bother = __pyx_t_8;
                      __pyx_t_8 = 0;

                      /* "PyClical.pyx":600
 *                     try:
 *                         bother = other.encode("UTF-8")
 *                         self.instance = new Clifford(<char *>bother)             # «««««««
 *                     except RuntimeError:
 *                         raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 */
                      __pyx_t_14 = __Pyx_PyObject_AsWritableString(__pyx_v_bother); if
(unlikely((!__pyx_t_14) && PyErr_Occurred())) __PYX_ERR(0, 600, __pyx_L11_error)
                      try {
                        __pyx_t_6 = new Clifford(((char *)__pyx_t_14));
                      } catch(...) {
                        __Pyx_CppExn2PyErr();
                        __PYX_ERR(0, 600, __pyx_L11_error)
                      }
                      __pyx_v_self->instance = __pyx_t_6;

                      /* "PyClical.pyx":598
 *                     self.instance = new Clifford(<scalar_t>other)
 *                 elif isinstance(other, str):
 *                     try:             # «««««««
 *                         bother = other.encode("UTF-8")
 *                         self.instance = new Clifford(<char *>bother)
 */
                    }
                    __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
                    __Pyx_XDECREF(__pyx_t_11); __pyx_t_11 = 0;
                    __Pyx_XDECREF(__pyx_t_12); __pyx_t_12 = 0;
                    goto __pyx_L16_try_end;
                    __pyx_L11_error:;
                    __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
                    __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;
                    __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;

                    /* "PyClical.pyx":601
 *                         bother = other.encode("UTF-8")
 *                         self.instance = new Clifford(<char *>bother)
 *                     except RuntimeError:             # «««««««
```

```
 *                     raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 *                 else:
 */
                    __pyx_t_15 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
                    if (__pyx_t_15) {
                      __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
                      if (__Pyx_GetException(&__pyx_t_8, &__pyx_t_7, &__pyx_t_13) < 0) __PYX_ERR(0, 601, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_8);
                      __Pyx_GOTREF(__pyx_t_7);
                      __Pyx_GOTREF(__pyx_t_13);

                      /* "PyClical.pyx":602
 *                         self.instance = new Clifford(<char *>bother)
 *                     except RuntimeError:
 *                         raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")           # <<<<<<<<<<<<<<
 *                 else:
 *                     raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 */
                      __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid_string); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 602, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_16);
                      __pyx_t_17 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_17)) __PYX_ERR(0, 602, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_17);
                      __pyx_t_18 = PyNumber_Add(__pyx_t_16, __pyx_t_17); if (unlikely(!__pyx_t_18)) __PYX_ERR(0, 602, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_18);
                      __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
                      __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
                      __pyx_t_17 = PyNumber_Add(__pyx_t_18, __pyx_kp_u_); if (unlikely(!__pyx_t_17)) __PYX_ERR(0, 602, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_17);
                      __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
                      __pyx_t_18 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_17); if (unlikely(!__pyx_t_18)) __PYX_ERR(0, 602, __pyx_L13_except_error)
                      __Pyx_GOTREF(__pyx_t_18);
                      __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
                      __Pyx_Raise(__pyx_t_18, 0, 0, 0);
                      __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
                      __PYX_ERR(0, 602, __pyx_L13_except_error)
                    }
                    goto __pyx_L13_except_error;
                    __pyx_L13_except_error:;

                    /* "PyClical.pyx":598
 *                     self.instance = new Clifford(<scalar_t>other)
 *                 elif isinstance(other, str):
 *                     try:             # <<<<<<<<<<<<<<
 *                         bother = other.encode("UTF-8")
 *                         self.instance = new Clifford(<char *>bother)
 */
                    __Pyx_XGIVEREF(__pyx_t_10);
                    __Pyx_XGIVEREF(__pyx_t_11);
                    __Pyx_XGIVEREF(__pyx_t_12);
                    __Pyx_ExceptionReset(__pyx_t_10, __pyx_t_11, __pyx_t_12);
                    goto __pyx_L4_error;
                    __pyx_L16_try_end:;
                  }

                  /* "PyClical.pyx":597
 *                 elif isinstance(other, numbers.Real):
 *                     self.instance = new Clifford(<scalar_t>other)
 *                 elif isinstance(other, str):             # <<<<<<<<<<<<<<
 *                     try:
 *                         bother = other.encode("UTF-8")
 */
                  goto __pyx_L10;
                }

                /* "PyClical.pyx":604
 *                         raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
 *                 else:
 *                     raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")             # <<<<<<<<<<<<<<
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " " + str(type(other))
 */
                /*else*/ {
                  __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u__2); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 604, __pyx_L4_error)
                  __Pyx_GOTREF(__pyx_t_13);
                  __pyx_t_7 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 604, __pyx_L4_error)
                  __Pyx_GOTREF(__pyx_t_7);
```

```
                    __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_t_7); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 604, __pyx_L4_error)
                    __Pyx_GOTREF(__pyx_t_8);
                    __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                    __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                    __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_kp_u_); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 604, __pyx_L4_error)
                    __Pyx_GOTREF(__pyx_t_7);
                    __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                    __pyx_t_8 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_7); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 604, __pyx_L4_error)
                    __Pyx_GOTREF(__pyx_t_8);
                    __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                    __Pyx_Raise(__pyx_t_8, 0, 0, 0);
                    __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                    __PYX_ERR(0, 604, __pyx_L4_error)
                  }
                  __pyx_L10:;

                  /* "PyClical.pyx":590
 *         error_msg_prefix = "Cannot initialize clifford object from"
 *         if ixt is None:
 *             try:             # «««««««
 *                 if   isinstance(other, clifford):
 *                     self.instance = new Clifford((<clifford>other).unwrap())
 */
                }
                __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
                __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                goto __pyx_L9_try_end;
                __pyx_L4_error:;
                __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
                __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
                __Pyx_XDECREF(__pyx_t_17); __pyx_t_17 = 0;
                __Pyx_XDECREF(__pyx_t_18); __pyx_t_18 = 0;
                __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;
                __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;

                /* "PyClical.pyx":605
 *                 else:
 *                     raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *             except RuntimeError as err:             # «««««««
 *                 raise ValueError(error_msg_prefix + " " + str(type(other))
 *                                                   + " value " + repr(other) + ":"
 */
                __pyx_t_15 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
                if (__pyx_t_15) {
                  __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  if (__Pyx_GetException(&__pyx_t_8, &__pyx_t_7, &__pyx_t_13) < 0) __PYX_ERR(0, 605, __pyx_L6_except_error)
                  __Pyx_GOTREF(__pyx_t_8);
                  __Pyx_GOTREF(__pyx_t_7);
                  __Pyx_GOTREF(__pyx_t_13);
                  __Pyx_INCREF(__pyx_t_7);
                  __pyx_v_err = __pyx_t_7;
                  /*try:*/ {

                    /* "PyClical.pyx":606
 *                     raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " " + str(type(other))             # «««««««
 *                                                   + " value " + repr(other) + ":"
 *                                                   + "\n\t" + str(err))
 */
                    __pyx_t_18 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u__2); if (unlikely(!__pyx_t_18)) __PYX_ERR(0, 606, __pyx_L24_error)
                    __Pyx_GOTREF(__pyx_t_18);
                    __pyx_t_17 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_17)) __PYX_ERR(0, 606, __pyx_L24_error)
                    __Pyx_GOTREF(__pyx_t_17);
                    __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_t_18, __pyx_t_17); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 606, __pyx_L24_error)
                    __Pyx_GOTREF(__pyx_t_16);
                    __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
                    __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;

                    /* "PyClical.pyx":607
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " " + str(type(other))
 *                                                   + " value " + repr(other) + ":"             # «««««««
 *                                                   + "\n\t" + str(err))
 *         elif isinstance(ixt, index_set):
 */
                    __pyx_t_17 = __Pyx_PyUnicode_Concat(__pyx_t_16, __pyx_kp_u_value); if
```

```
(unlikely(!__pyx_t_17)) __PYX_ERR(0, 607, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_17);
                  __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
                  __pyx_t_16 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 607, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_16);
                  __pyx_t_18 = PyNumber_Add(__pyx_t_17, __pyx_t_16); if (unlikely(!__pyx_t_18)) __PYX_ERR(0, 607, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_18);
                  __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
                  __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
                  __pyx_t_16 = PyNumber_Add(__pyx_t_18, __pyx_kp_u__5); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 607, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_16);
                  __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;

                  /* "PyClical.pyx":608
 *                    raise ValueError(error_msg_prefix + " " + str(type(other))
 *                                                      + " value " + repr(other) + ":"
 *                                                      + "\n\t" + str(err))             # <<<<<<<<
 *            elif isinstance(ixt, index_set):
 *                if   isinstance(other, numbers.Real):
 */
                  __pyx_t_18 = PyNumber_Add(__pyx_t_16, __pyx_kp_u__6); if (unlikely(!__pyx_t_18)) __PYX_ERR(0, 608, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_18);
                  __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
                  __pyx_t_16 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), __pyx_v_err); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 608, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_16);
                  __pyx_t_17 = PyNumber_Add(__pyx_t_18, __pyx_t_16); if (unlikely(!__pyx_t_17)) __PYX_ERR(0, 608, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_17);
                  __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
                  __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;

                  /* "PyClical.pyx":606
 *                        raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *                except RuntimeError as err:
 *                    raise ValueError(error_msg_prefix + " " + str(type(other))             # <<<<<<<<
 *                                                      + " value " + repr(other) + ":"
 *                                                      + "\n\t" + str(err))
 */
                  __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_17); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 606, __pyx_L24_error)
                  __Pyx_GOTREF(__pyx_t_16);
                  __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
                  __Pyx_Raise(__pyx_t_16, 0, 0, 0);
                  __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
                  __PYX_ERR(0, 606, __pyx_L24_error)
                }

                /* "PyClical.pyx":605
 *                    else:
 *                        raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
 *                except RuntimeError as err:             # <<<<<<<<
 *                    raise ValueError(error_msg_prefix + " " + str(type(other))
 *                                                      + " value " + repr(other) + ":"
 */
                /*finally:*/ {
                  __pyx_L24_error:;
                  /*exception exit:*/{
                    __Pyx_PyThreadState_declare
                    __Pyx_PyThreadState_assign
                    __pyx_t_12 = 0; __pyx_t_11 = 0; __pyx_t_10 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0; __pyx_t_23 = 0;
                    __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
                    __Pyx_XDECREF(__pyx_t_17); __pyx_t_17 = 0;
                    __Pyx_XDECREF(__pyx_t_18); __pyx_t_18 = 0;
                    if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_21, &__pyx_t_22, &__pyx_t_23);
                    if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_12, &__pyx_t_11, &__pyx_t_10) < 0)) __Pyx_ErrFetch(&__pyx_t_12, &__pyx_t_11, &__pyx_t_10);
                    __Pyx_XGOTREF(__pyx_t_12);
                    __Pyx_XGOTREF(__pyx_t_11);
                    __Pyx_XGOTREF(__pyx_t_10);
                    __Pyx_XGOTREF(__pyx_t_21);
                    __Pyx_XGOTREF(__pyx_t_22);
                    __Pyx_XGOTREF(__pyx_t_23);
                    __pyx_t_15 = __pyx_lineno; __pyx_t_19 = __pyx_clineno; __pyx_t_20 = __pyx_filename;
                    {
                      __Pyx_DECREF(__pyx_v_err);
                      __pyx_v_err = NULL;
                    }
                    if (PY_MAJOR_VERSION >= 3) {
                      __Pyx_XGIVEREF(__pyx_t_21);
```

```
                            __Pyx_XGIVEREF(__pyx_t_22);
                            __Pyx_XGIVEREF(__pyx_t_23);
                            __Pyx_ExceptionReset(__pyx_t_21, __pyx_t_22, __pyx_t_23);
                          }
                          __Pyx_XGIVEREF(__pyx_t_12);
                          __Pyx_XGIVEREF(__pyx_t_11);
                          __Pyx_XGIVEREF(__pyx_t_10);
                          __Pyx_ErrRestore(__pyx_t_12, __pyx_t_11, __pyx_t_10);
                          __pyx_t_12 = 0; __pyx_t_11 = 0; __pyx_t_10 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0; __pyx_t_23 = 0;
                          __pyx_lineno = __pyx_t_15; __pyx_clineno = __pyx_t_19; __pyx_filename = __pyx_t_20;
                          goto __pyx_L6_except_error;
                        }
                      }
                    }
                    goto __pyx_L6_except_error;
                    __pyx_L6_except_error:;

                    /* "PyClical.pyx":590
 *         error_msg_prefix = "Cannot initialize clifford object from"
 *         if ixt is None:
 *             try:              # «««««««
 *                 if   isinstance(other, clifford):
 *                     self.instance = new Clifford((<clifford>other).unwrap())
 */
                    __Pyx_XGIVEREF(__pyx_t_3);
                    __Pyx_XGIVEREF(__pyx_t_4);
                    __Pyx_XGIVEREF(__pyx_t_5);
                    __Pyx_ExceptionReset(__pyx_t_3, __pyx_t_4, __pyx_t_5);
                    goto __pyx_L1_error;
                    __pyx_L9_try_end:;
                  }

                  /* "PyClical.pyx":589
 *         """
 *         error_msg_prefix = "Cannot initialize clifford object from"
 *         if ixt is None:             # «««««««
 *             try:
 *                 if   isinstance(other, clifford):
 */
                  goto __pyx_L3;
                }

                /* "PyClical.pyx":609
 *                                                 + " value " + repr(other) + ":"
 *                                                 + "\n\t" + str(err))
 *         elif isinstance(ixt, index_set):                # «««««««
 *             if   isinstance(other, numbers.Real):
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 */
                __pyx_t_2 = __Pyx_TypeCheck(__pyx_v_ixt, __pyx_ptype_8PyClical_index_set);
                __pyx_t_1 = (__pyx_t_2 != 0);
                if (likely(__pyx_t_1)) {

                  /* "PyClical.pyx":610
 *                                                 + "\n\t" + str(err))
 *         elif isinstance(ixt, index_set):
 *             if   isinstance(other, numbers.Real):             # «««««««
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 *             elif isinstance(other, collections.abc.Sequence):
 */
                  __Pyx_GetModuleGlobalName(__pyx_t_13, __pyx_n_s_numbers); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 610, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_13);
                  __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_t_13, __pyx_n_s_Real); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 610, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_7);
                  __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                  __pyx_t_1 = PyObject_IsInstance(__pyx_v_other, __pyx_t_7); if (unlikely(__pyx_t_1 == ((int)-1))) __PYX_ERR(0, 610, __pyx_L1_error)
                  __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                  __pyx_t_2 = (__pyx_t_1 != 0);
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":611
 *         elif isinstance(ixt, index_set):
 *             if   isinstance(other, numbers.Real):
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other) # «««««««
 *             elif isinstance(other, collections.abc.Sequence):
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
 */
                    __pyx_t_9 = __pyx_PyFloat_AsDouble(__pyx_v_other); if (unlikely((__pyx_t_9 == ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 611, __pyx_L1_error)
                    try {
                      __pyx_t_6 = new Clifford(__pyx_f_8PyClical_9index_set_unwrap(((struct
```

```
__pyx_obj_8PyClical_index_set *)__pyx_v_ixt)), ((scalar_t)__pyx_t_9));
              } catch(...) {
                __Pyx_CppExn2PyErr();
                __PYX_ERR(0, 611, __pyx_L1_error)
              }
              __pyx_v_self->instance = __pyx_t_6;

              /* "PyClical.pyx":610
 *                                                   + "\n\t" + str(err))
 *         elif isinstance(ixt, index_set):
 *             if   isinstance(other, numbers.Real):              # «««««««
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 *             elif isinstance(other, collections.abc.Sequence):
 */
              goto __pyx_L30;
            }

            /* "PyClical.pyx":612
 *             if   isinstance(other, numbers.Real):
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 *             elif isinstance(other, collections.abc.Sequence):             # «««««««
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
 *             else:
 */
            __Pyx_GetModuleGlobalName(__pyx_t_7, __pyx_n_s_collections); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 612, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_7);
            __pyx_t_13 = __Pyx_PyObject_GetAttrStr(__pyx_t_7, __pyx_n_s_abc); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 612, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_13);
            __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
            __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_t_13, __pyx_n_s_Sequence); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 612, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_7);
            __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
            __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_7); if (unlikely(__pyx_t_2 ==
((int)-1))) __PYX_ERR(0, 612, __pyx_L1_error)
            __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
            __pyx_t_1 = (__pyx_t_2 != 0);
            if (likely(__pyx_t_1)) {

              /* "PyClical.pyx":613
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 *             elif isinstance(other, collections.abc.Sequence):
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
# «««««««
 *             else:
 *                 raise TypeError(error_msg_prefix + " (" + str(type(other))
 */
              try {
                __pyx_t_6 = new Clifford(__pyx_f_8PyClical_list_to_vector(__pyx_v_other),
__pyx_f_8PyClical_9index_set_unwrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_ixt)));
              } catch(...) {
                __Pyx_CppExn2PyErr();
                __PYX_ERR(0, 613, __pyx_L1_error)
              }
              __pyx_v_self->instance = __pyx_t_6;

              /* "PyClical.pyx":612
 *             if   isinstance(other, numbers.Real):
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 *             elif isinstance(other, collections.abc.Sequence):             # «««««««
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
 *             else:
 */
              goto __pyx_L30;
            }

            /* "PyClical.pyx":615
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
 *             else:
 *                 raise TypeError(error_msg_prefix + " (" + str(type(other))              # «««««««
 *                                                  + ", " + repr(ixt) + ").")
 *         else:
 */
            /*else*/ {
              __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u__7); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 615, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_7);
              __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 615, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_13);
              __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_t_13); if (unlikely(!__pyx_t_8))
__PYX_ERR(0, 615, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_8);
              __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
              __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
```

```
                /* "PyClical.pyx":616
 *             else:
 *                 raise TypeError(error_msg_prefix + " (" + str(type(other))
 *                                                 + ", " + repr(ixt) + ").")             # <<<<<<<<
 *         else:
 *             raise TypeError(error_msg_prefix + " (" + str(type(other))
 */
                __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_kp_u__8); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 616, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_13);
                __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                __pyx_t_8 = PyObject_Repr(__pyx_v_ixt); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 616, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_8);
                __pyx_t_7 = PyNumber_Add(__pyx_t_13, __pyx_t_8); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 616, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_7);
                __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                __pyx_t_8 = PyNumber_Add(__pyx_t_7, __pyx_kp_u__9); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 616, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_8);
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;

                /* "PyClical.pyx":615
 *                 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
 *             else:
 *                 raise TypeError(error_msg_prefix + " (" + str(type(other))             # <<<<<<<<
 *                                                 + ", " + repr(ixt) + ").")
 *         else:
 */
                __pyx_t_7 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_8); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 615, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_7);
                __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
                __Pyx_Raise(__pyx_t_7, 0, 0, 0);
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                __PYX_ERR(0, 615, __pyx_L1_error)
              }
              __pyx_L30:;

              /* "PyClical.pyx":609
 *                                                  + " value " + repr(other) + ":"
 *                                                  + "\n\t" + str(err))
 *         elif isinstance(ixt, index_set):                 # <<<<<<<<
 *             if   isinstance(other, numbers.Real):
 *                 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
 */
              goto __pyx_L3;
            }

            /* "PyClical.pyx":618
 *                                                 + ", " + repr(ixt) + ").")
 *         else:
 *             raise TypeError(error_msg_prefix + " (" + str(type(other))             # <<<<<<<<
 *                                             + ", " + str(type(ixt)) + ").")
 *
 */
            /*else*/ {
              __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u__7); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 618, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_7);
              __pyx_t_8 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 618, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_8);
              __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_t_8); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 618, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_13);
              __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
              __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;

              /* "PyClical.pyx":619
 *         else:
 *             raise TypeError(error_msg_prefix + " (" + str(type(other))
 *                                             + ", " + str(type(ixt)) + ").")             # <<<<<<<<
 *
 *     def __dealloc__(self):
 */
              __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u__8); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 619, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_8);
              __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
              __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_ixt))); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 619, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_13);
              __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_t_13); if (unlikely(!__pyx_t_7))
```

```
__PYX_ERR(0, 619, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_7);
            __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
            __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
            __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_kp_u__9); if (unlikely(!__pyx_t_13))
__PYX_ERR(0, 619, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_13);
            __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;

            /* "PyClical.pyx":618
 *                                                + ", " + repr(ixt) + ").")
 *         else:
 *             raise TypeError(error_msg_prefix + " (" + str(type(other))            # «««««««
 *                                            + ", " + str(type(ixt)) + ").")
 *
 */
            __pyx_t_7 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_13); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 618, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_7);
            __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
            __Pyx_Raise(__pyx_t_7, 0, 0, 0);
            __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
            __PYX_ERR(0, 618, __pyx_L1_error)
          }
          __pyx_L3:;

          /* "PyClical.pyx":565
 *         return clifford(self)
 *
 *     def __cinit__(self, other = 0, ixt = None):             # «««««««
 *         """
 *         Construct an object of type clifford.
 */

          /* function exit code */
          __pyx_r = 0;
          goto __pyx_L0;
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_7);
          __Pyx_XDECREF(__pyx_t_8);
          __Pyx_XDECREF(__pyx_t_13);
          __Pyx_XDECREF(__pyx_t_16);
          __Pyx_XDECREF(__pyx_t_17);
          __Pyx_XDECREF(__pyx_t_18);
          __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = -1;
          __pyx_L0:;
          __Pyx_XDECREF(__pyx_v_error_msg_prefix);
          __Pyx_XDECREF(__pyx_v_bother);
          __Pyx_XDECREF(__pyx_v_err);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":621
 *                                            + ", " + str(type(ixt)) + ").")
 *
 *     def __dealloc__(self):              # «««««««
 *         """
 *         Clean up by deallocating the instance of C++ class Clifford.
 */

        /* Python wrapper */
        static void __pyx_pw_8PyClical_8clifford_5__dealloc__(PyObject *__pyx_v_self); /*proto*/
        static void __pyx_pw_8PyClical_8clifford_5__dealloc__(PyObject *__pyx_v_self) {
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__dealloc__ (wrapper)", 0);
          __pyx_pf_8PyClical_8clifford_4__dealloc__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
        }

        static void __pyx_pf_8PyClical_8clifford_4__dealloc__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__dealloc__", 0);

          /* "PyClical.pyx":625
 *         Clean up by deallocating the instance of C++ class Clifford.
 *         """
 *         del self.instance              # «««««««
 *
 *     def __contains__(self, x):
 */
```

```
        delete __pyx_v_self->instance;

        /* "PyClical.pyx":621
 *                                             + ", " + str(type(ixt)) + ").")
 *
 *     def __dealloc__(self):             # «««««««
 *         """
 *         Clean up by deallocating the instance of C++ class Clifford.
 */

        /* function exit code */
        __Pyx_RefNannyFinishContext();
      }

      /* "PyClical.pyx":627
 *         del self.instance
 *
 *     def __contains__(self, x):             # «««««««
 *         """
 *         Not applicable.
 */

      /* Python wrapper */
      static int __pyx_pw_8PyClical_8clifford_7__contains__(PyObject *__pyx_v_self, PyObject *__pyx_v_x); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_6__contains__[] = "\n        Not applicable.\n\n»> x=clifford(index_set({-3,4,7})); -3 in x\n        Traceback (most recent call last):\n...\n        TypeError: Not applicable.\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_6__contains__;
      #endif
      static int __pyx_pw_8PyClical_8clifford_7__contains__(PyObject *__pyx_v_self, PyObject *__pyx_v_x) {
        int __pyx_r;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__contains__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_6__contains__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_x));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static int __pyx_pf_8PyClical_8clifford_6__contains__(CYTHON_UNUSED struct __pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v_x) {
        int __pyx_r;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__contains__", 0);

        /* "PyClical.pyx":636
 *         TypeError: Not applicable.
 *         """
 *         raise TypeError("Not applicable.")             # «««««««
 *
 *     def __iter__(self):
 */
        __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__10, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 636, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __Pyx_Raise(__pyx_t_1, 0, 0, 0);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __PYX_ERR(0, 636, __pyx_L1_error)

        /* "PyClical.pyx":627
 *         del self.instance
 *
 *     def __contains__(self, x):             # «««««««
 *         """
 *         Not applicable.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__contains__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = -1;
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":638
```

```
 *         raise TypeError("Not applicable.")
 *
 *     def __iter__(self):             # «««««««
 *         """
 *         Not applicable.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_9__iter__(PyObject *__pyx_v_self); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_8__iter__[] = "\n        Not applicable.\n\n
»> for a in clifford(index_set({-3,4,7})):print(a, end=\",\")\n        Traceback (most recent call
last):\n          ...\n        TypeError: Not applicable.\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_8__iter__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_9__iter__(PyObject *__pyx_v_self) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__iter__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_8__iter__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_8__iter__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__iter__", 0);

          /* "PyClical.pyx":647
 *         TypeError: Not applicable.
 *         """
 *         raise TypeError("Not applicable.")             # «««««««
 *
 *     def reframe(self, ixt):
 */
          __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__10, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 647, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __Pyx_Raise(__pyx_t_1, 0, 0, 0);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __PYX_ERR(0, 647, __pyx_L1_error)

          /* "PyClical.pyx":638
 *         raise TypeError("Not applicable.")
 *
 *     def __iter__(self):             # «««««««
 *         """
 *         Not applicable.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.__iter__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = NULL;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":649
 *         raise TypeError("Not applicable.")
 *
 *     def reframe(self, ixt):             # «««««««
 *         """
 *         Put self into a larger frame, containing the union of self.frame() and index set ixt.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_11reframe(PyObject *__pyx_v_self, PyObject
*__pyx_v_ixt); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_10reframe[] = "\n        Put self into a larger
frame, containing the union of self.frame() and index set ixt.\n        This can be used to make
multiplication faster, by multiplying within a common frame.\n\n        »>
clifford(\"2+3{1}\").reframe(index_set({1,2,3}))\n        clifford(\"2+3{1}\")\n        »>
s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() == (s|t);\n
True\n        ";
```

```
static PyObject *__pyx_pw_8PyClical_8clifford_11reframe(PyObject *__pyx_v_self, PyObject *__pyx_v_ixt) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("reframe (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_10reframe(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_ixt));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_10reframe(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_ixt) {
  PyObject *__pyx_v_error_msg_prefix = NULL;
  struct __pyx_obj_8PyClical_clifford *__pyx_v_result = NULL;
  PyObject *__pyx_v_err = NULL;
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_t_2;
  PyObject *__pyx_t_3 = NULL;
  PyObject *__pyx_t_4 = NULL;
  PyObject *__pyx_t_5 = NULL;
  PyObject *__pyx_t_6 = NULL;
  Clifford *__pyx_t_7;
  int __pyx_t_8;
  PyObject *__pyx_t_9 = NULL;
  PyObject *__pyx_t_10 = NULL;
  PyObject *__pyx_t_11 = NULL;
  PyObject *__pyx_t_12 = NULL;
  PyObject *__pyx_t_13 = NULL;
  int __pyx_t_14;
  char const *__pyx_t_15;
  PyObject *__pyx_t_16 = NULL;
  PyObject *__pyx_t_17 = NULL;
  PyObject *__pyx_t_18 = NULL;
  PyObject *__pyx_t_19 = NULL;
  PyObject *__pyx_t_20 = NULL;
  PyObject *__pyx_t_21 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("reframe", 0);

  /* "PyClical.pyx":659
 *         True
 *         """
 *         error_msg_prefix = "Cannot reframe"             # <<<<<<<<<<<<<<
 *         if isinstance(ixt, index_set):
 *             try:
 */
  __Pyx_INCREF(__pyx_kp_u_Cannot_reframe);
  __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_reframe;

  /* "PyClical.pyx":660
 *         """
 *         error_msg_prefix = "Cannot reframe"
 *         if isinstance(ixt, index_set):             # <<<<<<<<<<<<<<
 *             try:
 *                 result = clifford()
 */
  __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_ixt, __pyx_ptype_8PyClical_index_set);
  __pyx_t_2 = (__pyx_t_1 != 0);
  if (likely(__pyx_t_2)) {

    /* "PyClical.pyx":661
 *         error_msg_prefix = "Cannot reframe"
 *         if isinstance(ixt, index_set):
 *             try:             # <<<<<<<<<<<<<<
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 */
    {
      __Pyx_PyThreadState_declare
      __Pyx_PyThreadState_assign
      __Pyx_ExceptionSave(&__pyx_t_3, &__pyx_t_4, &__pyx_t_5);
      __Pyx_XGOTREF(__pyx_t_3);
      __Pyx_XGOTREF(__pyx_t_4);
      __Pyx_XGOTREF(__pyx_t_5);
      /*try:*/ {

        /* "PyClical.pyx":662
 *         if isinstance(ixt, index_set):
 *             try:
 *                 result = clifford()             # <<<<<<<<<<<<<<
```

```
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 *             except RuntimeError as err:
 */
              __pyx_t_6 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 662, __pyx_L4_error)
              __Pyx_GOTREF(__pyx_t_6);
              __pyx_v_result = ((struct __pyx_obj_8PyClical_clifford *)__pyx_t_6);
              __pyx_t_6 = 0;

              /* "PyClical.pyx":663
 *             try:
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap()) # «««««««
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
 */
              try {
                __pyx_t_7 = new Clifford(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self), __pyx_f_8PyClical_9index_set_unwrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_ixt)));
              } catch(...) {
                __Pyx_CppExn2PyErr();
                __PYX_ERR(0, 663, __pyx_L4_error)
              }
              __pyx_v_result->instance = __pyx_t_7;

              /* "PyClical.pyx":661
 *         error_msg_prefix = "Cannot reframe"
 *         if isinstance(ixt, index_set):
 *             try:             # «««««««
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 */
            }
            __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
            __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
            __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
            goto __pyx_L9_try_end;
            __pyx_L4_error:;
            __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;

            /* "PyClical.pyx":664
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 *             except RuntimeError as err:             # «««««««
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
 *                                                      + str(ixt) + ":"
 */
            __pyx_t_8 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
            if (__pyx_t_8) {
              __Pyx_AddTraceback("PyClical.clifford.reframe", __pyx_clineno, __pyx_lineno, __pyx_filename);
              if (__Pyx_GetException(&__pyx_t_6, &__pyx_t_9, &__pyx_t_10) < 0) __PYX_ERR(0, 664, __pyx_L6_except_error)
              __Pyx_GOTREF(__pyx_t_6);
              __Pyx_GOTREF(__pyx_t_9);
              __Pyx_GOTREF(__pyx_t_10);
              __Pyx_INCREF(__pyx_t_9);
              __pyx_v_err = __pyx_t_9;
              /*try:*/ {

                /* "PyClical.pyx":665
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame " # «««««««
 *                                                      + str(ixt) + ":"
 *                                                      + "\n\t" + str(err))
 */
                __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_from); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 665, __pyx_L15_error)
                __Pyx_GOTREF(__pyx_t_11);
                __pyx_t_12 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 665, __pyx_L15_error)
                __Pyx_GOTREF(__pyx_t_12);
                __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_t_12); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 665, __pyx_L15_error)
                __Pyx_GOTREF(__pyx_t_13);
                __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;
                __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
                __pyx_t_12 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u_to_frame); if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 665, __pyx_L15_error)
                __Pyx_GOTREF(__pyx_t_12);
                __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;

                /* "PyClical.pyx":666
 *             except RuntimeError as err:
```

```
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
 *                                                              + str(ixt) + ":"            # «««««««
 *                                                              + "\n\t" + str(err))
 *         else:
 */
                 __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), __pyx_v_ixt); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 666, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_13);
                 __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_t_12, __pyx_t_13); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 666, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_11);
                 __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
                 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_kp_u__5); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 666, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_13);
                 __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;

                 /* "PyClical.pyx":667
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
 *                                                              + str(ixt) + ":"
 *                                                              + "\n\t" + str(err))            # «««««««
 *         else:
 *             raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
 */
                 __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u__6); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 667, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_11);
                 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                 __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), __pyx_v_err); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 667, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_13);
                 __pyx_t_12 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_t_13); if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 667, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_12);
                 __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;
                 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;

                 /* "PyClical.pyx":665
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 *             except RuntimeError as err:
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame " # «««««««
 *                                                              + str(ixt) + ":"
 *                                                              + "\n\t" + str(err))
 */
                 __pyx_t_13 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_12); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 665, __pyx_L15_error)
                 __Pyx_GOTREF(__pyx_t_13);
                 __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
                 __Pyx_Raise(__pyx_t_13, 0, 0, 0);
                 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                 __PYX_ERR(0, 665, __pyx_L15_error)
               }

               /* "PyClical.pyx":664
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 *             except RuntimeError as err:             # «««««««
 *                 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
 *                                                              + str(ixt) + ":"
 */
               /*finally:*/ {
                 __pyx_L15_error:;
                 /*exception exit:*/{
                   __Pyx_PyThreadState_declare
                   __Pyx_PyThreadState_assign
                   __pyx_t_16 = 0; __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0;
                   __Pyx_XDECREF(__pyx_t_11); __pyx_t_11 = 0;
                   __Pyx_XDECREF(__pyx_t_12); __pyx_t_12 = 0;
                   __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
                   if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_19, &__pyx_t_20, &__pyx_t_21);
                   if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_16, &__pyx_t_17, &__pyx_t_18) < 0)) __Pyx_ErrFetch(&__pyx_t_16, &__pyx_t_17, &__pyx_t_18);
                   __Pyx_XGOTREF(__pyx_t_16);
                   __Pyx_XGOTREF(__pyx_t_17);
                   __Pyx_XGOTREF(__pyx_t_18);
                   __Pyx_XGOTREF(__pyx_t_19);
                   __Pyx_XGOTREF(__pyx_t_20);
                   __Pyx_XGOTREF(__pyx_t_21);
                   __pyx_t_8 = __pyx_lineno; __pyx_t_14 = __pyx_clineno; __pyx_t_15 = __pyx_filename;
                   {
                     __Pyx_DECREF(__pyx_v_err);
                     __pyx_v_err = NULL;
```

```
                    }
                    if (PY_MAJOR_VERSION >= 3) {
                      __Pyx_XGIVEREF(__pyx_t_19);
                      __Pyx_XGIVEREF(__pyx_t_20);
                      __Pyx_XGIVEREF(__pyx_t_21);
                      __Pyx_ExceptionReset(__pyx_t_19, __pyx_t_20, __pyx_t_21);
                    }
                    __Pyx_XGIVEREF(__pyx_t_16);
                    __Pyx_XGIVEREF(__pyx_t_17);
                    __Pyx_XGIVEREF(__pyx_t_18);
                    __Pyx_ErrRestore(__pyx_t_16, __pyx_t_17, __pyx_t_18);
                    __pyx_t_16 = 0; __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0;
                    __pyx_lineno = __pyx_t_8; __pyx_clineno = __pyx_t_14; __pyx_filename = __pyx_t_15;
                    goto __pyx_L6_except_error;
                  }
                }
              }
              goto __pyx_L6_except_error;
              __pyx_L6_except_error:;

              /* "PyClical.pyx":661
 *         error_msg_prefix = "Cannot reframe"
 *         if isinstance(ixt, index_set):
 *             try:             # <<<<<<<<<<<<<<
 *                 result = clifford()
 *                 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
 */
              __Pyx_XGIVEREF(__pyx_t_3);
              __Pyx_XGIVEREF(__pyx_t_4);
              __Pyx_XGIVEREF(__pyx_t_5);
              __Pyx_ExceptionReset(__pyx_t_3, __pyx_t_4, __pyx_t_5);
              goto __pyx_L1_error;
              __pyx_L9_try_end:;
            }

            /* "PyClical.pyx":660
 *         """
 *         error_msg_prefix = "Cannot reframe"
 *         if isinstance(ixt, index_set):             # <<<<<<<<<<<<<<
 *             try:
 *                 result = clifford()
 */
            goto __pyx_L3;
          }

          /* "PyClical.pyx":669
 *                                                               + "\n\t" + str(err))
 *         else:
 *             raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")             # <<<<<<<<<<<<<<
 *         return result
 *
 */
          /*else*/ {
            __pyx_t_10 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_using); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 669, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_10);
            __pyx_t_9 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_ixt))); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 669, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_9);
            __pyx_t_6 = __Pyx_PyUnicode_Concat(__pyx_t_10, __pyx_t_9); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 669, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_6);
            __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
            __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
            __pyx_t_9 = __Pyx_PyUnicode_Concat(__pyx_t_6, __pyx_kp_u__9); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 669, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_9);
            __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
            __pyx_t_6 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_9); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 669, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_6);
            __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
            __Pyx_Raise(__pyx_t_6, 0, 0, 0);
            __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
            __PYX_ERR(0, 669, __pyx_L1_error)
          }
          __pyx_L3:;

          /* "PyClical.pyx":670
 *         else:
 *             raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
 *         return result             # <<<<<<<<<<<<<<
 *
 *     def __richcmp__(lhs, rhs, int op):
 */
```

```
        __Pyx_XDECREF(__pyx_r);
        __Pyx_INCREF(((PyObject *)__pyx_v_result));
        __pyx_r = ((PyObject *)__pyx_v_result);
        goto __pyx_L0;

        /* "PyClical.pyx":649
 *         raise TypeError("Not applicable.")
 *
 *     def reframe(self, ixt):             # <<<<<<<<<<<<<<
 *         """
 *         Put self into a larger frame, containing the union of self.frame() and index set ixt.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_6);
        __Pyx_XDECREF(__pyx_t_9);
        __Pyx_XDECREF(__pyx_t_10);
        __Pyx_XDECREF(__pyx_t_11);
        __Pyx_XDECREF(__pyx_t_12);
        __Pyx_XDECREF(__pyx_t_13);
        __Pyx_AddTraceback("PyClical.clifford.reframe", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XDECREF(__pyx_v_error_msg_prefix);
        __Pyx_XDECREF((PyObject *)__pyx_v_result);
        __Pyx_XDECREF(__pyx_v_err);
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":672
 *         return result
 *
 *     def __richcmp__(lhs, rhs, int op):             # <<<<<<<<<<<<<<
 *         """
 *         Compare objects of type clifford.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_13__richcmp__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, int __pyx_v_op); /*proto*/
      static PyObject *__pyx_pw_8PyClical_8clifford_13__richcmp__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, int __pyx_v_op) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__richcmp__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_12__richcmp__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs), ((int)__pyx_v_op));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_12__richcmp__(struct
__pyx_obj_8PyClical_clifford *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        int __pyx_t_1;
        int __pyx_t_2;
        int __pyx_t_3;
        PyObject *__pyx_t_4 = NULL;
        PyObject *__pyx_t_5 = NULL;
        PyObject *__pyx_t_6 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__richcmp__", 0);

        /* "PyClical.pyx":691
 *         True
 *         """
 *         if op == 2: # ==             # <<<<<<<<<<<<<<
 *             if (lhs is None) or (rhs is None):
 *                 return bool(lhs is rhs)
 */
        __pyx_t_1 = ((__pyx_v_op == 2) != 0);
        if (__pyx_t_1) {

          /* "PyClical.pyx":692
 *         """
 *         if op == 2: # ==
 *             if (lhs is None) or (rhs is None):             # <<<<<<<<<<<<<<
 *                 return bool(lhs is rhs)
```

```
 *             else:
 */
            __pyx_t_2 = (((PyObject *)__pyx_v_lhs) == Py_None);
            __pyx_t_3 = (__pyx_t_2 != 0);
            if (!__pyx_t_3) {
            } else {
              __pyx_t_1 = __pyx_t_3;
              goto __pyx_L5_bool_binop_done;
            }
            __pyx_t_3 = (__pyx_v_rhs == Py_None);
            __pyx_t_2 = (__pyx_t_3 != 0);
            __pyx_t_1 = __pyx_t_2;
            __pyx_L5_bool_binop_done:;
            if (__pyx_t_1) {

              /* "PyClical.pyx":693
 *         if op == 2: # ==
 *             if (lhs is None) or (rhs is None):
 *                 return bool(lhs is rhs)             # «««««««
 *             else:
 *                 return bool( toClifford(lhs) == toClifford(rhs) )
 */
              __Pyx_XDECREF(__pyx_r);
              __pyx_t_1 = (((PyObject *)__pyx_v_lhs) == __pyx_v_rhs);
              __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 693, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_4);
              __pyx_r = __pyx_t_4;
              __pyx_t_4 = 0;
              goto __pyx_L0;

              /* "PyClical.pyx":692
 *         """
 *         if op == 2: # ==
 *             if (lhs is None) or (rhs is None):             # «««««««
 *                 return bool(lhs is rhs)
 *             else:
 */
            }

            /* "PyClical.pyx":695
 *                 return bool(lhs is rhs)
 *             else:
 *                 return bool( toClifford(lhs) == toClifford(rhs) )             # «««««««
 *         elif op == 3: # !=
 *             if (lhs is None) or (rhs is None):
 */
            /*else*/ {
              __Pyx_XDECREF(__pyx_r);
              __pyx_t_1 = (__pyx_f_8PyClical_toClifford(((PyObject *)__pyx_v_lhs)) ==
__pyx_f_8PyClical_toClifford(__pyx_v_rhs));
              __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 695, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_4);
              __pyx_r = __pyx_t_4;
              __pyx_t_4 = 0;
              goto __pyx_L0;
            }

            /* "PyClical.pyx":691
 *         True
 *         """
 *         if op == 2: # ==             # «««««««
 *             if (lhs is None) or (rhs is None):
 *                 return bool(lhs is rhs)
 */
          }

          /* "PyClical.pyx":696
 *             else:
 *                 return bool( toClifford(lhs) == toClifford(rhs) )
 *         elif op == 3: # !=             # «««««««
 *             if (lhs is None) or (rhs is None):
 *                 return not bool(lhs is rhs)
 */
          __pyx_t_1 = ((__pyx_v_op == 3) != 0);
          if (__pyx_t_1) {

            /* "PyClical.pyx":697
 *                 return bool( toClifford(lhs) == toClifford(rhs) )
 *         elif op == 3: # !=
 *             if (lhs is None) or (rhs is None):             # «««««««
 *                 return not bool(lhs is rhs)
 *             else:
 */
            __pyx_t_2 = (((PyObject *)__pyx_v_lhs) == Py_None);
            __pyx_t_3 = (__pyx_t_2 != 0);
```

```
          if (!__pyx_t_3) {
          } else {
            __pyx_t_1 = __pyx_t_3;
            goto __pyx_L8_bool_binop_done;
          }
          __pyx_t_3 = (__pyx_v_rhs == Py_None);
          __pyx_t_2 = (__pyx_t_3 != 0);
          __pyx_t_1 = __pyx_t_2;
          __pyx_L8_bool_binop_done:;
          if (__pyx_t_1) {

            /* "PyClical.pyx":698
 *         elif op == 3: # !=
 *             if (lhs is None) or (rhs is None):
 *                 return not bool(lhs is rhs)             # <<<<<<<<<<<<<<
 *             else:
 *                 return bool( toClifford(lhs) != toClifford(rhs) )
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = (((PyObject *)__pyx_v_lhs) == __pyx_v_rhs);
            __pyx_t_4 = __Pyx_PyBool_FromLong((!((!(!__pyx_t_1)) != 0))); if (unlikely(!__pyx_t_4))
  __PYX_ERR(0, 698, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_4);
            __pyx_r = __pyx_t_4;
            __pyx_t_4 = 0;
            goto __pyx_L0;

            /* "PyClical.pyx":697
 *                 return bool( toClifford(lhs) == toClifford(rhs) )
 *         elif op == 3: # !=
 *             if (lhs is None) or (rhs is None):             # <<<<<<<<<<<<<<
 *                 return not bool(lhs is rhs)
 *             else:
 */
          }

          /* "PyClical.pyx":700
 *                 return not bool(lhs is rhs)
 *             else:
 *                 return bool( toClifford(lhs) != toClifford(rhs) )             # <<<<<<<<<<<<<<
 *         elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
 *             raise TypeError("This comparison operator is not implemented for "
 */
          /*else*/ {
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = (__pyx_f_8PyClical_toClifford(((PyObject *)__pyx_v_lhs)) !=
  __pyx_f_8PyClical_toClifford(__pyx_v_rhs));
            __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4))
  __PYX_ERR(0, 700, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_4);
            __pyx_r = __pyx_t_4;
            __pyx_t_4 = 0;
            goto __pyx_L0;
          }

          /* "PyClical.pyx":696
 *             else:
 *                 return bool( toClifford(lhs) == toClifford(rhs) )
 *         elif op == 3: # !=             # <<<<<<<<<<<<<<
 *             if (lhs is None) or (rhs is None):
 *                 return not bool(lhs is rhs)
 */
        }

        /* "PyClical.pyx":701
 *             else:
 *                 return bool( toClifford(lhs) != toClifford(rhs) )
 *         elif isinstance(lhs, clifford) or isinstance(rhs, clifford):             # <<<<<<<<<<<<<<
 *             raise TypeError("This comparison operator is not implemented for "
 *                             + str(type(lhs)) + ", " + str(type(rhs)) + ".")
 */
        __pyx_t_2 = __Pyx_TypeCheck(((PyObject *)__pyx_v_lhs), __pyx_ptype_8PyClical_clifford);
        __pyx_t_3 = (__pyx_t_2 != 0);
        if (!__pyx_t_3) {
        } else {
          __pyx_t_1 = __pyx_t_3;
          goto __pyx_L10_bool_binop_done;
        }
        __pyx_t_3 = __Pyx_TypeCheck(__pyx_v_rhs, __pyx_ptype_8PyClical_clifford);
        __pyx_t_2 = (__pyx_t_3 != 0);
        __pyx_t_1 = __pyx_t_2;
        __pyx_L10_bool_binop_done:;
        if (unlikely(__pyx_t_1)) {

          /* "PyClical.pyx":703
 *         elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
 *             raise TypeError("This comparison operator is not implemented for "
```

```
 *                              + str(type(lhs)) + ", " + str(type(rhs)) + ".")              # «««««««
 *         else:
 *             return NotImplemented
 */
            __pyx_t_4 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(((PyObject *)__pyx_v_lhs)))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_4);
            __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_kp_u_This_comparison_operator_is_not, __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_5);
            __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
            __pyx_t_4 = __Pyx_PyUnicode_Concat(__pyx_t_5, __pyx_kp_u__8); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_4);
            __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
            __pyx_t_5 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)Py_TYPE(__pyx_v_rhs))); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_5);
            __pyx_t_6 = __Pyx_PyUnicode_Concat(__pyx_t_4, __pyx_t_5); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_6);
            __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
            __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
            __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_t_6, __pyx_kp_u_); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 703, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_5);
            __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;

            /* "PyClical.pyx":702
 *                 return bool( toClifford(lhs) != toClifford(rhs) )
 *         elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
 *             raise TypeError("This comparison operator is not implemented for "             # «««««««
 *                              + str(type(lhs)) + ", " + str(type(rhs)) + ".")
 *         else:
 */
            __pyx_t_6 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_5); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 702, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_6);
            __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
            __Pyx_Raise(__pyx_t_6, 0, 0, 0);
            __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
            __PYX_ERR(0, 702, __pyx_L1_error)

            /* "PyClical.pyx":701
 *             else:
 *                 return bool( toClifford(lhs) != toClifford(rhs) )
 *         elif isinstance(lhs, clifford) or isinstance(rhs, clifford):              # «««««««
 *             raise TypeError("This comparison operator is not implemented for "
 *                              + str(type(lhs)) + ", " + str(type(rhs)) + ".")
 */
          }

          /* "PyClical.pyx":705
 *                              + str(type(lhs)) + ", " + str(type(rhs)) + ".")
 *         else:
 *             return NotImplemented             # «««««««
 *
 *     def __getitem__(self, ixt):
 */
          /*else*/ {
            __Pyx_XDECREF(__pyx_r);
            __Pyx_INCREF(__pyx_builtin_NotImplemented);
            __pyx_r = __pyx_builtin_NotImplemented;
            goto __pyx_L0;
          }

          /* "PyClical.pyx":672
 *         return result
 *
 *     def __richcmp__(lhs, rhs, int op):             # «««««««
 *         """
 *         Compare objects of type clifford.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_4);
          __Pyx_XDECREF(__pyx_t_5);
          __Pyx_XDECREF(__pyx_t_6);
          __Pyx_AddTraceback("PyClical.clifford.__richcmp__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
```

```
        }

        /* "PyClical.pyx":707
 *             return NotImplemented
 *
 *     def __getitem__(self, ixt):              # «««««««
 *         """
 *         Subscripting: map from index set to scalar coordinate.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_15__getitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_ixt); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_14__getitem__[] = "\n        Subscripting: map from index set to scalar coordinate.\n\n        »> clifford(\"{1}\")[index_set(1)]\n        1.0\n        »> clifford(\"{1}\")[index_set({1})]\n        1.0\n        »> clifford(\"{1}\")[index_set({1,2})]\n        0.0\n        »> clifford(\"2{1,2}\")[index_set({1,2})]\n        2.0\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_14__getitem__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_15__getitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_ixt) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__getitem__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_14__getitem__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_ixt));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_14__getitem__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_ixt) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__getitem__", 0);

          /* "PyClical.pyx":720
 *         2.0
 *         """
 *         return self.instance.getitem(toIndexSet(ixt))              # «««««««
 *
 *     def __neg__(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->operator[](__pyx_f_8PyClical_toIndexSet(__pyx_v_ixt))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 720, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":707
 *             return NotImplemented
 *
 *     def __getitem__(self, ixt):              # «««««««
 *         """
 *         Subscripting: map from index set to scalar coordinate.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.__getitem__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":722
 *         return self.instance.getitem(toIndexSet(ixt))
 *
 *     def __neg__(self):              # «««««««
 *         """
 *         Unary -.
 */
```

```
/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_17__neg__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_8clifford_16__neg__[] = "\n        Unary -.\n\n        >>>
print(-clifford(\"{1}\"))\n        -{1}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_16__neg__;
#endif
static PyObject *__pyx_pw_8PyClical_8clifford_17__neg__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__neg__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_16__neg__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_16__neg__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__neg__", 0);

  /* "PyClical.pyx":729
 *         -{1}
 *         """
 *         return clifford().wrap( self.instance.neg() )             # <<<<<<<<<<<<<<
 *
 *     def __pos__(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 729, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->operator-()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 729,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_r = __pyx_t_2;
  __pyx_t_2 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":722
 *         return self.instance.getitem(toIndexSet(ixt))
 *
 *     def __neg__(self):             # <<<<<<<<<<<<<<
 *         """
 *         Unary -.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_AddTraceback("PyClical.clifford.__neg__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":731
 *         return clifford().wrap( self.instance.neg() )
 *
 *     def __pos__(self):             # <<<<<<<<<<<<<<
 *         """
 *         Unary +.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_19__pos__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_8clifford_18__pos__[] = "\n        Unary +.\n\n        >>>
print(+clifford(\"{1}\"))\n        {1}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_18__pos__;
#endif
static PyObject *__pyx_pw_8PyClical_8clifford_19__pos__(PyObject *__pyx_v_self) {
```

```
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__pos__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_18__pos__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_18__pos__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__pos__", 0);

        /* "PyClical.pyx":738
 *         {1}
 *         """
 *         return clifford(self)             # «««««<
 *
 *     def __add__(lhs, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), ((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 738, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":731
 *         return clifford().wrap( self.instance.neg() )
 *
 *     def __pos__(self):             # «««««<
 *         """
 *         Unary +.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__pos__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":740
 *         return clifford(self)
 *
 *     def __add__(lhs, rhs):             # «««««<
 *         """
 *         Geometric sum.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_21__add__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_20__add__[] = "\n        Geometric sum.\n\n        »> print(clifford(1) + clifford(\"{2}\"))\n        1+{2}\n        »> print(clifford(\"{1}\") + clifford(\"{2}\"))\n        {1}+{2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_20__add__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_21__add__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__add__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_20__add__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_20__add__(PyObject *__pyx_v_lhs, PyObject
```

```
*__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__add__", 0);

        /* "PyClical.pyx":749
 *         {1}+{2}
 *         """
 *         return clifford().wrap( toClifford(lhs) + toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __iadd__(self, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 749, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) +
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 749,
__pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":740
 *         return clifford(self)
 *
 *     def __add__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric sum.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.clifford.__add__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":751
 *         return clifford().wrap( toClifford(lhs) + toClifford(rhs) )
 *
 *     def __iadd__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric sum.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_23__iadd__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_22__iadd__[] = "\n        Geometric sum.\n\n
>>> x = clifford(1); x += clifford(\"{2}\"); print(x)\n        1+{2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_22__iadd__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_23__iadd__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__iadd__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_22__iadd__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_22__iadd__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
```

```
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("__iadd__", 0);

    /* "PyClical.pyx":758
 *         1+{2}
 *         """
 *         return self.wrap( self.unwrap() + toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __sub__(lhs, rhs):
 */
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) + __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 758, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_r = __pyx_t_1;
    __pyx_t_1 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":751
 *         return clifford().wrap( toClifford(lhs) + toClifford(rhs) )
 *
 *     def __iadd__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric sum.
 */

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_AddTraceback("PyClical.clifford.__iadd__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":760
 *         return self.wrap( self.unwrap() + toClifford(rhs) )
 *
 *     def __sub__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric difference.
 */

  /* Python wrapper */
  static PyObject *__pyx_pw_8PyClical_8clifford_25__sub__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs); /*proto*/
  static char __pyx_doc_8PyClical_8clifford_24__sub__[] = "\n        Geometric difference.\n\n
>>> print(clifford(1) - clifford(\"{2}\"))\n        1-{2}\n        >>> print(clifford(\"{1}\") -
clifford(\"{2}\"))\n        {1}-{2}\n        ";
  #if CYTHON_COMPILING_IN_CPYTHON
  struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_24__sub__;
  #endif
  static PyObject *__pyx_pw_8PyClical_8clifford_25__sub__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("__sub__ (wrapper)", 0);
    __pyx_r = __pyx_pf_8PyClical_8clifford_24__sub__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_8clifford_24__sub__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    PyObject *__pyx_t_2 = NULL;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("__sub__", 0);

    /* "PyClical.pyx":769
 *         {1}-{2}
 *         """
 *         return clifford().wrap( toClifford(lhs) - toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __isub__(self, rhs):
```

```
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 769, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) - __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 769, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":760
 *         return self.wrap( self.unwrap() + toClifford(rhs) )
 *
 *     def __sub__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric difference.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.clifford.__sub__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":771
 *         return clifford().wrap( toClifford(lhs) - toClifford(rhs) )
 *
 *     def __isub__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric difference.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_27__isub__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_26__isub__[] = "\n        Geometric difference.\n\n >>> x = clifford(1); x -= clifford(\"{2}\"); print(x)\n        1-{2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_26__isub__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_27__isub__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__isub__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_26__isub__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_26__isub__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__isub__", 0);

        /* "PyClical.pyx":778
 *         1-{2}
 *         """
 *         return self.wrap( self.unwrap() - toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __mul__(lhs, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self, (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) - __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 778, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
```

```
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":771
 *         return clifford().wrap( toClifford(lhs) - toClifford(rhs) )
 *
 *     def __isub__(self, rhs):             # «««««««
 *         """
 *         Geometric difference.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__isub__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":780
 *         return self.wrap( self.unwrap() - toClifford(rhs) )
 *
 *     def __mul__(lhs, rhs):             # «««««««
 *         """
 *         Geometric product.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_29__mul__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_28__mul__[] = "\n        Geometric product.\n\n        »> print(clifford(\"{1}\") * clifford(\"{2}\"))\n        {1,2}\n        »> print(clifford(2) * clifford(\"{2}\"))\n        2{2}\n        »> print(clifford(\"{1}\") * clifford(\"{1,2}\"))\n        {2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_28__mul__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_29__mul__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__mul__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_28__mul__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_28__mul__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__mul__", 0);

        /* "PyClical.pyx":791
 *         {2}
 *         """
 *         return clifford().wrap( toClifford(lhs) * toClifford(rhs) )             # «««««««
 *
 *     def __imul__(self, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 791, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) * __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 791, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

```

```
        /* "PyClical.pyx":780
 *         return self.wrap( self.unwrap() - toClifford(rhs) )
 *
 *     def __mul__(lhs, rhs):             # «««««««
 *         """
 *         Geometric product.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.clifford.__mul__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":793
 *         return clifford().wrap( toClifford(lhs) * toClifford(rhs) )
 *
 *     def __imul__(self, rhs):             # «««««««
 *         """
 *         Geometric product.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_31__imul__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_30__imul__[] = "\n        Geometric product.\n\n        »> x = clifford(2); x *= clifford(\"{2}\"); print(x)\n        2{2}\n        »> x = clifford(\"{1}\"); x *= clifford(\"{2}\"); print(x)\n        {1,2}\n        »> x = clifford(\"{1}\"); x *= clifford(\"{1,2}\"); print(x)\n        {2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_30__imul__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_31__imul__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__imul__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_30__imul__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_30__imul__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__imul__", 0);

        /* "PyClical.pyx":804
 *         {2}
 *         """
 *         return self.wrap( self.unwrap() * toClifford(rhs) )             # «««««««
 *
 *     def __mod__(lhs, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self, (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) * __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 804, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":793
 *         return clifford().wrap( toClifford(lhs) * toClifford(rhs) )
 *
 *     def __imul__(self, rhs):             # «««««««
 *         """
 *         Geometric product.
 */

        /* function exit code */
```

```
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_AddTraceback("PyClical.clifford.__imul__", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":806
 *         return self.wrap( self.unwrap() * toClifford(rhs) )
 *
 *     def __mod__(lhs, rhs):              # «««««««
 *         """
 *         Contraction.
 */

  /* Python wrapper */
  static PyObject *__pyx_pw_8PyClical_8clifford_33__mod__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
  static char __pyx_doc_8PyClical_8clifford_32__mod__[] = "\n        Contraction.\n\n        »> print(clifford(\"{1}\") % clifford(\"{2}\"))\n        0\n        »> print(clifford(2) % clifford(\"{2}\"))\n        2{2}\n        »> print(clifford(\"{1}\") % clifford(\"{1}\"))\n        1\n        »> print(clifford(\"{1}\") % clifford(\"{1,2}\"))\n        {2}\n        ";
  #if CYTHON_COMPILING_IN_CPYTHON
  struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_32__mod__;
  #endif
  static PyObject *__pyx_pw_8PyClical_8clifford_33__mod__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("__mod__ (wrapper)", 0);
    __pyx_r = __pyx_pf_8PyClical_8clifford_32__mod__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_8clifford_32__mod__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    PyObject *__pyx_t_2 = NULL;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("__mod__", 0);

    /* "PyClical.pyx":819
 *         {2}
 *         """
 *         return clifford().wrap( toClifford(lhs) % toClifford(rhs) )              # «««««««
 *
 *     def __imod__(self, rhs):
 */
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 819, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) % __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 819, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __pyx_r = __pyx_t_2;
    __pyx_t_2 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":806
 *         return self.wrap( self.unwrap() * toClifford(rhs) )
 *
 *     def __mod__(lhs, rhs):              # «««««««
 *         """
 *         Contraction.
 */

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_XDECREF(__pyx_t_2);
    __Pyx_AddTraceback("PyClical.clifford.__mod__", __pyx_clineno, __pyx_lineno,
```

```
__pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":821
 *          return clifford().wrap( toClifford(lhs) % toClifford(rhs) )
 *
 *     def __imod__(self, rhs):              # «««««««
 *          """
 *          Contraction.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_35__imod__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_34__imod__[] = "\n        Contraction.\n\n        »>
x = clifford(\"{1}\"); x %= clifford(\"{2}\"); print(x)\n        0\n        »> x = clifford(2); x %=
clifford(\"{2}\"); print(x)\n        2{2}\n        »> x = clifford(\"{1}\"); x %= clifford(\"{1}\");
print(x)\n        1\n        »> x = clifford(\"{1}\"); x %= clifford(\"{1,2}\"); print(x)\n
{2}\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_34__imod__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_35__imod__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__imod__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_34__imod__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_34__imod__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__imod__", 0);

          /* "PyClical.pyx":834
 *          {2}
 *          """
 *          return self.wrap( self.unwrap() % toClifford(rhs) )             # «««««««
 *
 *     def __and__(lhs, rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) % __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 834, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":821
 *          return clifford().wrap( toClifford(lhs) % toClifford(rhs) )
 *
 *     def __imod__(self, rhs):              # «««««««
 *          """
 *          Contraction.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.__imod__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":836
```

```
 *         return self.wrap( self.unwrap() % toClifford(rhs) )
 *
 *     def __and__(lhs, rhs):               # «««««««
 *         """
 *         Inner product.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_37__and__(PyObject *__pyx_v_lhs, PyObject
 *__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_36__and__[] = "\n        Inner product.\n\n
 >>> print(clifford(\"{1}\") & clifford(\"{2}\"))\n        0\n        >>> print(clifford(2) &
 clifford(\"{2}\"))\n        0\n        >>> print(clifford(\"{1}\") & clifford(\"{1}\"))\n        1\n
 >>> print(clifford(\"{1}\") & clifford(\"{1,2}\"))\n        {2}\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_36__and__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_37__and__(PyObject *__pyx_v_lhs, PyObject
 *__pyx_v_rhs) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__and__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_36__and__(((PyObject *)__pyx_v_lhs), ((PyObject
 *)__pyx_v_rhs));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_36__and__(PyObject *__pyx_v_lhs, PyObject
 *__pyx_v_rhs) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          PyObject *__pyx_t_2 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__and__", 0);

          /* "PyClical.pyx":849
 *         {2}
 *         """
 *         return clifford().wrap( toClifford(lhs) & toClifford(rhs) )             # «««««««
 *
 *     def __iand__(self, rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 849, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
 *)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) &
 __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 849,
 __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":836
 *         return self.wrap( self.unwrap() % toClifford(rhs) )
 *
 *     def __and__(lhs, rhs):               # «««««««
 *         """
 *         Inner product.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.__and__", __pyx_clineno, __pyx_lineno,
 __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":851
 *         return clifford().wrap( toClifford(lhs) & toClifford(rhs) )
 *
 *     def __iand__(self, rhs):               # «««««««
```

```
 *         """
 *         Inner product.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_39__iand__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_38__iand__[] = "\n        Inner product.\n\n        >>> x = clifford(\"{1}\"); x &= clifford(\"{2}\"); print(x)\n        0\n        >>> x = clifford(2); x &= clifford(\"{2}\"); print(x)\n        0\n        >>> x = clifford(\"{1}\"); x &= clifford(\"{1}\"); print(x)\n        1\n        >>> x = clifford(\"{1}\"); x &= clifford(\"{1,2}\"); print(x)\n        {2}\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_38__iand__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_39__iand__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__iand__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_38__iand__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_38__iand__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__iand__", 0);

          /* "PyClical.pyx":864
 *         {2}
 *         """
 *         return self.wrap( self.unwrap() & toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __xor__(lhs, rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self, (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) & __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 864, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":851
 *         return clifford().wrap( toClifford(lhs) & toClifford(rhs) )
 *
 *     def __iand__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Inner product.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.__iand__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":866
 *         return self.wrap( self.unwrap() & toClifford(rhs) )
 *
 *     def __xor__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Outer product.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_41__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_40__xor__[] = "\n        Outer product.\n\n        >>> print(clifford(\"{1}\") ^ clifford(\"{2}\"))\n        {1,2}\n        >>> print(clifford(2) ^
```

```
clifford(\"{2}\"))\n        2{2}\n        >>> print(clifford(\"{1}\") ^ clifford(\"{1}\"))\n        0\n
>>> print(clifford(\"{1}\") ^ clifford(\"{1,2}\"))\n        0\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_40__xor__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_41__xor__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__xor__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_40__xor__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_40__xor__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          PyObject *__pyx_t_2 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__xor__", 0);

          /* "PyClical.pyx":879
 *          0
 *          """
 *          return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *      def __ixor__(self, rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 879, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) ^
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 879,
__pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":866
 *          return self.wrap( self.unwrap() & toClifford(rhs) )
 *
 *      def __xor__(lhs, rhs):             # <<<<<<<<<<<<<<
 *          """
 *          Outer product.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.__xor__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":881
 *          return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )
 *
 *      def __ixor__(self, rhs):             # <<<<<<<<<<<<<<
 *          """
 *          Outer product.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_43__ixor__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_42__ixor__[] = "\n        Outer product.\n\n
>>> x = clifford(\"{1}\"); x ^= clifford(\"{2}\"); print(x)\n        {1,2}\n        >>> x = clifford(2);
x ^= clifford(\"{2}\"); print(x)\n        2{2}\n        >>> x = clifford(\"{1}\"); x ^=
clifford(\"{1}\"); print(x)\n        0\n        >>> x = clifford(\"{1}\"); x ^= clifford(\"{1,2}\");
print(x)\n        0\n        ";
```

```
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_42__ixor__;
#endif
static PyObject *__pyx_pw_8PyClical_8clifford_43__ixor__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__ixor__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_42__ixor__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_42__ixor__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__ixor__", 0);

  /* "PyClical.pyx":894
 *         0
 *         """
 *         return self.wrap( self.unwrap() ^ toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __truediv__(lhs, rhs):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) ^ __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 894, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":881
 *         return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )
 *
 *     def __ixor__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Outer product.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.clifford.__ixor__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":896
 *         return self.wrap( self.unwrap() ^ toClifford(rhs) )
 *
 *     def __truediv__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric quotient.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_45__truediv__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs); /*proto*/
static char __pyx_doc_8PyClical_8clifford_44__truediv__[] = "\n        Geometric quotient.\n\n
>>> print(clifford(\"{1}\") / clifford(\"{2}\"))\n        {1,2}\n        >>> print(clifford(2) /
clifford(\"{2}\"))\n        2{2}\n        >>> print(clifford(\"{1}\") / clifford(\"{1}\"))\n        1\n
>>> print(clifford(\"{1}\") / clifford(\"{1,2}\"))\n        -{2}\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_44__truediv__;
#endif
static PyObject *__pyx_pw_8PyClical_8clifford_45__truediv__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__truediv__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_44__truediv__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));
```

```

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_44__truediv__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__truediv__", 0);

        /* "PyClical.pyx":909
 *         -{2}
 *         """
 *         return clifford().wrap( toClifford(lhs) / toClifford(rhs) )             # <<<<<<<<<<<<<<
 *
 *     def __idiv__(self, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 909, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) /
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 909,
__pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":896
 *         return self.wrap( self.unwrap() ^ toClifford(rhs) )
 *
 *     def __truediv__(lhs, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric quotient.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.clifford.__truediv__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":911
 *         return clifford().wrap( toClifford(lhs) / toClifford(rhs) )
 *
 *     def __idiv__(self, rhs):             # <<<<<<<<<<<<<<
 *         """
 *         Geometric quotient.
 */

      /* Python wrapper */
      #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
      static PyObject *__pyx_pw_8PyClical_8clifford_47__idiv__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_46__idiv__[] = "\n        Geometric quotient.\n\n
>>> x = clifford(\"{1}\"); x /= clifford(\"{2}\"); print(x)\n        {1,2}\n        >>> x = clifford(2);
x /= clifford(\"{2}\"); print(x)\n        2{2}\n        >>> x = clifford(\"{1}\"); x /=
clifford(\"{1}\"); print(x)\n        1\n        >>> x = clifford(\"{1}\"); x /= clifford(\"{1,2}\");
print(x)\n        -{2}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_46__idiv__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_47__idiv__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__idiv__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_46__idiv__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

```

```
    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }
  #endif
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  static PyObject *__pyx_pf_8PyClical_8clifford_46__idiv__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("__idiv__", 0);

    /* "PyClical.pyx":924
 *         -{2}
 *         """
 *         return self.wrap( self.unwrap() / toClifford(rhs) )             # «««««««
 *
 *     def inv(self):
 */
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self, (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) / __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 924, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_r = __pyx_t_1;
    __pyx_t_1 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":911
 *         return clifford().wrap( toClifford(lhs) / toClifford(rhs) )
 *
 *     def __idiv__(self, rhs):             # «««««««
 *         """
 *         Geometric quotient.
 */

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_AddTraceback("PyClical.clifford.__idiv__", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }
  #endif
  /* "PyClical.pyx":926
 *         return self.wrap( self.unwrap() / toClifford(rhs) )
 *
 *     def inv(self):             # «««««««
 *         """
 *         Geometric multiplicative inverse.
 */

  /* Python wrapper */
  static PyObject *__pyx_pw_8PyClical_8clifford_49inv(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
  static char __pyx_doc_8PyClical_8clifford_48inv[] = "\n        Geometric multiplicative inverse.\n\n        >>> x = clifford(\"{1}\"); print(x.inv())\n        {1}\n        >>> x = clifford(2); print(x.inv())\n        0.5\n        >>> x = clifford(\"{1,2}\"); print(x.inv())\n        -{1,2}\n        ";
  static PyObject *__pyx_pw_8PyClical_8clifford_49inv(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("inv (wrapper)", 0);
    __pyx_r = __pyx_pf_8PyClical_8clifford_48inv(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_8clifford_48inv(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    PyObject *__pyx_t_2 = NULL;
    int __pyx_lineno = 0;
```

```
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("inv", 0);

        /* "PyClical.pyx":937
 *         -{1,2}
 *         """
 *         return clifford().wrap( self.instance.inv() )             # «««««««
 *
 *     def __or__(lhs, rhs):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 937, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->inv()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 937, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":926
 *         return self.wrap( self.unwrap() / toClifford(rhs) )
 *
 *     def inv(self):             # «««««««
 *         """
 *         Geometric multiplicative inverse.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.clifford.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":939
 *         return clifford().wrap( self.instance.inv() )
 *
 *     def __or__(lhs, rhs):             # «««««««
 *         """
 *         Transform left hand side, using right hand side as a transformation.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_51__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_50__or__[] = "\n        Transform left hand side, using right hand side as a transformation.\n\n        >>> x=clifford(\"{1,2}\") * pi/2; y=clifford(\"{1}\"); print(y|x)\n        -{1}\n        >>> x=clifford(\"{1,2}\") * pi/2; y=clifford(\"{1}\"); print(y|exp(x))\n        -{1}\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_50__or__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_51__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__or__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_50__or__(((PyObject *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_50__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__or__", 0);

        /* "PyClical.pyx":948
```

```
 *         -{1}
 *         """
 *         return clifford().wrap( toClifford(lhs) | toClifford(rhs) )             # «««««««
 *
 *     def __ior__(self, rhs):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 948, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) | __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 948, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":939
 *         return clifford().wrap( self.instance.inv() )
 *
 *     def __or__(lhs, rhs):             # «««««««
 *         """
 *         Transform left hand side, using right hand side as a transformation.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.__or__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":950
 *         return clifford().wrap( toClifford(lhs) | toClifford(rhs) )
 *
 *     def __ior__(self, rhs):             # «««««««
 *         """
 *         Transform left hand side, using right hand side as a transformation.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_53__ior__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_52__ior__[] = "\n        Transform left hand side, using right hand side as a transformation.\n\n        >>> x=clifford(\"{1,2}\") * pi/2; y=clifford(\"{1}\"); y|=x; print(y)\n        -{1}\n        >>> x=clifford(\"{1,2}\") * pi/2; y=clifford(\"{1}\"); y|=exp(x); print(y)\n        -{1}\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_52__ior__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_53__ior__(PyObject *__pyx_v_self, PyObject *__pyx_v_rhs) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__ior__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_52__ior__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_52__ior__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_rhs) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__ior__", 0);

          /* "PyClical.pyx":959
 *         -{1}
 *         """
 *         return self.wrap( self.unwrap() | toClifford(rhs) )             # «««««««
 *
 *     def __pow__(self, m, dummy):
```

```
*/
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self, (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) | __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 959, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":950
 *          return clifford().wrap( toClifford(lhs) | toClifford(rhs) )
 *
 *      def __ior__(self, rhs):             # <<<<<<<<
 *          """
 *          Transform left hand side, using right hand side as a transformation.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__ior__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":961
 *          return self.wrap( self.unwrap() | toClifford(rhs) )
 *
 *      def __pow__(self, m, dummy):             # <<<<<<<<
 *          """
 *          Power: self to the m.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_55__pow__(PyObject *__pyx_v_self, PyObject *__pyx_v_m, PyObject *__pyx_v_dummy); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_54__pow__[] = "\n        Power: self to the m.\n\n>>> x=clifford(\"{1}\"); print(x ** 2)\n        1\n        >>> x=clifford(\"2\"); print(x ** 2)\n        4\n        >>> x=clifford(\"2+{1}\"); print(x ** 0)\n        1\n        >>> x=clifford(\"2+{1}\"); print(x ** 1)\n        2+{1}\n        >>> x=clifford(\"2+{1}\"); print(x ** 2)\n        5+4{1}\n        >>> i=clifford(\"{1,2}\"); print(exp(pi/2) * (i ** i))\n        1\n        ";
      #if CYTHON_COMPILING_IN_CPYTHON
      struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_54__pow__;
      #endif
      static PyObject *__pyx_pw_8PyClical_8clifford_55__pow__(PyObject *__pyx_v_self, PyObject *__pyx_v_m, PyObject *__pyx_v_dummy) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__pow__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_54__pow__(((PyObject *)__pyx_v_self), ((PyObject *)__pyx_v_m), ((PyObject *)__pyx_v_dummy));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_54__pow__(PyObject *__pyx_v_self, PyObject *__pyx_v_m, CYTHON_UNUSED PyObject *__pyx_v_dummy) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__pow__", 0);

        /* "PyClical.pyx":978
 *          1
 *          """
 *          return pow(self, m)             # <<<<<<<<
 *
 *      def pow(self, m):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_pow(__pyx_v_self, __pyx_v_m, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 978, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

```

```
        /* "PyClical.pyx":961
 *         return self.wrap( self.unwrap() | toClifford(rhs) )
 *
 *     def __pow__(self, m, dummy):              # <<<<<<<<<<<<<<
 *         """
 *         Power: self to the m.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__pow__", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":980
 *         return pow(self, m)
 *
 *     def pow(self, m):              # <<<<<<<<<<<<<<
 *         """
 *         Power: self to the m.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_57pow(PyObject *__pyx_v_self, PyObject *__pyx_v_m); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_56pow[] = "\n        Power: self to the m.\n\n        >>> x=clifford(\"{1}\"); print(x.pow(2))\n        1\n        >>> x=clifford(\"2\"); print(x.pow(2))\n        4\n        >>> x=clifford(\"2+{1}\"); print(x.pow(0))\n        1\n        >>> x=clifford(\"2+{1}\"); print(x.pow(1))\n        2+{1}\n        >>> x=clifford(\"2+{1}\"); print(x.pow(2))\n        5+4{1}\n        >>> print(clifford(\"1+{1}+{1,2}\").pow(3))\n        1+3{1}+3{1,2}\n        >>> i=clifford(\"{1,2}\"); print(exp(pi/2) * i.pow(i))\n        1\n        ";
      static PyObject *__pyx_pw_8PyClical_8clifford_57pow(PyObject *__pyx_v_self, PyObject *__pyx_v_m) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("pow (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_56pow(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_m));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_56pow(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_m) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_t_3;
        int __pyx_t_4;
        int __pyx_t_5;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("pow", 0);

        /* "PyClical.pyx":999
 *         1
 *         """
 *         if isinstance(m, numbers.Integral):              # <<<<<<<<<<<<<<
 *             return clifford().wrap( self.instance.pow(m) )
 *         else:
 */
        __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_numbers); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 999, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __Pyx_PyObject_GetAttrStr(__pyx_t_1, __pyx_n_s_Integral); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 999, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_t_3 = PyObject_IsInstance(__pyx_v_m, __pyx_t_2); if (unlikely(__pyx_t_3 == ((int)-1))) __PYX_ERR(0, 999, __pyx_L1_error)
        __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
        __pyx_t_4 = (__pyx_t_3 != 0);
        if (__pyx_t_4) {

          /* "PyClical.pyx":1000
 *         """
 *         if isinstance(m, numbers.Integral):
```

```
 *             return clifford().wrap( self.instance.pow(m) )            # «««««««
 *         else:
 *             return exp(m * log(self))
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_2 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 1000, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_2);
            __pyx_t_5 = __Pyx_PyInt_As_int(__pyx_v_m); if (unlikely((__pyx_t_5 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 1000, __pyx_L1_error)
            __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_2), __pyx_v_self->instance->pow(__pyx_t_5)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1000,
__pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;

            /* "PyClical.pyx":999
 *         1
 *         """
 *         if isinstance(m, numbers.Integral):             # «««««««
 *             return clifford().wrap( self.instance.pow(m) )
 *         else:
 */
          }

          /* "PyClical.pyx":1002
 *             return clifford().wrap( self.instance.pow(m) )
 *         else:
 *             return exp(m * log(self))             # «««««««
 *
 *     def outer_pow(self, m):
 */
          /*else*/ {
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __pyx_f_8PyClical_log(((PyObject *)__pyx_v_self), 0, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1002, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_t_2 = PyNumber_Multiply(__pyx_v_m, __pyx_t_1); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1002, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_2);
            __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
            __pyx_t_1 = __pyx_f_8PyClical_exp(__pyx_t_2, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1002, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;
          }

          /* "PyClical.pyx":980
 *         return pow(self, m)
 *
 *     def pow(self, m):             # «««««««
 *         """
 *         Power: self to the m.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1004
 *             return exp(m * log(self))
 *
 *     def outer_pow(self, m):             # «««««««
 *         """
 *         Outer product power.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_59outer_pow(PyObject *__pyx_v_self, PyObject
*__pyx_v_m); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_58outer_pow[] = "\n        Outer product power.\n\n
»> x=clifford(\"2+{1}\"); print(x.outer_pow(0))\n        1\n        »> x=clifford(\"2+{1}\");
print(x.outer_pow(1))\n        2+{1}\n        »> x=clifford(\"2+{1}\"); print(x.outer_pow(2))\n
```

```
4+4{1}\n        >> print(clifford(\"1+{1}+{1,2}\").outer_pow(3))\n        1+3{1}+3{1,2}\n\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_59outer_pow(PyObject *__pyx_v_self, PyObject *__pyx_v_m) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("outer_pow (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_58outer_pow(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v_m));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_58outer_pow(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_m) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_t_2;
          PyObject *__pyx_t_3 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("outer_pow", 0);

          /* "PyClical.pyx":1018
 *
 *         """
 *         return clifford().wrap( self.instance.outer_pow(m) )             # <<<<<<<<<<<<<<
 *
 *     def __call__(self, grade):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1018, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_m); if (unlikely((__pyx_t_2 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 1018, __pyx_L1_error)
          __pyx_t_3 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->outer_pow(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1018, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_3);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_3;
          __pyx_t_3 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1004
 *             return exp(m * log(self))
 *
 *     def outer_pow(self, m):             # <<<<<<<<<<<<<<
 *         """
 *         Outer product power.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_3);
          __Pyx_AddTraceback("PyClical.clifford.outer_pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1020
 *         return clifford().wrap( self.instance.outer_pow(m) )
 *
 *     def __call__(self, grade):             # <<<<<<<<<<<<<<
 *         """
 *         Pure grade-vector part.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_61__call__(PyObject *__pyx_v_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_60__call__[] = "\n        Pure grade-vector part.\n\n        >> print(clifford(\"{1}\")(1))\n        {1}\n        >> print(clifford(\"{1}\")(0))\n        0\n        >> print(clifford(\"1+{1}+{1,2}\")(0))\n        1\n        >> print(clifford(\"1+{1}+{1,2}\")(1))\n        {1}\n        >> print(clifford(\"1+{1}+{1,2}\")(2))\n        {1,2}\n        >> print(clifford(\"1+{1}+{1,2}\")(3))\n        0\n        ";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_60__call__;
```

```
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_61__call__(PyObject *__pyx_v_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
          PyObject *__pyx_v_grade = 0;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__call__ (wrapper)", 0);
          {
            static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_grade,0};
            PyObject* values[1] = {0};
            if (unlikely(__pyx_kwds)) {
              Py_ssize_t kw_args;
              const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
              switch (pos_args) {
                case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                CYTHON_FALLTHROUGH;
                case  0: break;
                default: goto __pyx_L5_argtuple_error;
              }
              kw_args = PyDict_Size(__pyx_kwds);
              switch (pos_args) {
                case  0:
                if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_grade)) != 0)) kw_args--;
                else goto __pyx_L5_argtuple_error;
              }
              if (unlikely(kw_args > 0)) {
                if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "__call__") < 0)) __PYX_ERR(0, 1020, __pyx_L3_error)
              }
            } else if (PyTuple_GET_SIZE(__pyx_args) != 1) {
              goto __pyx_L5_argtuple_error;
            } else {
              values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
            }
            __pyx_v_grade = values[0];
          }
          goto __pyx_L4_argument_unpacking_done;
          __pyx_L5_argtuple_error:;
          __Pyx_RaiseArgtupleInvalid("__call__", 1, 1, 1, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1020, __pyx_L3_error)
          __pyx_L3_error:;
          __Pyx_AddTraceback("PyClical.clifford.__call__", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __Pyx_RefNannyFinishContext();
          return NULL;
          __pyx_L4_argument_unpacking_done:;
          __pyx_r = __pyx_pf_8PyClical_8clifford_60__call__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), __pyx_v_grade);

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_60__call__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_grade) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_t_2;
          PyObject *__pyx_t_3 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__call__", 0);

          /* "PyClical.pyx":1037
 *           0
 *           """
 *           return clifford().wrap( self.instance.call(grade) )              # <<<<<<<<<<<<<<
 *
 *      def scalar(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1037, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_grade); if (unlikely((__pyx_t_2 == (int)-1) && PyErr_Occurred())) __PYX_ERR(0, 1037, __pyx_L1_error)
          __pyx_t_3 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->operator()(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1037, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_3);
```

```
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __pyx_r = __pyx_t_3;
    __pyx_t_3 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":1020
 *         return clifford().wrap( self.instance.outer_pow(m) )
 *
 *     def __call__(self, grade):             # <<<<<<<<<<<<<<
 *         """
 *         Pure grade-vector part.
 */

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_XDECREF(__pyx_t_3);
    __Pyx_AddTraceback("PyClical.clifford.__call__", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":1039
 *         return clifford().wrap( self.instance.call(grade) )
 *
 *     def scalar(self):             # <<<<<<<<<<<<<<
 *         """
 *         Scalar part.
 */

  /* Python wrapper */
  static PyObject *__pyx_pw_8PyClical_8clifford_63scalar(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
  static char __pyx_doc_8PyClical_8clifford_62scalar[] = "\n        Scalar part.\n\n        >>> clifford(\"1+{1}+{1,2}\").scalar()\n        1.0\n        >>> clifford(\"{1,2}\").scalar()\n        0.0\n        ";
  static PyObject *__pyx_pw_8PyClical_8clifford_63scalar(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("scalar (wrapper)", 0);
    __pyx_r = __pyx_pf_8PyClical_8clifford_62scalar(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_8clifford_62scalar(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("scalar", 0);

    /* "PyClical.pyx":1048
 *         0.0
 *         """
 *         return self.instance.scalar()             # <<<<<<<<<<<<<<
 *
 *     def pure(self):
 */
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->scalar()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1048, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_r = __pyx_t_1;
    __pyx_t_1 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":1039
 *         return clifford().wrap( self.instance.call(grade) )
 *
 *     def scalar(self):             # <<<<<<<<<<<<<<
 *         """
 *         Scalar part.
 */

    /* function exit code */
```

```
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_AddTraceback("PyClical.clifford.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":1050
 *         return self.instance.scalar()
 *
 *     def pure(self):             # <<<<<<<<<<<<<<
 *         """
 *         Pure part.
 */

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_8clifford_65pure(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
    static char __pyx_doc_8PyClical_8clifford_64pure[] = "\n        Pure part.\n\n        >>> print(clifford(\"1+{1}+{1,2}\").pure())\n        {1}+{1,2}\n        >>> print(clifford(\"{1,2}\").pure())\n        {1,2}\n        ";
    static PyObject *__pyx_pw_8PyClical_8clifford_65pure(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
      __Pyx_RefNannySetupContext("pure (wrapper)", 0);
      __pyx_r = __pyx_pf_8PyClical_8clifford_64pure(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

      /* function exit code */
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    static PyObject *__pyx_pf_8PyClical_8clifford_64pure(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      PyObject *__pyx_t_2 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("pure", 0);

      /* "PyClical.pyx":1059
 *         {1,2}
 *         """
 *         return clifford().wrap( self.instance.pure() )             # <<<<<<<<<<<<<<
 *
 *     def even(self):
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1059, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->pure()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1059, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_2);
      __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
      __pyx_r = __pyx_t_2;
      __pyx_t_2 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":1050
 *         return self.instance.scalar()
 *
 *     def pure(self):             # <<<<<<<<<<<<<<
 *         """
 *         Pure part.
 */

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_XDECREF(__pyx_t_2);
      __Pyx_AddTraceback("PyClical.clifford.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }
```

```
    /* "PyClical.pyx":1061
 *         return clifford().wrap( self.instance.pure() )
 *
 *     def even(self):              # «««««««
 *         """
 *         Even part of multivector, sum of even grade terms.
 */

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_8clifford_67even(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
    static char __pyx_doc_8PyClical_8clifford_66even[] = "\n        Even part of multivector, sum
of even grade terms.\n\n        >>> print(clifford(\"1+{1}+{1,2}\").even())\n        1+{1,2}\n
";
    static PyObject *__pyx_pw_8PyClical_8clifford_67even(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
      __Pyx_RefNannySetupContext("even (wrapper)", 0);
      __pyx_r = __pyx_pf_8PyClical_8clifford_66even(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

      /* function exit code */
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    static PyObject *__pyx_pf_8PyClical_8clifford_66even(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      PyObject *__pyx_t_2 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("even", 0);

      /* "PyClical.pyx":1068
 *         1+{1,2}
 *         """
 *         return clifford().wrap( self.instance.even() )             # «««««««
 *
 *     def odd(self):
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1068, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->even()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1068,
__pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_2);
      __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
      __pyx_r = __pyx_t_2;
      __pyx_t_2 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":1061
 *         return clifford().wrap( self.instance.pure() )
 *
 *     def even(self):              # «««««««
 *         """
 *         Even part of multivector, sum of even grade terms.
 */

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_XDECREF(__pyx_t_2);
      __Pyx_AddTraceback("PyClical.clifford.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":1070
 *         return clifford().wrap( self.instance.even() )
 *
 *     def odd(self):              # «««««««
 *         """
 *         Odd part of multivector, sum of odd grade terms.
 */

```

```
        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_69odd(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_68odd[] = "\n        Odd part of multivector, sum of
odd grade terms.\n\n        >>> print(clifford(\"1+{1}+{1,2}\").odd())\n        {1}\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_69odd(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("odd (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_68odd(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_68odd(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          PyObject *__pyx_t_2 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("odd", 0);

          /* "PyClical.pyx":1077
 *           {1}
 *           """
 *           return clifford().wrap( self.instance.odd() )             # <<<<<<<<<<<<<<
 *
 *      def vector_part(self, frm = None):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1077, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->odd()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1077,
__pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1070
 *           return clifford().wrap( self.instance.even() )
 *
 *      def odd(self):             # <<<<<<<<<<<<<<
 *           """
 *           Odd part of multivector, sum of odd grade terms.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1079
 *           return clifford().wrap( self.instance.odd() )
 *
 *      def vector_part(self, frm = None):             # <<<<<<<<<<<<<<
 *           """
 *           Vector part of multivector, as a Python list, with respect to frm.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_71vector_part(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_70vector_part[] = "\n        Vector part of
multivector, as a Python list, with respect to frm.\n\n        >>>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part())\n        [2.0, 3.0]\n        >>>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part(index_set({-1,1,2})))\n        [0.0, 2.0, 3.0]\n
";
        static PyObject *__pyx_pw_8PyClical_8clifford_71vector_part(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds) {
```

```
  PyObject *__pyx_v_frm = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("vector_part (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_frm,0};
    PyObject* values[1] = {0};
    values[0] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_frm);
          if (value) { values[0] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "vector_part") < 0)) __PYX_ERR(0, 1079, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_frm = values[0];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("vector_part", 0, 0, 1, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1079, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_8clifford_70vector_part(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self), __pyx_v_frm);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_70vector_part(struct __pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_frm) {
  PyObject *__pyx_v_error_msg_prefix = NULL;
  std::vector<scalar_t>  __pyx_v_vec;
  int __pyx_v_n;
  int __pyx_v_i;
  PyObject *__pyx_v_lst = NULL;
  PyObject *__pyx_v_err = NULL;
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  int __pyx_t_4;
  int __pyx_t_5;
  std::vector<scalar_t>  __pyx_t_6;
  PyObject *__pyx_t_7 = NULL;
  int __pyx_t_8;
  int __pyx_t_9;
  int __pyx_t_10;
  PyObject *__pyx_t_11 = NULL;
  PyObject *__pyx_t_12 = NULL;
  PyObject *__pyx_t_13 = NULL;
  PyObject *__pyx_t_14 = NULL;
  PyObject *__pyx_t_15 = NULL;
  char const *__pyx_t_16;
  PyObject *__pyx_t_17 = NULL;
```

```
        PyObject *__pyx_t_18 = NULL;
        PyObject *__pyx_t_19 = NULL;
        PyObject *__pyx_t_20 = NULL;
        PyObject *__pyx_t_21 = NULL;
        PyObject *__pyx_t_22 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("vector_part", 0);

        /* "PyClical.pyx":1088
 *         [0.0, 2.0, 3.0]
 *         """
 *         error_msg_prefix = "Cannot take vector part of "             # «««««««
 *         cdef vector[scalar_t] vec
 *         cdef int n
 */
        __Pyx_INCREF(__pyx_kp_u_Cannot_take_vector_part_of);
        __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_take_vector_part_of;

        /* "PyClical.pyx":1092
 *         cdef int n
 *         cdef int i
 *         try:              # «««««««
 *             if frm is None:
 *                 vec = self.instance.vector_part()
 */
        {
          __Pyx_PyThreadState_declare
          __Pyx_PyThreadState_assign
          __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
          __Pyx_XGOTREF(__pyx_t_1);
          __Pyx_XGOTREF(__pyx_t_2);
          __Pyx_XGOTREF(__pyx_t_3);
          /*try:*/ {

            /* "PyClical.pyx":1093
 *         cdef int i
 *         try:
 *             if frm is None:              # «««««««
 *                 vec = self.instance.vector_part()
 *             else:
 */
            __pyx_t_4 = (__pyx_v_frm == Py_None);
            __pyx_t_5 = (__pyx_t_4 != 0);
            if (__pyx_t_5) {

              /* "PyClical.pyx":1094
 *         try:
 *             if frm is None:
 *                 vec = self.instance.vector_part()             # «««««««
 *             else:
 *                 vec = self.instance.vector_part((<index_set>frm).unwrap())
 */
              __pyx_t_6 = __pyx_v_self->instance->vector_part();
              __pyx_v_vec = __pyx_t_6;

              /* "PyClical.pyx":1093
 *         cdef int i
 *         try:
 *             if frm is None:              # «««««««
 *                 vec = self.instance.vector_part()
 *             else:
 */
              goto __pyx_L9;
            }

            /* "PyClical.pyx":1096
 *                 vec = self.instance.vector_part()
 *             else:
 *                 vec = self.instance.vector_part((<index_set>frm).unwrap())             # «««««««
 *             n = vec.size()
 *             lst = [0.0]*n
 */
            /*else*/ {
              try {
                __pyx_t_6 =
      __pyx_v_self->instance->vector_part(__pyx_f_8PyClical_9index_set_unwrap(((struct
      __pyx_obj_8PyClical_index_set *)__pyx_v_frm)));
              } catch(...) {
                __Pyx_CppExn2PyErr();
                __PYX_ERR(0, 1096, __pyx_L3_error)
              }
              __pyx_v_vec = __pyx_t_6;
            }
            __pyx_L9:;
```

```
              /* "PyClical.pyx":1097
 *             else:
 *                 vec = self.instance.vector_part((<index_set>frm).unwrap())
 *             n = vec.size()             # <<<<<<<<
 *             lst = [0.0]*n
 *             for i in xrange(n):
 */
              __pyx_v_n = __pyx_v_vec.size();

              /* "PyClical.pyx":1098
 *                 vec = self.instance.vector_part((<index_set>frm).unwrap())
 *             n = vec.size()
 *             lst = [0.0]*n             # <<<<<<<<
 *             for i in xrange(n):
 *                 lst[i] = vec[i]
 */
              __pyx_t_7 = PyList_New(1 * ((__pyx_v_n<0) ? 0:__pyx_v_n)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1098, __pyx_L3_error)
              __Pyx_GOTREF(__pyx_t_7);
              { Py_ssize_t __pyx_temp;
                for (__pyx_temp=0; __pyx_temp < __pyx_v_n; __pyx_temp++) {
                  __Pyx_INCREF(__pyx_float_0_0);
                  __Pyx_GIVEREF(__pyx_float_0_0);
                  PyList_SET_ITEM(__pyx_t_7, __pyx_temp, __pyx_float_0_0);
                }
              }
              __pyx_v_lst = ((PyObject*)__pyx_t_7);
              __pyx_t_7 = 0;

              /* "PyClical.pyx":1099
 *             n = vec.size()
 *             lst = [0.0]*n
 *             for i in xrange(n):             # <<<<<<<<
 *                 lst[i] = vec[i]
 *             return lst
 */
              __pyx_t_8 = __pyx_v_n;
              __pyx_t_9 = __pyx_t_8;
              for (__pyx_t_10 = 0; __pyx_t_10 < __pyx_t_9; __pyx_t_10+=1) {
                __pyx_v_i = __pyx_t_10;

                /* "PyClical.pyx":1100
 *             lst = [0.0]*n
 *             for i in xrange(n):
 *                 lst[i] = vec[i]             # <<<<<<<<
 *             return lst
 *         except RuntimeError as err:
 */
                __pyx_t_7 = PyFloat_FromDouble((__pyx_v_vec[__pyx_v_i])); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1100, __pyx_L3_error)
                __Pyx_GOTREF(__pyx_t_7);
                if (unlikely(__Pyx_SetItemInt(__pyx_v_lst, __pyx_v_i, __pyx_t_7, int, 1, __Pyx_PyInt_From_int, 1, 1, 1) < 0)) __PYX_ERR(0, 1100, __pyx_L3_error)
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
              }

              /* "PyClical.pyx":1101
 *             for i in xrange(n):
 *                 lst[i] = vec[i]
 *             return lst             # <<<<<<<<
 *         except RuntimeError as err:
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "
 */
              __Pyx_XDECREF(__pyx_r);
              __Pyx_INCREF(__pyx_v_lst);
              __pyx_r = __pyx_v_lst;
              goto __pyx_L7_try_return;

              /* "PyClical.pyx":1092
 *         cdef int n
 *         cdef int i
 *         try:             # <<<<<<<<
 *             if frm is None:
 *                 vec = self.instance.vector_part()
 */
            }
            __pyx_L3_error:;
            __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;

            /* "PyClical.pyx":1102
 *                 lst[i] = vec[i]
 *             return lst
 *         except RuntimeError as err:             # <<<<<<<<
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "
 *                                                + repr(frm) + " as frame:\n\t"
 */
            __pyx_t_8 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
```

```
            if (__pyx_t_8) {
              __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno, __pyx_filename);
              if (__Pyx_GetException(&__pyx_t_7, &__pyx_t_11, &__pyx_t_12) < 0) __PYX_ERR(0, 1102, __pyx_L5_except_error)
              __Pyx_GOTREF(__pyx_t_7);
              __Pyx_GOTREF(__pyx_t_11);
              __Pyx_GOTREF(__pyx_t_12);
              __Pyx_INCREF(__pyx_t_11);
              __pyx_v_err = __pyx_t_11;
              /*try:*/ {

                /* "PyClical.pyx":1103
 *             return lst
 *         except RuntimeError as err:
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "             # <<<<<<<<<<<<<<
 *                                               + repr(frm) + " as frame:\n\t"
 *                                               + str(err))
 */
                __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 1103, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_13);
                __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_t_13); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 1103, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_14);
                __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_14, __pyx_kp_u_using_invalid); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 1103, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_13);
                __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;

                /* "PyClical.pyx":1104
 *         except RuntimeError as err:
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "
 *                                               + repr(frm) + " as frame:\n\t"            # <<<<<<<<<<<<<<
 *                                               + str(err))
 *
 */
                __pyx_t_14 = PyObject_Repr(__pyx_v_frm); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 1104, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_14);
                __pyx_t_15 = PyNumber_Add(__pyx_t_13, __pyx_t_14); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 1104, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_15);
                __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
                __pyx_t_14 = PyNumber_Add(__pyx_t_15, __pyx_kp_u_as_frame); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 1104, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_14);
                __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;

                /* "PyClical.pyx":1105
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "
 *                                               + repr(frm) + " as frame:\n\t"
 *                                               + str(err))             # <<<<<<<<<<<<<<
 *
 *     def involute(self):
 */
                __pyx_t_15 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), __pyx_v_err); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 1105, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_15);
                __pyx_t_13 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 1105, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_13);
                __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
                __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;

                /* "PyClical.pyx":1103
 *             return lst
 *         except RuntimeError as err:
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "             # <<<<<<<<<<<<<<
 *                                               + repr(frm) + " as frame:\n\t"
 *                                               + str(err))
 */
                __pyx_t_15 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_13); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 1103, __pyx_L17_error)
                __Pyx_GOTREF(__pyx_t_15);
                __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
                __Pyx_Raise(__pyx_t_15, 0, 0, 0);
                __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
                __PYX_ERR(0, 1103, __pyx_L17_error)
              }

              /* "PyClical.pyx":1102
 *                 lst[i] = vec[i]
 *             return lst
 *         except RuntimeError as err:             # <<<<<<<<<<<<<<
```

```
 *             raise ValueError(error_msg_prefix + str(self) + " using invalid "
 *                                               + repr(frm) + " as frame:\n\t"
 */
              /*finally:*/ {
                __pyx_L17_error:;
                /*exception exit:*/{
                  __Pyx_PyThreadState_declare
                  __Pyx_PyThreadState_assign
                  __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0;
                  __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
                  __Pyx_XDECREF(__pyx_t_14); __pyx_t_14 = 0;
                  __Pyx_XDECREF(__pyx_t_15); __pyx_t_15 = 0;
                  if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_20, &__pyx_t_21, &__pyx_t_22);
                  if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_17, &__pyx_t_18, &__pyx_t_19) < 0)) __Pyx_ErrFetch(&__pyx_t_17, &__pyx_t_18, &__pyx_t_19);
                  __Pyx_XGOTREF(__pyx_t_17);
                  __Pyx_XGOTREF(__pyx_t_18);
                  __Pyx_XGOTREF(__pyx_t_19);
                  __Pyx_XGOTREF(__pyx_t_20);
                  __Pyx_XGOTREF(__pyx_t_21);
                  __Pyx_XGOTREF(__pyx_t_22);
                  __pyx_t_8 = __pyx_lineno; __pyx_t_9 = __pyx_clineno; __pyx_t_16 = __pyx_filename;
                  {
                    __Pyx_DECREF(__pyx_v_err);
                    __pyx_v_err = NULL;
                  }
                  if (PY_MAJOR_VERSION >= 3) {
                    __Pyx_XGIVEREF(__pyx_t_20);
                    __Pyx_XGIVEREF(__pyx_t_21);
                    __Pyx_XGIVEREF(__pyx_t_22);
                    __Pyx_ExceptionReset(__pyx_t_20, __pyx_t_21, __pyx_t_22);
                  }
                  __Pyx_XGIVEREF(__pyx_t_17);
                  __Pyx_XGIVEREF(__pyx_t_18);
                  __Pyx_XGIVEREF(__pyx_t_19);
                  __Pyx_ErrRestore(__pyx_t_17, __pyx_t_18, __pyx_t_19);
                  __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0;
                  __pyx_lineno = __pyx_t_8; __pyx_clineno = __pyx_t_9; __pyx_filename = __pyx_t_16;
                  goto __pyx_L5_except_error;
                }
              }
            }
            goto __pyx_L5_except_error;
            __pyx_L5_except_error:;

            /* "PyClical.pyx":1092
 *         cdef int n
 *         cdef int i
 *         try:                # <<<<<<<<<<<<<<
 *             if frm is None:
 *                 vec = self.instance.vector_part()
 */
            __Pyx_XGIVEREF(__pyx_t_1);
            __Pyx_XGIVEREF(__pyx_t_2);
            __Pyx_XGIVEREF(__pyx_t_3);
            __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
            goto __pyx_L1_error;
            __pyx_L7_try_return:;
            __Pyx_XGIVEREF(__pyx_t_1);
            __Pyx_XGIVEREF(__pyx_t_2);
            __Pyx_XGIVEREF(__pyx_t_3);
            __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
            goto __pyx_L0;
          }

          /* "PyClical.pyx":1079
 *         return clifford().wrap( self.instance.odd() )
 *
 *     def vector_part(self, frm = None):             # <<<<<<<<<<<<<<
 *         """
 *         Vector part of multivector, as a Python list, with respect to frm.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_7);
          __Pyx_XDECREF(__pyx_t_11);
          __Pyx_XDECREF(__pyx_t_12);
          __Pyx_XDECREF(__pyx_t_13);
          __Pyx_XDECREF(__pyx_t_14);
          __Pyx_XDECREF(__pyx_t_15);
          __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
```

```
      __pyx_L0:;
      __Pyx_XDECREF(__pyx_v_error_msg_prefix);
      __Pyx_XDECREF(__pyx_v_lst);
      __Pyx_XDECREF(__pyx_v_err);
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":1107
 *                                                + str(err))
 *
 *     def involute(self):              # <<<<<<<<<<<<<<
 *         """
 *         Main involution, each {i} is replaced by -{i} in each term,
 */

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_8clifford_73involute(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
    static char __pyx_doc_8PyClical_8clifford_72involute[] = "\n        Main involution, each {i} is replaced by -{i} in each term,\n        eg. clifford(\"{1}\") -> -clifford(\"{1}\").\n\n        >>> print(clifford(\"{1}\").involute())\n        -{1}\n        >>> print((clifford(\"{2}\") * clifford(\"{1}\")).involute())\n        -{1,2}\n        >>> print((clifford(\"{1}\") * clifford(\"{2}\")).involute())\n        {1,2}\n        >>> print(clifford(\"1+{1}+{1,2}\").involute())\n        1-{1}+{1,2}\n        ";
    static PyObject *__pyx_pw_8PyClical_8clifford_73involute(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
      __Pyx_RefNannySetupContext("involute (wrapper)", 0);
      __pyx_r = __pyx_pf_8PyClical_8clifford_72involute(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

      /* function exit code */
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    static PyObject *__pyx_pf_8PyClical_8clifford_72involute(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      PyObject *__pyx_t_2 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("involute", 0);

      /* "PyClical.pyx":1121
 *         1-{1}+{1,2}
 *         """
 *         return clifford().wrap( self.instance.involute() )              # <<<<<<<<<<<<<<
 *
 *     def reverse(self):
 */
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1121, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->involute()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1121, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_2);
      __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
      __pyx_r = __pyx_t_2;
      __pyx_t_2 = 0;
      goto __pyx_L0;

      /* "PyClical.pyx":1107
 *                                                + str(err))
 *
 *     def involute(self):              # <<<<<<<<<<<<<<
 *         """
 *         Main involution, each {i} is replaced by -{i} in each term,
 */

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_XDECREF(__pyx_t_2);
      __Pyx_AddTraceback("PyClical.clifford.involute", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
```

```
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1123
 *         return clifford().wrap( self.instance.involute() )
 *
 *     def reverse(self):             # <<<<<<<<<<<<<<
 *         """
 *         Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
 */

          /* Python wrapper */
          static PyObject *__pyx_pw_8PyClical_8clifford_75reverse(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
          static char __pyx_doc_8PyClical_8clifford_74reverse[] = "\n        Reversion, eg. clifford(\"{1}\")*clifford(\"{2}\") -> clifford(\"{2}\")*clifford(\"{1}\").\n\n        >>> print(clifford(\"{1}\").reverse())\n        {1}\n        >>> print((clifford(\"{2}\") * clifford(\"{1}\")).reverse())\n        {1,2}\n        >>> print((clifford(\"{1}\") * clifford(\"{2}\")).reverse())\n        -{1,2}\n        >>> print(clifford(\"1+{1}+{1,2}\").reverse())\n        1+{1}-{1,2}\n        ";
          static PyObject *__pyx_pw_8PyClical_8clifford_75reverse(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
            PyObject *__pyx_r = 0;
            __Pyx_RefNannyDeclarations
            __Pyx_RefNannySetupContext("reverse (wrapper)", 0);
            __pyx_r = __pyx_pf_8PyClical_8clifford_74reverse(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

            /* function exit code */
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          static PyObject *__pyx_pf_8PyClical_8clifford_74reverse(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            PyObject *__pyx_t_2 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("reverse", 0);

            /* "PyClical.pyx":1136
 *         1+{1}-{1,2}
 *         """
 *         return clifford().wrap( self.instance.reverse() )             # <<<<<<<<<<<<<<
 *
 *     def conj(self):
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1136, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->reverse()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1136, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_2);
            __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
            __pyx_r = __pyx_t_2;
            __pyx_t_2 = 0;
            goto __pyx_L0;

            /* "PyClical.pyx":1123
 *         return clifford().wrap( self.instance.involute() )
 *
 *     def reverse(self):             # <<<<<<<<<<<<<<
 *         """
 *         Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
 */

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_XDECREF(__pyx_t_2);
            __Pyx_AddTraceback("PyClical.clifford.reverse", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1138
```

```
 *         return clifford().wrap( self.instance.reverse() )
 *
 *     def conj(self):              # «««««««
 *         """
 *         Conjugation, reverse o involute == involute o reverse.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_77conj(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_76conj[] = "\n        Conjugation, reverse o involute == involute o reverse.\n\n        >>> print((clifford(\"{1}\")).conj())\n        -{1}\n        >>> print((clifford(\"{2}\") * clifford(\"{1}\")).conj())\n        {1,2}\n        >>> print((clifford(\"{1}\") * clifford(\"{2}\")).conj())\n        -{1,2}\n        >>> print(clifford(\"1+{1}+{1,2}\").conj())\n        1-{1}-{1,2}\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_77conj(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("conj (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_76conj(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_76conj(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          PyObject *__pyx_t_2 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("conj", 0);

          /* "PyClical.pyx":1151
 *         1-{1}-{1,2}
 *         """
 *         return clifford().wrap( self.instance.conj() )             # «««««««
 *
 *     def quad(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1151, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), __pyx_v_self->instance->conj()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1151, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1138
 *         return clifford().wrap( self.instance.reverse() )
 *
 *     def conj(self):              # «««««««
 *         """
 *         Conjugation, reverse o involute == involute o reverse.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1153
 *         return clifford().wrap( self.instance.conj() )
 *
 *     def quad(self):              # «««««««
 *         """
 *         Quadratic form == (rev(x)*x)(0).
 */

```

```
/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_79quad(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_8clifford_78quad[] = "\n        Quadratic form == (rev(x)*x)(0).\n\n        >>> print(clifford(\"1+{1}+{1,2}\").quad())\n        3.0\n        >>> print(clifford(\"1+{-1}+{1,2}+{1,2,3}\").quad())\n        2.0\n        ";
static PyObject *__pyx_pw_8PyClical_8clifford_79quad(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("quad (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_78quad(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_78quad(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("quad", 0);

  /* "PyClical.pyx":1162
 *         2.0
 *         """
 *         return self.instance.quad()             # <<<<<<<<<<<<<<
 *
 *     def norm(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->quad()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1162, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":1153
 *         return clifford().wrap( self.instance.conj() )
 *
 *     def quad(self):             # <<<<<<<<<<<<<<
 *         """
 *         Quadratic form == (rev(x)*x)(0).
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.clifford.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1164
 *         return self.instance.quad()
 *
 *     def norm(self):             # <<<<<<<<<<<<<<
 *         """
 *         Norm == sum of squares of coordinates.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_81norm(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_8clifford_80norm[] = "\n        Norm == sum of squares of coordinates.\n\n        >>> clifford(\"1+{1}+{1,2}\").norm()\n        3.0\n        >>> clifford(\"1+{-1}+{1,2}+{1,2,3}\").norm()\n        4.0\n        ";
static PyObject *__pyx_pw_8PyClical_8clifford_81norm(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("norm (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_80norm(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
```

```
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_80norm(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("norm", 0);

  /* "PyClical.pyx":1173
 *         4.0
 *         """
 *         return self.instance.norm()             # <<<<<<<<<<<<<<
 *
 *     def abs(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->norm()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1173, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":1164
 *         return self.instance.quad()
 *
 *     def norm(self):             # <<<<<<<<<<<<<<
 *         """
 *         Norm == sum of squares of coordinates.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.clifford.norm", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1175
 *         return self.instance.norm()
 *
 *     def abs(self):             # <<<<<<<<<<<<<<
 *         """
 *         Absolute value: square root of norm.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_83abs(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
static char __pyx_doc_8PyClical_8clifford_82abs[] = "\n        Absolute value: square root of
norm.\n\n        >>> clifford(\"1+{-1}+{1,2}+{1,2,3}\").abs()\n        2.0\n        ";
static PyObject *__pyx_pw_8PyClical_8clifford_83abs(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("abs (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_82abs(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_82abs(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("abs", 0);

  /* "PyClical.pyx":1182
 *         2.0
 *         """
 *         return glucat.abs( self.unwrap() )             # <<<<<<<<<<<<<<
```

```
 *
 *     def max_abs(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = PyFloat_FromDouble(abs(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1182, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1175
 *         return self.instance.norm()
 *
 *     def abs(self):              # «««««««
 *         """
 *         Absolute value: square root of norm.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1184
 *         return glucat.abs( self.unwrap() )
 *
 *     def max_abs(self):              # «««««««
 *         """
 *         Maximum of absolute values of components of multivector: multivector infinity norm.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_85max_abs(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_84max_abs[] = "\n        Maximum of absolute values of components of multivector: multivector infinity norm.\n\n        »> clifford(\"1+{-1}+{1,2}+{1,2,3}\").max_abs()\n        1.0\n        »> clifford(\"3+2{1}+{1,2}\").max_abs()\n        3.0\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_85max_abs(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("max_abs (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_84max_abs(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_84max_abs(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("max_abs", 0);

          /* "PyClical.pyx":1193
 *         3.0
 *         """
 *         return self.instance.max_abs()              # «««««««
 *
 *     def truncated(self, limit):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->max_abs()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1193, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1184
 *         return glucat.abs( self.unwrap() )
 *
 *     def max_abs(self):              # «««««««
```

```
 *         """
 *         Maximum of absolute values of components of multivector: multivector infinity norm.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.max_abs", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1195
 *         return self.instance.max_abs()
 *
 *     def truncated(self, limit):             # <<<<<<<<
 *         """
 *         Remove all terms of self with relative size smaller than limit.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_87truncated(PyObject *__pyx_v_self, PyObject
*__pyx_v_limit); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_86truncated[] = "\n        Remove all terms of self
with relative size smaller than limit.\n\n        >>>
clifford(\"1e8+{1}+1e-8{1,2}\").truncated(1.0e-6)\n        clifford(\"100000000\")\n        >>>
clifford(\"1e4+{1}+1e-4{1,2}\").truncated(1.0e-6)\n        clifford(\"10000+{1}\")\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_87truncated(PyObject *__pyx_v_self, PyObject
*__pyx_v_limit) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("truncated (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_86truncated(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_limit));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_86truncated(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_limit) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          scalar_t __pyx_t_2;
          PyObject *__pyx_t_3 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("truncated", 0);

          /* "PyClical.pyx":1204
 *         clifford("10000+{1}")
 *         """
 *         return clifford().wrap( self.instance.truncated(limit) )             # <<<<<<<<
 *
 *     def isinf(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1204, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_PyFloat_AsDouble(__pyx_v_limit); if (unlikely((__pyx_t_2 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1204, __pyx_L1_error)
          __pyx_t_3 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->truncated(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0,
1204, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_3);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_3;
          __pyx_t_3 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1195
 *         return self.instance.max_abs()
 *
 *     def truncated(self, limit):             # <<<<<<<<
 *         """
 *         Remove all terms of self with relative size smaller than limit.
 */

          /* function exit code */
```

```
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_XDECREF(__pyx_t_3);
    __Pyx_AddTraceback("PyClical.clifford.truncated", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":1206
 *         return clifford().wrap( self.instance.truncated(limit) )
 *
 *     def isinf(self):             # «««««««
 *         """
 *         Check if a multivector contains any infinite values.
 */

  /* Python wrapper */
  static PyObject *__pyx_pw_8PyClical_8clifford_89isinf(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused); /*proto*/
  static char __pyx_doc_8PyClical_8clifford_88isinf[] = "\n        Check if a multivector contains any infinite values.\n\n        >> clifford().isinf()\n        False\n        ";
  static PyObject *__pyx_pw_8PyClical_8clifford_89isinf(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject *unused) {
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("isinf (wrapper)", 0);
    __pyx_r = __pyx_pf_8PyClical_8clifford_88isinf(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_8clifford_88isinf(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("isinf", 0);

    /* "PyClical.pyx":1213
 *         False
 *         """
 *         return self.instance.isnan()             # «««««««
 *
 *     def isnan(self):
 */
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_1 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->isnan()); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1213, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_r = __pyx_t_1;
    __pyx_t_1 = 0;
    goto __pyx_L0;

    /* "PyClical.pyx":1206
 *         return clifford().wrap( self.instance.truncated(limit) )
 *
 *     def isinf(self):             # «««««««
 *         """
 *         Check if a multivector contains any infinite values.
 */

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
    __Pyx_AddTraceback("PyClical.clifford.isinf", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __pyx_r = NULL;
    __pyx_L0:;
    __Pyx_XGIVEREF(__pyx_r);
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  /* "PyClical.pyx":1215
 *         return self.instance.isnan()
 *
 *     def isnan(self):             # «««««««
 *         """
```

```
 *         Check if a multivector contains any IEEE NaN values.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_91isnan(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_90isnan[] = "\n        Check if a multivector
contains any IEEE NaN values.\n\n        >>> clifford().isnan()\n        False\n        ";
        static PyObject *__pyx_pw_8PyClical_8clifford_91isnan(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("isnan (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_90isnan(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_90isnan(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("isnan", 0);

          /* "PyClical.pyx":1222
 *         False
 *         """
 *         return self.instance.isnan()             # <<<<<<<<<<<<<<
 *
 *     def frame(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->isnan()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1222, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1215
 *         return self.instance.isnan()
 *
 *     def isnan(self):             # <<<<<<<<<<<<<<
 *         """
 *         Check if a multivector contains any IEEE NaN values.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.isnan", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1224
 *         return self.instance.isnan()
 *
 *     def frame(self):             # <<<<<<<<<<<<<<
 *         """
 *         Subalgebra generated by all generators of terms of given multivector.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_93frame(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_92frame[] = "\n        Subalgebra generated by all
generators of terms of given multivector.\n\n        >>>
print(clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame())\n        {-2,-1,1,2,7}\n        >>>
s=clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame(); type(s)\n        <class 'PyClical.index_set'>\n
";
        static PyObject *__pyx_pw_8PyClical_8clifford_93frame(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("frame (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_92frame(((struct __pyx_obj_8PyClical_clifford
```

```
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_92frame(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          PyObject *__pyx_t_2 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("frame", 0);

          /* "PyClical.pyx":1233
 *         <class 'PyClical.index_set'>
 *         """
 *         return index_set().wrap( self.instance.frame() )             # <<<<<<<<<<<<<<
 *
 *     def __repr__(self):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1233, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set
*)__pyx_t_1), __pyx_v_self->instance->frame()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1233,
__pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_2);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __pyx_r = __pyx_t_2;
          __pyx_t_2 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1224
 *         return self.instance.isnan()
 *
 *     def frame(self):             # <<<<<<<<<<<<<<
 *         """
 *         Subalgebra generated by all generators of terms of given multivector.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_XDECREF(__pyx_t_2);
          __Pyx_AddTraceback("PyClical.clifford.frame", __pyx_clineno, __pyx_lineno, __pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "PyClical.pyx":1235
 *         return index_set().wrap( self.instance.frame() )
 *
 *     def __repr__(self):             # <<<<<<<<<<<<<<
 *         """
 *         The official string representation of self.
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_95__repr__(PyObject *__pyx_v_self); /*proto*/
        static char __pyx_doc_8PyClical_8clifford_94__repr__[] = "\n        The
\342\200\234official\342\200\235 string representation of self.\n\n        >>>
clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").__repr__()\n        'clifford(\"1+3{-1}+2{1,2}+4{-2,7}\")'\n
";
        #if CYTHON_COMPILING_IN_CPYTHON
        struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_94__repr__;
        #endif
        static PyObject *__pyx_pw_8PyClical_8clifford_95__repr__(PyObject *__pyx_v_self) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__repr__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_94__repr__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

```

```
static PyObject *__pyx_pf_8PyClical_8clifford_94__repr__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__repr__", 0);

  /* "PyClical.pyx":1242
 *         'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
 *         """
 *         return clifford_to_repr( self.unwrap() ).decode()             # <<<<<<<<<<<<<<
 *
 *     def __str__(self):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __Pyx_decode_cpp_string(clifford_to_repr(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self)), 0, PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1242, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":1235
 *         return index_set().wrap( self.instance.frame() )
 *
 *     def __repr__(self):             # <<<<<<<<<<<<<<
 *         """
 *         The official string representation of self.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.clifford.__repr__", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1244
 *         return clifford_to_repr( self.unwrap() ).decode()
 *
 *     def __str__(self):             # <<<<<<<<<<<<<<
 *         """
 *         The informal string representation of self.
 */

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_8clifford_97__str__(PyObject *__pyx_v_self); /*proto*/
static char __pyx_doc_8PyClical_8clifford_96__str__[] = "\n        The \342\200\234informal\342\200\235 string representation of self.\n\n        >>> clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").__str__()\n        '1+3{-1}+2{1,2}+4{-2,7}'\n        ";
#if CYTHON_COMPILING_IN_CPYTHON
struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_96__str__;
#endif
static PyObject *__pyx_pw_8PyClical_8clifford_97__str__(PyObject *__pyx_v_self) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__str__ (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_8clifford_96__str__(((struct __pyx_obj_8PyClical_clifford *)__pyx_v_self));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_8clifford_96__str__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__str__", 0);

  /* "PyClical.pyx":1251
 *         '1+3{-1}+2{1,2}+4{-2,7}'
 *         """
```

```
 *         return clifford_to_str( self.unwrap() ).decode()             # «««««««
 *
 * def clifford_hidden_doctests():
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_1 =
__Pyx_decode_cpp_string(clifford_to_str(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self)), 0,
PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1251, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __pyx_r = __pyx_t_1;
          __pyx_t_1 = 0;
          goto __pyx_L0;

          /* "PyClical.pyx":1244
 *         return clifford_to_repr( self.unwrap() ).decode()
 *
 *     def __str__(self):             # «««««««
 *         """
 *         The informal string representation of self.
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
          __Pyx_AddTraceback("PyClical.clifford.__str__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
          __pyx_r = NULL;
          __pyx_L0:;
          __Pyx_XGIVEREF(__pyx_r);
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        /* "(tree fragment)":1
 * def __reduce_cython__(self):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 */

        /* Python wrapper */
        static PyObject *__pyx_pw_8PyClical_8clifford_99__reduce_cython__(PyObject *__pyx_v_self,
CYTHON_UNUSED PyObject *unused); /*proto*/
        static PyObject *__pyx_pw_8PyClical_8clifford_99__reduce_cython__(PyObject *__pyx_v_self,
CYTHON_UNUSED PyObject *unused) {
          PyObject *__pyx_r = 0;
          __Pyx_RefNannyDeclarations
          __Pyx_RefNannySetupContext("__reduce_cython__ (wrapper)", 0);
          __pyx_r = __pyx_pf_8PyClical_8clifford_98__reduce_cython__(((struct
__pyx_obj_8PyClical_clifford *)__pyx_v_self));

          /* function exit code */
          __Pyx_RefNannyFinishContext();
          return __pyx_r;
        }

        static PyObject *__pyx_pf_8PyClical_8clifford_98__reduce_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self) {
          PyObject *__pyx_r = NULL;
          __Pyx_RefNannyDeclarations
          PyObject *__pyx_t_1 = NULL;
          int __pyx_lineno = 0;
          const char *__pyx_filename = NULL;
          int __pyx_clineno = 0;
          __Pyx_RefNannySetupContext("__reduce_cython__", 0);

          /* "(tree fragment)":2
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # «««««««
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */
          __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__11, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 2, __pyx_L1_error)
          __Pyx_GOTREF(__pyx_t_1);
          __Pyx_Raise(__pyx_t_1, 0, 0, 0);
          __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
          __PYX_ERR(1, 2, __pyx_L1_error)

          /* "(tree fragment)":1
 * def __reduce_cython__(self):             # «««««««
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 */

          /* function exit code */
          __pyx_L1_error:;
          __Pyx_XDECREF(__pyx_t_1);
```

```
        __Pyx_AddTraceback("PyClical.clifford.__reduce_cython__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        __pyx_r = NULL;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "(tree fragment)":3
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):             # <<<<<<<<<<<<<<
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_8clifford_101__setstate_cython__(PyObject *__pyx_v_self,
PyObject *__pyx_v___pyx_state); /*proto*/
      static PyObject *__pyx_pw_8PyClical_8clifford_101__setstate_cython__(PyObject *__pyx_v_self,
PyObject *__pyx_v___pyx_state) {
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("__setstate_cython__ (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_8clifford_100__setstate_cython__(((struct
__pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v___pyx_state));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_8clifford_100__setstate_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v___pyx_state) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__setstate_cython__", 0);

        /* "(tree fragment)":4
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # <<<<<<<<<<<<<<
 */
        __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__12, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 4, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __Pyx_Raise(__pyx_t_1, 0, 0, 0);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __PYX_ERR(1, 4, __pyx_L1_error)

        /* "(tree fragment)":3
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):             # <<<<<<<<<<<<<<
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.clifford.__setstate_cython__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
        __pyx_r = NULL;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1253
 *         return clifford_to_str( self.unwrap() ).decode()
 *
 * def clifford_hidden_doctests():             # <<<<<<<<<<<<<<
 *     """
 *     Tests for functions that Doctest cannot see.
 */

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_9clifford_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused); /*proto*/
      static char __pyx_doc_8PyClical_8clifford_hidden_doctests[] = "\n    Tests for functions that
Doctest cannot see.\n\n    For clifford.__cinit__: Construct an object of type clifford.\n\n    >>>
print(clifford(2))\n    2\n    >>> print(clifford(2.0))\n    2\n    >>> print(clifford(1.0e-1))\n
0.1\n    >>> print(clifford(\"2\"))\n    2\n    >>> print(clifford(\"2{1,2,3}\"))\n    2{1,2,3}\n    >>>
print(clifford(clifford(\"2{1,2,3}\")))\n    2{1,2,3}\n    >>> print(clifford(\"-{1}\"))\n    -{1}\n
```

```
»> print(clifford(2,index_set({1,2})))\n    2{1,2}\n    »> print(clifford([2,3],index_set({1,2})))\n
2{1}+3{2}\n    »> print(clifford([1,2]))\n    Traceback (most recent call last):\n      ...\n
TypeError: Cannot initialize clifford object from <class ’list’>.\n    »> print(clifford(None))\n
Traceback (most recent call last):\n      ...\n    TypeError: Cannot initialize clifford object from
<class ’NoneType’>.\n    »> print(clifford(None,[1,2]))\n    Traceback (most recent call last):\n
...\n    TypeError: Cannot initialize clifford object from (<class ’NoneType’>, <class ’list’>).\n
»> print(clifford([1,2],[1,2]))\n    Traceback (most recent call last):\n      ...\n    TypeError:
Cannot initialize clifford object from (<class ’list’>, <class ’list’>).\n    »>
print(clifford(\"\"))\n    Traceback (most recent call last):\n      ...\n    ValueError: Cannot
initialize clifford object from invalid string ”.\n    »> print(clifford(\"{\"))\n    Traceback (most
recent call last):\n      ...\n    ValueError: Cannot initialize clifford object from invalid string
’{’.\n    »> print(clifford(\"{1\"))\n    Traceback (most recent call last):\n      ...\n
ValueError: Cannot initialize clifford object from invalid string ’{1’.\n    »>
print(clifford(\"+\"))\n    Traceback (most recent call last):\n      ...\n    ValueError: Cannot
initialize clifford object from invalid string ’+’.\n    »> print(clifford(\"-\"))\n    Traceback
(most recent call last):\n      ...\n    ValueError: Cannot initialize clifford object fro""m invalid
string ’-’.\n    »> print(clifford(\"{1}+\"))\n    Traceback (most recent call last):\n      ...\n
ValueError: Cannot initialize clifford object from invalid string ’{1}+’.\n\n    For
clifford.__richcmp__: Compare objects of type clifford.\n\n    »> clifford(\"{1}\") ==
clifford(\"1{1}\")\n    True\n    »> clifford(\"{1}\") != clifford(\"1.0{1}\")\n    False\n    »>
clifford(\"{1}\") != clifford(\"1.0\")\n    True\n    »> clifford(\"{1,2}\") == None\n    False\n
»> clifford(\"{1,2}\") != None\n    True\n    »> None == clifford(\"{1,2}\")\n    False\n    »> None
!= clifford(\"{1,2}\")\n    True\n    ";
                static PyMethodDef __pyx_mdef_8PyClical_9clifford_hidden_doctests =
{"clifford_hidden_doctests", (PyCFunction)__pyx_pw_8PyClical_9clifford_hidden_doctests, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_hidden_doctests};
                static PyObject *__pyx_pw_8PyClical_9clifford_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("clifford_hidden_doctests (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_8clifford_hidden_doctests(__pyx_self);

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_8clifford_hidden_doctests(CYTHON_UNUSED PyObject
*__pyx_self) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("clifford_hidden_doctests", 0);

                  /* "PyClical.pyx":1335
 *     True
 *     """
 *     return              # <<<<<<<<<<<<<<
 *
 * cpdef inline error_squared_tol(obj):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_r = Py_None; __Pyx_INCREF(Py_None);
                  goto __pyx_L0;

                  /* "PyClical.pyx":1253
 *         return clifford_to_str( self.unwrap() ).decode()
 *
 * def clifford_hidden_doctests():             # <<<<<<<<<<<<<<
 *     """
 *     Tests for functions that Doctest cannot see.
 */

                  /* function exit code */
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1337
 *     return
 *
 * cpdef inline error_squared_tol(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Quadratic norm error tolerance relative to a specific multivector.
 */

                static PyObject *__pyx_pw_8PyClical_11error_squared_tol(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_error_squared_tol(PyObject
*__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
```

```
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("error_squared_tol", 0);

                    /* "PyClical.pyx":1344
 *     0.0
 *     """
 *     return glucat.error_squared_tol(toClifford(obj))             # «««««««
 *
 * cpdef inline error_squared(lhs, rhs, threshold):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = PyFloat_FromDouble(error_squared_tol(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1344, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1337
 *     return
 *
 * cpdef inline error_squared_tol(obj):             # «««««««
 *     """
 *     Quadratic norm error tolerance relative to a specific multivector.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.error_squared_tol", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_11error_squared_tol(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_10error_squared_tol[] = "\n    Quadratic norm error tolerance relative to a specific multivector.\n\n    »> print(error_squared_tol(clifford(\"{1}\")) * 3.0 - error_squared_tol(clifford(\"1{1}-2{2}+3{3}\")))\n    0.0\n    ";
                  static PyObject *__pyx_pw_8PyClical_11error_squared_tol(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("error_squared_tol (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_10error_squared_tol(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_10error_squared_tol(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("error_squared_tol", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __pyx_f_8PyClical_error_squared_tol(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1337, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.error_squared_tol", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }
```

```
                /* "PyClical.pyx":1346
 *     return glucat.error_squared_tol(toClifford(obj))
 *
 * cpdef inline error_squared(lhs, rhs, threshold):             # «««««««
 *     """
 *     Relative or absolute error using the quadratic norm.
 */

                static PyObject *__pyx_pw_8PyClical_13error_squared(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_error_squared(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  scalar_t __pyx_t_1;
                  PyObject *__pyx_t_2 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("error_squared", 0);

                  /* "PyClical.pyx":1357
 *     25.0
 *     """
 *     return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)             # «««««««
 *
 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_PyFloat_AsDouble(__pyx_v_threshold); if (unlikely((__pyx_t_1 == ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1357, __pyx_L1_error)
                  __pyx_t_2 = PyFloat_FromDouble(error_squared(__pyx_f_8PyClical_toClifford(__pyx_v_lhs), __pyx_f_8PyClical_toClifford(__pyx_v_rhs), ((scalar_t)__pyx_t_1))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1357, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_r = __pyx_t_2;
                  __pyx_t_2 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1346
 *     return glucat.error_squared_tol(toClifford(obj))
 *
 * cpdef inline error_squared(lhs, rhs, threshold):             # «««««««
 *     """
 *     Relative or absolute error using the quadratic norm.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_AddTraceback("PyClical.error_squared", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_13error_squared(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_12error_squared[] = "\n    Relative or absolute error using the quadratic norm.\n\n    >>> err2=scalar_epsilon*scalar_epsilon\n\n    >>> print(error_squared(clifford(\"{1}\"), clifford(\"1{1}\"), err2))\n    0.0\n    >>> print(error_squared(clifford(\"1{1}-3{2}+4{3}\"), clifford(\"{1}\"), err2))\n    25.0\n    ";
                static PyObject *__pyx_pw_8PyClical_13error_squared(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_lhs = 0;
                  PyObject *__pyx_v_rhs = 0;
                  PyObject *__pyx_v_threshold = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("error_squared (wrapper)", 0);
                  {
                    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_lhs,&__pyx_n_s_rhs,&__pyx_n_s_threshold,0};
                    PyObject* values[3] = {0,0,0};
                    if (unlikely(__pyx_kwds)) {
                      Py_ssize_t kw_args;
                      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
                      switch (pos_args) {
```

```
        case  3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
        CYTHON_FALLTHROUGH;
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) != 0)) kw_args--;
        else {
          __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, 1); __PYX_ERR(0, 1346, __pyx_L3_error)
        }
        CYTHON_FALLTHROUGH;
        case  2:
        if (likely((values[2] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_threshold)) != 0)) kw_args--;
        else {
          __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, 2); __PYX_ERR(0, 1346, __pyx_L3_error)
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "error_squared") < 0)) __PYX_ERR(0, 1346, __pyx_L3_error)
      }
    } else if (PyTuple_GET_SIZE(__pyx_args) != 3) {
      goto __pyx_L5_argtuple_error;
    } else {
      values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
      values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
      values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
    }
    __pyx_v_lhs = values[0];
    __pyx_v_rhs = values[1];
    __pyx_v_threshold = values[2];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1346, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.error_squared", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_12error_squared(__pyx_self, __pyx_v_lhs, __pyx_v_rhs, __pyx_v_threshold);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_12error_squared(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("error_squared", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_error_squared(__pyx_v_lhs, __pyx_v_rhs, __pyx_v_threshold, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1346, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.error_squared", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":1359
 *     return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
 *
 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):             # «««««««
 *     """
 *     Test for approximate equality of multivectors.
 */

    static PyObject *__pyx_pw_8PyClical_15approx_equal(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
    static CYTHON_INLINE PyObject *__pyx_f_8PyClical_approx_equal(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_approx_equal *__pyx_optional_args) {
      PyObject *__pyx_v_threshold = ((PyObject *)Py_None);
      PyObject *__pyx_v_tol = ((PyObject *)Py_None);
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      int __pyx_t_2;
      PyObject *__pyx_t_3 = NULL;
      scalar_t __pyx_t_4;
      scalar_t __pyx_t_5;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("approx_equal", 0);
      if (__pyx_optional_args) {
        if (__pyx_optional_args->__pyx_n > 0) {
          __pyx_v_threshold = __pyx_optional_args->threshold;
          if (__pyx_optional_args->__pyx_n > 1) {
            __pyx_v_tol = __pyx_optional_args->tol;
          }
        }
      }
      __Pyx_INCREF(__pyx_v_threshold);
      __Pyx_INCREF(__pyx_v_tol);

      /* "PyClical.pyx":1374
 *     True
 *     """
 *     threshold = error_squared_tol(rhs) if threshold is None else threshold              # «««««««
 *     tol       = error_squared_tol(rhs) if tol       is None else tol
 *     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
 */
      __pyx_t_2 = (__pyx_v_threshold == Py_None);
      if ((__pyx_t_2 != 0)) {
        __pyx_t_3 = __pyx_f_8PyClical_error_squared_tol(__pyx_v_rhs, 0); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1374, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_3);
        __pyx_t_1 = __pyx_t_3;
        __pyx_t_3 = 0;
      } else {
        __Pyx_INCREF(__pyx_v_threshold);
        __pyx_t_1 = __pyx_v_threshold;
      }
      __Pyx_DECREF_SET(__pyx_v_threshold, __pyx_t_1);
      __pyx_t_1 = 0;

      /* "PyClical.pyx":1375
 *     """
 *     threshold = error_squared_tol(rhs) if threshold is None else threshold
 *     tol       = error_squared_tol(rhs) if tol       is None else tol              # «««««««
 *     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
 *
 */
      __pyx_t_2 = (__pyx_v_tol == Py_None);
      if ((__pyx_t_2 != 0)) {
        __pyx_t_3 = __pyx_f_8PyClical_error_squared_tol(__pyx_v_rhs, 0); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1375, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_3);
        __pyx_t_1 = __pyx_t_3;
        __pyx_t_3 = 0;
      } else {
        __Pyx_INCREF(__pyx_v_tol);
        __pyx_t_1 = __pyx_v_tol;
      }
      __Pyx_DECREF_SET(__pyx_v_tol, __pyx_t_1);
      __pyx_t_1 = 0;
```

```
                    /* "PyClical.pyx":1376
 *     threshold = error_squared_tol(rhs) if threshold is None else threshold
 *     tol       = error_squared_tol(rhs) if tol       is None else tol
 *     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
<scalar_t>tol)             # «««««««
 *
 * cpdef inline inv(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_4 = __pyx_PyFloat_AsDouble(__pyx_v_threshold); if (unlikely((__pyx_t_4 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1376, __pyx_L1_error)
                    __pyx_t_5 = __pyx_PyFloat_AsDouble(__pyx_v_tol); if (unlikely((__pyx_t_5 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1376, __pyx_L1_error)
                    __pyx_t_1 =
__Pyx_PyBool_FromLong(approx_equal(__pyx_f_8PyClical_toClifford(__pyx_v_lhs),
__pyx_f_8PyClical_toClifford(__pyx_v_rhs), ((scalar_t)__pyx_t_4), ((scalar_t)__pyx_t_5))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1376, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1359
 *     return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
 *
 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):             # «««««««
 *     """
 *     Test for approximate equality of multivectors.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno,
__pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XDECREF(__pyx_v_threshold);
                    __Pyx_XDECREF(__pyx_v_tol);
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_15approx_equal(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
                  static char __pyx_doc_8PyClical_14approx_equal[] = "\n    Test for approximate
equality of multivectors.\n\n    >>> err2=scalar_epsilon*scalar_epsilon\n\n    >>>
print(approx_equal(clifford(\"{1}\"), clifford(\"1{1}\")))\n    True\n    >>>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}\"), clifford(\"{1}\")))\n    False\n    >>>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}+0.001\"), clifford(\"1{1}-3{2}+4{3}\"), err2, err2))\n
False\n    >>> print(approx_equal(clifford(\"1{1}-3{2}+4{3}+1.0e-30\"), clifford(\"1{1}-3{2}+4{3}\"),
err2, err2))\n    True\n    ";
                  static PyObject *__pyx_pw_8PyClical_15approx_equal(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds) {
                    PyObject *__pyx_v_lhs = 0;
                    PyObject *__pyx_v_rhs = 0;
                    PyObject *__pyx_v_threshold = 0;
                    PyObject *__pyx_v_tol = 0;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("approx_equal (wrapper)", 0);
                    {
                      static PyObject **__pyx_pyargnames[] =
{&__pyx_n_s_lhs,&__pyx_n_s_rhs,&__pyx_n_s_threshold,&__pyx_n_s_tol,0};
                      PyObject* values[4] = {0,0,0,0};
                      values[2] = ((PyObject *)Py_None);
                      values[3] = ((PyObject *)Py_None);
                      if (unlikely(__pyx_kwds)) {
                        Py_ssize_t kw_args;
                        const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
                        switch (pos_args) {
                          case  4: values[3] = PyTuple_GET_ITEM(__pyx_args, 3);
                          CYTHON_FALLTHROUGH;
                          case  3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
                          CYTHON_FALLTHROUGH;
                          case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                          CYTHON_FALLTHROUGH;
                          case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                          CYTHON_FALLTHROUGH;
                          case  0: break;
```

```
                    default: goto __pyx_L5_argtuple_error;
                  }
                  kw_args = PyDict_Size(__pyx_kwds);
                  switch (pos_args) {
                    case  0:
                    if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) !=
0)) kw_args--;
                    else goto __pyx_L5_argtuple_error;
                    CYTHON_FALLTHROUGH;
                    case  1:
                    if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) !=
0)) kw_args--;
                    else {
                      __Pyx_RaiseArgtupleInvalid("approx_equal", 0, 2, 4, 1); __PYX_ERR(0, 1359,
__pyx_L3_error)
                    }
                    CYTHON_FALLTHROUGH;
                    case  2:
                    if (kw_args > 0) {
                      PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_threshold);
                      if (value) { values[2] = value; kw_args--; }
                    }
                    CYTHON_FALLTHROUGH;
                    case  3:
                    if (kw_args > 0) {
                      PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_tol);
                      if (value) { values[3] = value; kw_args--; }
                    }
                  }
                  if (unlikely(kw_args > 0)) {
                    if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "approx_equal") < 0)) __PYX_ERR(0, 1359, __pyx_L3_error)
                  }
                } else {
                  switch (PyTuple_GET_SIZE(__pyx_args)) {
                    case  4: values[3] = PyTuple_GET_ITEM(__pyx_args, 3);
                    CYTHON_FALLTHROUGH;
                    case  3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
                    CYTHON_FALLTHROUGH;
                    case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                    values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                    break;
                    default: goto __pyx_L5_argtuple_error;
                  }
                }
                __pyx_v_lhs = values[0];
                __pyx_v_rhs = values[1];
                __pyx_v_threshold = values[2];
                __pyx_v_tol = values[3];
              }
              goto __pyx_L4_argument_unpacking_done;
              __pyx_L5_argtuple_error:;
              __Pyx_RaiseArgtupleInvalid("approx_equal", 0, 2, 4, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1359, __pyx_L3_error)
              __pyx_L3_error:;
              __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno,
__pyx_filename);
              __Pyx_RefNannyFinishContext();
              return NULL;
              __pyx_L4_argument_unpacking_done:;
              __pyx_r = __pyx_pf_8PyClical_14approx_equal(__pyx_self, __pyx_v_lhs, __pyx_v_rhs,
__pyx_v_threshold, __pyx_v_tol);

              /* function exit code */
              __Pyx_RefNannyFinishContext();
              return __pyx_r;
            }

            static PyObject *__pyx_pf_8PyClical_14approx_equal(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, PyObject *__pyx_v_tol) {
              PyObject *__pyx_r = NULL;
              __Pyx_RefNannyDeclarations
              PyObject *__pyx_t_1 = NULL;
              struct __pyx_opt_args_8PyClical_approx_equal __pyx_t_2;
              int __pyx_lineno = 0;
              const char *__pyx_filename = NULL;
              int __pyx_clineno = 0;
              __Pyx_RefNannySetupContext("approx_equal", 0);
              __Pyx_XDECREF(__pyx_r);
              __pyx_t_2.__pyx_n = 2;
              __pyx_t_2.threshold = __pyx_v_threshold;
              __pyx_t_2.tol = __pyx_v_tol;
              __pyx_t_1 = __pyx_f_8PyClical_approx_equal(__pyx_v_lhs, __pyx_v_rhs, 0, &__pyx_t_2);
if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1359, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_1);
              __pyx_r = __pyx_t_1;
              __pyx_t_1 = 0;
```

```
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1378
 *     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
 *
 * cpdef inline inv(obj):             # «««««««
 *     """
 *     Geometric multiplicative inverse.
 */

                  static PyObject *__pyx_pw_8PyClical_17inv(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_inv(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    PyObject *__pyx_t_2 = NULL;
                    PyObject *__pyx_t_3 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("inv", 0);

                    /* "PyClical.pyx":1391
 *     nan
 *     """
 *     return clifford(obj).inv()             # «««««««
 *
 * cpdef inline scalar(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1391, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_2);
                    __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_inv); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1391, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                    __pyx_t_2 = NULL;
                    if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                      __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                      if (likely(__pyx_t_2)) {
                        PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                        __Pyx_INCREF(__pyx_t_2);
                        __Pyx_INCREF(function);
                        __Pyx_DECREF_SET(__pyx_t_3, function);
                      }
                    }
                    __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                    __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                    if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1391, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1378
 *     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
 *
 * cpdef inline inv(obj):             # «««««««
 *     """
 *     Geometric multiplicative inverse.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_XDECREF(__pyx_t_2);
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_AddTraceback("PyClical.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
        __pyx_r = 0;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_17inv(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
      static char __pyx_doc_8PyClical_16inv[] = "\n    Geometric multiplicative inverse.\n\n
>>> print(inv(clifford(\"{1}\")))\n    {1}\n    >>> print(inv(clifford(\"{-1}\")))\n    -{-1}\n    >>>
print(inv(clifford(\"{-2,-1}\")))\n    -{-2,-1}\n    >>> print(inv(clifford(\"{-1}+{1}\")))\n    nan\n
";
      static PyObject *__pyx_pw_8PyClical_17inv(PyObject *__pyx_self, PyObject *__pyx_v_obj)
{
        PyObject *__pyx_r = 0;
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("inv (wrapper)", 0);
        __pyx_r = __pyx_pf_8PyClical_16inv(__pyx_self, ((PyObject *)__pyx_v_obj));

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_16inv(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("inv", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_inv(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1378, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1393
 *     return clifford(obj).inv()
 *
 * cpdef inline scalar(obj):             # <<<<<<<<
 *     """
 *     Scalar part.
 */

      static PyObject *__pyx_pw_8PyClical_19scalar(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_scalar(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        PyObject *__pyx_t_3 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("scalar", 0);

        /* "PyClical.pyx":1402
 *     0.0
 *     """
 *     return clifford(obj).scalar()             # <<<<<<<<
 *
 * cpdef inline real(obj):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1402, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
```

```
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_scalar); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1402, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1402, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1393
 *     return clifford(obj).inv()
 *
 * cpdef inline scalar(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Scalar part.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_19scalar(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_18scalar[] = "\n    Scalar part.\n\n    >>> scalar(clifford(\"1+{1}+{1,2}\"))\n    1.0\n    >>> scalar(clifford(\"{1,2}\"))\n    0.0\n    ";
                static PyObject *__pyx_pw_8PyClical_19scalar(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("scalar (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_18scalar(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_18scalar(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("scalar", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_scalar(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1393, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
```

```
        return __pyx_r;
      }

      /* "PyClical.pyx":1404
 *     return clifford(obj).scalar()
 *
 * cpdef inline real(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Real part: synonym for scalar part.
 */

      static PyObject *__pyx_pw_8PyClical_21real(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_real(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        PyObject *__pyx_t_3 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("real", 0);

        /* "PyClical.pyx":1413
 *     0.0
 *     """
 *     return clifford(obj).scalar()             # <<<<<<<<<<<<<<
 *
 * cpdef inline imag(obj):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1413, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_scalar); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1413, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_3);
        __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
        __pyx_t_2 = NULL;
        if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
          __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
          if (likely(__pyx_t_2)) {
            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
            __Pyx_INCREF(__pyx_t_2);
            __Pyx_INCREF(function);
            __Pyx_DECREF_SET(__pyx_t_3, function);
          }
        }
        __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
        __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
        if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1413, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":1404
 *     return clifford(obj).scalar()
 *
 * cpdef inline real(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Real part: synonym for scalar part.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_XDECREF(__pyx_t_3);
        __Pyx_AddTraceback("PyClical.real", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = 0;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* Python wrapper */
      static PyObject *__pyx_pw_8PyClical_21real(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
      static char __pyx_doc_8PyClical_20real[] = "\n    Real part: synonym for scalar part.\n\n    >>> real(clifford(\"1+{1}+{1,2}\"))\n    1.0\n    >>> real(clifford(\"{1,2}\"))\n    0.0\n    ";
```

```
static PyObject *__pyx_pw_8PyClical_21real(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("real (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_20real(__pyx_self, ((PyObject *)__pyx_v_obj));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_20real(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("real", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_real(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1404, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.real", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1415
 *     return clifford(obj).scalar()
 *
 * cpdef inline imag(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Imaginary part: deprecated (always 0).
 */

static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_imag(CYTHON_UNUSED PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("imag", 0);

  /* "PyClical.pyx":1424
 *     0.0
 *     """
 *     return 0.0             # <<<<<<<<<<<<<<
 *
 * cpdef inline pure(obj):
 */
  __Pyx_XDECREF(__pyx_r);
  __Pyx_INCREF(__pyx_float_0_0);
  __pyx_r = __pyx_float_0_0;
  goto __pyx_L0;

  /* "PyClical.pyx":1415
 *     return clifford(obj).scalar()
 *
 * cpdef inline imag(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Imaginary part: deprecated (always 0).
 */

  /* function exit code */
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static char __pyx_doc_8PyClical_22imag[] = "\n    Imaginary part: deprecated (always
```

```
0).\n\n    >> imag(clifford(\"1+{1}+{1,2}\"))\n    0.0\n    >> imag(clifford(\"{1,2}\"))\n    0.0\n
";
                static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("imag (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_22imag(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_22imag(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("imag", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_imag(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1415, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.imag", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1426
 *      return 0.0
 *
 * cpdef inline pure(obj):             # «««««««
 *      """
 *      Pure part
 */

                static PyObject *__pyx_pw_8PyClical_25pure(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_pure(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("pure", 0);

                  /* "PyClical.pyx":1435
 *      {1,2}
 *      """
 *      return clifford(obj).pure()             # «««««««
 *
 * cpdef inline even(obj):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1435, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_pure); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1435, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
```

```
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1435, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1426
 *     return 0.0
 *
 * cpdef inline pure(obj):             # «««««««
 *     """
 *     Pure part
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_25pure(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_24pure[] = "\n    Pure part\n\n    »> print(pure(clifford(\"1+{1}+{1,2}\")))\n    {1}+{1,2}\n    »> print(pure(clifford(\"{1,2}\")))\n    {1,2}\n    ";
                static PyObject *__pyx_pw_8PyClical_25pure(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("pure (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_24pure(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_24pure(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("pure", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_pure(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1426, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1437
 *     return clifford(obj).pure()
 *
 * cpdef inline even(obj):             # «««««««
 *     """
 *     Even part of multivector, sum of even grade terms.
 */

```

```
static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_even(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("even", 0);

  /* "PyClical.pyx":1444
 *     1+{1,2}
 *     """
 *     return clifford(obj).even()             # <<<<<<<<<<<<<<
 *
 * cpdef inline odd(obj):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1444, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_even); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1444, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __pyx_t_2 = NULL;
  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
    if (likely(__pyx_t_2)) {
      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
      __Pyx_INCREF(__pyx_t_2);
      __Pyx_INCREF(function);
      __Pyx_DECREF_SET(__pyx_t_3, function);
    }
  }
  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1444, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":1437
 *     return clifford(obj).pure()
 *
 * cpdef inline even(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Even part of multivector, sum of even grade terms.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_XDECREF(__pyx_t_3);
  __Pyx_AddTraceback("PyClical.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = 0;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static char __pyx_doc_8PyClical_26even[] = "\n    Even part of multivector, sum of even grade terms.\n\n    >>> print(even(clifford(\"1+{1}+{1,2}\")))\n    1+{1,2}\n    ";
static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("even (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_26even(__pyx_self, ((PyObject *)__pyx_v_obj));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}
```

```
static PyObject *__pyx_pf_8PyClical_26even(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("even", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_even(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1437, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1446
 *     return clifford(obj).even()
 *
 * cpdef inline odd(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Odd part of multivector, sum of odd grade terms.
 */

static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_odd(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("odd", 0);

  /* "PyClical.pyx":1453
 *     {1}
 *     """
 *     return clifford(obj).odd()             # <<<<<<<<<<<<<<
 *
 * cpdef inline involute(obj):
 */
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1453, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_odd); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1453, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __pyx_t_2 = NULL;
  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
    if (likely(__pyx_t_2)) {
      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
      __Pyx_INCREF(__pyx_t_2);
      __Pyx_INCREF(function);
      __Pyx_DECREF_SET(__pyx_t_3, function);
    }
  }
  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1453, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* "PyClical.pyx":1446
 *     return clifford(obj).even()
```

```
 *
 * cpdef inline odd(obj):             # «««««««
 *     """
 *     Odd part of multivector, sum of odd grade terms.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_28odd[] = "\n    Odd part of multivector, sum of odd grade terms.\n\n    »> print(odd(clifford(\"1+{1}+{1,2}\")))\n    {1}\n    ";
                static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("odd (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_28odd(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_28odd(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("odd", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_odd(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1446, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1455
 *     return clifford(obj).odd()
 *
 * cpdef inline involute(obj):             # «««««««
 *     """
 *     Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
 */

                static PyObject *__pyx_pw_8PyClical_31involute(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_involute(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("involute", 0);

                  /* "PyClical.pyx":1468
```

```
 *     1-{1}+{1,2}
 *     """
 *     return clifford(obj).involute()             # <<<<<<<<<<<<<<
 *
 * cpdef inline reverse(obj):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1468, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_involute); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1468, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1468, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1455
 *     return clifford(obj).odd()
 *
 * cpdef inline involute(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.involute", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_31involute(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_30involute[] = "\n    Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})\n\n    >>> print(involute(clifford(\"{1}\")))\n    -{1}\n    >>> print(involute(clifford(\"{2}\") * clifford(\"{1}\")))\n    -{1,2}\n    >>> print(involute(clifford(\"{1}\") * clifford(\"{2}\")))\n    {1,2}\n    >>> print(involute(clifford(\"1+{1}+{1,2}\")))\n    1-{1}+{1,2}\n    ";
                static PyObject *__pyx_pw_8PyClical_31involute(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("involute (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_30involute(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_30involute(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("involute", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_involute(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
```

```
__PYX_ERR(0, 1455, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.involute", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1470
 *     return clifford(obj).involute()
 *
 * cpdef inline reverse(obj):             # «««««««
 *     """
 *     Reversion, eg. {1}*{2} -> {2}*{1}
 */

                static PyObject *__pyx_pw_8PyClical_33reverse(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_reverse(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("reverse", 0);

                  /* "PyClical.pyx":1483
 *     1+{1}-{1,2}
 *     """
 *     return clifford(obj).reverse()             # «««««««
 *
 * cpdef inline conj(obj):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1483, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_reverse); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1483, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1483, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1470
 *     return clifford(obj).involute()
 *
 * cpdef inline reverse(obj):             # «««««««
 *     """
 *     Reversion, eg. {1}*{2} -> {2}*{1}
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
```

```
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_AddTraceback("PyClical.reverse", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_33reverse(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_32reverse[] = "\n    Reversion, eg. {1}*{2} ->
{2}*{1}\n\n    >>> print(reverse(clifford(\"{1}\")))\n    {1}\n    >>> print(reverse(clifford(\"{2}\") *
clifford(\"{1}\")))\n    {1,2}\n    >>> print(reverse(clifford(\"{1}\") * clifford(\"{2}\")))\n
-{1,2}\n    >>> print(reverse(clifford(\"1+{1}+{1,2}\")))\n    1+{1}-{1,2}\n    ";
                  static PyObject *__pyx_pw_8PyClical_33reverse(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("reverse (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_32reverse(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_32reverse(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("reverse", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __pyx_f_8PyClical_reverse(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1470, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.reverse", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1485
 *     return clifford(obj).reverse()
 *
 * cpdef inline conj(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Conjugation, reverse o involute == involute o reverse.
 */

                  static PyObject *__pyx_pw_8PyClical_35conj(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_conj(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    PyObject *__pyx_t_2 = NULL;
                    PyObject *__pyx_t_3 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("conj", 0);

                    /* "PyClical.pyx":1498
 *     1-{1}-{1,2}
 *     """
 *     return clifford(obj).conj()             # <<<<<<<<<<<<<<
 *
 * cpdef inline quad(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
```

```
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1498, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_conj); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1498, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1498, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1485
 *     return clifford(obj).reverse()
 *
 * cpdef inline conj(obj):             # «««««««
 *     """
 *     Conjugation, reverse o involute == involute o reverse.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_35conj(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_34conj[] = "\n    Conjugation, reverse o involute == involute o reverse.\n\n    >>> print(conj(clifford(\"{1}\")))\n    -{1}\n    >>> print(conj(clifford(\"{2}\") * clifford(\"{1}\")))\n    {1,2}\n    >>> print(conj(clifford(\"{1}\") * clifford(\"{2}\")))\n    -{1,2}\n    >>> print(conj(clifford(\"1+{1}+{1,2}\")))\n    1-{1}-{1,2}\n    ";
                static PyObject *__pyx_pw_8PyClical_35conj(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("conj (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_34conj(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_34conj(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("conj", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_conj(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1485, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
```

```
                __Pyx_AddTraceback("PyClical.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
                __pyx_r = NULL;
                __pyx_L0:;
                __Pyx_XGIVEREF(__pyx_r);
                __Pyx_RefNannyFinishContext();
                return __pyx_r;
              }

              /* "PyClical.pyx":1500
 *      return clifford(obj).conj()
 *
 * cpdef inline quad(obj):              # <<<<<<<<<<<<<<
 *      """
 *      Quadratic form == (rev(x)*x)(0).
 */

              static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
              static CYTHON_INLINE PyObject *__pyx_f_8PyClical_quad(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
                PyObject *__pyx_r = NULL;
                __Pyx_RefNannyDeclarations
                PyObject *__pyx_t_1 = NULL;
                PyObject *__pyx_t_2 = NULL;
                PyObject *__pyx_t_3 = NULL;
                int __pyx_lineno = 0;
                const char *__pyx_filename = NULL;
                int __pyx_clineno = 0;
                __Pyx_RefNannySetupContext("quad", 0);

                /* "PyClical.pyx":1509
 *      2.0
 *      """
 *      return clifford(obj).quad()              # <<<<<<<<<<<<<<
 *
 * cpdef inline norm(obj):
 */
                __Pyx_XDECREF(__pyx_r);
                __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1509, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_2);
                __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_quad); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1509, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_3);
                __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                __pyx_t_2 = NULL;
                if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                  __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                  if (likely(__pyx_t_2)) {
                    PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                    __Pyx_INCREF(__pyx_t_2);
                    __Pyx_INCREF(function);
                    __Pyx_DECREF_SET(__pyx_t_3, function);
                  }
                }
                __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
                __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1509, __pyx_L1_error)
                __Pyx_GOTREF(__pyx_t_1);
                __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                __pyx_r = __pyx_t_1;
                __pyx_t_1 = 0;
                goto __pyx_L0;

                /* "PyClical.pyx":1500
 *      return clifford(obj).conj()
 *
 * cpdef inline quad(obj):              # <<<<<<<<<<<<<<
 *      """
 *      Quadratic form == (rev(x)*x)(0).
 */

                /* function exit code */
                __pyx_L1_error:;
                __Pyx_XDECREF(__pyx_t_1);
                __Pyx_XDECREF(__pyx_t_2);
                __Pyx_XDECREF(__pyx_t_3);
                __Pyx_AddTraceback("PyClical.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
                __pyx_r = 0;
                __pyx_L0:;
                __Pyx_XGIVEREF(__pyx_r);
                __Pyx_RefNannyFinishContext();
                return __pyx_r;
              }

              /* Python wrapper */
```

```
                static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_36quad[] = "\n    Quadratic form == (rev(x)*x)(0).\n\n    >>> print(quad(clifford(\"1+{1}+{1,2}\")))\n    3.0\n    >>> print(quad(clifford(\"1+{-1}+{1,2}+{1,2,3}\")))\n    2.0\n    ";
                static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("quad (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_36quad(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_36quad(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("quad", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_quad(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1500, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1511
 *     return clifford(obj).quad()
 *
 * cpdef inline norm(obj):             # <<<<<<<<<<<<<<
 *     """
 *     norm == sum of squares of coordinates.
 */

                static PyObject *__pyx_pw_8PyClical_39norm(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_norm(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("norm", 0);

                  /* "PyClical.pyx":1520
 *     4.0
 *     """
 *     return clifford(obj).norm()             # <<<<<<<<<<<<<<
 *
 * cpdef inline abs(obj):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1520, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_norm); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1520, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                    __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                    if (likely(__pyx_t_2)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
```

```
                      __Pyx_INCREF(__pyx_t_2);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_3, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) : __Pyx_PyObject_CallNoArg(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1520, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1511
 *     return clifford(obj).quad()
 *
 * cpdef inline norm(obj):             # «««««««
 *     """
 *     norm == sum of squares of coordinates.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.norm", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_39norm(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_38norm[] = "\n    norm == sum of squares of coordinates.\n\n    >>> norm(clifford(\"1+{1}+{1,2}\"))\n    3.0\n    >>> norm(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    4.0\n    ";
                static PyObject *__pyx_pw_8PyClical_39norm(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("norm (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_38norm(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_38norm(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("norm", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_norm(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1511, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.norm", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1522
 *     return clifford(obj).norm()
 *
 * cpdef inline abs(obj):             # «««««««
 *     """
```

```
 *      Absolute value of multivector: multivector 2-norm.
 */

                  static PyObject *__pyx_pw_8PyClical_41abs(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_abs(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("abs", 0);

                    /* "PyClical.pyx":1529
 *      2.0
 *      """
 *      return glucat.abs(toClifford(obj))              # <<<<<<<<<<<<<<
 *
 * cpdef inline max_abs(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = PyFloat_FromDouble(abs(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1529, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1522
 *      return clifford(obj).norm()
 *
 * cpdef inline abs(obj):              # <<<<<<<<<<<<<<
 *      """
 *      Absolute value of multivector: multivector 2-norm.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_41abs(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_40abs[] = "\n    Absolute value of multivector: multivector 2-norm.\n\n    >>> abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    2.0\n    ";
                  static PyObject *__pyx_pw_8PyClical_41abs(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("abs (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_40abs(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_40abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("abs", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __pyx_f_8PyClical_abs(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1522, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1531
 *     return glucat.abs(toClifford(obj))
 *
 * cpdef inline max_abs(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Maximum absolute value of coordinates multivector: multivector infinity-norm.
 */

                  static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_abs(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("max_abs", 0);

                    /* "PyClical.pyx":1541
 *
 *     """
 *     return glucat.max_abs(toClifford(obj))             # <<<<<<<<<<<<<<
 *
 * cpdef inline pow(obj, m):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = PyFloat_FromDouble(max_abs(__pyx_f_8PyClical_toClifford(__pyx_v_obj)));
if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1541, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1531
 *     return glucat.abs(toClifford(obj))
 *
 * cpdef inline max_abs(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Maximum absolute value of coordinates multivector: multivector infinity-norm.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.max_abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_42max_abs[] = "\n    Maximum absolute value of
coordinates multivector: multivector infinity-norm.\n\n    >>>
max_abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n    1.0\n    >>> max_abs(clifford(\"3+2{1}+{1,2}\"))\n
3.0\n\n    ";
                  static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("max_abs (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_42max_abs(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_42max_abs(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
```

```
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("max_abs", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_max_abs(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1531, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.max_abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1543
 *     return glucat.max_abs(toClifford(obj))
 *
 * cpdef inline pow(obj, m):             # <<<<<<<<<<<<<<
 *     """
 *     Integer power of multivector: obj to the m.
 */

      static PyObject *__pyx_pw_8PyClical_45pow(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_pow(PyObject *__pyx_v_obj, PyObject *__pyx_v_m, CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        PyObject *__pyx_t_3 = NULL;
        PyObject *__pyx_t_4 = NULL;
        PyObject *__pyx_t_5 = NULL;
        PyObject *__pyx_t_6 = NULL;
        int __pyx_t_7;
        PyObject *__pyx_t_8 = NULL;
        PyObject *__pyx_t_9 = NULL;
        PyObject *__pyx_t_10 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("pow", 0);

        /* "PyClical.pyx":1562
 *     1
 *     """
 *     try:             # <<<<<<<<<<<<<<
 *         math.pow(obj, m)
 *     except:
 */
        {
          __Pyx_PyThreadState_declare
          __Pyx_PyThreadState_assign
          __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
          __Pyx_XGOTREF(__pyx_t_1);
          __Pyx_XGOTREF(__pyx_t_2);
          __Pyx_XGOTREF(__pyx_t_3);
          /*try:*/ {

            /* "PyClical.pyx":1563
 *     """
 *     try:
 *         math.pow(obj, m)             # <<<<<<<<<<<<<<
 *     except:
 *         return clifford(obj).pow(m)
 */
            __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_GOTREF(__pyx_t_5);
            __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_pow); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_GOTREF(__pyx_t_6);
            __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
            __pyx_t_5 = NULL;
            __pyx_t_7 = 0;
            if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
              __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
              if (likely(__pyx_t_5)) {
                PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
                __Pyx_INCREF(__pyx_t_5);
```

```
              __Pyx_INCREF(function);
              __Pyx_DECREF_SET(__pyx_t_6, function);
              __pyx_t_7 = 1;
            }
          }
          #if CYTHON_FAST_PYCALL
          if (PyFunction_Check(__pyx_t_6)) {
            PyObject *__pyx_temp[3] = {__pyx_t_5, __pyx_v_obj, __pyx_v_m};
            __pyx_t_4 = __Pyx_PyFunction_FastCall(__pyx_t_6, __pyx_temp+1-__pyx_t_7, 2+__pyx_t_7); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
            __Pyx_GOTREF(__pyx_t_4);
          } else
          #endif
          #if CYTHON_FAST_PYCCALL
          if (__Pyx_PyFastCFunction_Check(__pyx_t_6)) {
            PyObject *__pyx_temp[3] = {__pyx_t_5, __pyx_v_obj, __pyx_v_m};
            __pyx_t_4 = __Pyx_PyCFunction_FastCall(__pyx_t_6, __pyx_temp+1-__pyx_t_7, 2+__pyx_t_7); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
            __Pyx_GOTREF(__pyx_t_4);
          } else
          #endif
          {
            __pyx_t_8 = PyTuple_New(2+__pyx_t_7); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_GOTREF(__pyx_t_8);
            if (__pyx_t_5) {
              __Pyx_GIVEREF(__pyx_t_5); PyTuple_SET_ITEM(__pyx_t_8, 0, __pyx_t_5); __pyx_t_5 = NULL;
            }
            __Pyx_INCREF(__pyx_v_obj);
            __Pyx_GIVEREF(__pyx_v_obj);
            PyTuple_SET_ITEM(__pyx_t_8, 0+__pyx_t_7, __pyx_v_obj);
            __Pyx_INCREF(__pyx_v_m);
            __Pyx_GIVEREF(__pyx_v_m);
            PyTuple_SET_ITEM(__pyx_t_8, 1+__pyx_t_7, __pyx_v_m);
            __pyx_t_4 = __Pyx_PyObject_Call(__pyx_t_6, __pyx_t_8, NULL); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
            __Pyx_GOTREF(__pyx_t_4);
            __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
          }
          __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
          __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;

          /* "PyClical.pyx":1562
 *     1
 *     """
 *     try:             # <<<<<<<<<<<<<<
 *         math.pow(obj, m)
 *     except:
 */
        }
        __Pyx_XDECREF(__pyx_t_1); __pyx_t_1 = 0;
        __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
        goto __pyx_L8_try_end;
        __pyx_L3_error:;
        __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
        __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
        __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
        __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;

        /* "PyClical.pyx":1564
 *     try:
 *         math.pow(obj, m)
 *     except:             # <<<<<<<<<<<<<<
 *         return clifford(obj).pow(m)
 *
 */
        /*except:*/ {
          __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
          if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_8) < 0) __PYX_ERR(0, 1564, __pyx_L5_except_error)
          __Pyx_GOTREF(__pyx_t_4);
          __Pyx_GOTREF(__pyx_t_6);
          __Pyx_GOTREF(__pyx_t_8);

          /* "PyClical.pyx":1565
 *         math.pow(obj, m)
 *     except:
 *         return clifford(obj).pow(m)             # <<<<<<<<<<<<<<
 *
 * cpdef inline outer_pow(obj, m):
 */
          __Pyx_XDECREF(__pyx_r);
          __pyx_t_9 = __Pyx_PyObject_CallOneArg(((PyObject
```

```
*)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1565, __pyx_L5_except_error)
          __Pyx_GOTREF(__pyx_t_9);
          __pyx_t_10 = __Pyx_PyObject_GetAttrStr(__pyx_t_9, __pyx_n_s_pow); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1565, __pyx_L5_except_error)
          __Pyx_GOTREF(__pyx_t_10);
          __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
          __pyx_t_9 = NULL;
          if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_10))) {
            __pyx_t_9 = PyMethod_GET_SELF(__pyx_t_10);
            if (likely(__pyx_t_9)) {
              PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_10);
              __Pyx_INCREF(__pyx_t_9);
              __Pyx_INCREF(function);
              __Pyx_DECREF_SET(__pyx_t_10, function);
            }
          }
          __pyx_t_5 = (__pyx_t_9) ? __Pyx_PyObject_Call2Args(__pyx_t_10, __pyx_t_9, __pyx_v_m) : __Pyx_PyObject_CallOneArg(__pyx_t_10, __pyx_v_m);
          __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
          if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1565, __pyx_L5_except_error)
          __Pyx_GOTREF(__pyx_t_5);
          __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
          __pyx_r = __pyx_t_5;
          __pyx_t_5 = 0;
          __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
          __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
          __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
          goto __pyx_L6_except_return;
        }
        __pyx_L5_except_error:;

        /* "PyClical.pyx":1562
 *     1
 *     """
 *     try:             # <<<<<<<<<<<<<<
 *       math.pow(obj, m)
 *     except:
 */
        __Pyx_XGIVEREF(__pyx_t_1);
        __Pyx_XGIVEREF(__pyx_t_2);
        __Pyx_XGIVEREF(__pyx_t_3);
        __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
        goto __pyx_L1_error;
        __pyx_L6_except_return:;
        __Pyx_XGIVEREF(__pyx_t_1);
        __Pyx_XGIVEREF(__pyx_t_2);
        __Pyx_XGIVEREF(__pyx_t_3);
        __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
        goto __pyx_L0;
        __pyx_L8_try_end:;
      }

      /* "PyClical.pyx":1543
 *     return glucat.max_abs(toClifford(obj))
 *
 * cpdef inline pow(obj, m):             # <<<<<<<<<<<<<<
 *     """
 *     Integer power of multivector: obj to the m.
 */

      /* function exit code */
      __pyx_r = Py_None; __Pyx_INCREF(Py_None);
      goto __pyx_L0;
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_4);
      __Pyx_XDECREF(__pyx_t_5);
      __Pyx_XDECREF(__pyx_t_6);
      __Pyx_XDECREF(__pyx_t_8);
      __Pyx_XDECREF(__pyx_t_9);
      __Pyx_XDECREF(__pyx_t_10);
      __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = 0;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_45pow(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
    static char __pyx_doc_8PyClical_44pow[] = "\n    Integer power of multivector: obj to the m.\n\n    >>> x=clifford(\"{1}\"); print(pow(x,2))\n    1\n    >>> x=clifford(\"2\"); print(pow(x,2))\n    4\n    >>> x=clifford(\"2+{1}\"); print(pow(x,0))\n    1\n    >>> x=clifford(\"2+{1}\"); print(pow(x,1))\n    2+{1}\n    >>> x=clifford(\"2+{1}\"); print(pow(x,2))\n    5+4{1}\n    >>> print(pow(clifford(\"1+{1}+{1,2}\"),3))\n    1+3{1}+3{1,2}\n    >>>
```

```
i=clifford(\"{1,2}\"); print(exp(pi/2) * pow(i, i))\n    1\n    ";
static PyObject *__pyx_pw_8PyClical_45pow(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_m = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("pow (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_m,0};
    PyObject* values[2] = {0,0};
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_m)) !=
0)) kw_args--;
        else {
          __Pyx_RaiseArgtupleInvalid("pow", 1, 2, 2, 1); __PYX_ERR(0, 1543,
__pyx_L3_error)
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "pow") < 0)) __PYX_ERR(0, 1543, __pyx_L3_error)
      }
    } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
      goto __pyx_L5_argtuple_error;
    } else {
      values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
      values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
    }
    __pyx_v_obj = values[0];
    __pyx_v_m = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("pow", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1543, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_44pow(__pyx_self, __pyx_v_obj, __pyx_v_m);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_44pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj, PyObject *__pyx_v_m) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("pow", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_pow(__pyx_v_obj, __pyx_v_m, 0); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1543, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
```

```
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1567
 *          return clifford(obj).pow(m)
 *
 * cpdef inline outer_pow(obj, m):             # <<<<<<<<<<<<<<
 *     """
 *     Outer product power of multivector.
 */

                  static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject
 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_outer_pow(PyObject *__pyx_v_obj,
 PyObject *__pyx_v_m, CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    PyObject *__pyx_t_2 = NULL;
                    PyObject *__pyx_t_3 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("outer_pow", 0);

                    /* "PyClical.pyx":1574
 *     1+3{1}+3{1,2}
 *     """
 *     return clifford(obj).outer_pow(m)             # <<<<<<<<<<<<<<
 *
 * cpdef inline complexifier(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
 __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1574, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_2);
                    __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_outer_pow); if
 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1574, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                    __pyx_t_2 = NULL;
                    if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
                      __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
                      if (likely(__pyx_t_2)) {
                        PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
                        __Pyx_INCREF(__pyx_t_2);
                        __Pyx_INCREF(function);
                        __Pyx_DECREF_SET(__pyx_t_3, function);
                      }
                    }
                    __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_Call2Args(__pyx_t_3, __pyx_t_2, __pyx_v_m)
 : __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_v_m);
                    __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
                    if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1574, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1567
 *          return clifford(obj).pow(m)
 *
 * cpdef inline outer_pow(obj, m):             # <<<<<<<<<<<<<<
 *     """
 *     Outer product power of multivector.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_XDECREF(__pyx_t_2);
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno,
 __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
```

```
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_8PyClical_46outer_pow[] = "\n    Outer product power of multivector.\n\n    >>> print(outer_pow(clifford(\"1+{1}+{1,2}\"),3))\n    1+3{1}+3{1,2}\n    ";
static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_m = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("outer_pow (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_m,0};
    PyObject* values[2] = {0,0};
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_m)) != 0)) kw_args--;
        else {
          __Pyx_RaiseArgtupleInvalid("outer_pow", 1, 2, 2, 1); __PYX_ERR(0, 1567, __pyx_L3_error)
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "outer_pow") < 0)) __PYX_ERR(0, 1567, __pyx_L3_error)
      }
    } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
      goto __pyx_L5_argtuple_error;
    } else {
      values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
      values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
    }
    __pyx_v_obj = values[0];
    __pyx_v_m = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("outer_pow", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1567, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_46outer_pow(__pyx_self, __pyx_v_obj, __pyx_v_m);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_46outer_pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_m) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("outer_pow", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_outer_pow(__pyx_v_obj, __pyx_v_m, 0); if
```

```
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1567, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno,
__pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1576
 *     return clifford(obj).outer_pow(m)
 *
 * cpdef inline complexifier(obj):             # «««««««
 *     """
 *     Square root of -1 which commutes with all members of the frame of the given multivector.
 */

                static PyObject *__pyx_pw_8PyClical_49complexifier(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_complexifier(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("complexifier", 0);

                  /* "PyClical.pyx":1589
 *     {-1}
 *     """
 *     return clifford().wrap( glucat.complexifier(toClifford(obj)) )             # «««««««
 *
 * cpdef inline sqrt(obj, i = None):
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford));
if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1589, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), complexifier(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1589, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
                  __pyx_r = __pyx_t_2;
                  __pyx_t_2 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1576
 *     return clifford(obj).outer_pow(m)
 *
 * cpdef inline complexifier(obj):             # «««««««
 *     """
 *     Square root of -1 which commutes with all members of the frame of the given multivector.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_AddTraceback("PyClical.complexifier", __pyx_clineno, __pyx_lineno,
__pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_49complexifier(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                static char __pyx_doc_8PyClical_48complexifier[] = "\n    Square root of -1 which
commutes with all members of the frame of the given multivector.\n\n    »>
print(complexifier(clifford(index_set({1}))))\n    {1,2,3}\n    »>
print(complexifier(clifford(index_set({-1}))))\n    {-1}\n    »> print(complexifier(index_set({1})))\n
```

```
{1,2,3}\n    »> print(complexifier(index_set({-1})))\n    {-1}\n    ";
                static PyObject *__pyx_pw_8PyClical_49complexifier(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("complexifier (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_48complexifier(__pyx_self, ((PyObject *)__pyx_v_obj));

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_48complexifier(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("complexifier", 0);
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_1 = __pyx_f_8PyClical_complexifier(__pyx_v_obj, 0); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1576, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_AddTraceback("PyClical.complexifier", __pyx_clineno, __pyx_lineno,
__pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1591
 *     return clifford().wrap( glucat.complexifier(toClifford(obj)) )
 *
 * cpdef inline sqrt(obj, i = None):              # «««««««
 *     """
 *     Square root of multivector with optional complexifier.
 */

                static PyObject *__pyx_pw_8PyClical_51sqrt(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sqrt(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_sqrt *__pyx_optional_args) {
                  PyObject *__pyx_v_i = ((PyObject *)Py_None);
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  int __pyx_t_1;
                  int __pyx_t_2;
                  PyObject *__pyx_t_3 = NULL;
                  Clifford __pyx_t_4;
                  PyObject *__pyx_t_5 = NULL;
                  PyObject *__pyx_t_6 = NULL;
                  PyObject *__pyx_t_7 = NULL;
                  PyObject *__pyx_t_8 = NULL;
                  PyObject *__pyx_t_9 = NULL;
                  PyObject *__pyx_t_10 = NULL;
                  PyObject *__pyx_t_11 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("sqrt", 0);
                  if (__pyx_optional_args) {
                    if (__pyx_optional_args->__pyx_n > 0) {
                      __pyx_v_i = __pyx_optional_args->i;
                    }
                  }

                  /* "PyClical.pyx":1606
 *     -1
 *     """
 *     if not (i is None):              # «««««««
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  __pyx_t_1 = (__pyx_v_i != Py_None);
                  __pyx_t_2 = (__pyx_t_1 != 0);
```

```
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":1607
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )             # <<<<<<<<<<<<<<
 *     else:
 *         try:
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1607, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    try {
                      __pyx_t_4 = sqrt(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1607, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1607, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1606
 *     -1
 *     """
 *     if not (i is None):             # <<<<<<<<<<<<<<
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  }

                  /* "PyClical.pyx":1609
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<<<<<<<
 *             return math.sqrt(obj)
 *         except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1610
 *     else:
 *         try:
 *             return math.sqrt(obj)             # <<<<<<<<<<<<<<
 *         except:
 *             return clifford().wrap( glucat.sqrt(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1610, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_sqrt); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1610, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1610, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
```

```
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1609
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # «««««««
 *             return math.sqrt(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1611
 *         try:
 *             return math.sqrt(obj)
 *         except:             # «««««««
 *             return clifford().wrap( glucat.sqrt(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1611, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1612
 *             return math.sqrt(obj)
 *         except:
 *             return clifford().wrap( glucat.sqrt(toClifford(obj)) )             # «««««««
 *
 * cpdef inline exp(obj):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1612, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), sqrt(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1612, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1609
 *         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # «««««««
 *             return math.sqrt(obj)
 *         except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1591
 *     return clifford().wrap( glucat.complexifier(toClifford(obj)) )
 *
```

```
 * cpdef inline sqrt(obj, i = None):             # <<<<<<<<
 *     """
 *     Square root of multivector with optional complexifier.
 */

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_3);
  __Pyx_XDECREF(__pyx_t_5);
  __Pyx_XDECREF(__pyx_t_9);
  __Pyx_XDECREF(__pyx_t_10);
  __Pyx_XDECREF(__pyx_t_11);
  __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = 0;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_51sqrt(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_8PyClical_50sqrt[] = "\n    Square root of multivector with
optional complexifier.\n\n    >>> print(sqrt(-1))\n    {-1}\n    >>> print(sqrt(clifford(\"2{-1}\")))\n
1+{-1}\n    >>> j=sqrt(-1,complexifier(index_set({1}))); print(j); print(j*j)\n    {1,2,3}\n    -1\n
>>> j=sqrt(-1,\"{1,2,3}\"); print(j); print(j*j)\n    {1,2,3}\n    -1\n    ";
static PyObject *__pyx_pw_8PyClical_51sqrt(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_i = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("sqrt (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "sqrt") < 0)) __PYX_ERR(0, 1591, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("sqrt", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1591, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
        __Pyx_RefNannyFinishContext();
        return NULL;
        __pyx_L4_argument_unpacking_done:;
        __pyx_r = __pyx_pf_8PyClical_50sqrt(__pyx_self, __pyx_v_obj, __pyx_v_i);

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_50sqrt(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        struct __pyx_opt_args_8PyClical_sqrt __pyx_t_2;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("sqrt", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2.__pyx_n = 1;
        __pyx_t_2.i = __pyx_v_i;
        __pyx_t_1 = __pyx_f_8PyClical_sqrt(__pyx_v_obj, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1591, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1614
 *             return clifford().wrap( glucat.sqrt(toClifford(obj)) )
 *
 * cpdef inline exp(obj):            # «««««««
 *     """
 *     Exponential of multivector.
 */

      static PyObject *__pyx_pw_8PyClical_53exp(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_exp(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        PyObject *__pyx_t_3 = NULL;
        PyObject *__pyx_t_4 = NULL;
        PyObject *__pyx_t_5 = NULL;
        PyObject *__pyx_t_6 = NULL;
        PyObject *__pyx_t_7 = NULL;
        PyObject *__pyx_t_8 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("exp", 0);

        /* "PyClical.pyx":1623
 *     {1,2}
 *     """
 *     try:             # «««««««
 *         return math.exp(obj)
 *     except:
 */
        {
          __Pyx_PyThreadState_declare
          __Pyx_PyThreadState_assign
          __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
          __Pyx_XGOTREF(__pyx_t_1);
          __Pyx_XGOTREF(__pyx_t_2);
          __Pyx_XGOTREF(__pyx_t_3);
          /*try:*/ {

            /* "PyClical.pyx":1624
 *     """
 *     try:
```

```
 *         return math.exp(obj)             # «««««««
 *     except:
 *         return clifford().wrap( glucat.exp(toClifford(obj)) )
 */
                __Pyx_XDECREF(__pyx_r);
                __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1624, __pyx_L3_error)
                __Pyx_GOTREF(__pyx_t_5);
                __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_exp); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1624, __pyx_L3_error)
                __Pyx_GOTREF(__pyx_t_6);
                __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                __pyx_t_5 = NULL;
                if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
                  __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
                  if (likely(__pyx_t_5)) {
                    PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
                    __Pyx_INCREF(__pyx_t_5);
                    __Pyx_INCREF(function);
                    __Pyx_DECREF_SET(__pyx_t_6, function);
                  }
                }
                __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
                __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1624, __pyx_L3_error)
                __Pyx_GOTREF(__pyx_t_4);
                __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                __pyx_r = __pyx_t_4;
                __pyx_t_4 = 0;
                goto __pyx_L7_try_return;

                /* "PyClical.pyx":1623
 *     {1,2}
 *     """
 *     try:             # «««««««
 *         return math.exp(obj)
 *     except:
 */
              }
              __pyx_L3_error:;
              __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
              __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
              __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;

              /* "PyClical.pyx":1625
 *     try:
 *         return math.exp(obj)
 *     except:             # «««««««
 *         return clifford().wrap( glucat.exp(toClifford(obj)) )
 *
 */
              /*except:*/ {
                __Pyx_AddTraceback("PyClical.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);
                if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0, 1625, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_4);
                __Pyx_GOTREF(__pyx_t_6);
                __Pyx_GOTREF(__pyx_t_5);

                /* "PyClical.pyx":1626
 *         return math.exp(obj)
 *     except:
 *         return clifford().wrap( glucat.exp(toClifford(obj)) )             # «««««««
 *
 * cpdef inline log(obj,i = None):
 */
                __Pyx_XDECREF(__pyx_r);
                __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1626, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_7);
                __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_7), exp(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 1626, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_8);
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                __pyx_r = __pyx_t_8;
                __pyx_t_8 = 0;
                __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
                __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                goto __pyx_L6_except_return;
              }
              __pyx_L5_except_error:;

              /* "PyClical.pyx":1623
```

```
 *     {1,2}
 *     """
 *     try:              # <<<<<<<<
 *         return math.exp(obj)
 *     except:
 */
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L1_error;
                      __pyx_L7_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L0;
                      __pyx_L6_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L0;
                    }

                    /* "PyClical.pyx":1614
 *              return clifford().wrap( glucat.sqrt(toClifford(obj)) )
 *
 * cpdef inline exp(obj):             # <<<<<<<<
 *     """
 *     Exponential of multivector.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_4);
                    __Pyx_XDECREF(__pyx_t_5);
                    __Pyx_XDECREF(__pyx_t_6);
                    __Pyx_XDECREF(__pyx_t_7);
                    __Pyx_XDECREF(__pyx_t_8);
                    __Pyx_AddTraceback("PyClical.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_53exp(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_52exp[] = "\n    Exponential of multivector.\n\n    >>>
x=clifford(\"{1,2}\") * pi/4; print(exp(x))\n    0.7071+0.7071{1,2}\n    >>> x=clifford(\"{1,2}\") *
pi/2; print(exp(x))\n    {1,2}\n    ";
                  static PyObject *__pyx_pw_8PyClical_53exp(PyObject *__pyx_self, PyObject *__pyx_v_obj)
{
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("exp (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_52exp(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_52exp(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("exp", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __pyx_f_8PyClical_exp(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1614, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1628
 *         return clifford().wrap( glucat.exp(toClifford(obj)) )
 *
 * cpdef inline log(obj,i = None):             # «««««««
 *     """
 *     Natural logarithm of multivector with optional complexifier.
 */

                static PyObject *__pyx_pw_8PyClical_55log(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_log(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_log *__pyx_optional_args) {
                  PyObject *__pyx_v_i = ((PyObject *)Py_None);
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  int __pyx_t_1;
                  int __pyx_t_2;
                  PyObject *__pyx_t_3 = NULL;
                  Clifford __pyx_t_4;
                  PyObject *__pyx_t_5 = NULL;
                  PyObject *__pyx_t_6 = NULL;
                  PyObject *__pyx_t_7 = NULL;
                  PyObject *__pyx_t_8 = NULL;
                  PyObject *__pyx_t_9 = NULL;
                  PyObject *__pyx_t_10 = NULL;
                  PyObject *__pyx_t_11 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("log", 0);
                  if (__pyx_optional_args) {
                    if (__pyx_optional_args->__pyx_n > 0) {
                      __pyx_v_i = __pyx_optional_args->i;
                    }
                  }

                  /* "PyClical.pyx":1643
 *     RuntimeError: check_complex(val, i): i is not a valid complexifier for val
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  __pyx_t_1 = (__pyx_v_i != Py_None);
                  __pyx_t_2 = (__pyx_t_1 != 0);
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":1644
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )             # «««««««
 *     else:
 *         try:
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1644, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    try {
                      __pyx_t_4 = log(__pyx_f_8PyClical_toClifford(__pyx_v_obj), __pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1644, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1644, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1643
 *     RuntimeError: check_complex(val, i): i is not a valid complexifier for val
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
 *     else:
 */
```

```
                  }

                  /* "PyClical.pyx":1646
 *         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:              # <<<<<<<<<<<<<<
 *             return math.log(obj)
 *         except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1647
 *     else:
 *         try:
 *             return math.log(obj)             # <<<<<<<<<<<<<<
 *         except:
 *             return clifford().wrap( glucat.log(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1647, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_log); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1647, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1647, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1646
 *         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:              # <<<<<<<<<<<<<<
 *             return math.log(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1648
 *         try:
 *             return math.log(obj)
 *         except:              # <<<<<<<<<<<<<<
 *             return clifford().wrap( glucat.log(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1648, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1649
 *             return math.log(obj)
 *         except:
 *             return clifford().wrap( glucat.log(toClifford(obj)) )             # <<<<<<<<<<<<<<
```

```
 *
 * cpdef inline cos(obj,i = None):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1649, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), log(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1649, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1646
 *           return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
 *       else:
 *           try:              # <<<<<<<<<<<<<<
 *               return math.log(obj)
 *           except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1628
 *           return clifford().wrap( glucat.exp(toClifford(obj)) )
 *
 * cpdef inline log(obj,i = None):              # <<<<<<<<<<<<<<
 *       """
 *       Natural logarithm of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_55log(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_54log[] = "\n    Natural logarithm of multivector with optional complexifier.\n\n    >>> x=clifford(\"{-1}\"); print((log(x,\"{-1}\") * 2/pi))\n    {-1}\n    >>> x=clifford(\"{1,2}\"); print((log(x,\"{1,2,3}\") * 2/pi))\n    {1,2}\n    >>> x=clifford(\"{1,2}\"); print((log(x) * 2/pi))\n    {1,2}\n    >>> x=clifford(\"{1,2}\"); print((log(x,\"{1,2}\") * 2/pi))\n    Traceback (most recent call last):\n    ...\n    RuntimeError: check_complex(val, i): i is not a valid complexifier for val\n    ";
                static PyObject *__pyx_pw_8PyClical_55log(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_obj = 0;
                  PyObject *__pyx_v_i = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
```

```
    int __pyx_clineno = 0;
    PyObject *__pyx_r = 0;
    __Pyx_RefNannyDeclarations
    __Pyx_RefNannySetupContext("log (wrapper)", 0);
    {
      static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
      PyObject* values[2] = {0,0};
      values[1] = ((PyObject *)Py_None);
      if (unlikely(__pyx_kwds)) {
        Py_ssize_t kw_args;
        const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
        switch (pos_args) {
          case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
          CYTHON_FALLTHROUGH;
          case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
          CYTHON_FALLTHROUGH;
          case  0: break;
          default: goto __pyx_L5_argtuple_error;
        }
        kw_args = PyDict_Size(__pyx_kwds);
        switch (pos_args) {
          case  0:
          if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
          else goto __pyx_L5_argtuple_error;
          CYTHON_FALLTHROUGH;
          case  1:
          if (kw_args > 0) {
            PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
            if (value) { values[1] = value; kw_args--; }
          }
        }
        if (unlikely(kw_args > 0)) {
          if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "log") < 0)) __PYX_ERR(0, 1628, __pyx_L3_error)
        }
      } else {
        switch (PyTuple_GET_SIZE(__pyx_args)) {
          case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
          CYTHON_FALLTHROUGH;
          case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
          break;
          default: goto __pyx_L5_argtuple_error;
        }
      }
      __pyx_v_obj = values[0];
      __pyx_v_i = values[1];
    }
    goto __pyx_L4_argument_unpacking_done;
    __pyx_L5_argtuple_error:;
    __Pyx_RaiseArgtupleInvalid("log", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1628, __pyx_L3_error)
    __pyx_L3_error:;
    __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
    __Pyx_RefNannyFinishContext();
    return NULL;
    __pyx_L4_argument_unpacking_done:;
    __pyx_r = __pyx_pf_8PyClical_54log(__pyx_self, __pyx_v_obj, __pyx_v_i);

    /* function exit code */
    __Pyx_RefNannyFinishContext();
    return __pyx_r;
  }

  static PyObject *__pyx_pf_8PyClical_54log(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj, PyObject *__pyx_v_i) {
    PyObject *__pyx_r = NULL;
    __Pyx_RefNannyDeclarations
    PyObject *__pyx_t_1 = NULL;
    struct __pyx_opt_args_8PyClical_log __pyx_t_2;
    int __pyx_lineno = 0;
    const char *__pyx_filename = NULL;
    int __pyx_clineno = 0;
    __Pyx_RefNannySetupContext("log", 0);
    __Pyx_XDECREF(__pyx_r);
    __pyx_t_2.__pyx_n = 1;
    __pyx_t_2.i = __pyx_v_i;
    __pyx_t_1 = __pyx_f_8PyClical_log(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1628, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_r = __pyx_t_1;
    __pyx_t_1 = 0;
    goto __pyx_L0;

    /* function exit code */
    __pyx_L1_error:;
    __Pyx_XDECREF(__pyx_t_1);
```

```
            __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1651
 *             return clifford().wrap( glucat.log(toClifford(obj)) )
 *
 * cpdef inline cos(obj,i = None):             # «««««««
 *     """
 *     Cosine of multivector with optional complexifier.
 */

          static PyObject *__pyx_pw_8PyClical_57cos(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
          static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cos(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_cos *__pyx_optional_args) {
            PyObject *__pyx_v_i = ((PyObject *)Py_None);
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            int __pyx_t_1;
            int __pyx_t_2;
            PyObject *__pyx_t_3 = NULL;
            Clifford __pyx_t_4;
            PyObject *__pyx_t_5 = NULL;
            PyObject *__pyx_t_6 = NULL;
            PyObject *__pyx_t_7 = NULL;
            PyObject *__pyx_t_8 = NULL;
            PyObject *__pyx_t_9 = NULL;
            PyObject *__pyx_t_10 = NULL;
            PyObject *__pyx_t_11 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("cos", 0);
            if (__pyx_optional_args) {
              if (__pyx_optional_args->__pyx_n > 0) {
                __pyx_v_i = __pyx_optional_args->i;
              }
            }

            /* "PyClical.pyx":1660
 *     {1,2}
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
 *     else:
 */
            __pyx_t_1 = (__pyx_v_i != Py_None);
            __pyx_t_2 = (__pyx_t_1 != 0);
            if (__pyx_t_2) {

              /* "PyClical.pyx":1661
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )             # «««««««
 *     else:
 *         try:
 */
              __Pyx_XDECREF(__pyx_r);
              __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1661, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_3);
              try {
                __pyx_t_4 = cos(__pyx_f_8PyClical_toClifford(__pyx_v_obj), __pyx_f_8PyClical_toClifford(__pyx_v_i));
              } catch(...) {
                __Pyx_CppExn2PyErr();
                __PYX_ERR(0, 1661, __pyx_L1_error)
              }
              __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1661, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_5);
              __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
              __pyx_r = __pyx_t_5;
              __pyx_t_5 = 0;
              goto __pyx_L0;

              /* "PyClical.pyx":1660
 *     {1,2}
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
 *     else:
```

```
*/
                    }

                    /* "PyClical.pyx":1663
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.cos(obj)
 *         except:
 */
                    /*else*/ {
                      {
                        __Pyx_PyThreadState_declare
                        __Pyx_PyThreadState_assign
                        __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                        __Pyx_XGOTREF(__pyx_t_6);
                        __Pyx_XGOTREF(__pyx_t_7);
                        __Pyx_XGOTREF(__pyx_t_8);
                        /*try:*/ {

                          /* "PyClical.pyx":1664
 *     else:
 *         try:
 *             return math.cos(obj)             # <<<<<<<<
 *         except:
 *             return clifford().wrap( glucat.cos(toClifford(obj)) )
 */
                          __Pyx_XDECREF(__pyx_r);
                          __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1664, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_3);
                          __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_cos); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1664, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_9);
                          __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                          __pyx_t_3 = NULL;
                          if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                            __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                            if (likely(__pyx_t_3)) {
                              PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                              __Pyx_INCREF(__pyx_t_3);
                              __Pyx_INCREF(function);
                              __Pyx_DECREF_SET(__pyx_t_9, function);
                            }
                          }
                          __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                          __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                          if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1664, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_5);
                          __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                          __pyx_r = __pyx_t_5;
                          __pyx_t_5 = 0;
                          goto __pyx_L8_try_return;

                          /* "PyClical.pyx":1663
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.cos(obj)
 *         except:
 */
                        }
                        __pyx_L4_error:;
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                        /* "PyClical.pyx":1665
 *         try:
 *             return math.cos(obj)
 *         except:             # <<<<<<<<
 *             return clifford().wrap( glucat.cos(toClifford(obj)) )
 *
 */
                        /*except:*/ {
                          __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno,
__pyx_filename);
                          if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1665, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_5);
                          __Pyx_GOTREF(__pyx_t_9);
                          __Pyx_GOTREF(__pyx_t_3);

                          /* "PyClical.pyx":1666
 *             return math.cos(obj)
 *         except:
```

```
 *             return clifford().wrap( glucat.cos(toClifford(obj)) )             # <<<<<<<<
 *
 * cpdef inline acos(obj,i = None):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1666, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), cos(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1666, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1663
 *         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.cos(obj)
 *         except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1651
 *             return clifford().wrap( glucat.log(toClifford(obj)) )
 *
 * cpdef inline cos(obj,i = None):             # <<<<<<<<
 *     """
 *     Cosine of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_57cos(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_56cos[] = "\n    Cosine of multivector with optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{1,2,3}\"))\n    {1,2}\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x)))\n    {1,2}\n    ";
                static PyObject *__pyx_pw_8PyClical_57cos(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_obj = 0;
                  PyObject *__pyx_v_i = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  PyObject *__pyx_r = 0;
```

```
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("cos (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "cos") < 0)) __PYX_ERR(0, 1651, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("cos", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1651, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_56cos(__pyx_self, __pyx_v_obj, __pyx_v_i);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_56cos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  struct __pyx_opt_args_8PyClical_cos __pyx_t_2;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("cos", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_2.__pyx_n = 1;
  __pyx_t_2.i = __pyx_v_i;
  __pyx_t_1 = __pyx_f_8PyClical_cos(__pyx_v_obj, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1651, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
```

```
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1668
 *              return clifford().wrap( glucat.cos(toClifford(obj)) )
 *
 * cpdef inline acos(obj,i = None):             # <<<<<<<<
 *     """
 *     Inverse cosine of multivector with optional complexifier.
 */

                static PyObject *__pyx_pw_8PyClical_59acos(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_acos(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_acos *__pyx_optional_args) {
                  PyObject *__pyx_v_i = ((PyObject *)Py_None);
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  int __pyx_t_1;
                  int __pyx_t_2;
                  PyObject *__pyx_t_3 = NULL;
                  Clifford __pyx_t_4;
                  PyObject *__pyx_t_5 = NULL;
                  PyObject *__pyx_t_6 = NULL;
                  PyObject *__pyx_t_7 = NULL;
                  PyObject *__pyx_t_8 = NULL;
                  PyObject *__pyx_t_9 = NULL;
                  PyObject *__pyx_t_10 = NULL;
                  PyObject *__pyx_t_11 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("acos", 0);
                  if (__pyx_optional_args) {
                    if (__pyx_optional_args->__pyx_n > 0) {
                      __pyx_v_i = __pyx_optional_args->i;
                    }
                  }

                  /* "PyClical.pyx":1681
 *     {1,2}
 *     """
 *     if not (i is None):             # <<<<<<<<
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  __pyx_t_1 = (__pyx_v_i != Py_None);
                  __pyx_t_2 = (__pyx_t_1 != 0);
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":1682
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )             # <<<<<<<<
 *     else:
 *         try:
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1682, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    try {
                      __pyx_t_4 = acos(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1682, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1682, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1681
 *     {1,2}
 *     """
 *     if not (i is None):             # <<<<<<<<
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  }
```

```

                  /* "PyClical.pyx":1684
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # «««««««
 *             return math.acos(obj)
 *         except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1685
 *     else:
 *         try:
 *             return math.acos(obj)             # «««««««
 *         except:
 *             return clifford().wrap( glucat.acos(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1685, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_acos); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1685, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1685, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1684
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # «««««««
 *             return math.acos(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1686
 *         try:
 *             return math.acos(obj)
 *         except:             # «««««««
 *             return clifford().wrap( glucat.acos(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1686, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1687
 *             return math.acos(obj)
 *         except:
 *             return clifford().wrap( glucat.acos(toClifford(obj)) )             # «««««««
 *
```

```
 * cpdef inline cosh(obj):
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1687, __pyx_L6_except_error)
            __Pyx_GOTREF(__pyx_t_10);
            __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), acos(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1687, __pyx_L6_except_error)
            __Pyx_GOTREF(__pyx_t_11);
            __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
            __pyx_r = __pyx_t_11;
            __pyx_t_11 = 0;
            __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
            __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
            __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
            goto __pyx_L7_except_return;
          }
          __pyx_L6_except_error:;

          /* "PyClical.pyx":1684
 *         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.acos(obj)
 *         except:
 */
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L1_error;
          __pyx_L8_try_return:;
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L0;
          __pyx_L7_except_return:;
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L0;
        }
      }

      /* "PyClical.pyx":1668
 *             return clifford().wrap( glucat.cos(toClifford(obj)) )
 *
 * cpdef inline acos(obj,i = None):             # <<<<<<<<
 *     """
 *     Inverse cosine of multivector with optional complexifier.
 */

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_3);
      __Pyx_XDECREF(__pyx_t_5);
      __Pyx_XDECREF(__pyx_t_9);
      __Pyx_XDECREF(__pyx_t_10);
      __Pyx_XDECREF(__pyx_t_11);
      __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = 0;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_59acos(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
    static char __pyx_doc_8PyClical_58acos[] = "\n    Inverse cosine of multivector with optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{1,2,3}\"))\n    {1,2}\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x),\"{-1,1,2,3,4}\"))\n    {1,2}\n    >>> print(acos(0) / pi)\n    0.5\n    >>> x=clifford(\"{1,2}\"); print(cos(acos(x)))\n    {1,2}\n    ";
    static PyObject *__pyx_pw_8PyClical_59acos(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
      PyObject *__pyx_v_obj = 0;
      PyObject *__pyx_v_i = 0;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
```

```
  __Pyx_RefNannySetupContext("acos (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "acos") < 0)) __PYX_ERR(0, 1668, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("acos", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1668, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_58acos(__pyx_self, __pyx_v_obj, __pyx_v_i);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_58acos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  struct __pyx_opt_args_8PyClical_acos __pyx_t_2;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("acos", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_2.__pyx_n = 1;
  __pyx_t_2.i = __pyx_v_i;
  __pyx_t_1 = __pyx_f_8PyClical_acos(__pyx_v_obj, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1668, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
```

```
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1689
 *              return clifford().wrap( glucat.acos(toClifford(obj)) )
 *
 * cpdef inline cosh(obj):              # «««««««
 *     """
 *     Hyperbolic cosine of multivector.
 */

                static PyObject *__pyx_pw_8PyClical_61cosh(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cosh(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  PyObject *__pyx_t_4 = NULL;
                  PyObject *__pyx_t_5 = NULL;
                  PyObject *__pyx_t_6 = NULL;
                  PyObject *__pyx_t_7 = NULL;
                  PyObject *__pyx_t_8 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("cosh", 0);

                  /* "PyClical.pyx":1700
 *     {1,2}
 *     """
 *     try:              # «««««««
 *         return math.cosh(obj)
 *     except:
 */
                  {
                    __Pyx_PyThreadState_declare
                    __Pyx_PyThreadState_assign
                    __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
                    __Pyx_XGOTREF(__pyx_t_1);
                    __Pyx_XGOTREF(__pyx_t_2);
                    __Pyx_XGOTREF(__pyx_t_3);
                    /*try:*/ {

                      /* "PyClical.pyx":1701
 *     """
 *     try:
 *         return math.cosh(obj)             # «««««««
 *     except:
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )
 */
                      __Pyx_XDECREF(__pyx_r);
                      __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1701, __pyx_L3_error)
                      __Pyx_GOTREF(__pyx_t_5);
                      __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_cosh); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1701, __pyx_L3_error)
                      __Pyx_GOTREF(__pyx_t_6);
                      __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __pyx_t_5 = NULL;
                      if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
                        __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
                        if (likely(__pyx_t_5)) {
                          PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
                          __Pyx_INCREF(__pyx_t_5);
                          __Pyx_INCREF(function);
                          __Pyx_DECREF_SET(__pyx_t_6, function);
                        }
                      }
                      __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1701, __pyx_L3_error)
                      __Pyx_GOTREF(__pyx_t_4);
                      __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                      __pyx_r = __pyx_t_4;
                      __pyx_t_4 = 0;
                      goto __pyx_L7_try_return;

                      /* "PyClical.pyx":1700
 *     {1,2}
 *     """
 *     try:              # «««««««
```

```
 *         return math.cosh(obj)
 *     except:
 */
                    }
                    __pyx_L3_error:;
                    __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
                    __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                    __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;

                    /* "PyClical.pyx":1702
 *     try:
 *         return math.cosh(obj)
 *     except:             # <<<<<<<<<
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )
 *
 */
                    /*except:*/ {
                      __Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno,
 __pyx_filename);
                      if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0,
 1702, __pyx_L5_except_error)
                      __Pyx_GOTREF(__pyx_t_4);
                      __Pyx_GOTREF(__pyx_t_6);
                      __Pyx_GOTREF(__pyx_t_5);

                      /* "PyClical.pyx":1703
 *         return math.cosh(obj)
 *     except:
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )             # <<<<<<<<<
 *
 * cpdef inline acosh(obj,i = None):
 */
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject
 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1703,
 __pyx_L5_except_error)
                      __Pyx_GOTREF(__pyx_t_7);
                      __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct
 __pyx_obj_8PyClical_clifford *)__pyx_t_7), cosh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
 (unlikely(!__pyx_t_8)) __PYX_ERR(0, 1703, __pyx_L5_except_error)
                      __Pyx_GOTREF(__pyx_t_8);
                      __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                      __pyx_r = __pyx_t_8;
                      __pyx_t_8 = 0;
                      __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
                      __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                      goto __pyx_L6_except_return;
                    }
                    __pyx_L5_except_error:;

                    /* "PyClical.pyx":1700
 *     {1,2}
 *     """
 *     try:             # <<<<<<<<<
 *         return math.cosh(obj)
 *     except:
 */
                    __Pyx_XGIVEREF(__pyx_t_1);
                    __Pyx_XGIVEREF(__pyx_t_2);
                    __Pyx_XGIVEREF(__pyx_t_3);
                    __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                    goto __pyx_L1_error;
                    __pyx_L7_try_return:;
                    __Pyx_XGIVEREF(__pyx_t_1);
                    __Pyx_XGIVEREF(__pyx_t_2);
                    __Pyx_XGIVEREF(__pyx_t_3);
                    __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                    goto __pyx_L0;
                    __pyx_L6_except_return:;
                    __Pyx_XGIVEREF(__pyx_t_1);
                    __Pyx_XGIVEREF(__pyx_t_2);
                    __Pyx_XGIVEREF(__pyx_t_3);
                    __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                    goto __pyx_L0;
                  }

                  /* "PyClical.pyx":1689
 *             return clifford().wrap( glucat.acos(toClifford(obj)) )
 *
 * cpdef inline cosh(obj):             # <<<<<<<<<
 *     """
 *     Hyperbolic cosine of multivector.
 */

                  /* function exit code */
                  __pyx_L1_error:;
```

```
__Pyx_XDECREF(__pyx_t_4);
__Pyx_XDECREF(__pyx_t_5);
__Pyx_XDECREF(__pyx_t_6);
__Pyx_XDECREF(__pyx_t_7);
__Pyx_XDECREF(__pyx_t_8);
__Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
__pyx_r = 0;
__pyx_L0:;
__Pyx_XGIVEREF(__pyx_r);
__Pyx_RefNannyFinishContext();
return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_61cosh(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
static char __pyx_doc_8PyClical_60cosh[] = "\n    Hyperbolic cosine of multivector.\n\n    >>> x=clifford(\"{1,2}\") * pi; print(cosh(x))\n    -1\n    >>> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x)))\n    {1,2,3}\n    >>> x=clifford(\"{1,2}\"); print(cosh(acosh(x)))\n    {1,2}\n    ";
static PyObject *__pyx_pw_8PyClical_61cosh(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("cosh (wrapper)", 0);
  __pyx_r = __pyx_pf_8PyClical_60cosh(__pyx_self, ((PyObject *)__pyx_v_obj));

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_60cosh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("cosh", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_1 = __pyx_f_8PyClical_cosh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1689, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* "PyClical.pyx":1705
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )
 *
 * cpdef inline acosh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic cosine of multivector with optional complexifier.
 */

static PyObject *__pyx_pw_8PyClical_63acosh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static CYTHON_INLINE PyObject *__pyx_f_8PyClical_acosh(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_acosh *__pyx_optional_args) {
  PyObject *__pyx_v_i = ((PyObject *)Py_None);
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  int __pyx_t_1;
  int __pyx_t_2;
  PyObject *__pyx_t_3 = NULL;
  Clifford __pyx_t_4;
  PyObject *__pyx_t_5 = NULL;
  PyObject *__pyx_t_6 = NULL;
  PyObject *__pyx_t_7 = NULL;
  PyObject *__pyx_t_8 = NULL;
  PyObject *__pyx_t_9 = NULL;
  PyObject *__pyx_t_10 = NULL;
  PyObject *__pyx_t_11 = NULL;
  int __pyx_lineno = 0;
```

```
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("acosh", 0);
                  if (__pyx_optional_args) {
                    if (__pyx_optional_args->__pyx_n > 0) {
                      __pyx_v_i = __pyx_optional_args->i;
                    }
                  }

                  /* "PyClical.pyx":1720
 *     {1,2}
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  __pyx_t_1 = (__pyx_v_i != Py_None);
                  __pyx_t_2 = (__pyx_t_1 != 0);
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":1721
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )             # «««««««
 *     else:
 *         try:
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1721, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    try {
                      __pyx_t_4 = acosh(__pyx_f_8PyClical_toClifford(__pyx_v_obj), __pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1721, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1721, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1720
 *     {1,2}
 *     """
 *     if not (i is None):             # «««««««
 *         return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  }

                  /* "PyClical.pyx":1723
 *         return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # «««««««
 *             return math.acosh(obj)
 *         except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1724
 *     else:
 *         try:
 *             return math.acosh(obj)             # «««««««
 *         except:
 *             return clifford().wrap( glucat.acosh(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1724, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_acosh); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1724, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
```

```
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
      __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1724, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1723
 *             return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
 *         else:
 *             try:                 # <<<<<<<<<<<<<<
 *                 return math.acosh(obj)
 *             except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1725
 *             try:
 *                 return math.acosh(obj)
 *             except:             # <<<<<<<<<<<<<<
 *                 return clifford().wrap( glucat.acosh(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.acosh", __pyx_clineno, __pyx_lineno,
      __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
      1725, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1726
 *                 return math.acosh(obj)
 *             except:
 *                 return clifford().wrap( glucat.acosh(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * cpdef inline sin(obj,i = None):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
      *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1726,
      __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
      __pyx_obj_8PyClical_clifford *)__pyx_t_10), acosh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
      (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1726, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1723
 *             return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
 *         else:
 *             try:                 # <<<<<<<<<<<<<<
 *                 return math.acosh(obj)
 *             except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
```

```
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1705
 *         return clifford().wrap( glucat.cosh(toClifford(obj)) )
 *
 * cpdef inline acosh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic cosine of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_63acosh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_62acosh[] = "\n    Inverse hyperbolic cosine of multivector with optional complexifier.\n\n    >>> print(acosh(0,\"{-2,-1,1}\"))\n    1.571{-2,-1,1}\n    >>> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x,\"{-1,1,2,3,4}\")))\n    {1,2,3}\n    >>> print(acosh(0))\n    1.571{-1}\n    >>> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x)))\n    {1,2,3}\n    >>> x=clifford(\"{1,2}\"); print(cosh(acosh(x)))\n    {1,2}\n    ";
                static PyObject *__pyx_pw_8PyClical_63acosh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_obj = 0;
                  PyObject *__pyx_v_i = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("acosh (wrapper)", 0);
                  {
                    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
                    PyObject* values[2] = {0,0};
                    values[1] = ((PyObject *)Py_None);
                    if (unlikely(__pyx_kwds)) {
                      Py_ssize_t kw_args;
                      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
                      switch (pos_args) {
                        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                        CYTHON_FALLTHROUGH;
                        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                        CYTHON_FALLTHROUGH;
                        case  0: break;
                        default: goto __pyx_L5_argtuple_error;
                      }
                      kw_args = PyDict_Size(__pyx_kwds);
                      switch (pos_args) {
                        case  0:
                        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
                        else goto __pyx_L5_argtuple_error;
                        CYTHON_FALLTHROUGH;
                        case  1:
                        if (kw_args > 0) {
                          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
                          if (value) { values[1] = value; kw_args--; }
                        }
                      }
                      if (unlikely(kw_args > 0)) {
                        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
```

```
values, pos_args, "acosh") < 0)) __PYX_ERR(0, 1705, __pyx_L3_error)
            }
          } else {
            switch (PyTuple_GET_SIZE(__pyx_args)) {
              case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
              CYTHON_FALLTHROUGH;
              case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
              break;
              default: goto __pyx_L5_argtuple_error;
            }
          }
          __pyx_v_obj = values[0];
          __pyx_v_i = values[1];
        }
        goto __pyx_L4_argument_unpacking_done;
        __pyx_L5_argtuple_error:;
        __Pyx_RaiseArgtupleInvalid("acosh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1705, __pyx_L3_error)
        __pyx_L3_error:;
        __Pyx_AddTraceback("PyClical.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __Pyx_RefNannyFinishContext();
        return NULL;
        __pyx_L4_argument_unpacking_done:;
        __pyx_r = __pyx_pf_8PyClical_62acosh(__pyx_self, __pyx_v_obj, __pyx_v_i);

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_62acosh(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        struct __pyx_opt_args_8PyClical_acosh __pyx_t_2;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("acosh", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2.__pyx_n = 1;
        __pyx_t_2.i = __pyx_v_i;
        __pyx_t_1 = __pyx_f_8PyClical_acosh(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1705, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1728
 *     return clifford().wrap( glucat.acosh(toClifford(obj)) )
 *
 * cpdef inline sin(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Sine of multivector with optional complexifier.
 */

      static PyObject *__pyx_pw_8PyClical_65sin(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sin(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_sin *__pyx_optional_args) {
        PyObject *__pyx_v_i = ((PyObject *)Py_None);
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        int __pyx_t_1;
        int __pyx_t_2;
        PyObject *__pyx_t_3 = NULL;
        Clifford __pyx_t_4;
        PyObject *__pyx_t_5 = NULL;
        PyObject *__pyx_t_6 = NULL;
        PyObject *__pyx_t_7 = NULL;
        PyObject *__pyx_t_8 = NULL;
        PyObject *__pyx_t_9 = NULL;
        PyObject *__pyx_t_10 = NULL;
        PyObject *__pyx_t_11 = NULL;
```

```
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("sin", 0);
                    if (__pyx_optional_args) {
                      if (__pyx_optional_args->__pyx_n > 0) {
                        __pyx_v_i = __pyx_optional_args->i;
                      }
                    }

                    /* "PyClical.pyx":1739
 *      {1,2,3}
 *      """
 *      if not (i is None):             # <<<<<<<<
 *          return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
 *      else:
 */
                    __pyx_t_1 = (__pyx_v_i != Py_None);
                    __pyx_t_2 = (__pyx_t_1 != 0);
                    if (__pyx_t_2) {

                      /* "PyClical.pyx":1740
 *      """
 *      if not (i is None):
 *          return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )             # <<<<<<<<
 *      else:
 *          try:
 */
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1740, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_3);
                      try {
                        __pyx_t_4 = sin(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));
                      } catch(...) {
                        __Pyx_CppExn2PyErr();
                        __PYX_ERR(0, 1740, __pyx_L1_error)
                      }
                      __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1740, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_5);
                      __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __pyx_r = __pyx_t_5;
                      __pyx_t_5 = 0;
                      goto __pyx_L0;

                      /* "PyClical.pyx":1739
 *      {1,2,3}
 *      """
 *      if not (i is None):             # <<<<<<<<
 *          return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
 *      else:
 */
                    }

                    /* "PyClical.pyx":1742
 *          return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:             # <<<<<<<<
 *              return math.sin(obj)
 *          except:
 */
                    /*else*/ {
                      {
                        __Pyx_PyThreadState_declare
                        __Pyx_PyThreadState_assign
                        __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                        __Pyx_XGOTREF(__pyx_t_6);
                        __Pyx_XGOTREF(__pyx_t_7);
                        __Pyx_XGOTREF(__pyx_t_8);
                        /*try:*/ {

                          /* "PyClical.pyx":1743
 *      else:
 *          try:
 *              return math.sin(obj)             # <<<<<<<<
 *          except:
 *              return clifford().wrap( glucat.sin(toClifford(obj)) )
 */
                          __Pyx_XDECREF(__pyx_r);
                          __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1743, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_3);
                          __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_sin); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1743, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_9);
```

```
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1743, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1742
 *         return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<<<<<<<
 *             return math.sin(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1744
 *         try:
 *             return math.sin(obj)
 *         except:             # <<<<<<<<<<<<<<
 *             return clifford().wrap( glucat.sin(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1744, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1745
 *             return math.sin(obj)
 *         except:
 *             return clifford().wrap( glucat.sin(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * cpdef inline asin(obj,i = None):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1745, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), sin(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1745, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1742
 *         return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<<<<<<<
 *             return math.sin(obj)
 *         except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
```

```
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1728
 *              return clifford().wrap( glucat.acosh(toClifford(obj)) )
 *
 * cpdef inline sin(obj,i = None):              # «««««««
 *     """
 *     Sine of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_65sin(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_64sin[] = "\n    Sine of multivector with optional complexifier.\n\n    »> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n    {-1}\n    »> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n    {-1}\n    »> x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n    {1,2,3}\n    ";
                static PyObject *__pyx_pw_8PyClical_65sin(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_obj = 0;
                  PyObject *__pyx_v_i = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("sin (wrapper)", 0);
                  {
                    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
                    PyObject* values[2] = {0,0};
                    values[1] = ((PyObject *)Py_None);
                    if (unlikely(__pyx_kwds)) {
                      Py_ssize_t kw_args;
                      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
                      switch (pos_args) {
                        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                        CYTHON_FALLTHROUGH;
                        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                        CYTHON_FALLTHROUGH;
                        case  0: break;
                        default: goto __pyx_L5_argtuple_error;
                      }
                      kw_args = PyDict_Size(__pyx_kwds);
                      switch (pos_args) {
                        case  0:
                        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
                        else goto __pyx_L5_argtuple_error;
                        CYTHON_FALLTHROUGH;
                        case  1:
                        if (kw_args > 0) {
                          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
                          if (value) { values[1] = value; kw_args--; }
                        }
                      }
                      if (unlikely(kw_args > 0)) {
                        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "sin") < 0)) __PYX_ERR(0, 1728, __pyx_L3_error)
```

```
            }
          } else {
            switch (PyTuple_GET_SIZE(__pyx_args)) {
              case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
              CYTHON_FALLTHROUGH;
              case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
              break;
              default: goto __pyx_L5_argtuple_error;
            }
          }
          __pyx_v_obj = values[0];
          __pyx_v_i = values[1];
        }
        goto __pyx_L4_argument_unpacking_done;
        __pyx_L5_argtuple_error:;
        __Pyx_RaiseArgtupleInvalid("sin", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
 __PYX_ERR(0, 1728, __pyx_L3_error)
        __pyx_L3_error:;
        __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __Pyx_RefNannyFinishContext();
        return NULL;
        __pyx_L4_argument_unpacking_done:;
        __pyx_r = __pyx_pf_8PyClical_64sin(__pyx_self, __pyx_v_obj, __pyx_v_i);

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_64sin(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 *__pyx_v_obj, PyObject *__pyx_v_i) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        struct __pyx_opt_args_8PyClical_sin __pyx_t_2;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("sin", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_2.__pyx_n = 1;
        __pyx_t_2.i = __pyx_v_i;
        __pyx_t_1 = __pyx_f_8PyClical_sin(__pyx_v_obj, 0, &__pyx_t_2); if
 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1728, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1747
 *             return clifford().wrap( glucat.sin(toClifford(obj)) )
 *
 * cpdef inline asin(obj,i = None):             # «««««««
 *     """
 *     Inverse sine of multivector with optional complexifier.
 */

      static PyObject *__pyx_pw_8PyClical_67asin(PyObject *__pyx_self, PyObject *__pyx_args,
 PyObject *__pyx_kwds); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asin(PyObject *__pyx_v_obj,
 CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asin *__pyx_optional_args) {
        PyObject *__pyx_v_i = ((PyObject *)Py_None);
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        int __pyx_t_1;
        int __pyx_t_2;
        PyObject *__pyx_t_3 = NULL;
        Clifford __pyx_t_4;
        PyObject *__pyx_t_5 = NULL;
        PyObject *__pyx_t_6 = NULL;
        PyObject *__pyx_t_7 = NULL;
        PyObject *__pyx_t_8 = NULL;
        PyObject *__pyx_t_9 = NULL;
        PyObject *__pyx_t_10 = NULL;
        PyObject *__pyx_t_11 = NULL;
        int __pyx_lineno = 0;
```

```
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("asin", 0);
                  if (__pyx_optional_args) {
                    if (__pyx_optional_args->__pyx_n > 0) {
                      __pyx_v_i = __pyx_optional_args->i;
                    }
                  }

                  /* "PyClical.pyx":1760
 *      {1,2,3}
 *      """
 *      if not (i is None):             # <<<<<<<<
 *          return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
 *      else:
 */
                  __pyx_t_1 = (__pyx_v_i != Py_None);
                  __pyx_t_2 = (__pyx_t_1 != 0);
                  if (__pyx_t_2) {

                    /* "PyClical.pyx":1761
 *      """
 *      if not (i is None):
 *          return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )             # <<<<<<<<
 *      else:
 *          try:
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1761, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_3);
                    try {
                      __pyx_t_4 = asin(__pyx_f_8PyClical_toClifford(__pyx_v_obj), __pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1761, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1761, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1760
 *      {1,2,3}
 *      """
 *      if not (i is None):             # <<<<<<<<
 *          return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
 *      else:
 */
                  }

                  /* "PyClical.pyx":1763
 *          return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:             # <<<<<<<<
 *              return math.asin(obj)
 *          except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1764
 *      else:
 *          try:
 *              return math.asin(obj)             # <<<<<<<<
 *          except:
 *              return clifford().wrap( glucat.asin(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1764, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_asin); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1764, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
```

```
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
      __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1764, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1763
 *           return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
 *       else:
 *           try:              # <<<<<<<<<<<<<<
 *               return math.asin(obj)
 *           except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1765
 *           try:
 *               return math.asin(obj)
 *           except:              # <<<<<<<<<<<<<<
 *               return clifford().wrap( glucat.asin(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno,
      __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
      1765, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1766
 *               return math.asin(obj)
 *           except:
 *               return clifford().wrap( glucat.asin(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * cpdef inline sinh(obj):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
      *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1766,
      __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
      __pyx_obj_8PyClical_clifford *)__pyx_t_10), asin(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
      (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1766, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1763
 *           return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
 *       else:
 *           try:              # <<<<<<<<<<<<<<
 *               return math.asin(obj)
 *           except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L1_error;
```

```
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1747
 *               return clifford().wrap( glucat.sin(toClifford(obj)) )
 *
 * cpdef inline asin(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse sine of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_67asin(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                static char __pyx_doc_8PyClical_66asin[] = "\n    Inverse sine of multivector with optional complexifier.\n\n    >>> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n    {-1}\n    >>> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n    {-1}\n    >>> print(asin(1) / pi)\n    0.5\n    >>> x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n    {1,2,3}\n    ";
                static PyObject *__pyx_pw_8PyClical_67asin(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                  PyObject *__pyx_v_obj = 0;
                  PyObject *__pyx_v_i = 0;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("asin (wrapper)", 0);
                  {
                    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
                    PyObject* values[2] = {0,0};
                    values[1] = ((PyObject *)Py_None);
                    if (unlikely(__pyx_kwds)) {
                      Py_ssize_t kw_args;
                      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
                      switch (pos_args) {
                        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                        CYTHON_FALLTHROUGH;
                        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                        CYTHON_FALLTHROUGH;
                        case  0: break;
                        default: goto __pyx_L5_argtuple_error;
                      }
                      kw_args = PyDict_Size(__pyx_kwds);
                      switch (pos_args) {
                        case  0:
                        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
                        else goto __pyx_L5_argtuple_error;
                        CYTHON_FALLTHROUGH;
                        case  1:
                        if (kw_args > 0) {
                          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
                          if (value) { values[1] = value; kw_args--; }
                        }
                      }
                      if (unlikely(kw_args > 0)) {
                        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "asin") < 0)) __PYX_ERR(0, 1747, __pyx_L3_error)
                      }
```

```
        } else {
          switch (PyTuple_GET_SIZE(__pyx_args)) {
            case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
            CYTHON_FALLTHROUGH;
            case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
            break;
            default: goto __pyx_L5_argtuple_error;
          }
        }
        __pyx_v_obj = values[0];
        __pyx_v_i = values[1];
      }
      goto __pyx_L4_argument_unpacking_done;
      __pyx_L5_argtuple_error:;
      __Pyx_RaiseArgtupleInvalid("asin", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1747, __pyx_L3_error)
      __pyx_L3_error:;
      __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __Pyx_RefNannyFinishContext();
      return NULL;
      __pyx_L4_argument_unpacking_done:;
      __pyx_r = __pyx_pf_8PyClical_66asin(__pyx_self, __pyx_v_obj, __pyx_v_i);

      /* function exit code */
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    static PyObject *__pyx_pf_8PyClical_66asin(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      struct __pyx_opt_args_8PyClical_asin __pyx_t_2;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("asin", 0);
      __Pyx_XDECREF(__pyx_r);
      __pyx_t_2.__pyx_n = 1;
      __pyx_t_2.i = __pyx_v_i;
      __pyx_t_1 = __pyx_f_8PyClical_asin(__pyx_v_obj, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1747, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_1);
      __pyx_r = __pyx_t_1;
      __pyx_t_1 = 0;
      goto __pyx_L0;

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_1);
      __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = NULL;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* "PyClical.pyx":1768
 *     return clifford().wrap( glucat.asin(toClifford(obj)) )
 *
 * cpdef inline sinh(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Hyperbolic sine of multivector.
 */

    static PyObject *__pyx_pw_8PyClical_69sinh(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
    static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sinh(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
      PyObject *__pyx_r = NULL;
      __Pyx_RefNannyDeclarations
      PyObject *__pyx_t_1 = NULL;
      PyObject *__pyx_t_2 = NULL;
      PyObject *__pyx_t_3 = NULL;
      PyObject *__pyx_t_4 = NULL;
      PyObject *__pyx_t_5 = NULL;
      PyObject *__pyx_t_6 = NULL;
      PyObject *__pyx_t_7 = NULL;
      PyObject *__pyx_t_8 = NULL;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      __Pyx_RefNannySetupContext("sinh", 0);

      /* "PyClical.pyx":1777
```

```
 *     0.5{1,2}
 *     """
 *     try:              # <<<<<<<<<<
 *         return math.sinh(obj)
 *     except:
 */
                {
                  __Pyx_PyThreadState_declare
                  __Pyx_PyThreadState_assign
                  __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
                  __Pyx_XGOTREF(__pyx_t_1);
                  __Pyx_XGOTREF(__pyx_t_2);
                  __Pyx_XGOTREF(__pyx_t_3);
                  /*try:*/ {

                    /* "PyClical.pyx":1778
 *     """
 *     try:
 *         return math.sinh(obj)             # <<<<<<<<<<
 *     except:
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )
 */
                    __Pyx_XDECREF(__pyx_r);
                    __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1778, __pyx_L3_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_sinh); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1778, __pyx_L3_error)
                    __Pyx_GOTREF(__pyx_t_6);
                    __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                    __pyx_t_5 = NULL;
                    if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
                      __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
                      if (likely(__pyx_t_5)) {
                        PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
                        __Pyx_INCREF(__pyx_t_5);
                        __Pyx_INCREF(function);
                        __Pyx_DECREF_SET(__pyx_t_6, function);
                      }
                    }
                    __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
                    __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                    if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1778, __pyx_L3_error)
                    __Pyx_GOTREF(__pyx_t_4);
                    __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                    __pyx_r = __pyx_t_4;
                    __pyx_t_4 = 0;
                    goto __pyx_L7_try_return;

                    /* "PyClical.pyx":1777
 *     0.5{1,2}
 *     """
 *     try:              # <<<<<<<<<<
 *         return math.sinh(obj)
 *     except:
 */
                  }
                  __pyx_L3_error:;
                  __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
                  __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                  __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;

                  /* "PyClical.pyx":1779
 *     try:
 *         return math.sinh(obj)
 *     except:             # <<<<<<<<<<
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )
 *
 */
                  /*except:*/ {
                    __Pyx_AddTraceback("PyClical.sinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0, 1779, __pyx_L5_except_error)
                    __Pyx_GOTREF(__pyx_t_4);
                    __Pyx_GOTREF(__pyx_t_6);
                    __Pyx_GOTREF(__pyx_t_5);

                    /* "PyClical.pyx":1780
 *         return math.sinh(obj)
 *     except:
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )             # <<<<<<<<<<
 *
 * cpdef inline asinh(obj,i = None):
 */
                    __Pyx_XDECREF(__pyx_r);
```

```
                        __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1780, __pyx_L5_except_error)
                        __Pyx_GOTREF(__pyx_t_7);
                        __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_7), sinh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 1780, __pyx_L5_except_error)
                        __Pyx_GOTREF(__pyx_t_8);
                        __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                        __pyx_r = __pyx_t_8;
                        __pyx_t_8 = 0;
                        __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                        goto __pyx_L6_except_return;
                      }
                      __pyx_L5_except_error:;

                      /* "PyClical.pyx":1777
 *      0.5{1,2}
 *      """
 *      try:              # <<<<<<<<<<<<<<
 *          return math.sinh(obj)
 *      except:
 */
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L1_error;
                      __pyx_L7_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L0;
                      __pyx_L6_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_1);
                      __Pyx_XGIVEREF(__pyx_t_2);
                      __Pyx_XGIVEREF(__pyx_t_3);
                      __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                      goto __pyx_L0;
                    }

                    /* "PyClical.pyx":1768
 *             return clifford().wrap( glucat.asin(toClifford(obj)) )
 *
 * cpdef inline sinh(obj):             # <<<<<<<<<<<<<<
 *      """
 *      Hyperbolic sine of multivector.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_4);
                    __Pyx_XDECREF(__pyx_t_5);
                    __Pyx_XDECREF(__pyx_t_6);
                    __Pyx_XDECREF(__pyx_t_7);
                    __Pyx_XDECREF(__pyx_t_8);
                    __Pyx_AddTraceback("PyClical.sinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_69sinh(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static char __pyx_doc_8PyClical_68sinh[] = "\n    Hyperbolic sine of multivector.\n\n >>> x=clifford(\"{1,2}\") * pi/2; print(sinh(x))\n    {1,2}\n    >>> x=clifford(\"{1,2}\") * pi/6; print(sinh(x))\n    0.5{1,2}\n    ";
                  static PyObject *__pyx_pw_8PyClical_69sinh(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("sinh (wrapper)", 0);
                    __pyx_r = __pyx_pf_8PyClical_68sinh(__pyx_self, ((PyObject *)__pyx_v_obj));

                    /* function exit code */
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  static PyObject *__pyx_pf_8PyClical_68sinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
```

```
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("sinh", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __pyx_f_8PyClical_sinh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1768, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.sinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1782
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )
 *
 * cpdef inline asinh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic sine of multivector with optional complexifier.
 */

                  static PyObject *__pyx_pw_8PyClical_71asinh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asinh(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asinh *__pyx_optional_args) {
                    PyObject *__pyx_v_i = ((PyObject *)Py_None);
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    int __pyx_t_1;
                    int __pyx_t_2;
                    PyObject *__pyx_t_3 = NULL;
                    Clifford __pyx_t_4;
                    PyObject *__pyx_t_5 = NULL;
                    PyObject *__pyx_t_6 = NULL;
                    PyObject *__pyx_t_7 = NULL;
                    PyObject *__pyx_t_8 = NULL;
                    PyObject *__pyx_t_9 = NULL;
                    PyObject *__pyx_t_10 = NULL;
                    PyObject *__pyx_t_11 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("asinh", 0);
                    if (__pyx_optional_args) {
                      if (__pyx_optional_args->__pyx_n > 0) {
                        __pyx_v_i = __pyx_optional_args->i;
                      }
                    }

                    /* "PyClical.pyx":1793
 *     {1,2}
 *     """
 *     if not (i is None):             # <<<<<<<<<<<<<<
 *         return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
 *     else:
 */
                    __pyx_t_1 = (__pyx_v_i != Py_None);
                    __pyx_t_2 = (__pyx_t_1 != 0);
                    if (__pyx_t_2) {

                      /* "PyClical.pyx":1794
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )             # <<<<<<<<<<<<<<
 *     else:
 *         try:
 */
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1794, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_3);
                      try {
                        __pyx_t_4 = asinh(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
```

```
__pyx_f_8PyClical_toClifford(__pyx_v_i));
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1794, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1794, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1793
 *      {1,2}
 *      """
 *      if not (i is None):             # <<<<<<<<
 *          return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
 *      else:
 */
                  }

                  /* "PyClical.pyx":1796
 *          return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:              # <<<<<<<<
 *              return math.asinh(obj)
 *          except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1797
 *      else:
 *          try:
 *              return math.asinh(obj)             # <<<<<<<<
 *          except:
 *              return clifford().wrap( glucat.asinh(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1797, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_asinh); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1797, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1797, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1796
 *          return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:              # <<<<<<<<
 *              return math.asinh(obj)
 *          except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

```

```
                        /* "PyClical.pyx":1798
 *         try:
 *             return math.asinh(obj)
 *         except:             # <<<<<<<<
 *             return clifford().wrap( glucat.asinh(toClifford(obj)) )
 *
 */
                        /*except:*/ {
                          __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                          if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1798, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_5);
                          __Pyx_GOTREF(__pyx_t_9);
                          __Pyx_GOTREF(__pyx_t_3);

                          /* "PyClical.pyx":1799
 *             return math.asinh(obj)
 *         except:
 *             return clifford().wrap( glucat.asinh(toClifford(obj)) )             # <<<<<<<<
 *
 * cpdef inline tan(obj,i = None):
 */
                          __Pyx_XDECREF(__pyx_r);
                          __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1799, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_10);
                          __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), asinh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1799, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_11);
                          __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                          __pyx_r = __pyx_t_11;
                          __pyx_t_11 = 0;
                          __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                          __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                          __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                          goto __pyx_L7_except_return;
                        }
                        __pyx_L6_except_error:;

                        /* "PyClical.pyx":1796
 *         return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.asinh(obj)
 *         except:
 */
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L1_error;
                        __pyx_L8_try_return:;
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L0;
                        __pyx_L7_except_return:;
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L0;
                      }
                    }

                    /* "PyClical.pyx":1782
 *         return clifford().wrap( glucat.sinh(toClifford(obj)) )
 *
 * cpdef inline asinh(obj,i = None):             # <<<<<<<<
 *     """
 *     Inverse hyperbolic sine of multivector with optional complexifier.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_XDECREF(__pyx_t_5);
                    __Pyx_XDECREF(__pyx_t_9);
                    __Pyx_XDECREF(__pyx_t_10);
                    __Pyx_XDECREF(__pyx_t_11);
                    __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
```

```
  __Pyx_XGIVEREF(__pyx_r);
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_71asinh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_8PyClical_70asinh[] = "\n    Inverse hyperbolic sine of multivector with optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(asinh(x,\"{1,2,3}\") * 2/pi)\n    {1,2}\n    >>> x=clifford(\"{1,2}\"); print(asinh(x) * 2/pi)\n    {1,2}\n    >>> x=clifford(\"{1,2}\") / 2; print(asinh(x) * 6/pi)\n    {1,2}\n    ";
static PyObject *__pyx_pw_8PyClical_71asinh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_i = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("asinh (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "asinh") < 0)) __PYX_ERR(0, 1782, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("asinh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1782, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_70asinh(__pyx_self, __pyx_v_obj, __pyx_v_i);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_70asinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
```

```
                    struct __pyx_opt_args_8PyClical_asinh __pyx_t_2;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("asinh", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2.__pyx_n = 1;
                    __pyx_t_2.i = __pyx_v_i;
                    __pyx_t_1 = __pyx_f_8PyClical_asinh(__pyx_v_obj, 0, &__pyx_t_2); if
      (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1782, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1801
       *             return clifford().wrap( glucat.asinh(toClifford(obj)) )
       *
       * cpdef inline tan(obj,i = None):             # <<<<<<<<
       *     """
       *     Tangent of multivector with optional complexifier.
       */

                  static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args,
      PyObject *__pyx_kwds); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_tan(PyObject *__pyx_v_obj,
      CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_tan *__pyx_optional_args) {
                    PyObject *__pyx_v_i = ((PyObject *)Py_None);
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    int __pyx_t_1;
                    int __pyx_t_2;
                    PyObject *__pyx_t_3 = NULL;
                    Clifford __pyx_t_4;
                    PyObject *__pyx_t_5 = NULL;
                    PyObject *__pyx_t_6 = NULL;
                    PyObject *__pyx_t_7 = NULL;
                    PyObject *__pyx_t_8 = NULL;
                    PyObject *__pyx_t_9 = NULL;
                    PyObject *__pyx_t_10 = NULL;
                    PyObject *__pyx_t_11 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("tan", 0);
                    if (__pyx_optional_args) {
                      if (__pyx_optional_args->__pyx_n > 0) {
                        __pyx_v_i = __pyx_optional_args->i;
                      }
                    }

                    /* "PyClical.pyx":1810
       *     0.7616{1,2}
       *     """
       *     if not (i is None):             # <<<<<<<<
       *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
       *     else:
       */
                    __pyx_t_1 = (__pyx_v_i != Py_None);
                    __pyx_t_2 = (__pyx_t_1 != 0);
                    if (__pyx_t_2) {

                      /* "PyClical.pyx":1811
       *     """
       *     if not (i is None):
       *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )             # <<<<<<<<
       *     else:
       *         try:
       */
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
      *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1811, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_3);
                      try {
                        __pyx_t_4 = tan(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
      __pyx_f_8PyClical_toClifford(__pyx_v_i));
```

```
                    } catch(...) {
                      __Pyx_CppExn2PyErr();
                      __PYX_ERR(0, 1811, __pyx_L1_error)
                    }
                    __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1811, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_5);
                    __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                    __pyx_r = __pyx_t_5;
                    __pyx_t_5 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1810
 *     0.7616{1,2}
 *     """
 *     if not (i is None):              # <<<<<<<<
 *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
 *     else:
 */
                  }

                  /* "PyClical.pyx":1813
 *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.tan(obj)
 *         except:
 */
                  /*else*/ {
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                      __Pyx_XGOTREF(__pyx_t_6);
                      __Pyx_XGOTREF(__pyx_t_7);
                      __Pyx_XGOTREF(__pyx_t_8);
                      /*try:*/ {

                        /* "PyClical.pyx":1814
 *     else:
 *         try:
 *             return math.tan(obj)             # <<<<<<<<
 *         except:
 *             return clifford().wrap( glucat.tan(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1814, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_tan); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1814, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1814, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1813
 *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.tan(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1815
```

```
 *         try:
 *             return math.tan(obj)
 *         except:              # <<<<<<<<<<<<<<
 *             return clifford().wrap( glucat.tan(toClifford(obj)) )
 *
 */
                        /*except:*/ {
                          __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno,
      __pyx_filename);
                          if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
      1815, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_5);
                          __Pyx_GOTREF(__pyx_t_9);
                          __Pyx_GOTREF(__pyx_t_3);

                          /* "PyClical.pyx":1816
 *             return math.tan(obj)
 *         except:
 *             return clifford().wrap( glucat.tan(toClifford(obj)) )              # <<<<<<<<<<<<<<
 *
 * cpdef inline atan(obj,i = None):
 */
                          __Pyx_XDECREF(__pyx_r);
                          __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
      *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1816,
      __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_10);
                          __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
      __pyx_obj_8PyClical_clifford *)__pyx_t_10), tan(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
      (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1816, __pyx_L6_except_error)
                          __Pyx_GOTREF(__pyx_t_11);
                          __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                          __pyx_r = __pyx_t_11;
                          __pyx_t_11 = 0;
                          __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                          __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                          __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                          goto __pyx_L7_except_return;
                        }
                        __pyx_L6_except_error:;

                        /* "PyClical.pyx":1813
 *         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<<<<<<<
 *             return math.tan(obj)
 *         except:
 */
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L1_error;
                        __pyx_L8_try_return:;
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L0;
                        __pyx_L7_except_return:;
                        __Pyx_XGIVEREF(__pyx_t_6);
                        __Pyx_XGIVEREF(__pyx_t_7);
                        __Pyx_XGIVEREF(__pyx_t_8);
                        __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                        goto __pyx_L0;
                      }
                    }

                    /* "PyClical.pyx":1801
 *             return clifford().wrap( glucat.asinh(toClifford(obj)) )
 *
 * cpdef inline tan(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Tangent of multivector with optional complexifier.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_3);
                    __Pyx_XDECREF(__pyx_t_5);
                    __Pyx_XDECREF(__pyx_t_9);
                    __Pyx_XDECREF(__pyx_t_10);
                    __Pyx_XDECREF(__pyx_t_11);
                    __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
```

```
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_8PyClical_72tan[] = "\n    Tangent of multivector with optional complexifier.\n\n    >>> x=clifford(\"{1,2}\"); print(tan(x,\"{1,2,3}\"))\n    0.7616{1,2}\n    >>> x=clifford(\"{1,2}\"); print(tan(x))\n    0.7616{1,2}\n    ";
static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_i = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("tan (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "tan") < 0)) __PYX_ERR(0, 1801, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("tan", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1801, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_72tan(__pyx_self, __pyx_v_obj, __pyx_v_i);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_72tan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  struct __pyx_opt_args_8PyClical_tan __pyx_t_2;
  int __pyx_lineno = 0;
```

```
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("tan", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2.__pyx_n = 1;
                    __pyx_t_2.i = __pyx_v_i;
                    __pyx_t_1 = __pyx_f_8PyClical_tan(__pyx_v_obj, 0, &__pyx_t_2); if
      (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1801, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1818
 *               return clifford().wrap( glucat.tan(toClifford(obj)) )
 *
 * cpdef inline atan(obj,i = None):             # <<<<<<<<
 *     """
 *     Inverse tangent of multivector with optional complexifier.
 */

                  static PyObject *__pyx_pw_8PyClical_75atan(PyObject *__pyx_self, PyObject *__pyx_args,
      PyObject *__pyx_kwds); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_atan(PyObject *__pyx_v_obj,
      CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atan *__pyx_optional_args) {
                    PyObject *__pyx_v_i = ((PyObject *)Py_None);
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    int __pyx_t_1;
                    int __pyx_t_2;
                    PyObject *__pyx_t_3 = NULL;
                    Clifford __pyx_t_4;
                    PyObject *__pyx_t_5 = NULL;
                    PyObject *__pyx_t_6 = NULL;
                    PyObject *__pyx_t_7 = NULL;
                    PyObject *__pyx_t_8 = NULL;
                    PyObject *__pyx_t_9 = NULL;
                    PyObject *__pyx_t_10 = NULL;
                    PyObject *__pyx_t_11 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("atan", 0);
                    if (__pyx_optional_args) {
                      if (__pyx_optional_args->__pyx_n > 0) {
                        __pyx_v_i = __pyx_optional_args->i;
                      }
                    }

                    /* "PyClical.pyx":1827
 *     {1}
 *     """
 *     if not (i is None):            # <<<<<<<<
 *         return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
 *     else:
 */
                    __pyx_t_1 = (__pyx_v_i != Py_None);
                    __pyx_t_2 = (__pyx_t_1 != 0);
                    if (__pyx_t_2) {

                      /* "PyClical.pyx":1828
 *     """
 *     if not (i is None):
 *         return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )             # <<<<<<<<
 *     else:
 *         try:
 */
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
      *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1828, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_3);
                      try {
                        __pyx_t_4 = atan(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
      __pyx_f_8PyClical_toClifford(__pyx_v_i));
                      } catch(...) {
                        __Pyx_CppExn2PyErr();
```

```
                        __PYX_ERR(0, 1828, __pyx_L1_error)
                      }
                      __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1828, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_5);
                      __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __pyx_r = __pyx_t_5;
                      __pyx_t_5 = 0;
                      goto __pyx_L0;

                      /* "PyClical.pyx":1827
 *      {1}
 *      """
 *      if not (i is None):              # <<<<<<<<
 *          return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
 *      else:
 */
                    }

                    /* "PyClical.pyx":1830
 *          return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:             # <<<<<<<<
 *              return math.atan(obj)
 *          except:
 */
                    /*else*/ {
                      {
                        __Pyx_PyThreadState_declare
                        __Pyx_PyThreadState_assign
                        __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                        __Pyx_XGOTREF(__pyx_t_6);
                        __Pyx_XGOTREF(__pyx_t_7);
                        __Pyx_XGOTREF(__pyx_t_8);
                        /*try:*/ {

                          /* "PyClical.pyx":1831
 *      else:
 *          try:
 *              return math.atan(obj)             # <<<<<<<<
 *          except:
 *              return clifford().wrap( glucat.atan(toClifford(obj)) )
 */
                          __Pyx_XDECREF(__pyx_r);
                          __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1831, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_3);
                          __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_atan); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1831, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_9);
                          __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                          __pyx_t_3 = NULL;
                          if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                            __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                            if (likely(__pyx_t_3)) {
                              PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                              __Pyx_INCREF(__pyx_t_3);
                              __Pyx_INCREF(function);
                              __Pyx_DECREF_SET(__pyx_t_9, function);
                            }
                          }
                          __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                          __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                          if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1831, __pyx_L4_error)
                          __Pyx_GOTREF(__pyx_t_5);
                          __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                          __pyx_r = __pyx_t_5;
                          __pyx_t_5 = 0;
                          goto __pyx_L8_try_return;

                          /* "PyClical.pyx":1830
 *          return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:             # <<<<<<<<
 *              return math.atan(obj)
 *          except:
 */
                        }
                        __pyx_L4_error:;
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                        /* "PyClical.pyx":1832
 *          try:
 *              return math.atan(obj)
```

```
 *         except:              # «««««««
 *             return clifford().wrap( glucat.atan(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1832, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1833
 *             return math.atan(obj)
 *         except:
 *             return clifford().wrap( glucat.atan(toClifford(obj)) )             # «««««««
 *
 * cpdef inline tanh(obj):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1833, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), atan(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1833, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        goto __pyx_L7_except_return;
                      }
                      __pyx_L6_except_error:;

                      /* "PyClical.pyx":1830
 *         return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:              # «««««««
 *             return math.atan(obj)
 *         except:
 */
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L1_error;
                      __pyx_L8_try_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                      __pyx_L7_except_return:;
                      __Pyx_XGIVEREF(__pyx_t_6);
                      __Pyx_XGIVEREF(__pyx_t_7);
                      __Pyx_XGIVEREF(__pyx_t_8);
                      __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
                      goto __pyx_L0;
                    }
                  }

                  /* "PyClical.pyx":1818
 *             return clifford().wrap( glucat.tan(toClifford(obj)) )
 *
 * cpdef inline atan(obj,i = None):              # «««««««
 *     """
 *     Inverse tangent of multivector with optional complexifier.
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_XDECREF(__pyx_t_5);
                  __Pyx_XDECREF(__pyx_t_9);
                  __Pyx_XDECREF(__pyx_t_10);
                  __Pyx_XDECREF(__pyx_t_11);
                  __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = 0;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
```

```
}

/* Python wrapper */
static PyObject *__pyx_pw_8PyClical_75atan(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
static char __pyx_doc_8PyClical_74atan[] = "\n    Inverse tangent of multivector with optional complexifier.\n\n    >>> s=index_set({1,2,3}); x=clifford(\"{1}\"); print(tan(atan(x,s),s))\n    {1}\n    >>> x=clifford(\"{1}\"); print(tan(atan(x)))\n    {1}\n    ";
static PyObject *__pyx_pw_8PyClical_75atan(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
  PyObject *__pyx_v_obj = 0;
  PyObject *__pyx_v_i = 0;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  PyObject *__pyx_r = 0;
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("atan (wrapper)", 0);
  {
    static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
    PyObject* values[2] = {0,0};
    values[1] = ((PyObject *)Py_None);
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "atan") < 0)) __PYX_ERR(0, 1818, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_obj = values[0];
    __pyx_v_i = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("atan", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1818, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  __pyx_r = __pyx_pf_8PyClical_74atan(__pyx_self, __pyx_v_obj, __pyx_v_i);

  /* function exit code */
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_74atan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  struct __pyx_opt_args_8PyClical_atan __pyx_t_2;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
```

```
                    __Pyx_RefNannySetupContext("atan", 0);
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_2.__pyx_n = 1;
                    __pyx_t_2.i = __pyx_v_i;
                    __pyx_t_1 = __pyx_f_8PyClical_atan(__pyx_v_obj, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1818, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = NULL;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "PyClical.pyx":1835
 *             return clifford().wrap( glucat.atan(toClifford(obj)) )
 *
 * cpdef inline tanh(obj):              # <<<<<<<<<<<<<<
 *     """
 *     Hyperbolic tangent of multivector.
 */

                  static PyObject *__pyx_pw_8PyClical_77tanh(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
                  static CYTHON_INLINE PyObject *__pyx_f_8PyClical_tanh(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    PyObject *__pyx_t_2 = NULL;
                    PyObject *__pyx_t_3 = NULL;
                    PyObject *__pyx_t_4 = NULL;
                    PyObject *__pyx_t_5 = NULL;
                    PyObject *__pyx_t_6 = NULL;
                    PyObject *__pyx_t_7 = NULL;
                    PyObject *__pyx_t_8 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("tanh", 0);

                    /* "PyClical.pyx":1842
 *     {1,2}
 *     """
 *     try:              # <<<<<<<<<<<<<<
 *         return math.tanh(obj)
 *     except:
 */
                    {
                      __Pyx_PyThreadState_declare
                      __Pyx_PyThreadState_assign
                      __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
                      __Pyx_XGOTREF(__pyx_t_1);
                      __Pyx_XGOTREF(__pyx_t_2);
                      __Pyx_XGOTREF(__pyx_t_3);
                      /*try:*/ {

                        /* "PyClical.pyx":1843
 *     """
 *     try:
 *         return math.tanh(obj)              # <<<<<<<<<<<<<<
 *     except:
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1843, __pyx_L3_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_tanh); if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1843, __pyx_L3_error)
                        __Pyx_GOTREF(__pyx_t_6);
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __pyx_t_5 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
                          __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
                          if (likely(__pyx_t_5)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
                            __Pyx_INCREF(__pyx_t_5);
                            __Pyx_INCREF(function);
```

```
                    __Pyx_DECREF_SET(__pyx_t_6, function);
                  }
                }
                __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
                __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1843, __pyx_L3_error)
                __Pyx_GOTREF(__pyx_t_4);
                __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                __pyx_r = __pyx_t_4;
                __pyx_t_4 = 0;
                goto __pyx_L7_try_return;

                /* "PyClical.pyx":1842
 *     {1,2}
 *     """
 *     try:             # <<<<<<<<<<<<<<
 *         return math.tanh(obj)
 *     except:
 */
              }
              __pyx_L3_error:;
              __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
              __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
              __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;

              /* "PyClical.pyx":1844
 *     try:
 *         return math.tanh(obj)
 *     except:             # <<<<<<<<<<<<<<
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )
 *
 */
              /*except:*/ {
                __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0, 1844, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_4);
                __Pyx_GOTREF(__pyx_t_6);
                __Pyx_GOTREF(__pyx_t_5);

                /* "PyClical.pyx":1845
 *         return math.tanh(obj)
 *     except:
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * cpdef inline atanh(obj,i = None):
 */
                __Pyx_XDECREF(__pyx_r);
                __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1845, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_7);
                __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_7), tanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 1845, __pyx_L5_except_error)
                __Pyx_GOTREF(__pyx_t_8);
                __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
                __pyx_r = __pyx_t_8;
                __pyx_t_8 = 0;
                __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
                __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
                goto __pyx_L6_except_return;
              }
              __pyx_L5_except_error:;

              /* "PyClical.pyx":1842
 *     {1,2}
 *     """
 *     try:             # <<<<<<<<<<<<<<
 *         return math.tanh(obj)
 *     except:
 */
              __Pyx_XGIVEREF(__pyx_t_1);
              __Pyx_XGIVEREF(__pyx_t_2);
              __Pyx_XGIVEREF(__pyx_t_3);
              __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
              goto __pyx_L1_error;
              __pyx_L7_try_return:;
              __Pyx_XGIVEREF(__pyx_t_1);
              __Pyx_XGIVEREF(__pyx_t_2);
              __Pyx_XGIVEREF(__pyx_t_3);
              __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
              goto __pyx_L0;
              __pyx_L6_except_return:;
```

```
                        __Pyx_XGIVEREF(__pyx_t_1);
                        __Pyx_XGIVEREF(__pyx_t_2);
                        __Pyx_XGIVEREF(__pyx_t_3);
                        __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
                        goto __pyx_L0;
                      }

                      /* "PyClical.pyx":1835
 *             return clifford().wrap( glucat.atan(toClifford(obj)) )
 *
 * cpdef inline tanh(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Hyperbolic tangent of multivector.
 */

                      /* function exit code */
                      __pyx_L1_error:;
                      __Pyx_XDECREF(__pyx_t_4);
                      __Pyx_XDECREF(__pyx_t_5);
                      __Pyx_XDECREF(__pyx_t_6);
                      __Pyx_XDECREF(__pyx_t_7);
                      __Pyx_XDECREF(__pyx_t_8);
                      __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                      __pyx_r = 0;
                      __pyx_L0:;
                      __Pyx_XGIVEREF(__pyx_r);
                      __Pyx_RefNannyFinishContext();
                      return __pyx_r;
                    }

                    /* Python wrapper */
                    static PyObject *__pyx_pw_8PyClical_77tanh(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
                    static char __pyx_doc_8PyClical_76tanh[] = "\n    Hyperbolic tangent of
multivector.\n\n    >>> x=clifford(\"{1,2}\") * pi/4; print(tanh(x))\n    {1,2}\n    ";
                    static PyObject *__pyx_pw_8PyClical_77tanh(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
                      PyObject *__pyx_r = 0;
                      __Pyx_RefNannyDeclarations
                      __Pyx_RefNannySetupContext("tanh (wrapper)", 0);
                      __pyx_r = __pyx_pf_8PyClical_76tanh(__pyx_self, ((PyObject *)__pyx_v_obj));

                      /* function exit code */
                      __Pyx_RefNannyFinishContext();
                      return __pyx_r;
                    }

                    static PyObject *__pyx_pf_8PyClical_76tanh(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
                      PyObject *__pyx_r = NULL;
                      __Pyx_RefNannyDeclarations
                      PyObject *__pyx_t_1 = NULL;
                      int __pyx_lineno = 0;
                      const char *__pyx_filename = NULL;
                      int __pyx_clineno = 0;
                      __Pyx_RefNannySetupContext("tanh", 0);
                      __Pyx_XDECREF(__pyx_r);
                      __pyx_t_1 = __pyx_f_8PyClical_tanh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1835, __pyx_L1_error)
                      __Pyx_GOTREF(__pyx_t_1);
                      __pyx_r = __pyx_t_1;
                      __pyx_t_1 = 0;
                      goto __pyx_L0;

                      /* function exit code */
                      __pyx_L1_error:;
                      __Pyx_XDECREF(__pyx_t_1);
                      __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                      __pyx_r = NULL;
                      __pyx_L0:;
                      __Pyx_XGIVEREF(__pyx_r);
                      __Pyx_RefNannyFinishContext();
                      return __pyx_r;
                    }

                    /* "PyClical.pyx":1847
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )
 *
 * cpdef inline atanh(obj,i = None):             # <<<<<<<<<<<<<<
 *     """
 *     Inverse hyperbolic tangent of multivector with optional complexifier.
 */

                    static PyObject *__pyx_pw_8PyClical_79atanh(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
                    static CYTHON_INLINE PyObject *__pyx_f_8PyClical_atanh(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atanh *__pyx_optional_args) {
```

```
                PyObject *__pyx_v_i = ((PyObject *)Py_None);
                PyObject *__pyx_r = NULL;
                __Pyx_RefNannyDeclarations
                int __pyx_t_1;
                int __pyx_t_2;
                PyObject *__pyx_t_3 = NULL;
                Clifford __pyx_t_4;
                PyObject *__pyx_t_5 = NULL;
                PyObject *__pyx_t_6 = NULL;
                PyObject *__pyx_t_7 = NULL;
                PyObject *__pyx_t_8 = NULL;
                PyObject *__pyx_t_9 = NULL;
                PyObject *__pyx_t_10 = NULL;
                PyObject *__pyx_t_11 = NULL;
                int __pyx_lineno = 0;
                const char *__pyx_filename = NULL;
                int __pyx_clineno = 0;
                __Pyx_RefNannySetupContext("atanh", 0);
                if (__pyx_optional_args) {
                  if (__pyx_optional_args->__pyx_n > 0) {
                    __pyx_v_i = __pyx_optional_args->i;
                  }
                }

                /* "PyClical.pyx":1856
 *      {1,2}
 *      """
 *      if not (i is None):              # <<<<<<<<
 *          return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
 *      else:
 */
                __pyx_t_1 = (__pyx_v_i != Py_None);
                __pyx_t_2 = (__pyx_t_1 != 0);
                if (__pyx_t_2) {

                  /* "PyClical.pyx":1857
 *      """
 *      if not (i is None):
 *          return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )              # <<<<<<<<
 *      else:
 *          try:
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1857, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  try {
                    __pyx_t_4 = atanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj), __pyx_f_8PyClical_toClifford(__pyx_v_i));
                  } catch(...) {
                    __Pyx_CppExn2PyErr();
                    __PYX_ERR(0, 1857, __pyx_L1_error)
                  }
                  __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1857, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_5);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_5;
                  __pyx_t_5 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1856
 *      {1,2}
 *      """
 *      if not (i is None):              # <<<<<<<<
 *          return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
 *      else:
 */
                }

                /* "PyClical.pyx":1859
 *          return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
 *      else:
 *          try:              # <<<<<<<<
 *              return math.atanh(obj)
 *          except:
 */
                /*else*/ {
                  {
                    __Pyx_PyThreadState_declare
                    __Pyx_PyThreadState_assign
                    __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
                    __Pyx_XGOTREF(__pyx_t_6);
                    __Pyx_XGOTREF(__pyx_t_7);
                    __Pyx_XGOTREF(__pyx_t_8);
                    /*try:*/ {
```

```

                        /* "PyClical.pyx":1860
 *     else:
 *         try:
 *             return math.atanh(obj)             # <<<<<<<<
 *         except:
 *             return clifford().wrap( glucat.atanh(toClifford(obj)) )
 */
                        __Pyx_XDECREF(__pyx_r);
                        __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1860, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_3);
                        __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_atanh); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1860, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __pyx_t_3 = NULL;
                        if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
                          __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
                          if (likely(__pyx_t_3)) {
                            PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
                            __Pyx_INCREF(__pyx_t_3);
                            __Pyx_INCREF(function);
                            __Pyx_DECREF_SET(__pyx_t_9, function);
                          }
                        }
                        __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3, __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
                        __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                        if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1860, __pyx_L4_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
                        __pyx_r = __pyx_t_5;
                        __pyx_t_5 = 0;
                        goto __pyx_L8_try_return;

                        /* "PyClical.pyx":1859
 *         return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:             # <<<<<<<<
 *             return math.atanh(obj)
 *         except:
 */
                      }
                      __pyx_L4_error:;
                      __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                      __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
                      __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

                      /* "PyClical.pyx":1861
 *         try:
 *             return math.atanh(obj)
 *         except:             # <<<<<<<<
 *             return clifford().wrap( glucat.atanh(toClifford(obj)) )
 *
 */
                      /*except:*/ {
                        __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
                        if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0, 1861, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_5);
                        __Pyx_GOTREF(__pyx_t_9);
                        __Pyx_GOTREF(__pyx_t_3);

                        /* "PyClical.pyx":1862
 *             return math.atanh(obj)
 *         except:
 *             return clifford().wrap( glucat.atanh(toClifford(obj)) )             # <<<<<<<<
 *
 * cpdef inline random_clifford(index_set ixt, fill = 1.0):
 */
                        __Pyx_XDECREF(__pyx_r);
                        __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1862, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_10);
                        __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_10), atanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1862, __pyx_L6_except_error)
                        __Pyx_GOTREF(__pyx_t_11);
                        __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
                        __pyx_r = __pyx_t_11;
                        __pyx_t_11 = 0;
                        __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                        __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
                        __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
```

```
            goto __pyx_L7_except_return;
          }
          __pyx_L6_except_error:;

          /* "PyClical.pyx":1859
 *         return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
 *     else:
 *         try:              # «««««««
 *             return math.atanh(obj)
 *         except:
 */
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L1_error;
          __pyx_L8_try_return:;
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L0;
          __pyx_L7_except_return:;
          __Pyx_XGIVEREF(__pyx_t_6);
          __Pyx_XGIVEREF(__pyx_t_7);
          __Pyx_XGIVEREF(__pyx_t_8);
          __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
          goto __pyx_L0;
        }
      }

      /* "PyClical.pyx":1847
 *         return clifford().wrap( glucat.tanh(toClifford(obj)) )
 *
 * cpdef inline atanh(obj,i = None):             # «««««««
 *     """
 *     Inverse hyperbolic tangent of multivector with optional complexifier.
 */

      /* function exit code */
      __pyx_L1_error:;
      __Pyx_XDECREF(__pyx_t_3);
      __Pyx_XDECREF(__pyx_t_5);
      __Pyx_XDECREF(__pyx_t_9);
      __Pyx_XDECREF(__pyx_t_10);
      __Pyx_XDECREF(__pyx_t_11);
      __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
      __pyx_r = 0;
      __pyx_L0:;
      __Pyx_XGIVEREF(__pyx_r);
      __Pyx_RefNannyFinishContext();
      return __pyx_r;
    }

    /* Python wrapper */
    static PyObject *__pyx_pw_8PyClical_79atanh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
    static char __pyx_doc_8PyClical_78atanh[] = "\n    Inverse hyperbolic tangent of multivector with optional complexifier.\n\n    »> s=index_set({1,2,3}); x=clifford(\"{1,2}\"); print(tanh(atanh(x,s)))\n    {1,2}\n    »> x=clifford(\"{1,2}\"); print(tanh(atanh(x)))\n    {1,2}\n    ";
    static PyObject *__pyx_pw_8PyClical_79atanh(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
      PyObject *__pyx_v_obj = 0;
      PyObject *__pyx_v_i = 0;
      int __pyx_lineno = 0;
      const char *__pyx_filename = NULL;
      int __pyx_clineno = 0;
      PyObject *__pyx_r = 0;
      __Pyx_RefNannyDeclarations
      __Pyx_RefNannySetupContext("atanh (wrapper)", 0);
      {
        static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
        PyObject* values[2] = {0,0};
        values[1] = ((PyObject *)Py_None);
        if (unlikely(__pyx_kwds)) {
          Py_ssize_t kw_args;
          const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
          switch (pos_args) {
            case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
            CYTHON_FALLTHROUGH;
            case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
            CYTHON_FALLTHROUGH;
            case  0: break;
            default: goto __pyx_L5_argtuple_error;
          }
          kw_args = PyDict_Size(__pyx_kwds);
```

```
                switch (pos_args) {
                  case  0:
                  if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
                  else goto __pyx_L5_argtuple_error;
                  CYTHON_FALLTHROUGH;
                  case  1:
                  if (kw_args > 0) {
                    PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
                    if (value) { values[1] = value; kw_args--; }
                  }
                }
                if (unlikely(kw_args > 0)) {
                  if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "atanh") < 0)) __PYX_ERR(0, 1847, __pyx_L3_error)
                }
              } else {
                switch (PyTuple_GET_SIZE(__pyx_args)) {
                  case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
                  CYTHON_FALLTHROUGH;
                  case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
                  break;
                  default: goto __pyx_L5_argtuple_error;
                }
              }
              __pyx_v_obj = values[0];
              __pyx_v_i = values[1];
            }
            goto __pyx_L4_argument_unpacking_done;
            __pyx_L5_argtuple_error:;
            __Pyx_RaiseArgtupleInvalid("atanh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1847, __pyx_L3_error)
            __pyx_L3_error:;
            __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __Pyx_RefNannyFinishContext();
            return NULL;
            __pyx_L4_argument_unpacking_done:;
            __pyx_r = __pyx_pf_8PyClical_78atanh(__pyx_self, __pyx_v_obj, __pyx_v_i);

            /* function exit code */
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          static PyObject *__pyx_pf_8PyClical_78atanh(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            struct __pyx_opt_args_8PyClical_atanh __pyx_t_2;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("atanh", 0);
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_2.__pyx_n = 1;
            __pyx_t_2.i = __pyx_v_i;
            __pyx_t_1 = __pyx_f_8PyClical_atanh(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1847, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1864
 *             return clifford().wrap( glucat.atanh(toClifford(obj)) )
 *
 * cpdef inline random_clifford(index_set ixt, fill = 1.0):              # <<<<<<<<<<<<<<
 *     """
 *     Random multivector within a frame.
 */

          static PyObject *__pyx_pw_8PyClical_81random_clifford(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
          static CYTHON_INLINE PyObject *__pyx_f_8PyClical_random_clifford(struct
__pyx_obj_8PyClical_index_set *__pyx_v_ixt, CYTHON_UNUSED int __pyx_skip_dispatch, struct
```

```
__pyx_opt_args_8PyClical_random_clifford *__pyx_optional_args) {
                    PyObject *__pyx_v_fill = ((PyObject *)__pyx_float_1_0);
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    PyObject *__pyx_t_2 = NULL;
                    scalar_t __pyx_t_3;
                    PyObject *__pyx_t_4 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("random_clifford", 0);
                    if (__pyx_optional_args) {
                      if (__pyx_optional_args->__pyx_n > 0) {
                        __pyx_v_fill = __pyx_optional_args->fill;
                      }
                    }

                    /* "PyClical.pyx":1871
 *      {-3,-1,2}
 *      """
 *      return clifford().wrap( clifford().instance.random(ixt.unwrap(), <scalar_t>fill) ) # «««««««
 *
 * cpdef inline cga3(obj):
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1871, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_t_2 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1871, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_2);
                    __pyx_t_3 = __pyx_PyFloat_AsDouble(__pyx_v_fill); if (unlikely((__pyx_t_3 == ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1871, __pyx_L1_error)
                    __pyx_t_4 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), ((struct __pyx_obj_8PyClical_clifford *)__pyx_t_2)->instance->random(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_ixt), ((scalar_t)__pyx_t_3))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1871, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_4);
                    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
                    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                    __pyx_r = __pyx_t_4;
                    __pyx_t_4 = 0;
                    goto __pyx_L0;

                    /* "PyClical.pyx":1864
 *              return clifford().wrap( glucat.atanh(toClifford(obj)) )
 *
 * cpdef inline random_clifford(index_set ixt, fill = 1.0):             # «««««««
 *      """
 *      Random multivector within a frame.
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
                    __Pyx_XDECREF(__pyx_t_2);
                    __Pyx_XDECREF(__pyx_t_4);
                    __Pyx_AddTraceback("PyClical.random_clifford", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* Python wrapper */
                  static PyObject *__pyx_pw_8PyClical_81random_clifford(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
                  static char __pyx_doc_8PyClical_80random_clifford[] = "\n    Random multivector within a frame.\n\n    >>> print(random_clifford(index_set({-3,-1,2})).frame())\n    {-3,-1,2}\n    ";
                  static PyObject *__pyx_pw_8PyClical_81random_clifford(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
                    struct __pyx_obj_8PyClical_index_set *__pyx_v_ixt = 0;
                    PyObject *__pyx_v_fill = 0;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    PyObject *__pyx_r = 0;
                    __Pyx_RefNannyDeclarations
                    __Pyx_RefNannySetupContext("random_clifford (wrapper)", 0);
                    {
                      static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_ixt,&__pyx_n_s_fill,0};
                      PyObject* values[2] = {0,0};
                      values[1] = ((PyObject *)__pyx_float_1_0);
```

```
    if (unlikely(__pyx_kwds)) {
      Py_ssize_t kw_args;
      const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
      switch (pos_args) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        CYTHON_FALLTHROUGH;
        case  0: break;
        default: goto __pyx_L5_argtuple_error;
      }
      kw_args = PyDict_Size(__pyx_kwds);
      switch (pos_args) {
        case  0:
        if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_ixt)) != 0)) kw_args--;
        else goto __pyx_L5_argtuple_error;
        CYTHON_FALLTHROUGH;
        case  1:
        if (kw_args > 0) {
          PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_fill);
          if (value) { values[1] = value; kw_args--; }
        }
      }
      if (unlikely(kw_args > 0)) {
        if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args, "random_clifford") < 0)) __PYX_ERR(0, 1864, __pyx_L3_error)
      }
    } else {
      switch (PyTuple_GET_SIZE(__pyx_args)) {
        case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
        CYTHON_FALLTHROUGH;
        case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
        break;
        default: goto __pyx_L5_argtuple_error;
      }
    }
    __pyx_v_ixt = ((struct __pyx_obj_8PyClical_index_set *)values[0]);
    __pyx_v_fill = values[1];
  }
  goto __pyx_L4_argument_unpacking_done;
  __pyx_L5_argtuple_error:;
  __Pyx_RaiseArgtupleInvalid("random_clifford", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1864, __pyx_L3_error)
  __pyx_L3_error:;
  __Pyx_AddTraceback("PyClical.random_clifford", __pyx_clineno, __pyx_lineno, __pyx_filename);
  __Pyx_RefNannyFinishContext();
  return NULL;
  __pyx_L4_argument_unpacking_done:;
  if (unlikely(!__Pyx_ArgTypeTest(((PyObject *)__pyx_v_ixt), __pyx_ptype_8PyClical_index_set, 1, "ixt", 0))) __PYX_ERR(0, 1864, __pyx_L1_error)
  __pyx_r = __pyx_pf_8PyClical_80random_clifford(__pyx_self, __pyx_v_ixt, __pyx_v_fill);

  /* function exit code */
  goto __pyx_L0;
  __pyx_L1_error:;
  __pyx_r = NULL;
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  return __pyx_r;
}

static PyObject *__pyx_pf_8PyClical_80random_clifford(CYTHON_UNUSED PyObject *__pyx_self, struct __pyx_obj_8PyClical_index_set *__pyx_v_ixt, PyObject *__pyx_v_fill) {
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  struct __pyx_opt_args_8PyClical_random_clifford __pyx_t_2;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("random_clifford", 0);
  __Pyx_XDECREF(__pyx_r);
  __pyx_t_2.__pyx_n = 1;
  __pyx_t_2.fill = __pyx_v_fill;
  __pyx_t_1 = __pyx_f_8PyClical_random_clifford(__pyx_v_ixt, 0, &__pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1864, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_r = __pyx_t_1;
  __pyx_t_1 = 0;
  goto __pyx_L0;

  /* function exit code */
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
```

```
            __Pyx_AddTraceback("PyClical.random_clifford", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1873
 *     return clifford().wrap( clifford().instance.random(ixt.unwrap(), <scalar_t>fill) )
 *
 * cpdef inline cga3(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby definition.
 */

          static PyObject *__pyx_pw_8PyClical_83cga3(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
          static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cga3(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            PyObject *__pyx_t_2 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("cga3", 0);

            /* "PyClical.pyx":1880
 *     87{-1}+4{1}+18{2}+2{3}+85{4}
 *     """
 *     return clifford().wrap( glucat.cga3(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * cpdef inline cga3std(obj):
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1880, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), cga3::cga3(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1880, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_2);
            __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
            __pyx_r = __pyx_t_2;
            __pyx_t_2 = 0;
            goto __pyx_L0;

            /* "PyClical.pyx":1873
 *     return clifford().wrap( clifford().instance.random(ixt.unwrap(), <scalar_t>fill) )
 *
 * cpdef inline cga3(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby definition.
 */

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_XDECREF(__pyx_t_2);
            __Pyx_AddTraceback("PyClical.cga3", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = 0;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* Python wrapper */
          static PyObject *__pyx_pw_8PyClical_83cga3(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
          static char __pyx_doc_8PyClical_82cga3[] = "\n    Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby definition.\n\n    >>> x=clifford(\"2{1}+9{2}+{3}\"); print(cga3(x))\n    87{-1}+4{1}+18{2}+2{3}+85{4}\n    ";
          static PyObject *__pyx_pw_8PyClical_83cga3(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
            PyObject *__pyx_r = 0;
            __Pyx_RefNannyDeclarations
            __Pyx_RefNannySetupContext("cga3 (wrapper)", 0);
            __pyx_r = __pyx_pf_8PyClical_82cga3(__pyx_self, ((PyObject *)__pyx_v_obj));

            /* function exit code */
            __Pyx_RefNannyFinishContext();
```

```
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_82cga3(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("cga3", 0);
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __pyx_f_8PyClical_cga3(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1873, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("PyClical.cga3", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = NULL;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      /* "PyClical.pyx":1882
 *     return clifford().wrap( glucat.cga3(toClifford(obj)) )
 *
 * cpdef inline cga3std(obj):             # <<<<<<<<
 *     """
 *     Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
 */

      static PyObject *__pyx_pw_8PyClical_85cga3std(PyObject *__pyx_self, PyObject *__pyx_v_obj); /*proto*/
      static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cga3std(PyObject *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("cga3std", 0);

        /* "PyClical.pyx":1891
 *     0
 *     """
 *     return clifford().wrap( glucat.cga3std(toClifford(obj)) )             # <<<<<<<<
 *
 * cpdef inline agc3(obj):
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1891, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1), cga3::cga3std(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1891, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
        __pyx_r = __pyx_t_2;
        __pyx_t_2 = 0;
        goto __pyx_L0;

        /* "PyClical.pyx":1882
 *     return clifford().wrap( glucat.cga3(toClifford(obj)) )
 *
 * cpdef inline cga3std(obj):             # <<<<<<<<
 *     """
 *     Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_XDECREF(__pyx_t_2);
        __Pyx_AddTraceback("PyClical.cga3std", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = 0;
        __pyx_L0:;
```

```
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* Python wrapper */
          static PyObject *__pyx_pw_8PyClical_85cga3std(PyObject *__pyx_self, PyObject
   *__pyx_v_obj); /*proto*/
          static char __pyx_doc_8PyClical_84cga3std[] = "\n    Convert CGA3 null vector to
   standard conformal null vector using Doran and Lasenby definition.\n\n    >>>
   x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x)))\n    87{-1}+4{1}+18{2}+2{3}+85{4}\n    >>>
   x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x))-cga3(x))\n    0\n    ";
          static PyObject *__pyx_pw_8PyClical_85cga3std(PyObject *__pyx_self, PyObject
   *__pyx_v_obj) {
            PyObject *__pyx_r = 0;
            __Pyx_RefNannyDeclarations
            __Pyx_RefNannySetupContext("cga3std (wrapper)", 0);
            __pyx_r = __pyx_pf_8PyClical_84cga3std(__pyx_self, ((PyObject *)__pyx_v_obj));

            /* function exit code */
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          static PyObject *__pyx_pf_8PyClical_84cga3std(CYTHON_UNUSED PyObject *__pyx_self,
   PyObject *__pyx_v_obj) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("cga3std", 0);
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __pyx_f_8PyClical_cga3std(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
   __PYX_ERR(0, 1882, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_AddTraceback("PyClical.cga3std", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1893
 *     return clifford().wrap( glucat.cga3std(toClifford(obj)) )
 *
 * cpdef inline agc3(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
 */

          static PyObject *__pyx_pw_8PyClical_87agc3(PyObject *__pyx_self, PyObject
   *__pyx_v_obj); /*proto*/
          static CYTHON_INLINE PyObject *__pyx_f_8PyClical_agc3(PyObject *__pyx_v_obj,
   CYTHON_UNUSED int __pyx_skip_dispatch) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            PyObject *__pyx_t_2 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("agc3", 0);

            /* "PyClical.pyx":1902
 *     0
 *     """
 *     return clifford().wrap( glucat.agc3(toClifford(obj)) )             # <<<<<<<<<<<<<<
 *
 * # Some abbreviations.
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford));
   if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1902, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
   *)__pyx_t_1), cga3::agc3(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
   __PYX_ERR(0, 1902, __pyx_L1_error)
```

```
            __Pyx_GOTREF(__pyx_t_2);
            __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
            __pyx_r = __pyx_t_2;
            __pyx_t_2 = 0;
            goto __pyx_L0;

            /* "PyClical.pyx":1893
 *     return clifford().wrap( glucat.cga3std(toClifford(obj)) )
 *
 * cpdef inline agc3(obj):              # «««««««
 *     """
 *     Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
 */

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_XDECREF(__pyx_t_2);
            __Pyx_AddTraceback("PyClical.agc3", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = 0;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* Python wrapper */
          static PyObject *__pyx_pw_8PyClical_87agc3(PyObject *__pyx_self, PyObject
      *__pyx_v_obj); /*proto*/
          static char __pyx_doc_8PyClical_86agc3[] = "\n    Convert CGA3 null vector to
      Euclidean 3D vector using Doran and Lasenby definition.\n\n    >>> x=clifford(\"2{1}+9{2}+{3}\");
      print(agc3(cga3(x)))\n    2{1}+9{2}+{3}\n    >>> x=clifford(\"2{1}+9{2}+{3}\");
      print(agc3(cga3(x))-x)\n    0\n    ";
          static PyObject *__pyx_pw_8PyClical_87agc3(PyObject *__pyx_self, PyObject
      *__pyx_v_obj) {
            PyObject *__pyx_r = 0;
            __Pyx_RefNannyDeclarations
            __Pyx_RefNannySetupContext("agc3 (wrapper)", 0);
            __pyx_r = __pyx_pf_8PyClical_86agc3(__pyx_self, ((PyObject *)__pyx_v_obj));

            /* function exit code */
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          static PyObject *__pyx_pf_8PyClical_86agc3(CYTHON_UNUSED PyObject *__pyx_self,
      PyObject *__pyx_v_obj) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("agc3", 0);
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __pyx_f_8PyClical_agc3(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
      __PYX_ERR(0, 1893, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_AddTraceback("PyClical.agc3", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = NULL;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "PyClical.pyx":1936
 * """
 *
 * def e(obj):              # «««««««
 *     """
 *     Abbreviation for clifford(index_set(obj)).
 */

          /* Python wrapper */
          static PyObject *__pyx_pw_8PyClical_89e(PyObject *__pyx_self, PyObject *__pyx_v_obj);
      /*proto*/
          static char __pyx_doc_8PyClical_88e[] = "\n    Abbreviation for
      clifford(index_set(obj)).\n\n    >>> print(e(1))\n    {1}\n    >>> print(e(-1))\n    {-1}\n    >>>
      print(e(0))\n    1\n    ";
```

```
            static PyMethodDef __pyx_mdef_8PyClical_89e = {"e", (PyCFunction)__pyx_pw_8PyClical_89e, METH_O, __pyx_doc_8PyClical_88e};
            static PyObject *__pyx_pw_8PyClical_89e(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
              PyObject *__pyx_r = 0;
              __Pyx_RefNannyDeclarations
              __Pyx_RefNannySetupContext("e (wrapper)", 0);
              __pyx_r = __pyx_pf_8PyClical_88e(__pyx_self, ((PyObject *)__pyx_v_obj));

              /* function exit code */
              __Pyx_RefNannyFinishContext();
              return __pyx_r;
            }

            static PyObject *__pyx_pf_8PyClical_88e(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj) {
              PyObject *__pyx_r = NULL;
              __Pyx_RefNannyDeclarations
              PyObject *__pyx_t_1 = NULL;
              PyObject *__pyx_t_2 = NULL;
              int __pyx_lineno = 0;
              const char *__pyx_filename = NULL;
              int __pyx_clineno = 0;
              __Pyx_RefNannySetupContext("e", 0);

              /* "PyClical.pyx":1947
 *     1
 *     """
 *     return clifford(index_set(obj))             # <<<<<<<<<<<<<<
 *
 * def istpq(p, q):
 */
              __Pyx_XDECREF(__pyx_r);
              __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set), __pyx_v_obj); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1947, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_1);
              __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_t_1); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1947, __pyx_L1_error)
              __Pyx_GOTREF(__pyx_t_2);
              __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
              __pyx_r = __pyx_t_2;
              __pyx_t_2 = 0;
              goto __pyx_L0;

              /* "PyClical.pyx":1936
 * """
 *
 * def e(obj):             # <<<<<<<<<<<<<<
 *     """
 *     Abbreviation for clifford(index_set(obj)).
 */

              /* function exit code */
              __pyx_L1_error:;
              __Pyx_XDECREF(__pyx_t_1);
              __Pyx_XDECREF(__pyx_t_2);
              __Pyx_AddTraceback("PyClical.e", __pyx_clineno, __pyx_lineno, __pyx_filename);
              __pyx_r = NULL;
              __pyx_L0:;
              __Pyx_XGIVEREF(__pyx_r);
              __Pyx_RefNannyFinishContext();
              return __pyx_r;
            }

            /* "PyClical.pyx":1949
 *     return clifford(index_set(obj))
 *
 * def istpq(p, q):             # <<<<<<<<<<<<<<
 *     """
 *     Abbreviation for index_set({-q,...p}).
 */

            /* Python wrapper */
            static PyObject *__pyx_pw_8PyClical_91istpq(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds); /*proto*/
            static char __pyx_doc_8PyClical_90istpq[] = "\n    Abbreviation for index_set({-q,...p}).\n\n    >>> print(istpq(2,3))\n    {-3,-2,-1,1,2}\n    ";
            static PyMethodDef __pyx_mdef_8PyClical_91istpq = {"istpq", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_91istpq, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_90istpq};
            static PyObject *__pyx_pw_8PyClical_91istpq(PyObject *__pyx_self, PyObject *__pyx_args, PyObject *__pyx_kwds) {
              PyObject *__pyx_v_p = 0;
              PyObject *__pyx_v_q = 0;
              int __pyx_lineno = 0;
              const char *__pyx_filename = NULL;
              int __pyx_clineno = 0;
              PyObject *__pyx_r = 0;
```

```
        __Pyx_RefNannyDeclarations
        __Pyx_RefNannySetupContext("istpq (wrapper)", 0);
        {
          static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_p,&__pyx_n_s_q,0};
          PyObject* values[2] = {0,0};
          if (unlikely(__pyx_kwds)) {
            Py_ssize_t kw_args;
            const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
            switch (pos_args) {
              case  2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
              CYTHON_FALLTHROUGH;
              case  1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
              CYTHON_FALLTHROUGH;
              case  0: break;
              default: goto __pyx_L5_argtuple_error;
            }
            kw_args = PyDict_Size(__pyx_kwds);
            switch (pos_args) {
              case  0:
              if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_p)) !=
0)) kw_args--;
              else goto __pyx_L5_argtuple_error;
              CYTHON_FALLTHROUGH;
              case  1:
              if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_q)) !=
0)) kw_args--;
              else {
                __Pyx_RaiseArgtupleInvalid("istpq", 1, 2, 2, 1); __PYX_ERR(0, 1949,
__pyx_L3_error)
              }
            }
            if (unlikely(kw_args > 0)) {
              if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "istpq") < 0)) __PYX_ERR(0, 1949, __pyx_L3_error)
            }
          } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
            goto __pyx_L5_argtuple_error;
          } else {
            values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
            values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
          }
          __pyx_v_p = values[0];
          __pyx_v_q = values[1];
        }
        goto __pyx_L4_argument_unpacking_done;
        __pyx_L5_argtuple_error:;
        __Pyx_RaiseArgtupleInvalid("istpq", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1949, __pyx_L3_error)
        __pyx_L3_error:;
        __Pyx_AddTraceback("PyClical.istpq", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __Pyx_RefNannyFinishContext();
        return NULL;
        __pyx_L4_argument_unpacking_done:;
        __pyx_r = __pyx_pf_8PyClical_90istpq(__pyx_self, __pyx_v_p, __pyx_v_q);

        /* function exit code */
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
      }

      static PyObject *__pyx_pf_8PyClical_90istpq(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_p, PyObject *__pyx_v_q) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        PyObject *__pyx_t_2 = NULL;
        PyObject *__pyx_t_3 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("istpq", 0);

        /* "PyClical.pyx":1956
 *     {-3,-2,-1,1,2}
 *     """
 *     return index_set(set(range(-q,p+1)))             # <<<<<<<<<<<<<<
 *
 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = PyNumber_Negative(__pyx_v_q); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1956, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_t_2 = __Pyx_PyInt_AddObjC(__pyx_v_p, __pyx_int_1, 1, 0, 0); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_2);
        __pyx_t_3 = PyTuple_New(2); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1956,
```

```
__pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_GIVEREF(__pyx_t_1);
                  PyTuple_SET_ITEM(__pyx_t_3, 0, __pyx_t_1);
                  __Pyx_GIVEREF(__pyx_t_2);
                  PyTuple_SET_ITEM(__pyx_t_3, 1, __pyx_t_2);
                  __pyx_t_1 = 0;
                  __pyx_t_2 = 0;
                  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_builtin_range, __pyx_t_3, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_t_3 = PySet_New(__pyx_t_2); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1956, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_3);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set), __pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
                  __pyx_r = __pyx_t_2;
                  __pyx_t_2 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1949
 *     return clifford(index_set(obj))
 *
 * def istpq(p, q):             # «««««««
 *     """
 *     Abbreviation for index_set({-q,...p}).
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical.istpq", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "PyClical.pyx":1962
 *
 * # Doctest interface.
 * def _test():             # «««««««
 *     import PyClical, doctest
 *     return doctest.testmod(PyClical)
 */

                /* Python wrapper */
                static PyObject *__pyx_pw_8PyClical_93_test(PyObject *__pyx_self, CYTHON_UNUSED PyObject *unused); /*proto*/
                static PyMethodDef __pyx_mdef_8PyClical_93_test = {"_test", (PyCFunction)__pyx_pw_8PyClical_93_test, METH_NOARGS, 0};
                static PyObject *__pyx_pw_8PyClical_93_test(PyObject *__pyx_self, CYTHON_UNUSED PyObject *unused) {
                  PyObject *__pyx_r = 0;
                  __Pyx_RefNannyDeclarations
                  __Pyx_RefNannySetupContext("_test (wrapper)", 0);
                  __pyx_r = __pyx_pf_8PyClical_92_test(__pyx_self);

                  /* function exit code */
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                static PyObject *__pyx_pf_8PyClical_92_test(CYTHON_UNUSED PyObject *__pyx_self) {
                  PyObject *__pyx_v_PyClical = NULL;
                  PyObject *__pyx_v_doctest = NULL;
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  PyObject *__pyx_t_2 = NULL;
                  PyObject *__pyx_t_3 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("_test", 0);

                  /* "PyClical.pyx":1963
 * # Doctest interface.
 * def _test():
 *     import PyClical, doctest             # «««««««
```

```
 *     return doctest.testmod(PyClical)
 *
 */
                  __pyx_t_1 = __Pyx_Import(__pyx_n_s_PyClical, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1963, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_v_PyClical = __pyx_t_1;
                  __pyx_t_1 = 0;
                  __pyx_t_1 = __Pyx_Import(__pyx_n_s_doctest, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1963, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __pyx_v_doctest = __pyx_t_1;
                  __pyx_t_1 = 0;

                  /* "PyClical.pyx":1964
 * def _test():
 *     import PyClical, doctest
 *     return doctest.testmod(PyClical)             # <<<<<<<<<<<<<<
 *
 * if __name__ == "__main__":
 */
                  __Pyx_XDECREF(__pyx_r);
                  __pyx_t_2 = __Pyx_PyObject_GetAttrStr(__pyx_v_doctest, __pyx_n_s_testmod); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1964, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_2);
                  __pyx_t_3 = NULL;
                  if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_2))) {
                    __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_2);
                    if (likely(__pyx_t_3)) {
                      PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_2);
                      __Pyx_INCREF(__pyx_t_3);
                      __Pyx_INCREF(function);
                      __Pyx_DECREF_SET(__pyx_t_2, function);
                    }
                  }
                  __pyx_t_1 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_2, __pyx_t_3, __pyx_v_PyClical) : __Pyx_PyObject_CallOneArg(__pyx_t_2, __pyx_v_PyClical);
                  __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
                  if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1964, __pyx_L1_error)
                  __Pyx_GOTREF(__pyx_t_1);
                  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
                  __pyx_r = __pyx_t_1;
                  __pyx_t_1 = 0;
                  goto __pyx_L0;

                  /* "PyClical.pyx":1962
 *
 * # Doctest interface.
 * def _test():             # <<<<<<<<<<<<<<
 *     import PyClical, doctest
 *     return doctest.testmod(PyClical)
 */

                  /* function exit code */
                  __pyx_L1_error:;
                  __Pyx_XDECREF(__pyx_t_1);
                  __Pyx_XDECREF(__pyx_t_2);
                  __Pyx_XDECREF(__pyx_t_3);
                  __Pyx_AddTraceback("PyClical._test", __pyx_clineno, __pyx_lineno, __pyx_filename);
                  __pyx_r = NULL;
                  __pyx_L0:;
                  __Pyx_XDECREF(__pyx_v_PyClical);
                  __Pyx_XDECREF(__pyx_v_doctest);
                  __Pyx_XGIVEREF(__pyx_r);
                  __Pyx_RefNannyFinishContext();
                  return __pyx_r;
                }

                /* "string.to_py":31
 *
 * @cname("__pyx_convert_PyObject_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyObject_string_to_py_std__in_string(const string& s): # <<<<<<<<<<<<<<
 *     return __Pyx_PyObject_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

                static CYTHON_INLINE PyObject *__pyx_convert_PyObject_string_to_py_std__in_string(std::string const &__pyx_v_s) {
                  PyObject *__pyx_r = NULL;
                  __Pyx_RefNannyDeclarations
                  PyObject *__pyx_t_1 = NULL;
                  int __pyx_lineno = 0;
                  const char *__pyx_filename = NULL;
                  int __pyx_clineno = 0;
                  __Pyx_RefNannySetupContext("__pyx_convert_PyObject_string_to_py_std__in_string", 0);

```

```
                    /* "string.to_py":32
 * @cname("__pyx_convert_PyObject_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyObject_string_to_py_std__in_string(const string& s):
 *     return __Pyx_PyObject_FromStringAndSize(s.data(), s.size())             # <<<<<<<<<<<<<<
 * cdef extern from *:
 *     cdef object __Pyx_PyUnicode_FromStringAndSize(const char*, size_t)
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __Pyx_PyObject_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 32, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "string.to_py":31
 *
 * @cname("__pyx_convert_PyObject_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyObject_string_to_py_std__in_string(const string& s):             # <<<<<<<<<<<<<<
 *     return __Pyx_PyObject_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
__Pyx_AddTraceback("string.to_py.__pyx_convert_PyObject_string_to_py_std__in_string", __pyx_clineno, __pyx_lineno, __pyx_filename);
                    __pyx_r = 0;
                    __pyx_L0:;
                    __Pyx_XGIVEREF(__pyx_r);
                    __Pyx_RefNannyFinishContext();
                    return __pyx_r;
                  }

                  /* "string.to_py":37
 *
 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):             # <<<<<<<<<<<<<<
 *     return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

                  static CYTHON_INLINE PyObject *__pyx_convert_PyUnicode_string_to_py_std__in_string(std::string const &__pyx_v_s) {
                    PyObject *__pyx_r = NULL;
                    __Pyx_RefNannyDeclarations
                    PyObject *__pyx_t_1 = NULL;
                    int __pyx_lineno = 0;
                    const char *__pyx_filename = NULL;
                    int __pyx_clineno = 0;
                    __Pyx_RefNannySetupContext("__pyx_convert_PyUnicode_string_to_py_std__in_string", 0);

                    /* "string.to_py":38
 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):
 *     return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size())             # <<<<<<<<<<<<<<
 * cdef extern from *:
 *     cdef object __Pyx_PyStr_FromStringAndSize(const char*, size_t)
 */
                    __Pyx_XDECREF(__pyx_r);
                    __pyx_t_1 = __Pyx_PyUnicode_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 38, __pyx_L1_error)
                    __Pyx_GOTREF(__pyx_t_1);
                    __pyx_r = __pyx_t_1;
                    __pyx_t_1 = 0;
                    goto __pyx_L0;

                    /* "string.to_py":37
 *
 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):             # <<<<<<<<<<<<<<
 *     return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

                    /* function exit code */
                    __pyx_L1_error:;
                    __Pyx_XDECREF(__pyx_t_1);
__Pyx_AddTraceback("string.to_py.__pyx_convert_PyUnicode_string_to_py_std__in_string", __pyx_clineno, __pyx_lineno, __pyx_filename);
```

```
            __pyx_r = 0;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "string.to_py":43
 *
 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s):             # «««««««
 *     return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

          static CYTHON_INLINE PyObject *__pyx_convert_PyStr_string_to_py_std__in_string(std::string const &__pyx_v_s) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("__pyx_convert_PyStr_string_to_py_std__in_string", 0);

            /* "string.to_py":44
 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s):
 *     return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())             # «««««««
 * cdef extern from *:
 *     cdef object __Pyx_PyBytes_FromStringAndSize(const char*, size_t)
 */
            __Pyx_XDECREF(__pyx_r);
            __pyx_t_1 = __Pyx_PyStr_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 44, __pyx_L1_error)
            __Pyx_GOTREF(__pyx_t_1);
            __pyx_r = __pyx_t_1;
            __pyx_t_1 = 0;
            goto __pyx_L0;

            /* "string.to_py":43
 *
 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s):             # «««««««
 *     return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

            /* function exit code */
            __pyx_L1_error:;
            __Pyx_XDECREF(__pyx_t_1);
            __Pyx_AddTraceback("string.to_py.__pyx_convert_PyStr_string_to_py_std__in_string", __pyx_clineno, __pyx_lineno, __pyx_filename);
            __pyx_r = 0;
            __pyx_L0:;
            __Pyx_XGIVEREF(__pyx_r);
            __Pyx_RefNannyFinishContext();
            return __pyx_r;
          }

          /* "string.to_py":49
 *
 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s):             # «««««««
 *     return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

          static CYTHON_INLINE PyObject *__pyx_convert_PyBytes_string_to_py_std__in_string(std::string const &__pyx_v_s) {
            PyObject *__pyx_r = NULL;
            __Pyx_RefNannyDeclarations
            PyObject *__pyx_t_1 = NULL;
            int __pyx_lineno = 0;
            const char *__pyx_filename = NULL;
            int __pyx_clineno = 0;
            __Pyx_RefNannySetupContext("__pyx_convert_PyBytes_string_to_py_std__in_string", 0);

            /* "string.to_py":50
 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s):
 *     return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())             # «««««««
 * cdef extern from *:
 *     cdef object __Pyx_PyByteArray_FromStringAndSize(const char*, size_t)
```

```
*/
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyBytes_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 50, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "string.to_py":49
 *
 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s): # «««««««
 *     return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())
 * cdef extern from *:
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
        __Pyx_AddTraceback("string.to_py.__pyx_convert_PyBytes_string_to_py_std__in_string", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = 0;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
       }

       /* "string.to_py":55
 *
 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s): # «««««««
 *     return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
 *
 */

       static CYTHON_INLINE PyObject *__pyx_convert_PyByteArray_string_to_py_std__in_string(std::string const &__pyx_v_s) {
        PyObject *__pyx_r = NULL;
        __Pyx_RefNannyDeclarations
        PyObject *__pyx_t_1 = NULL;
        int __pyx_lineno = 0;
        const char *__pyx_filename = NULL;
        int __pyx_clineno = 0;
        __Pyx_RefNannySetupContext("__pyx_convert_PyByteArray_string_to_py_std__in_string", 0);

        /* "string.to_py":56
 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s):
 *     return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())             # «««««««
 *
 */
        __Pyx_XDECREF(__pyx_r);
        __pyx_t_1 = __Pyx_PyByteArray_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 56, __pyx_L1_error)
        __Pyx_GOTREF(__pyx_t_1);
        __pyx_r = __pyx_t_1;
        __pyx_t_1 = 0;
        goto __pyx_L0;

        /* "string.to_py":55
 *
 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s): # «««««««
 *     return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
 *
 */

        /* function exit code */
        __pyx_L1_error:;
        __Pyx_XDECREF(__pyx_t_1);
__Pyx_AddTraceback("string.to_py.__pyx_convert_PyByteArray_string_to_py_std__in_string", __pyx_clineno, __pyx_lineno, __pyx_filename);
        __pyx_r = 0;
        __pyx_L0:;
        __Pyx_XGIVEREF(__pyx_r);
        __Pyx_RefNannyFinishContext();
        return __pyx_r;
       }
       static struct __pyx_vtabstruct_8PyClical_index_set __pyx_vtable_8PyClical_index_set;
```

```
static PyObject *__pyx_tp_new_8PyClical_index_set(PyTypeObject *t, PyObject *a, PyObject *k) {
  struct __pyx_obj_8PyClical_index_set *p;
  PyObject *o;
  if (likely((t->tp_flags & Py_TPFLAGS_IS_ABSTRACT) == 0)) {
    o = (*t->tp_alloc)(t, 0);
  } else {
    o = (PyObject *) PyBaseObject_Type.tp_new(t, __pyx_empty_tuple, 0);
  }
  if (unlikely(!o)) return 0;
  p = ((struct __pyx_obj_8PyClical_index_set *)o);
  p->__pyx_vtab = __pyx_vtabptr_8PyClical_index_set;
  if (unlikely(__pyx_pw_8PyClical_9index_set_3__cinit__(o, a, k) < 0)) goto bad;
  return o;
  bad:
  Py_DECREF(o); o = 0;
  return NULL;
}

static void __pyx_tp_dealloc_8PyClical_index_set(PyObject *o) {
  #if CYTHON_USE_TP_FINALIZE
  if (unlikely(PyType_HasFeature(Py_TYPE(o), Py_TPFLAGS_HAVE_FINALIZE) && Py_TYPE(o)->tp_finalize) && (!PyType_IS_GC(Py_TYPE(o)) || !_PyGC_FINALIZED(o))) {
    if (PyObject_CallFinalizerFromDealloc(o)) return;
  }
  #endif
  {
    PyObject *etype, *eval, *etb;
    PyErr_Fetch(&etype, &eval, &etb);
    __Pyx_SET_REFCNT(o, Py_REFCNT(o) + 1);
    __pyx_pw_8PyClical_9index_set_5__dealloc__(o);
    __Pyx_SET_REFCNT(o, Py_REFCNT(o) - 1);
    PyErr_Restore(etype, eval, etb);
  }
  (*Py_TYPE(o)->tp_free)(o);
}
static PyObject *__pyx_sq_item_8PyClical_index_set(PyObject *o, Py_ssize_t i) {
  PyObject *r;
  PyObject *x = PyInt_FromSsize_t(i); if(!x) return 0;
  r = Py_TYPE(o)->tp_as_mapping->mp_subscript(o, x);
  Py_DECREF(x);
  return r;
}

static int __pyx_mp_ass_subscript_8PyClical_index_set(PyObject *o, PyObject *i, PyObject *v) {
  if (v) {
    return __pyx_pw_8PyClical_9index_set_9__setitem__(o, i, v);
  }
  else {
    PyErr_Format(PyExc_NotImplementedError,
      "Subscript deletion not supported by %.200s", Py_TYPE(o)->tp_name);
    return -1;
  }
}

static PyMethodDef __pyx_methods_8PyClical_index_set[] = {
  {"copy", (PyCFunction)__pyx_pw_8PyClical_9index_set_1copy, METH_NOARGS, __pyx_doc_8PyClical_9index_set_copy},
  {"count", (PyCFunction)__pyx_pw_8PyClical_9index_set_32count, METH_NOARGS, __pyx_doc_8PyClical_9index_set_31count},
  {"count_neg", (PyCFunction)__pyx_pw_8PyClical_9index_set_34count_neg, METH_NOARGS, __pyx_doc_8PyClical_9index_set_33count_neg},
  {"count_pos", (PyCFunction)__pyx_pw_8PyClical_9index_set_36count_pos, METH_NOARGS, __pyx_doc_8PyClical_9index_set_35count_pos},
  {"min", (PyCFunction)__pyx_pw_8PyClical_9index_set_38min, METH_NOARGS, __pyx_doc_8PyClical_9index_set_37min},
  {"max", (PyCFunction)__pyx_pw_8PyClical_9index_set_40max, METH_NOARGS, __pyx_doc_8PyClical_9index_set_39max},
  {"hash_fn", (PyCFunction)__pyx_pw_8PyClical_9index_set_42hash_fn, METH_NOARGS, __pyx_doc_8PyClical_9index_set_41hash_fn},
  {"sign_of_mult", (PyCFunction)__pyx_pw_8PyClical_9index_set_44sign_of_mult, METH_O, __pyx_doc_8PyClical_9index_set_43sign_of_mult},
  {"sign_of_square", (PyCFunction)__pyx_pw_8PyClical_9index_set_46sign_of_square, METH_NOARGS, __pyx_doc_8PyClical_9index_set_45sign_of_square},
  {"__reduce_cython__", (PyCFunction)__pyx_pw_8PyClical_9index_set_52__reduce_cython__, METH_NOARGS, 0},
  {"__setstate_cython__", (PyCFunction)__pyx_pw_8PyClical_9index_set_54__setstate_cython__, METH_O, 0},
  {0, 0, 0, 0}
};

static PyNumberMethods __pyx_tp_as_number_index_set = {
  0, /*nb_add*/
  0, /*nb_subtract*/
  0, /*nb_multiply*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
```

```
0x03050000)
  0, /*nb_divide*/
  #endif
  0, /*nb_remainder*/
  0, /*nb_divmod*/
  0, /*nb_power*/
  0, /*nb_negative*/
  0, /*nb_positive*/
  0, /*nb_absolute*/
  0, /*nb_nonzero*/
  __pyx_pw_8PyClical_9index_set_18__invert__, /*nb_invert*/
  0, /*nb_lshift*/
  0, /*nb_rshift*/
  __pyx_pw_8PyClical_9index_set_24__and__, /*nb_and*/
  __pyx_pw_8PyClical_9index_set_20__xor__, /*nb_xor*/
  __pyx_pw_8PyClical_9index_set_28__or__, /*nb_or*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_coerce*/
  #endif
  0, /*nb_int*/
  #if PY_MAJOR_VERSION < 3
  0, /*nb_long*/
  #else
  0, /*reserved*/
  #endif
  0, /*nb_float*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_oct*/
  #endif
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_hex*/
  #endif
  0, /*nb_inplace_add*/
  0, /*nb_inplace_subtract*/
  0, /*nb_inplace_multiply*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_inplace_divide*/
  #endif
  0, /*nb_inplace_remainder*/
  0, /*nb_inplace_power*/
  0, /*nb_inplace_lshift*/
  0, /*nb_inplace_rshift*/
  __pyx_pw_8PyClical_9index_set_26__iand__, /*nb_inplace_and*/
  __pyx_pw_8PyClical_9index_set_22__ixor__, /*nb_inplace_xor*/
  __pyx_pw_8PyClical_9index_set_30__ior__, /*nb_inplace_or*/
  0, /*nb_floor_divide*/
  0, /*nb_true_divide*/
  0, /*nb_inplace_floor_divide*/
  0, /*nb_inplace_true_divide*/
  0, /*nb_index*/
  #if PY_VERSION_HEX >= 0x03050000
  0, /*nb_matrix_multiply*/
  #endif
  #if PY_VERSION_HEX >= 0x03050000
  0, /*nb_inplace_matrix_multiply*/
  #endif
};

static PySequenceMethods __pyx_tp_as_sequence_index_set = {
  0, /*sq_length*/
  0, /*sq_concat*/
  0, /*sq_repeat*/
  __pyx_sq_item_8PyClical_index_set, /*sq_item*/
  0, /*sq_slice*/
  0, /*sq_ass_item*/
  0, /*sq_ass_slice*/
  __pyx_pw_8PyClical_9index_set_13__contains__, /*sq_contains*/
  0, /*sq_inplace_concat*/
  0, /*sq_inplace_repeat*/
};

static PyMappingMethods __pyx_tp_as_mapping_index_set = {
  0, /*mp_length*/
  __pyx_pw_8PyClical_9index_set_11__getitem__, /*mp_subscript*/
  __pyx_mp_ass_subscript_8PyClical_index_set, /*mp_ass_subscript*/
};

static PyTypeObject __pyx_type_8PyClical_index_set = {
  PyVarObject_HEAD_INIT(0, 0)
  "PyClical.index_set", /*tp_name*/
  sizeof(struct __pyx_obj_8PyClical_index_set), /*tp_basicsize*/
  0, /*tp_itemsize*/
  __pyx_tp_dealloc_8PyClical_index_set, /*tp_dealloc*/
```

```
    #if PY_VERSION_HEX < 0x030800b4
    0, /*tp_print*/
    #endif
    #if PY_VERSION_HEX >= 0x030800b4
    0, /*tp_vectorcall_offset*/
    #endif
    0, /*tp_getattr*/
    0, /*tp_setattr*/
    #if PY_MAJOR_VERSION < 3
    0, /*tp_compare*/
    #endif
    #if PY_MAJOR_VERSION >= 3
    0, /*tp_as_async*/
    #endif
    __pyx_pw_8PyClical_9index_set_48__repr__, /*tp_repr*/
    &__pyx_tp_as_number_index_set, /*tp_as_number*/
    &__pyx_tp_as_sequence_index_set, /*tp_as_sequence*/
    &__pyx_tp_as_mapping_index_set, /*tp_as_mapping*/
    0, /*tp_hash*/
    0, /*tp_call*/
    __pyx_pw_8PyClical_9index_set_50__str__, /*tp_str*/
    0, /*tp_getattro*/
    0, /*tp_setattro*/
    0, /*tp_as_buffer*/
Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_BASETYPE,
/*tp_flags*/
    "\n    Python class index_set wraps C++ class IndexSet.\n    ", /*tp_doc*/
    0, /*tp_traverse*/
    0, /*tp_clear*/
    __pyx_pw_8PyClical_9index_set_7__richcmp__, /*tp_richcompare*/
    0, /*tp_weaklistoffset*/
    __pyx_pw_8PyClical_9index_set_15__iter__, /*tp_iter*/
    0, /*tp_iternext*/
    __pyx_methods_8PyClical_index_set, /*tp_methods*/
    0, /*tp_members*/
    0, /*tp_getset*/
    0, /*tp_base*/
    0, /*tp_dict*/
    0, /*tp_descr_get*/
    0, /*tp_descr_set*/
    0, /*tp_dictoffset*/
    0, /*tp_init*/
    0, /*tp_alloc*/
    __pyx_tp_new_8PyClical_index_set, /*tp_new*/
    0, /*tp_free*/
    0, /*tp_is_gc*/
    0, /*tp_bases*/
    0, /*tp_mro*/
    0, /*tp_cache*/
    0, /*tp_subclasses*/
    0, /*tp_weaklist*/
    0, /*tp_del*/
    0, /*tp_version_tag*/
    #if PY_VERSION_HEX >= 0x030400a1
    0, /*tp_finalize*/
    #endif
    #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM
>= 0x07030800)
    0, /*tp_vectorcall*/
    #endif
    #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
    0, /*tp_print*/
    #endif
    #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
    0, /*tp_pypy_flags*/
    #endif
  };
  static struct __pyx_vtabstruct_8PyClical_clifford __pyx_vtable_8PyClical_clifford;

  static PyObject *__pyx_tp_new_8PyClical_clifford(PyTypeObject *t, PyObject *a,
PyObject *k) {
    struct __pyx_obj_8PyClical_clifford *p;
    PyObject *o;
    if (likely((t->tp_flags & Py_TPFLAGS_IS_ABSTRACT) == 0)) {
      o = (*t->tp_alloc)(t, 0);
    } else {
      o = (PyObject *) PyBaseObject_Type.tp_new(t, __pyx_empty_tuple, 0);
    }
    if (unlikely(!o)) return 0;
    p = ((struct __pyx_obj_8PyClical_clifford *)o);
    p->__pyx_vtab = __pyx_vtabptr_8PyClical_clifford;
    if (unlikely(__pyx_pw_8PyClical_8clifford_3__cinit__(o, a, k) < 0)) goto bad;
    return o;
    bad:
    Py_DECREF(o); o = 0;
    return NULL;
```

```
}

static void __pyx_tp_dealloc_8PyClical_clifford(PyObject *o) {
  #if CYTHON_USE_TP_FINALIZE
  if (unlikely(PyType_HasFeature(Py_TYPE(o), Py_TPFLAGS_HAVE_FINALIZE) && Py_TYPE(o)->tp_finalize) && (!PyType_IS_GC(Py_TYPE(o)) || !_PyGC_FINALIZED(o))) {
    if (PyObject_CallFinalizerFromDealloc(o)) return;
  }
  #endif
  {
    PyObject *etype, *eval, *etb;
    PyErr_Fetch(&etype, &eval, &etb);
    __Pyx_SET_REFCNT(o, Py_REFCNT(o) + 1);
    __pyx_pw_8PyClical_8clifford_5__dealloc__(o);
    __Pyx_SET_REFCNT(o, Py_REFCNT(o) - 1);
    PyErr_Restore(etype, eval, etb);
  }
  (*Py_TYPE(o)->tp_free)(o);
}
static PyObject *__pyx_sq_item_8PyClical_clifford(PyObject *o, Py_ssize_t i) {
  PyObject *r;
  PyObject *x = PyInt_FromSsize_t(i); if(!x) return 0;
  r = Py_TYPE(o)->tp_as_mapping->mp_subscript(o, x);
  Py_DECREF(x);
  return r;
}

static PyMethodDef __pyx_methods_8PyClical_clifford[] = {
  {"copy", (PyCFunction)__pyx_pw_8PyClical_8clifford_1copy, METH_NOARGS, __pyx_doc_8PyClical_8clifford_copy},
  {"reframe", (PyCFunction)__pyx_pw_8PyClical_8clifford_11reframe, METH_O, __pyx_doc_8PyClical_8clifford_10reframe},
  {"inv", (PyCFunction)__pyx_pw_8PyClical_8clifford_49inv, METH_NOARGS, __pyx_doc_8PyClical_8clifford_48inv},
  {"pow", (PyCFunction)__pyx_pw_8PyClical_8clifford_57pow, METH_O, __pyx_doc_8PyClical_8clifford_56pow},
  {"outer_pow", (PyCFunction)__pyx_pw_8PyClical_8clifford_59outer_pow, METH_O, __pyx_doc_8PyClical_8clifford_58outer_pow},
  {"scalar", (PyCFunction)__pyx_pw_8PyClical_8clifford_63scalar, METH_NOARGS, __pyx_doc_8PyClical_8clifford_62scalar},
  {"pure", (PyCFunction)__pyx_pw_8PyClical_8clifford_65pure, METH_NOARGS, __pyx_doc_8PyClical_8clifford_64pure},
  {"even", (PyCFunction)__pyx_pw_8PyClical_8clifford_67even, METH_NOARGS, __pyx_doc_8PyClical_8clifford_66even},
  {"odd", (PyCFunction)__pyx_pw_8PyClical_8clifford_69odd, METH_NOARGS, __pyx_doc_8PyClical_8clifford_68odd},
  {"vector_part", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_8clifford_71vector_part, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_8clifford_70vector_part},
  {"involute", (PyCFunction)__pyx_pw_8PyClical_8clifford_73involute, METH_NOARGS, __pyx_doc_8PyClical_8clifford_72involute},
  {"reverse", (PyCFunction)__pyx_pw_8PyClical_8clifford_75reverse, METH_NOARGS, __pyx_doc_8PyClical_8clifford_74reverse},
  {"conj", (PyCFunction)__pyx_pw_8PyClical_8clifford_77conj, METH_NOARGS, __pyx_doc_8PyClical_8clifford_76conj},
  {"quad", (PyCFunction)__pyx_pw_8PyClical_8clifford_79quad, METH_NOARGS, __pyx_doc_8PyClical_8clifford_78quad},
  {"norm", (PyCFunction)__pyx_pw_8PyClical_8clifford_81norm, METH_NOARGS, __pyx_doc_8PyClical_8clifford_80norm},
  {"abs", (PyCFunction)__pyx_pw_8PyClical_8clifford_83abs, METH_NOARGS, __pyx_doc_8PyClical_8clifford_82abs},
  {"max_abs", (PyCFunction)__pyx_pw_8PyClical_8clifford_85max_abs, METH_NOARGS, __pyx_doc_8PyClical_8clifford_84max_abs},
  {"truncated", (PyCFunction)__pyx_pw_8PyClical_8clifford_87truncated, METH_O, __pyx_doc_8PyClical_8clifford_86truncated},
  {"isinf", (PyCFunction)__pyx_pw_8PyClical_8clifford_89isinf, METH_NOARGS, __pyx_doc_8PyClical_8clifford_88isinf},
  {"isnan", (PyCFunction)__pyx_pw_8PyClical_8clifford_91isnan, METH_NOARGS, __pyx_doc_8PyClical_8clifford_90isnan},
  {"frame", (PyCFunction)__pyx_pw_8PyClical_8clifford_93frame, METH_NOARGS, __pyx_doc_8PyClical_8clifford_92frame},
  {"__reduce_cython__", (PyCFunction)__pyx_pw_8PyClical_8clifford_99__reduce_cython__, METH_NOARGS, 0},
  {"__setstate_cython__", (PyCFunction)__pyx_pw_8PyClical_8clifford_101__setstate_cython__, METH_O, 0},
  {0, 0, 0, 0}
};

static PyNumberMethods __pyx_tp_as_number_clifford = {
  __pyx_pw_8PyClical_8clifford_21__add__, /*nb_add*/
  __pyx_pw_8PyClical_8clifford_25__sub__, /*nb_subtract*/
  __pyx_pw_8PyClical_8clifford_29__mul__, /*nb_multiply*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_divide*/
  #endif
  __pyx_pw_8PyClical_8clifford_33__mod__, /*nb_remainder*/
```

```
  0, /*nb_divmod*/
  __pyx_pw_8PyClical_8clifford_55__pow__, /*nb_power*/
  __pyx_pw_8PyClical_8clifford_17__neg__, /*nb_negative*/
  __pyx_pw_8PyClical_8clifford_19__pos__, /*nb_positive*/
  0, /*nb_absolute*/
  0, /*nb_nonzero*/
  0, /*nb_invert*/
  0, /*nb_lshift*/
  0, /*nb_rshift*/
  __pyx_pw_8PyClical_8clifford_37__and__, /*nb_and*/
  __pyx_pw_8PyClical_8clifford_41__xor__, /*nb_xor*/
  __pyx_pw_8PyClical_8clifford_51__or__, /*nb_or*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_coerce*/
  #endif
  0, /*nb_int*/
  #if PY_MAJOR_VERSION < 3
  0, /*nb_long*/
  #else
  0, /*reserved*/
  #endif
  0, /*nb_float*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_oct*/
  #endif
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  0, /*nb_hex*/
  #endif
  __pyx_pw_8PyClical_8clifford_23__iadd__, /*nb_inplace_add*/
  __pyx_pw_8PyClical_8clifford_27__isub__, /*nb_inplace_subtract*/
  __pyx_pw_8PyClical_8clifford_31__imul__, /*nb_inplace_multiply*/
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  __pyx_pw_8PyClical_8clifford_47__idiv__, /*nb_inplace_divide*/
  #endif
  __pyx_pw_8PyClical_8clifford_35__imod__, /*nb_inplace_remainder*/
  0, /*nb_inplace_power*/
  0, /*nb_inplace_lshift*/
  0, /*nb_inplace_rshift*/
  __pyx_pw_8PyClical_8clifford_39__iand__, /*nb_inplace_and*/
  __pyx_pw_8PyClical_8clifford_43__ixor__, /*nb_inplace_xor*/
  __pyx_pw_8PyClical_8clifford_53__ior__, /*nb_inplace_or*/
  0, /*nb_floor_divide*/
  __pyx_pw_8PyClical_8clifford_45__truediv__, /*nb_true_divide*/
  0, /*nb_inplace_floor_divide*/
  0, /*nb_inplace_true_divide*/
  0, /*nb_index*/
  #if PY_VERSION_HEX >= 0x03050000
  0, /*nb_matrix_multiply*/
  #endif
  #if PY_VERSION_HEX >= 0x03050000
  0, /*nb_inplace_matrix_multiply*/
  #endif
};

static PySequenceMethods __pyx_tp_as_sequence_clifford = {
  0, /*sq_length*/
  0, /*sq_concat*/
  0, /*sq_repeat*/
  __pyx_sq_item_8PyClical_clifford, /*sq_item*/
  0, /*sq_slice*/
  0, /*sq_ass_item*/
  0, /*sq_ass_slice*/
  __pyx_pw_8PyClical_8clifford_7__contains__, /*sq_contains*/
  0, /*sq_inplace_concat*/
  0, /*sq_inplace_repeat*/
};

static PyMappingMethods __pyx_tp_as_mapping_clifford = {
  0, /*mp_length*/
  __pyx_pw_8PyClical_8clifford_15__getitem__, /*mp_subscript*/
  0, /*mp_ass_subscript*/
};

static PyTypeObject __pyx_type_8PyClical_clifford = {
  PyVarObject_HEAD_INIT(0, 0)
  "PyClical.clifford", /*tp_name*/
  sizeof(struct __pyx_obj_8PyClical_clifford), /*tp_basicsize*/
  0, /*tp_itemsize*/
  __pyx_tp_dealloc_8PyClical_clifford, /*tp_dealloc*/
  #if PY_VERSION_HEX < 0x030800b4
  0, /*tp_print*/
  #endif
  #if PY_VERSION_HEX >= 0x030800b4
```

```
  0, /*tp_vectorcall_offset*/
  #endif
  0, /*tp_getattr*/
  0, /*tp_setattr*/
  #if PY_MAJOR_VERSION < 3
  0, /*tp_compare*/
  #endif
  #if PY_MAJOR_VERSION >= 3
  0, /*tp_as_async*/
  #endif
  __pyx_pw_8PyClical_8clifford_95__repr__, /*tp_repr*/
  &__pyx_tp_as_number_clifford, /*tp_as_number*/
  &__pyx_tp_as_sequence_clifford, /*tp_as_sequence*/
  &__pyx_tp_as_mapping_clifford, /*tp_as_mapping*/
  0, /*tp_hash*/
  __pyx_pw_8PyClical_8clifford_61__call__, /*tp_call*/
  __pyx_pw_8PyClical_8clifford_97__str__, /*tp_str*/
  0, /*tp_getattro*/
  0, /*tp_setattro*/
  0, /*tp_as_buffer*/
  Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_BASETYPE, /*tp_flags*/
  "\n    Python class clifford wraps C++ class Clifford.\n    ", /*tp_doc*/
  0, /*tp_traverse*/
  0, /*tp_clear*/
  __pyx_pw_8PyClical_8clifford_13__richcmp__, /*tp_richcompare*/
  0, /*tp_weaklistoffset*/
  __pyx_pw_8PyClical_8clifford_9__iter__, /*tp_iter*/
  0, /*tp_iternext*/
  __pyx_methods_8PyClical_clifford, /*tp_methods*/
  0, /*tp_members*/
  0, /*tp_getset*/
  0, /*tp_base*/
  0, /*tp_dict*/
  0, /*tp_descr_get*/
  0, /*tp_descr_set*/
  0, /*tp_dictoffset*/
  0, /*tp_init*/
  0, /*tp_alloc*/
  __pyx_tp_new_8PyClical_clifford, /*tp_new*/
  0, /*tp_free*/
  0, /*tp_is_gc*/
  0, /*tp_bases*/
  0, /*tp_mro*/
  0, /*tp_cache*/
  0, /*tp_subclasses*/
  0, /*tp_weaklist*/
  0, /*tp_del*/
  0, /*tp_version_tag*/
  #if PY_VERSION_HEX >= 0x030400a1
  0, /*tp_finalize*/
  #endif
  #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM >= 0x07030800)
  0, /*tp_vectorcall*/
  #endif
  #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
  0, /*tp_print*/
  #endif
  #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
  0, /*tp_pypy_flags*/
  #endif
};

static struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *__pyx_freelist_8PyClical___pyx_scope_struct____iter__[8];
static int __pyx_freecount_8PyClical___pyx_scope_struct____iter__ = 0;

static PyObject *__pyx_tp_new_8PyClical___pyx_scope_struct____iter__(PyTypeObject *t, CYTHON_UNUSED PyObject *a, CYTHON_UNUSED PyObject *k) {
  PyObject *o;
  if (CYTHON_COMPILING_IN_CPYTHON && likely((__pyx_freecount_8PyClical___pyx_scope_struct____iter__ > 0) & (t->tp_basicsize == sizeof(struct __pyx_obj_8PyClical___pyx_scope_struct____iter__)))) {
    o = (PyObject*)__pyx_freelist_8PyClical___pyx_scope_struct____iter__[--__pyx_freecount_8PyClical___pyx_scope_struct____iter_
    memset(o, 0, sizeof(struct __pyx_obj_8PyClical___pyx_scope_struct____iter__));
    (void) PyObject_INIT(o, t);
    PyObject_GC_Track(o);
  } else {
    o = (*t->tp_alloc)(t, 0);
    if (unlikely(!o)) return 0;
  }
  return o;
}
```

```
static void __pyx_tp_dealloc_8PyClical___pyx_scope_struct____iter__(PyObject *o) {
  struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *p = (struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *)o;
  PyObject_GC_UnTrack(o);
  Py_CLEAR(p->__pyx_v_idx);
  Py_CLEAR(p->__pyx_v_self);
  Py_CLEAR(p->__pyx_t_0);
  if (CYTHON_COMPILING_IN_CPYTHON && ((__pyx_freecount_8PyClical___pyx_scope_struct____iter__ < 8) & (Py_TYPE(o)->tp_basicsize == sizeof(struct __pyx_obj_8PyClical___pyx_scope_struct____iter__)))) {
    __pyx_freelist_8PyClical___pyx_scope_struct____iter__[__pyx_freecount_8PyClical___pyx_scope_struct____iter__++] = ((struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *)o);
  } else {
    (*Py_TYPE(o)->tp_free)(o);
  }
}

static int __pyx_tp_traverse_8PyClical___pyx_scope_struct____iter__(PyObject *o, visitproc v, void *a) {
  int e;
  struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *p = (struct __pyx_obj_8PyClical___pyx_scope_struct____iter__ *)o;
  if (p->__pyx_v_idx) {
    e = (*v)(p->__pyx_v_idx, a); if (e) return e;
  }
  if (p->__pyx_v_self) {
    e = (*v)(((PyObject *)p->__pyx_v_self), a); if (e) return e;
  }
  if (p->__pyx_t_0) {
    e = (*v)(p->__pyx_t_0, a); if (e) return e;
  }
  return 0;
}

static PyTypeObject __pyx_type_8PyClical___pyx_scope_struct____iter__ = {
  PyVarObject_HEAD_INIT(0, 0)
  "PyClical.__pyx_scope_struct____iter__", /*tp_name*/
  sizeof(struct __pyx_obj_8PyClical___pyx_scope_struct____iter__), /*tp_basicsize*/
  0, /*tp_itemsize*/
  __pyx_tp_dealloc_8PyClical___pyx_scope_struct____iter__, /*tp_dealloc*/
  #if PY_VERSION_HEX < 0x030800b4
  0, /*tp_print*/
  #endif
  #if PY_VERSION_HEX >= 0x030800b4
  0, /*tp_vectorcall_offset*/
  #endif
  0, /*tp_getattr*/
  0, /*tp_setattr*/
  #if PY_MAJOR_VERSION < 3
  0, /*tp_compare*/
  #endif
  #if PY_MAJOR_VERSION >= 3
  0, /*tp_as_async*/
  #endif
  0, /*tp_repr*/
  0, /*tp_as_number*/
  0, /*tp_as_sequence*/
  0, /*tp_as_mapping*/
  0, /*tp_hash*/
  0, /*tp_call*/
  0, /*tp_str*/
  0, /*tp_getattro*/
  0, /*tp_setattro*/
  0, /*tp_as_buffer*/
  Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_HAVE_GC, /*tp_flags*/
  0, /*tp_doc*/
  __pyx_tp_traverse_8PyClical___pyx_scope_struct____iter__, /*tp_traverse*/
  0, /*tp_clear*/
  0, /*tp_richcompare*/
  0, /*tp_weaklistoffset*/
  0, /*tp_iter*/
  0, /*tp_iternext*/
  0, /*tp_methods*/
  0, /*tp_members*/
  0, /*tp_getset*/
  0, /*tp_base*/
  0, /*tp_dict*/
  0, /*tp_descr_get*/
  0, /*tp_descr_set*/
  0, /*tp_dictoffset*/
  0, /*tp_init*/
  0, /*tp_alloc*/
  __pyx_tp_new_8PyClical___pyx_scope_struct____iter__, /*tp_new*/
  0, /*tp_free*/
```

```
    0, /*tp_is_gc*/
    0, /*tp_bases*/
    0, /*tp_mro*/
    0, /*tp_cache*/
    0, /*tp_subclasses*/
    0, /*tp_weaklist*/
    0, /*tp_del*/
    0, /*tp_version_tag*/
    #if PY_VERSION_HEX >= 0x030400a1
    0, /*tp_finalize*/
    #endif
    #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM >= 0x07030800)
    0, /*tp_vectorcall*/
    #endif
    #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
    0, /*tp_print*/
    #endif
    #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
    0, /*tp_pypy_flags*/
    #endif
  };

  static PyMethodDef __pyx_methods[] = {
    {"compare", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_3compare, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_2compare},
    {"min_neg", (PyCFunction)__pyx_pw_8PyClical_5min_neg, METH_O, __pyx_doc_8PyClical_4min_neg},
    {"max_pos", (PyCFunction)__pyx_pw_8PyClical_7max_pos, METH_O, __pyx_doc_8PyClical_6max_pos},
    {"error_squared_tol", (PyCFunction)__pyx_pw_8PyClical_11error_squared_tol, METH_O, __pyx_doc_8PyClical_10error_squared_tol},
    {"error_squared", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_13error_squared, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_12error_squared},
    {"approx_equal", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_15approx_equal, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_14approx_equal},
    {"inv", (PyCFunction)__pyx_pw_8PyClical_17inv, METH_O, __pyx_doc_8PyClical_16inv},
    {"scalar", (PyCFunction)__pyx_pw_8PyClical_19scalar, METH_O, __pyx_doc_8PyClical_18scalar},
    {"real", (PyCFunction)__pyx_pw_8PyClical_21real, METH_O, __pyx_doc_8PyClical_20real},
    {"imag", (PyCFunction)__pyx_pw_8PyClical_23imag, METH_O, __pyx_doc_8PyClical_22imag},
    {"pure", (PyCFunction)__pyx_pw_8PyClical_25pure, METH_O, __pyx_doc_8PyClical_24pure},
    {"even", (PyCFunction)__pyx_pw_8PyClical_27even, METH_O, __pyx_doc_8PyClical_26even},
    {"odd", (PyCFunction)__pyx_pw_8PyClical_29odd, METH_O, __pyx_doc_8PyClical_28odd},
    {"involute", (PyCFunction)__pyx_pw_8PyClical_31involute, METH_O, __pyx_doc_8PyClical_30involute},
    {"reverse", (PyCFunction)__pyx_pw_8PyClical_33reverse, METH_O, __pyx_doc_8PyClical_32reverse},
    {"conj", (PyCFunction)__pyx_pw_8PyClical_35conj, METH_O, __pyx_doc_8PyClical_34conj},
    {"quad", (PyCFunction)__pyx_pw_8PyClical_37quad, METH_O, __pyx_doc_8PyClical_36quad},
    {"norm", (PyCFunction)__pyx_pw_8PyClical_39norm, METH_O, __pyx_doc_8PyClical_38norm},
    {"abs", (PyCFunction)__pyx_pw_8PyClical_41abs, METH_O, __pyx_doc_8PyClical_40abs},
    {"max_abs", (PyCFunction)__pyx_pw_8PyClical_43max_abs, METH_O, __pyx_doc_8PyClical_42max_abs},
    {"pow", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_45pow, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_44pow},
    {"outer_pow", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_47outer_pow, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_46outer_pow},
    {"complexifier", (PyCFunction)__pyx_pw_8PyClical_49complexifier, METH_O, __pyx_doc_8PyClical_48complexifier},
    {"sqrt", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_51sqrt, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_50sqrt},
    {"exp", (PyCFunction)__pyx_pw_8PyClical_53exp, METH_O, __pyx_doc_8PyClical_52exp},
    {"log", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_55log, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_54log},
    {"cos", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_57cos, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_56cos},
    {"acos", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_59acos, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_58acos},
    {"cosh", (PyCFunction)__pyx_pw_8PyClical_61cosh, METH_O, __pyx_doc_8PyClical_60cosh},
    {"acosh", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_63acosh, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_62acosh},
    {"sin", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_65sin, METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_64sin},
    {"asin", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_67asin,
```

```
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_66asin},
                  {"sinh", (PyCFunction)__pyx_pw_8PyClical_69sinh, METH_O,
__pyx_doc_8PyClical_68sinh},
                  {"asinh", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_71asinh,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_70asinh},
                  {"tan", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_73tan,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_72tan},
                  {"atan", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_75atan,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_74atan},
                  {"tanh", (PyCFunction)__pyx_pw_8PyClical_77tanh, METH_O,
__pyx_doc_8PyClical_76tanh},
                  {"atanh", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_79atanh,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_78atanh},
                  {"random_clifford",
(PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_81random_clifford,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_80random_clifford},
                  {"cga3", (PyCFunction)__pyx_pw_8PyClical_83cga3, METH_O,
__pyx_doc_8PyClical_82cga3},
                  {"cga3std", (PyCFunction)__pyx_pw_8PyClical_85cga3std, METH_O,
__pyx_doc_8PyClical_84cga3std},
                  {"agc3", (PyCFunction)__pyx_pw_8PyClical_87agc3, METH_O,
__pyx_doc_8PyClical_86agc3},
                  {0, 0, 0, 0}
                };

                #if PY_MAJOR_VERSION >= 3
                #if CYTHON_PEP489_MULTI_PHASE_INIT
                static PyObject* __pyx_pymod_create(PyObject *spec, PyModuleDef *def); /*proto*/
                static int __pyx_pymod_exec_PyClical(PyObject* module); /*proto*/
                static PyModuleDef_Slot __pyx_moduledef_slots[] = {
                  {Py_mod_create, (void*)__pyx_pymod_create},
                  {Py_mod_exec, (void*)__pyx_pymod_exec_PyClical},
                  {0, NULL}
                };
                #endif

                static struct PyModuleDef __pyx_moduledef = {
                    PyModuleDef_HEAD_INIT,
                    "PyClical",
                    0, /* m_doc */
                  #if CYTHON_PEP489_MULTI_PHASE_INIT
                    0, /* m_size */
                  #else
                    -1, /* m_size */
                  #endif
                    __pyx_methods /* m_methods */,
                  #if CYTHON_PEP489_MULTI_PHASE_INIT
                    __pyx_moduledef_slots, /* m_slots */
                  #else
                    NULL, /* m_reload */
                  #endif
                    NULL, /* m_traverse */
                    NULL, /* m_clear */
                    NULL /* m_free */
                };
                #endif
                #ifndef CYTHON_SMALL_CODE
#if defined(__clang__)
    #define CYTHON_SMALL_CODE
#elif defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ >= 3))
    #define CYTHON_SMALL_CODE __attribute__((cold))
#else
    #define CYTHON_SMALL_CODE
#endif
#endif

static __Pyx_StringTabEntry __pyx_string_tab[] = {
  {&__pyx_kp_u_, __pyx_k_, sizeof(__pyx_k_), 0, 1, 0, 0},
  {&__pyx_kp_u_Abbreviation_for_clifford_index, __pyx_k_Abbreviation_for_clifford_index,
sizeof(__pyx_k_Abbreviation_for_clifford_index), 0, 1, 0, 0},
  {&__pyx_kp_u_Abbreviation_for_index_set_q_p, __pyx_k_Abbreviation_for_index_set_q_p,
sizeof(__pyx_k_Abbreviation_for_index_set_q_p), 0, 1, 0, 0},
  {&__pyx_kp_u_Absolute_value_of_multivector_m, __pyx_k_Absolute_value_of_multivector_m,
sizeof(__pyx_k_Absolute_value_of_multivector_m), 0, 1, 0, 0},
  {&__pyx_kp_u_Absolute_value_square_root_of_n, __pyx_k_Absolute_value_square_root_of_n,
sizeof(__pyx_k_Absolute_value_square_root_of_n), 0, 1, 0, 0},
  {&__pyx_kp_u_Cannot_initialize_clifford_objec, __pyx_k_Cannot_initialize_clifford_objec,
sizeof(__pyx_k_Cannot_initialize_clifford_objec), 0, 1, 0, 0},
  {&__pyx_kp_u_Cannot_initialize_index_set_obje, __pyx_k_Cannot_initialize_index_set_obje,
sizeof(__pyx_k_Cannot_initialize_index_set_obje), 0, 1, 0, 0},
  {&__pyx_kp_u_Cannot_reframe, __pyx_k_Cannot_reframe, sizeof(__pyx_k_Cannot_reframe), 0, 1, 0, 0},
  {&__pyx_kp_u_Cannot_take_vector_part_of, __pyx_k_Cannot_take_vector_part_of,
sizeof(__pyx_k_Cannot_take_vector_part_of), 0, 1, 0, 0},
  {&__pyx_kp_u_Cardinality_Number_of_indices_i, __pyx_k_Cardinality_Number_of_indices_i,
sizeof(__pyx_k_Cardinality_Number_of_indices_i), 0, 1, 0, 0},
  {&__pyx_kp_u_Check_if_a_multivector_contains, __pyx_k_Check_if_a_multivector_contains,
sizeof(__pyx_k_Check_if_a_multivector_contains), 0, 1, 0, 0},
```

```
{&__pyx_kp_u_Check_if_a_multivector_contains_2, __pyx_k_Check_if_a_multivector_contains_2, sizeof(__pyx_k_Check_if_a_multivector_contains_2), 0, 1, 0, 0},
{&__pyx_kp_u_Conjugation_reverse_o_involute, __pyx_k_Conjugation_reverse_o_involute, sizeof(__pyx_k_Conjugation_reverse_o_involute), 0, 1, 0, 0},
{&__pyx_kp_u_Conjugation_reverse_o_involute_2, __pyx_k_Conjugation_reverse_o_involute_2, sizeof(__pyx_k_Conjugation_reverse_o_involute_2), 0, 1, 0, 0},
{&__pyx_kp_u_Contraction_print_clifford_1_cl, __pyx_k_Contraction_print_clifford_1_cl, sizeof(__pyx_k_Contraction_print_clifford_1_cl), 0, 1, 0, 0},
{&__pyx_kp_u_Contraction_x_clifford_1_x_clif, __pyx_k_Contraction_x_clifford_1_x_clif, sizeof(__pyx_k_Contraction_x_clifford_1_x_clif), 0, 1, 0, 0},
{&__pyx_kp_u_Convert_CGA3_null_vector_to_Euc, __pyx_k_Convert_CGA3_null_vector_to_Euc, sizeof(__pyx_k_Convert_CGA3_null_vector_to_Euc), 0, 1, 0, 0},
{&__pyx_kp_u_Convert_CGA3_null_vector_to_sta, __pyx_k_Convert_CGA3_null_vector_to_sta, sizeof(__pyx_k_Convert_CGA3_null_vector_to_sta), 0, 1, 0, 0},
{&__pyx_kp_u_Convert_Euclidean_3D_multivecto, __pyx_k_Convert_Euclidean_3D_multivecto, sizeof(__pyx_k_Convert_Euclidean_3D_multivecto), 0, 1, 0, 0},
{&__pyx_kp_u_Copy_this_clifford_object_x_cli, __pyx_k_Copy_this_clifford_object_x_cli, sizeof(__pyx_k_Copy_this_clifford_object_x_cli), 0, 1, 0, 0},
{&__pyx_kp_u_Copy_this_index_set_object_s_in, __pyx_k_Copy_this_index_set_object_s_in, sizeof(__pyx_k_Copy_this_index_set_object_s_in), 0, 1, 0, 0},
{&__pyx_kp_u_Cosine_of_multivector_with_opti, __pyx_k_Cosine_of_multivector_with_opti, sizeof(__pyx_k_Cosine_of_multivector_with_opti), 0, 1, 0, 0},
{&__pyx_kp_u_Even_part_of_multivector_sum_of, __pyx_k_Even_part_of_multivector_sum_of, sizeof(__pyx_k_Even_part_of_multivector_sum_of), 0, 1, 0, 0},
{&__pyx_kp_u_Even_part_of_multivector_sum_of_2, __pyx_k_Even_part_of_multivector_sum_of_2, sizeof(__pyx_k_Even_part_of_multivector_sum_of_2), 0, 1, 0, 0},
{&__pyx_kp_u_Exponential_of_multivector_x_cl, __pyx_k_Exponential_of_multivector_x_cl, sizeof(__pyx_k_Exponential_of_multivector_x_cl), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_difference_print_clif, __pyx_k_Geometric_difference_print_clif, sizeof(__pyx_k_Geometric_difference_print_clif), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_difference_x_clifford, __pyx_k_Geometric_difference_x_clifford, sizeof(__pyx_k_Geometric_difference_x_clifford), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_multiplicative_invers, __pyx_k_Geometric_multiplicative_invers, sizeof(__pyx_k_Geometric_multiplicative_invers), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_multiplicative_invers_2, __pyx_k_Geometric_multiplicative_invers_2, sizeof(__pyx_k_Geometric_multiplicative_invers_2), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_product_print_cliffor, __pyx_k_Geometric_product_print_cliffor, sizeof(__pyx_k_Geometric_product_print_cliffor), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_product_x_clifford_2, __pyx_k_Geometric_product_x_clifford_2, sizeof(__pyx_k_Geometric_product_x_clifford_2), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_quotient_print_cliffo, __pyx_k_Geometric_quotient_print_cliffo, sizeof(__pyx_k_Geometric_quotient_print_cliffo), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_quotient_x_clifford_1, __pyx_k_Geometric_quotient_x_clifford_1, sizeof(__pyx_k_Geometric_quotient_x_clifford_1), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_sum_print_clifford_1, __pyx_k_Geometric_sum_print_clifford_1, sizeof(__pyx_k_Geometric_sum_print_clifford_1), 0, 1, 0, 0},
{&__pyx_kp_u_Geometric_sum_x_clifford_1_x_cl, __pyx_k_Geometric_sum_x_clifford_1_x_cl, sizeof(__pyx_k_Geometric_sum_x_clifford_1_x_cl), 0, 1, 0, 0},
{&__pyx_kp_u_Get_the_value_of_an_index_set_o, __pyx_k_Get_the_value_of_an_index_set_o, sizeof(__pyx_k_Get_the_value_of_an_index_set_o), 0, 1, 0, 0},
{&__pyx_kp_u_Hyperbolic_cosine_of_multivecto, __pyx_k_Hyperbolic_cosine_of_multivecto, sizeof(__pyx_k_Hyperbolic_cosine_of_multivecto), 0, 1, 0, 0},
{&__pyx_kp_u_Hyperbolic_sine_of_multivector, __pyx_k_Hyperbolic_sine_of_multivector, sizeof(__pyx_k_Hyperbolic_sine_of_multivector), 0, 1, 0, 0},
{&__pyx_kp_u_Hyperbolic_tangent_of_multivect, __pyx_k_Hyperbolic_tangent_of_multivect, sizeof(__pyx_k_Hyperbolic_tangent_of_multivect), 0, 1, 0, 0},
{&__pyx_kp_u_Imaginary_part_deprecated_alway, __pyx_k_Imaginary_part_deprecated_alway, sizeof(__pyx_k_Imaginary_part_deprecated_alway), 0, 1, 0, 0},
{&__pyx_n_s_IndexError, __pyx_k_IndexError, sizeof(__pyx_k_IndexError), 0, 0, 1, 1},
{&__pyx_kp_u_Inner_product_print_clifford_1, __pyx_k_Inner_product_print_clifford_1, sizeof(__pyx_k_Inner_product_print_clifford_1), 0, 1, 0, 0},
{&__pyx_kp_u_Inner_product_x_clifford_1_x_cl, __pyx_k_Inner_product_x_clifford_1_x_cl, sizeof(__pyx_k_Inner_product_x_clifford_1_x_cl), 0, 1, 0, 0},
{&__pyx_kp_u_Integer_power_of_multivector_ob, __pyx_k_Integer_power_of_multivector_ob, sizeof(__pyx_k_Integer_power_of_multivector_ob), 0, 1, 0, 0},
{&__pyx_n_s_Integral, __pyx_k_Integral, sizeof(__pyx_k_Integral), 0, 0, 1, 1},
{&__pyx_kp_u_Inverse_cosine_of_multivector_w, __pyx_k_Inverse_cosine_of_multivector_w, sizeof(__pyx_k_Inverse_cosine_of_multivector_w), 0, 1, 0, 0},
{&__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu, __pyx_k_Inverse_hyperbolic_cosine_of_mu, sizeof(__pyx_k_Inverse_hyperbolic_cosine_of_mu), 0, 1, 0, 0},
{&__pyx_kp_u_Inverse_hyperbolic_sine_of_mult, __pyx_k_Inverse_hyperbolic_sine_of_mult, sizeof(__pyx_k_Inverse_hyperbolic_sine_of_mult), 0, 1, 0, 0},
{&__pyx_kp_u_Inverse_hyperbolic_tangent_of_m, __pyx_k_Inverse_hyperbolic_tangent_of_m, sizeof(__pyx_k_Inverse_hyperbolic_tangent_of_m), 0, 1, 0, 0},
{&__pyx_kp_u_Inverse_sine_of_multivector_wit, __pyx_k_Inverse_sine_of_multivector_wit, sizeof(__pyx_k_Inverse_sine_of_multivector_wit), 0, 1, 0, 0},
{&__pyx_kp_u_Inverse_tangent_of_multivector, __pyx_k_Inverse_tangent_of_multivector, sizeof(__pyx_k_Inverse_tangent_of_multivector), 0, 1, 0, 0},
{&__pyx_kp_u_Iterate_over_the_indices_of_an, __pyx_k_Iterate_over_the_indices_of_an, sizeof(__pyx_k_Iterate_over_the_indices_of_an), 0, 1, 0, 0},
{&__pyx_kp_u_Main_involution_each_i_is_repla, __pyx_k_Main_involution_each_i_is_repla, sizeof(__pyx_k_Main_involution_each_i_is_repla), 0, 1, 0, 0},
{&__pyx_kp_u_Main_involution_each_i_is_repla_2, __pyx_k_Main_involution_each_i_is_repla_2, sizeof(__pyx_k_Main_involution_each_i_is_repla_2), 0, 1, 0, 0},
{&__pyx_kp_u_Maximum_absolute_value_of_coord, __pyx_k_Maximum_absolute_value_of_coord, sizeof(__pyx_k_Maximum_absolute_value_of_coord), 0, 1, 0, 0},
{&__pyx_kp_u_Maximum_member_index_set_1_1_2, __pyx_k_Maximum_member_index_set_1_1_2,
```

```
sizeof(__pyx_k_Maximum_member_index_set_1_1_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Maximum_of_absolute_values_of_c, __pyx_k_Maximum_of_absolute_values_of_c,
sizeof(__pyx_k_Maximum_of_absolute_values_of_c), 0, 1, 0, 0},
  {&__pyx_kp_u_Maximum_positive_index_or_0_if, __pyx_k_Maximum_positive_index_or_0_if,
sizeof(__pyx_k_Maximum_positive_index_or_0_if), 0, 1, 0, 0},
  {&__pyx_kp_u_Minimum_member_index_set_1_1_2, __pyx_k_Minimum_member_index_set_1_1_2,
sizeof(__pyx_k_Minimum_member_index_set_1_1_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Minimum_negative_index_or_0_if, __pyx_k_Minimum_negative_index_or_0_if,
sizeof(__pyx_k_Minimum_negative_index_or_0_if), 0, 1, 0, 0},
  {&__pyx_kp_u_Natural_logarithm_of_multivecto, __pyx_k_Natural_logarithm_of_multivecto,
sizeof(__pyx_k_Natural_logarithm_of_multivecto), 0, 1, 0, 0},
  {&__pyx_kp_u_Norm_sum_of_squares_of_coordina, __pyx_k_Norm_sum_of_squares_of_coordina,
sizeof(__pyx_k_Norm_sum_of_squares_of_coordina), 0, 1, 0, 0},
  {&__pyx_n_s_NotImplemented, __pyx_k_NotImplemented, sizeof(__pyx_k_NotImplemented), 0, 0, 1, 1},
  {&__pyx_kp_u_Not_applicable, __pyx_k_Not_applicable, sizeof(__pyx_k_Not_applicable), 0, 1, 0, 0},
  {&__pyx_kp_u_Not_applicable_for_a_in_cliffor, __pyx_k_Not_applicable_for_a_in_cliffor,
sizeof(__pyx_k_Not_applicable_for_a_in_cliffor), 0, 1, 0, 0},
  {&__pyx_kp_u_Number_of_negative_indices_incl, __pyx_k_Number_of_negative_indices_incl,
sizeof(__pyx_k_Number_of_negative_indices_incl), 0, 1, 0, 0},
  {&__pyx_kp_u_Number_of_positive_indices_incl, __pyx_k_Number_of_positive_indices_incl,
sizeof(__pyx_k_Number_of_positive_indices_incl), 0, 1, 0, 0},
  {&__pyx_kp_u_Odd_part_of_multivector_sum_of, __pyx_k_Odd_part_of_multivector_sum_of,
sizeof(__pyx_k_Odd_part_of_multivector_sum_of), 0, 1, 0, 0},
  {&__pyx_kp_u_Odd_part_of_multivector_sum_of_2, __pyx_k_Odd_part_of_multivector_sum_of_2,
sizeof(__pyx_k_Odd_part_of_multivector_sum_of_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Outer_product_power_of_multivec, __pyx_k_Outer_product_power_of_multivec,
sizeof(__pyx_k_Outer_product_power_of_multivec), 0, 1, 0, 0},
  {&__pyx_kp_u_Outer_product_power_x_clifford, __pyx_k_Outer_product_power_x_clifford,
sizeof(__pyx_k_Outer_product_power_x_clifford), 0, 1, 0, 0},
  {&__pyx_kp_u_Outer_product_print_clifford_1, __pyx_k_Outer_product_print_clifford_1,
sizeof(__pyx_k_Outer_product_print_clifford_1), 0, 1, 0, 0},
  {&__pyx_kp_u_Outer_product_x_clifford_1_x_cl, __pyx_k_Outer_product_x_clifford_1_x_cl,
sizeof(__pyx_k_Outer_product_x_clifford_1_x_cl), 0, 1, 0, 0},
  {&__pyx_kp_u_Power_self_to_the_m_x_clifford, __pyx_k_Power_self_to_the_m_x_clifford,
sizeof(__pyx_k_Power_self_to_the_m_x_clifford), 0, 1, 0, 0},
  {&__pyx_kp_u_Power_self_to_the_m_x_clifford_2, __pyx_k_Power_self_to_the_m_x_clifford_2,
sizeof(__pyx_k_Power_self_to_the_m_x_clifford_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Pure_grade_vector_part_print_cl, __pyx_k_Pure_grade_vector_part_print_cl,
sizeof(__pyx_k_Pure_grade_vector_part_print_cl), 0, 1, 0, 0},
  {&__pyx_kp_u_Pure_part_print_clifford_1_1_1, __pyx_k_Pure_part_print_clifford_1_1_1,
sizeof(__pyx_k_Pure_part_print_clifford_1_1_1), 0, 1, 0, 0},
  {&__pyx_kp_u_Pure_part_print_pure_clifford_1, __pyx_k_Pure_part_print_pure_clifford_1,
sizeof(__pyx_k_Pure_part_print_pure_clifford_1), 0, 1, 0, 0},
  {&__pyx_kp_u_Put_self_into_a_larger_frame_co, __pyx_k_Put_self_into_a_larger_frame_co,
sizeof(__pyx_k_Put_self_into_a_larger_frame_co), 0, 1, 0, 0},
  {&__pyx_n_s_PyClical, __pyx_k_PyClical, sizeof(__pyx_k_PyClical), 0, 0, 1, 1},
  {&__pyx_kp_s_PyClical_pyx, __pyx_k_PyClical_pyx, sizeof(__pyx_k_PyClical_pyx), 0, 0, 1, 0},
  {&__pyx_kp_u_Quadratic_form_rev_x_x_0_print, __pyx_k_Quadratic_form_rev_x_x_0_print,
sizeof(__pyx_k_Quadratic_form_rev_x_x_0_print), 0, 1, 0, 0},
  {&__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2, __pyx_k_Quadratic_form_rev_x_x_0_print_2,
sizeof(__pyx_k_Quadratic_form_rev_x_x_0_print_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Quadratic_norm_error_tolerance, __pyx_k_Quadratic_norm_error_tolerance,
sizeof(__pyx_k_Quadratic_norm_error_tolerance), 0, 1, 0, 0},
  {&__pyx_kp_u_Random_multivector_within_a_fra, __pyx_k_Random_multivector_within_a_fra,
sizeof(__pyx_k_Random_multivector_within_a_fra), 0, 1, 0, 0},
  {&__pyx_n_s_Real, __pyx_k_Real, sizeof(__pyx_k_Real), 0, 0, 1, 1},
  {&__pyx_kp_u_Real_part_synonym_for_scalar_pa, __pyx_k_Real_part_synonym_for_scalar_pa,
sizeof(__pyx_k_Real_part_synonym_for_scalar_pa), 0, 1, 0, 0},
  {&__pyx_kp_u_Relative_or_absolute_error_usin, __pyx_k_Relative_or_absolute_error_usin,
sizeof(__pyx_k_Relative_or_absolute_error_usin), 0, 1, 0, 0},
  {&__pyx_kp_u_Remove_all_terms_of_self_with_r, __pyx_k_Remove_all_terms_of_self_with_r,
sizeof(__pyx_k_Remove_all_terms_of_self_with_r), 0, 1, 0, 0},
  {&__pyx_kp_u_Reversion_eg_1_2_2_1_print_reve, __pyx_k_Reversion_eg_1_2_2_1_print_reve,
sizeof(__pyx_k_Reversion_eg_1_2_2_1_print_reve), 0, 1, 0, 0},
  {&__pyx_kp_u_Reversion_eg_clifford_1_cliffor, __pyx_k_Reversion_eg_clifford_1_cliffor,
sizeof(__pyx_k_Reversion_eg_clifford_1_cliffor), 0, 1, 0, 0},
  {&__pyx_n_s_RuntimeError, __pyx_k_RuntimeError, sizeof(__pyx_k_RuntimeError), 0, 0, 1, 1},
  {&__pyx_kp_u_Scalar_part_clifford_1_1_1_2_sc, __pyx_k_Scalar_part_clifford_1_1_1_2_sc,
sizeof(__pyx_k_Scalar_part_clifford_1_1_1_2_sc), 0, 1, 0, 0},
  {&__pyx_kp_u_Scalar_part_scalar_clifford_1_1, __pyx_k_Scalar_part_scalar_clifford_1_1,
sizeof(__pyx_k_Scalar_part_scalar_clifford_1_1), 0, 1, 0, 0},
  {&__pyx_n_s_Sequence, __pyx_k_Sequence, sizeof(__pyx_k_Sequence), 0, 0, 1, 1},
  {&__pyx_kp_u_Set_complement_not_print_index, __pyx_k_Set_complement_not_print_index,
sizeof(__pyx_k_Set_complement_not_print_index), 0, 1, 0, 0},
  {&__pyx_kp_u_Set_intersection_and_print_inde, __pyx_k_Set_intersection_and_print_inde,
sizeof(__pyx_k_Set_intersection_and_print_inde), 0, 1, 0, 0},
  {&__pyx_kp_u_Set_intersection_and_x_index_se, __pyx_k_Set_intersection_and_x_index_se,
sizeof(__pyx_k_Set_intersection_and_x_index_se), 0, 1, 0, 0},
  {&__pyx_kp_u_Set_the_value_of_an_index_set_o, __pyx_k_Set_the_value_of_an_index_set_o,
sizeof(__pyx_k_Set_the_value_of_an_index_set_o), 0, 1, 0, 0},
  {&__pyx_kp_u_Set_union_or_print_index_set_1, __pyx_k_Set_union_or_print_index_set_1,
sizeof(__pyx_k_Set_union_or_print_index_set_1), 0, 1, 0, 0},
  {&__pyx_kp_u_Set_union_or_x_index_set_1_x_in, __pyx_k_Set_union_or_x_index_set_1_x_in,
sizeof(__pyx_k_Set_union_or_x_index_set_1_x_in), 0, 1, 0, 0},
  {&__pyx_kp_u_Sign_of_geometric_product_of_tw, __pyx_k_Sign_of_geometric_product_of_tw,
sizeof(__pyx_k_Sign_of_geometric_product_of_tw), 0, 1, 0, 0},
  {&__pyx_kp_u_Sign_of_geometric_square_of_a_C, __pyx_k_Sign_of_geometric_square_of_a_C,
```

```
sizeof(__pyx_k_Sign_of_geometric_square_of_a_C), 0, 1, 0, 0},
  {&__pyx_kp_u_Sine_of_multivector_with_option, __pyx_k_Sine_of_multivector_with_option,
sizeof(__pyx_k_Sine_of_multivector_with_option), 0, 1, 0, 0},
  {&__pyx_kp_u_Square_root_of_1_which_commutes, __pyx_k_Square_root_of_1_which_commutes,
sizeof(__pyx_k_Square_root_of_1_which_commutes), 0, 1, 0, 0},
  {&__pyx_kp_u_Square_root_of_multivector_with, __pyx_k_Square_root_of_multivector_with,
sizeof(__pyx_k_Square_root_of_multivector_with), 0, 1, 0, 0},
  {&__pyx_kp_u_Subalgebra_generated_by_all_gen, __pyx_k_Subalgebra_generated_by_all_gen,
sizeof(__pyx_k_Subalgebra_generated_by_all_gen), 0, 1, 0, 0},
  {&__pyx_kp_u_Subscripting_map_from_index_set, __pyx_k_Subscripting_map_from_index_set,
sizeof(__pyx_k_Subscripting_map_from_index_set), 0, 1, 0, 0},
  {&__pyx_kp_u_Symmetric_set_difference_exclus, __pyx_k_Symmetric_set_difference_exclus,
sizeof(__pyx_k_Symmetric_set_difference_exclus), 0, 1, 0, 0},
  {&__pyx_kp_u_Symmetric_set_difference_exclus_2, __pyx_k_Symmetric_set_difference_exclus_2,
sizeof(__pyx_k_Symmetric_set_difference_exclus_2), 0, 1, 0, 0},
  {&__pyx_kp_u_Tangent_of_multivector_with_opt, __pyx_k_Tangent_of_multivector_with_opt,
sizeof(__pyx_k_Tangent_of_multivector_with_opt), 0, 1, 0, 0},
  {&__pyx_kp_u_Test_for_approximate_equality_o, __pyx_k_Test_for_approximate_equality_o,
sizeof(__pyx_k_Test_for_approximate_equality_o), 0, 1, 0, 0},
  {&__pyx_kp_u_Tests_for_functions_that_Doctes, __pyx_k_Tests_for_functions_that_Doctes,
sizeof(__pyx_k_Tests_for_functions_that_Doctes), 0, 1, 0, 0},
  {&__pyx_kp_u_Tests_for_functions_that_Doctes_2, __pyx_k_Tests_for_functions_that_Doctes_2,
sizeof(__pyx_k_Tests_for_functions_that_Doctes_2), 0, 1, 0, 0},
  {&__pyx_kp_u_The_informal_string_representat, __pyx_k_The_informal_string_representat,
sizeof(__pyx_k_The_informal_string_representat), 0, 1, 0, 0},
  {&__pyx_kp_u_The_informal_string_representat_2, __pyx_k_The_informal_string_representat_2,
sizeof(__pyx_k_The_informal_string_representat_2), 0, 1, 0, 0},
  {&__pyx_kp_u_The_official_string_representat, __pyx_k_The_official_string_representat,
sizeof(__pyx_k_The_official_string_representat), 0, 1, 0, 0},
  {&__pyx_kp_u_The_official_string_representat_2, __pyx_k_The_official_string_representat_2,
sizeof(__pyx_k_The_official_string_representat_2), 0, 1, 0, 0},
  {&__pyx_kp_u_This_comparison_operator_is_not, __pyx_k_This_comparison_operator_is_not,
sizeof(__pyx_k_This_comparison_operator_is_not), 0, 1, 0, 0},
  {&__pyx_kp_u_Transform_left_hand_side_using, __pyx_k_Transform_left_hand_side_using,
sizeof(__pyx_k_Transform_left_hand_side_using), 0, 1, 0, 0},
  {&__pyx_kp_u_Transform_left_hand_side_using_2, __pyx_k_Transform_left_hand_side_using_2,
sizeof(__pyx_k_Transform_left_hand_side_using_2), 0, 1, 0, 0},
  {&__pyx_n_s_TypeError, __pyx_k_TypeError, sizeof(__pyx_k_TypeError), 0, 0, 1, 1},
  {&__pyx_kp_u_UTF_8, __pyx_k_UTF_8, sizeof(__pyx_k_UTF_8), 0, 1, 0, 0},
  {&__pyx_kp_u_Unary_print_clifford_1_1, __pyx_k_Unary_print_clifford_1_1,
sizeof(__pyx_k_Unary_print_clifford_1_1), 0, 1, 0, 0},
  {&__pyx_kp_u_Unary_print_clifford_1_1_2, __pyx_k_Unary_print_clifford_1_1_2,
sizeof(__pyx_k_Unary_print_clifford_1_1_2), 0, 1, 0, 0},
  {&__pyx_n_s_ValueError, __pyx_k_ValueError, sizeof(__pyx_k_ValueError), 0, 0, 1, 1},
  {&__pyx_kp_u_Vector_part_of_multivector_as_a, __pyx_k_Vector_part_of_multivector_as_a,
sizeof(__pyx_k_Vector_part_of_multivector_as_a), 0, 1, 0, 0},
  {&__pyx_kp_u__2, __pyx_k__2, sizeof(__pyx_k__2), 0, 1, 0, 0},
  {&__pyx_kp_u__5, __pyx_k__5, sizeof(__pyx_k__5), 0, 1, 0, 0},
  {&__pyx_kp_u__6, __pyx_k__6, sizeof(__pyx_k__6), 0, 1, 0, 0},
  {&__pyx_kp_u__7, __pyx_k__7, sizeof(__pyx_k__7), 0, 1, 0, 0},
  {&__pyx_kp_u__8, __pyx_k__8, sizeof(__pyx_k__8), 0, 1, 0, 0},
  {&__pyx_kp_u__9, __pyx_k__9, sizeof(__pyx_k__9), 0, 1, 0, 0},
  {&__pyx_n_s_abc, __pyx_k_abc, sizeof(__pyx_k_abc), 0, 0, 1, 1},
  {&__pyx_kp_u_abs_line_1522, __pyx_k_abs_line_1522, sizeof(__pyx_k_abs_line_1522), 0, 1, 0, 0},
  {&__pyx_n_s_acos, __pyx_k_acos, sizeof(__pyx_k_acos), 0, 0, 1, 1},
  {&__pyx_kp_u_acos_line_1668, __pyx_k_acos_line_1668, sizeof(__pyx_k_acos_line_1668), 0, 1, 0, 0},
  {&__pyx_n_s_acosh, __pyx_k_acosh, sizeof(__pyx_k_acosh), 0, 0, 1, 1},
  {&__pyx_kp_u_acosh_line_1705, __pyx_k_acosh_line_1705, sizeof(__pyx_k_acosh_line_1705), 0, 1, 0, 0},
  {&__pyx_kp_u_agc3_line_1893, __pyx_k_agc3_line_1893, sizeof(__pyx_k_agc3_line_1893), 0, 1, 0, 0},
  {&__pyx_kp_u_approx_equal_line_1359, __pyx_k_approx_equal_line_1359,
sizeof(__pyx_k_approx_equal_line_1359), 0, 1, 0, 0},
  {&__pyx_n_s_args, __pyx_k_args, sizeof(__pyx_k_args), 0, 0, 1, 1},
  {&__pyx_kp_u_as_frame, __pyx_k_as_frame, sizeof(__pyx_k_as_frame), 0, 1, 0, 0},
  {&__pyx_n_s_asin, __pyx_k_asin, sizeof(__pyx_k_asin), 0, 0, 1, 1},
  {&__pyx_kp_u_asin_line_1747, __pyx_k_asin_line_1747, sizeof(__pyx_k_asin_line_1747), 0, 1, 0, 0},
  {&__pyx_n_s_asinh, __pyx_k_asinh, sizeof(__pyx_k_asinh), 0, 0, 1, 1},
  {&__pyx_kp_u_asinh_line_1782, __pyx_k_asinh_line_1782, sizeof(__pyx_k_asinh_line_1782), 0, 1, 0, 0},
  {&__pyx_n_s_atan, __pyx_k_atan, sizeof(__pyx_k_atan), 0, 0, 1, 1},
  {&__pyx_kp_u_atan_line_1818, __pyx_k_atan_line_1818, sizeof(__pyx_k_atan_line_1818), 0, 1, 0, 0},
  {&__pyx_n_s_atanh, __pyx_k_atanh, sizeof(__pyx_k_atanh), 0, 0, 1, 1},
  {&__pyx_kp_u_atanh_line_1847, __pyx_k_atanh_line_1847, sizeof(__pyx_k_atanh_line_1847), 0, 1, 0, 0},
  {&__pyx_kp_u_cga3_line_1873, __pyx_k_cga3_line_1873, sizeof(__pyx_k_cga3_line_1873), 0, 1, 0, 0},
  {&__pyx_kp_u_cga3std_line_1882, __pyx_k_cga3std_line_1882, sizeof(__pyx_k_cga3std_line_1882), 0, 1,
0, 0},
  {&__pyx_n_s_cl, __pyx_k_cl, sizeof(__pyx_k_cl), 0, 0, 1, 1},
  {&__pyx_n_s_clifford, __pyx_k_clifford, sizeof(__pyx_k_clifford), 0, 0, 1, 1},
  {&__pyx_kp_u_clifford___add___line_740, __pyx_k_clifford___add___line_740,
sizeof(__pyx_k_clifford___add___line_740), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___and___line_836, __pyx_k_clifford___and___line_836,
sizeof(__pyx_k_clifford___and___line_836), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___call___line_1020, __pyx_k_clifford___call___line_1020,
sizeof(__pyx_k_clifford___call___line_1020), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___getitem___line_707, __pyx_k_clifford___getitem___line_707,
sizeof(__pyx_k_clifford___getitem___line_707), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___iadd___line_751, __pyx_k_clifford___iadd___line_751,
sizeof(__pyx_k_clifford___iadd___line_751), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___iand___line_851, __pyx_k_clifford___iand___line_851,
```

```
sizeof(__pyx_k_clifford___iand___line_851), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___idiv___line_911, __pyx_k_clifford___idiv___line_911,
sizeof(__pyx_k_clifford___idiv___line_911), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___imod___line_821, __pyx_k_clifford___imod___line_821,
sizeof(__pyx_k_clifford___imod___line_821), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___imul___line_793, __pyx_k_clifford___imul___line_793,
sizeof(__pyx_k_clifford___imul___line_793), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___ior___line_950, __pyx_k_clifford___ior___line_950,
sizeof(__pyx_k_clifford___ior___line_950), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___isub___line_771, __pyx_k_clifford___isub___line_771,
sizeof(__pyx_k_clifford___isub___line_771), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___iter___line_638, __pyx_k_clifford___iter___line_638,
sizeof(__pyx_k_clifford___iter___line_638), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___ixor___line_881, __pyx_k_clifford___ixor___line_881,
sizeof(__pyx_k_clifford___ixor___line_881), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___mod___line_806, __pyx_k_clifford___mod___line_806,
sizeof(__pyx_k_clifford___mod___line_806), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___mul___line_780, __pyx_k_clifford___mul___line_780,
sizeof(__pyx_k_clifford___mul___line_780), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___neg___line_722, __pyx_k_clifford___neg___line_722,
sizeof(__pyx_k_clifford___neg___line_722), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___or___line_939, __pyx_k_clifford___or___line_939,
sizeof(__pyx_k_clifford___or___line_939), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___pos___line_731, __pyx_k_clifford___pos___line_731,
sizeof(__pyx_k_clifford___pos___line_731), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___pow___line_961, __pyx_k_clifford___pow___line_961,
sizeof(__pyx_k_clifford___pow___line_961), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___repr___line_1235, __pyx_k_clifford___repr___line_1235,
sizeof(__pyx_k_clifford___repr___line_1235), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___str___line_1244, __pyx_k_clifford___str___line_1244,
sizeof(__pyx_k_clifford___str___line_1244), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___sub___line_760, __pyx_k_clifford___sub___line_760,
sizeof(__pyx_k_clifford___sub___line_760), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___truediv___line_896, __pyx_k_clifford___truediv___line_896,
sizeof(__pyx_k_clifford___truediv___line_896), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford___xor___line_866, __pyx_k_clifford___xor___line_866,
sizeof(__pyx_k_clifford___xor___line_866), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_abs_line_1175, __pyx_k_clifford_abs_line_1175,
sizeof(__pyx_k_clifford_abs_line_1175), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_conj_line_1138, __pyx_k_clifford_conj_line_1138,
sizeof(__pyx_k_clifford_conj_line_1138), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_copy_line_556, __pyx_k_clifford_copy_line_556,
sizeof(__pyx_k_clifford_copy_line_556), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_even_line_1061, __pyx_k_clifford_even_line_1061,
sizeof(__pyx_k_clifford_even_line_1061), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_frame_line_1224, __pyx_k_clifford_frame_line_1224,
sizeof(__pyx_k_clifford_frame_line_1224), 0, 1, 0, 0},
  {&__pyx_n_s_clifford_hidden_doctests, __pyx_k_clifford_hidden_doctests,
sizeof(__pyx_k_clifford_hidden_doctests), 0, 0, 1, 1},
  {&__pyx_kp_u_clifford_hidden_doctests_line_12, __pyx_k_clifford_hidden_doctests_line_12,
sizeof(__pyx_k_clifford_hidden_doctests_line_12), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_inv_line_926, __pyx_k_clifford_inv_line_926,
sizeof(__pyx_k_clifford_inv_line_926), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_involute_line_1107, __pyx_k_clifford_involute_line_1107,
sizeof(__pyx_k_clifford_involute_line_1107), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_isinf_line_1206, __pyx_k_clifford_isinf_line_1206,
sizeof(__pyx_k_clifford_isinf_line_1206), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_isnan_line_1215, __pyx_k_clifford_isnan_line_1215,
sizeof(__pyx_k_clifford_isnan_line_1215), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_max_abs_line_1184, __pyx_k_clifford_max_abs_line_1184,
sizeof(__pyx_k_clifford_max_abs_line_1184), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_norm_line_1164, __pyx_k_clifford_norm_line_1164,
sizeof(__pyx_k_clifford_norm_line_1164), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_odd_line_1070, __pyx_k_clifford_odd_line_1070,
sizeof(__pyx_k_clifford_odd_line_1070), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_outer_pow_line_1004, __pyx_k_clifford_outer_pow_line_1004,
sizeof(__pyx_k_clifford_outer_pow_line_1004), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_pow_line_980, __pyx_k_clifford_pow_line_980,
sizeof(__pyx_k_clifford_pow_line_980), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_pure_line_1050, __pyx_k_clifford_pure_line_1050,
sizeof(__pyx_k_clifford_pure_line_1050), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_quad_line_1153, __pyx_k_clifford_quad_line_1153,
sizeof(__pyx_k_clifford_quad_line_1153), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_reframe_line_649, __pyx_k_clifford_reframe_line_649,
sizeof(__pyx_k_clifford_reframe_line_649), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_reverse_line_1123, __pyx_k_clifford_reverse_line_1123,
sizeof(__pyx_k_clifford_reverse_line_1123), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_scalar_line_1039, __pyx_k_clifford_scalar_line_1039,
sizeof(__pyx_k_clifford_scalar_line_1039), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_truncated_line_1195, __pyx_k_clifford_truncated_line_1195,
sizeof(__pyx_k_clifford_truncated_line_1195), 0, 1, 0, 0},
  {&__pyx_kp_u_clifford_vector_part_line_1079, __pyx_k_clifford_vector_part_line_1079,
sizeof(__pyx_k_clifford_vector_part_line_1079), 0, 1, 0, 0},
  {&__pyx_n_s_cline_in_traceback, __pyx_k_cline_in_traceback, sizeof(__pyx_k_cline_in_traceback), 0,
0, 1, 1},
  {&__pyx_n_s_close, __pyx_k_close, sizeof(__pyx_k_close), 0, 0, 1, 1},
  {&__pyx_n_s_collections, __pyx_k_collections, sizeof(__pyx_k_collections), 0, 0, 1, 1},
```

```
{&__pyx_kp_u_compare_line_492, __pyx_k_compare_line_492, sizeof(__pyx_k_compare_line_492), 0, 1, 0, 0},
{&__pyx_kp_u_complexifier_line_1576, __pyx_k_complexifier_line_1576, sizeof(__pyx_k_complexifier_line_1576), 0, 1, 0, 0},
{&__pyx_n_s_conj, __pyx_k_conj, sizeof(__pyx_k_conj), 0, 0, 1, 1},
{&__pyx_kp_u_conj_line_1485, __pyx_k_conj_line_1485, sizeof(__pyx_k_conj_line_1485), 0, 1, 0, 0},
{&__pyx_n_s_copy, __pyx_k_copy, sizeof(__pyx_k_copy), 0, 0, 1, 1},
{&__pyx_n_s_cos, __pyx_k_cos, sizeof(__pyx_k_cos), 0, 0, 1, 1},
{&__pyx_kp_u_cos_line_1651, __pyx_k_cos_line_1651, sizeof(__pyx_k_cos_line_1651), 0, 1, 0, 0},
{&__pyx_n_s_cosh, __pyx_k_cosh, sizeof(__pyx_k_cosh), 0, 0, 1, 1},
{&__pyx_kp_u_cosh_line_1689, __pyx_k_cosh_line_1689, sizeof(__pyx_k_cosh_line_1689), 0, 1, 0, 0},
{&__pyx_n_s_doctest, __pyx_k_doctest, sizeof(__pyx_k_doctest), 0, 0, 1, 1},
{&__pyx_n_s_e, __pyx_k_e, sizeof(__pyx_k_e), 0, 0, 1, 1},
{&__pyx_kp_u_e_line_1936, __pyx_k_e_line_1936, sizeof(__pyx_k_e_line_1936), 0, 1, 0, 0},
{&__pyx_n_s_encode, __pyx_k_encode, sizeof(__pyx_k_encode), 0, 0, 1, 1},
{&__pyx_kp_u_error_squared_line_1346, __pyx_k_error_squared_line_1346, sizeof(__pyx_k_error_squared_line_1346), 0, 1, 0, 0},
{&__pyx_kp_u_error_squared_tol_line_1337, __pyx_k_error_squared_tol_line_1337, sizeof(__pyx_k_error_squared_tol_line_1337), 0, 1, 0, 0},
{&__pyx_n_s_even, __pyx_k_even, sizeof(__pyx_k_even), 0, 0, 1, 1},
{&__pyx_kp_u_even_line_1437, __pyx_k_even_line_1437, sizeof(__pyx_k_even_line_1437), 0, 1, 0, 0},
{&__pyx_n_s_exp, __pyx_k_exp, sizeof(__pyx_k_exp), 0, 0, 1, 1},
{&__pyx_kp_u_exp_line_1614, __pyx_k_exp_line_1614, sizeof(__pyx_k_exp_line_1614), 0, 1, 0, 0},
{&__pyx_n_s_fill, __pyx_k_fill, sizeof(__pyx_k_fill), 0, 0, 1, 1},
{&__pyx_n_s_frm, __pyx_k_frm, sizeof(__pyx_k_frm), 0, 0, 1, 1},
{&__pyx_kp_u_from, __pyx_k_from, sizeof(__pyx_k_from), 0, 1, 0, 0},
{&__pyx_n_s_getstate, __pyx_k_getstate, sizeof(__pyx_k_getstate), 0, 0, 1, 1},
{&__pyx_n_s_grade, __pyx_k_grade, sizeof(__pyx_k_grade), 0, 0, 1, 1},
{&__pyx_n_s_i, __pyx_k_i, sizeof(__pyx_k_i), 0, 0, 1, 1},
{&__pyx_kp_u_imag_line_1415, __pyx_k_imag_line_1415, sizeof(__pyx_k_imag_line_1415), 0, 1, 0, 0},
{&__pyx_n_s_import, __pyx_k_import, sizeof(__pyx_k_import), 0, 0, 1, 1},
{&__pyx_n_s_index_set, __pyx_k_index_set, sizeof(__pyx_k_index_set), 0, 0, 1, 1},
{&__pyx_kp_u_index_set___and___line_271, __pyx_k_index_set___and___line_271, sizeof(__pyx_k_index_set___and___line_271), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___getitem___line_191, __pyx_k_index_set___getitem___line_191, sizeof(__pyx_k_index_set___getitem___line_191), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___iand___line_282, __pyx_k_index_set___iand___line_282, sizeof(__pyx_k_index_set___iand___line_282), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___invert___line_240, __pyx_k_index_set___invert___line_240, sizeof(__pyx_k_index_set___invert___line_240), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___ior___line_304, __pyx_k_index_set___ior___line_304, sizeof(__pyx_k_index_set___ior___line_304), 0, 1, 0, 0},
{&__pyx_n_s_index_set___iter, __pyx_k_index_set___iter, sizeof(__pyx_k_index_set___iter), 0, 0, 1, 1},
{&__pyx_kp_u_index_set___iter___line_229, __pyx_k_index_set___iter___line_229, sizeof(__pyx_k_index_set___iter___line_229), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___ixor___line_260, __pyx_k_index_set___ixor___line_260, sizeof(__pyx_k_index_set___ixor___line_260), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___or___line_293, __pyx_k_index_set___or___line_293, sizeof(__pyx_k_index_set___or___line_293), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___repr___line_384, __pyx_k_index_set___repr___line_384, sizeof(__pyx_k_index_set___repr___line_384), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___setitem___line_179, __pyx_k_index_set___setitem___line_179, sizeof(__pyx_k_index_set___setitem___line_179), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___str___line_395, __pyx_k_index_set___str___line_395, sizeof(__pyx_k_index_set___str___line_395), 0, 1, 0, 0},
{&__pyx_kp_u_index_set___xor___line_249, __pyx_k_index_set___xor___line_249, sizeof(__pyx_k_index_set___xor___line_249), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_copy_line_65, __pyx_k_index_set_copy_line_65, sizeof(__pyx_k_index_set_copy_line_65), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_count_line_315, __pyx_k_index_set_count_line_315, sizeof(__pyx_k_index_set_count_line_315), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_count_neg_line_324, __pyx_k_index_set_count_neg_line_324, sizeof(__pyx_k_index_set_count_neg_line_324), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_count_pos_line_333, __pyx_k_index_set_count_pos_line_333, sizeof(__pyx_k_index_set_count_pos_line_333), 0, 1, 0, 0},
{&__pyx_n_s_index_set_hidden_doctests, __pyx_k_index_set_hidden_doctests, sizeof(__pyx_k_index_set_hidden_doctests), 0, 0, 1, 1},
{&__pyx_kp_u_index_set_hidden_doctests_line_4, __pyx_k_index_set_hidden_doctests_line_4, sizeof(__pyx_k_index_set_hidden_doctests_line_4), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_max_line_351, __pyx_k_index_set_max_line_351, sizeof(__pyx_k_index_set_max_line_351), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_min_line_342, __pyx_k_index_set_min_line_342, sizeof(__pyx_k_index_set_min_line_342), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_sign_of_mult_line_366, __pyx_k_index_set_sign_of_mult_line_366, sizeof(__pyx_k_index_set_sign_of_mult_line_366), 0, 1, 0, 0},
{&__pyx_kp_u_index_set_sign_of_square_line_37, __pyx_k_index_set_sign_of_square_line_37, sizeof(__pyx_k_index_set_sign_of_square_line_37), 0, 1, 0, 0},
{&__pyx_n_s_inv, __pyx_k_inv, sizeof(__pyx_k_inv), 0, 0, 1, 1},
{&__pyx_kp_u_inv_line_1378, __pyx_k_inv_line_1378, sizeof(__pyx_k_inv_line_1378), 0, 1, 0, 0},
{&__pyx_kp_u_invalid, __pyx_k_invalid, sizeof(__pyx_k_invalid), 0, 1, 0, 0},
{&__pyx_kp_u_invalid_string, __pyx_k_invalid_string, sizeof(__pyx_k_invalid_string), 0, 1, 0, 0},
{&__pyx_n_s_involute, __pyx_k_involute, sizeof(__pyx_k_involute), 0, 0, 1, 1},
{&__pyx_kp_u_involute_line_1455, __pyx_k_involute_line_1455, sizeof(__pyx_k_involute_line_1455), 0, 1, 0, 0},
{&__pyx_n_s_ist, __pyx_k_ist, sizeof(__pyx_k_ist), 0, 0, 1, 1},
{&__pyx_n_s_istpq, __pyx_k_istpq, sizeof(__pyx_k_istpq), 0, 0, 1, 1},
```

```
{&__pyx_kp_u_istpq_line_1949, __pyx_k_istpq_line_1949, sizeof(__pyx_k_istpq_line_1949), 0, 1, 0, 0},
{&__pyx_n_s_iter, __pyx_k_iter, sizeof(__pyx_k_iter), 0, 0, 1, 1},
{&__pyx_n_s_ixt, __pyx_k_ixt, sizeof(__pyx_k_ixt), 0, 0, 1, 1},
{&__pyx_kp_u_lexicographic_compare_eg_3_4_5, __pyx_k_lexicographic_compare_eg_3_4_5, sizeof(__pyx_k_lexicographic_compare_eg_3_4_5), 0, 1, 0, 0},
{&__pyx_n_s_lhs, __pyx_k_lhs, sizeof(__pyx_k_lhs), 0, 0, 1, 1},
{&__pyx_n_s_log, __pyx_k_log, sizeof(__pyx_k_log), 0, 0, 1, 1},
{&__pyx_kp_u_log_line_1628, __pyx_k_log_line_1628, sizeof(__pyx_k_log_line_1628), 0, 1, 0, 0},
{&__pyx_n_s_m, __pyx_k_m, sizeof(__pyx_k_m), 0, 0, 1, 1},
{&__pyx_n_s_main, __pyx_k_main, sizeof(__pyx_k_main), 0, 0, 1, 1},
{&__pyx_n_u_main, __pyx_k_main, sizeof(__pyx_k_main), 0, 1, 0, 1},
{&__pyx_n_s_math, __pyx_k_math, sizeof(__pyx_k_math), 0, 0, 1, 1},
{&__pyx_n_s_max, __pyx_k_max, sizeof(__pyx_k_max), 0, 0, 1, 1},
{&__pyx_kp_u_max_abs_line_1531, __pyx_k_max_abs_line_1531, sizeof(__pyx_k_max_abs_line_1531), 0, 1, 0, 0},
{&__pyx_kp_u_max_pos_line_513, __pyx_k_max_pos_line_513, sizeof(__pyx_k_max_pos_line_513), 0, 1, 0, 0},
{&__pyx_n_s_min, __pyx_k_min, sizeof(__pyx_k_min), 0, 0, 1, 1},
{&__pyx_kp_u_min_neg_line_504, __pyx_k_min_neg_line_504, sizeof(__pyx_k_min_neg_line_504), 0, 1, 0, 0},
{&__pyx_n_s_name, __pyx_k_name, sizeof(__pyx_k_name), 0, 0, 1, 1},
{&__pyx_n_s_nbar3, __pyx_k_nbar3, sizeof(__pyx_k_nbar3), 0, 0, 1, 1},
{&__pyx_n_s_ninf3, __pyx_k_ninf3, sizeof(__pyx_k_ninf3), 0, 0, 1, 1},
{&__pyx_kp_s_no_default___reduce___due_to_non, __pyx_k_no_default___reduce___due_to_non, sizeof(__pyx_k_no_default___reduce___due_to_non), 0, 0, 1, 0},
{&__pyx_n_s_norm, __pyx_k_norm, sizeof(__pyx_k_norm), 0, 0, 1, 1},
{&__pyx_kp_u_norm_line_1511, __pyx_k_norm_line_1511, sizeof(__pyx_k_norm_line_1511), 0, 1, 0, 0},
{&__pyx_kp_u_norm_sum_of_squares_of_coordina, __pyx_k_norm_sum_of_squares_of_coordina, sizeof(__pyx_k_norm_sum_of_squares_of_coordina), 0, 1, 0, 0},
{&__pyx_n_s_numbers, __pyx_k_numbers, sizeof(__pyx_k_numbers), 0, 0, 1, 1},
{&__pyx_n_s_obj, __pyx_k_obj, sizeof(__pyx_k_obj), 0, 0, 1, 1},
{&__pyx_n_s_odd, __pyx_k_odd, sizeof(__pyx_k_odd), 0, 0, 1, 1},
{&__pyx_kp_u_odd_line_1446, __pyx_k_odd_line_1446, sizeof(__pyx_k_odd_line_1446), 0, 1, 0, 0},
{&__pyx_n_s_other, __pyx_k_other, sizeof(__pyx_k_other), 0, 0, 1, 1},
{&__pyx_n_s_outer_pow, __pyx_k_outer_pow, sizeof(__pyx_k_outer_pow), 0, 0, 1, 1},
{&__pyx_kp_u_outer_pow_line_1567, __pyx_k_outer_pow_line_1567, sizeof(__pyx_k_outer_pow_line_1567), 0, 1, 0, 0},
{&__pyx_n_s_p, __pyx_k_p, sizeof(__pyx_k_p), 0, 0, 1, 1},
{&__pyx_n_s_pi, __pyx_k_pi, sizeof(__pyx_k_pi), 0, 0, 1, 1},
{&__pyx_n_s_pow, __pyx_k_pow, sizeof(__pyx_k_pow), 0, 0, 1, 1},
{&__pyx_kp_u_pow_line_1543, __pyx_k_pow_line_1543, sizeof(__pyx_k_pow_line_1543), 0, 1, 0, 0},
{&__pyx_n_s_pure, __pyx_k_pure, sizeof(__pyx_k_pure), 0, 0, 1, 1},
{&__pyx_kp_u_pure_line_1426, __pyx_k_pure_line_1426, sizeof(__pyx_k_pure_line_1426), 0, 1, 0, 0},
{&__pyx_n_s_pyx_vtable, __pyx_k_pyx_vtable, sizeof(__pyx_k_pyx_vtable), 0, 0, 1, 1},
{&__pyx_n_s_q, __pyx_k_q, sizeof(__pyx_k_q), 0, 0, 1, 1},
{&__pyx_n_s_quad, __pyx_k_quad, sizeof(__pyx_k_quad), 0, 0, 1, 1},
{&__pyx_kp_u_quad_line_1500, __pyx_k_quad_line_1500, sizeof(__pyx_k_quad_line_1500), 0, 1, 0, 0},
{&__pyx_kp_u_random_clifford_line_1864, __pyx_k_random_clifford_line_1864, sizeof(__pyx_k_random_clifford_line_1864), 0, 1, 0, 0},
{&__pyx_n_s_range, __pyx_k_range, sizeof(__pyx_k_range), 0, 0, 1, 1},
{&__pyx_kp_u_real_line_1404, __pyx_k_real_line_1404, sizeof(__pyx_k_real_line_1404), 0, 1, 0, 0},
{&__pyx_n_s_reduce, __pyx_k_reduce, sizeof(__pyx_k_reduce), 0, 0, 1, 1},
{&__pyx_n_s_reduce_cython, __pyx_k_reduce_cython, sizeof(__pyx_k_reduce_cython), 0, 0, 1, 1},
{&__pyx_n_s_reduce_ex, __pyx_k_reduce_ex, sizeof(__pyx_k_reduce_ex), 0, 0, 1, 1},
{&__pyx_n_s_reverse, __pyx_k_reverse, sizeof(__pyx_k_reverse), 0, 0, 1, 1},
{&__pyx_kp_u_reverse_line_1470, __pyx_k_reverse_line_1470, sizeof(__pyx_k_reverse_line_1470), 0, 1, 0, 0},
{&__pyx_n_s_rhs, __pyx_k_rhs, sizeof(__pyx_k_rhs), 0, 0, 1, 1},
{&__pyx_n_s_scalar, __pyx_k_scalar, sizeof(__pyx_k_scalar), 0, 0, 1, 1},
{&__pyx_n_s_scalar_epsilon, __pyx_k_scalar_epsilon, sizeof(__pyx_k_scalar_epsilon), 0, 0, 1, 1},
{&__pyx_kp_u_scalar_line_1393, __pyx_k_scalar_line_1393, sizeof(__pyx_k_scalar_line_1393), 0, 1, 0, 0},
{&__pyx_n_s_send, __pyx_k_send, sizeof(__pyx_k_send), 0, 0, 1, 1},
{&__pyx_n_s_setstate, __pyx_k_setstate, sizeof(__pyx_k_setstate), 0, 0, 1, 1},
{&__pyx_n_s_setstate_cython, __pyx_k_setstate_cython, sizeof(__pyx_k_setstate_cython), 0, 0, 1, 1},
{&__pyx_n_s_sin, __pyx_k_sin, sizeof(__pyx_k_sin), 0, 0, 1, 1},
{&__pyx_kp_u_sin_line_1728, __pyx_k_sin_line_1728, sizeof(__pyx_k_sin_line_1728), 0, 1, 0, 0},
{&__pyx_n_s_sinh, __pyx_k_sinh, sizeof(__pyx_k_sinh), 0, 0, 1, 1},
{&__pyx_kp_u_sinh_line_1768, __pyx_k_sinh_line_1768, sizeof(__pyx_k_sinh_line_1768), 0, 1, 0, 0},
{&__pyx_n_s_sqrt, __pyx_k_sqrt, sizeof(__pyx_k_sqrt), 0, 0, 1, 1},
{&__pyx_kp_u_sqrt_line_1591, __pyx_k_sqrt_line_1591, sizeof(__pyx_k_sqrt_line_1591), 0, 1, 0, 0},
{&__pyx_n_s_tan, __pyx_k_tan, sizeof(__pyx_k_tan), 0, 0, 1, 1},
{&__pyx_kp_u_tan_line_1801, __pyx_k_tan_line_1801, sizeof(__pyx_k_tan_line_1801), 0, 1, 0, 0},
{&__pyx_n_s_tanh, __pyx_k_tanh, sizeof(__pyx_k_tanh), 0, 0, 1, 1},
{&__pyx_kp_u_tanh_line_1835, __pyx_k_tanh_line_1835, sizeof(__pyx_k_tanh_line_1835), 0, 1, 0, 0},
{&__pyx_n_s_tau, __pyx_k_tau, sizeof(__pyx_k_tau), 0, 0, 1, 1},
{&__pyx_n_s_test, __pyx_k_test, sizeof(__pyx_k_test), 0, 0, 1, 1},
{&__pyx_n_s_test_2, __pyx_k_test_2, sizeof(__pyx_k_test_2), 0, 0, 1, 1},
{&__pyx_n_s_testmod, __pyx_k_testmod, sizeof(__pyx_k_testmod), 0, 0, 1, 1},
{&__pyx_n_s_threshold, __pyx_k_threshold, sizeof(__pyx_k_threshold), 0, 0, 1, 1},
{&__pyx_n_s_throw, __pyx_k_throw, sizeof(__pyx_k_throw), 0, 0, 1, 1},
{&__pyx_kp_u_to_frame, __pyx_k_to_frame, sizeof(__pyx_k_to_frame), 0, 1, 0, 0},
{&__pyx_n_s_tol, __pyx_k_tol, sizeof(__pyx_k_tol), 0, 0, 1, 1},
{&__pyx_kp_u_using, __pyx_k_using, sizeof(__pyx_k_using), 0, 1, 0, 0},
{&__pyx_kp_u_using_invalid, __pyx_k_using_invalid, sizeof(__pyx_k_using_invalid), 0, 1, 0, 0},
{&__pyx_kp_u_utf_8, __pyx_k_utf_8, sizeof(__pyx_k_utf_8), 0, 1, 0, 0},
{&__pyx_kp_u_value, __pyx_k_value, sizeof(__pyx_k_value), 0, 1, 0, 0},
```

```
  {&__pyx_n_s_version, __pyx_k_version, sizeof(__pyx_k_version), 0, 0, 1, 1},
  {&__pyx_n_s_xrange, __pyx_k_xrange, sizeof(__pyx_k_xrange), 0, 0, 1, 1},
  {0, 0, 0, 0, 0, 0, 0}
};
static CYTHON_SMALL_CODE int __Pyx_InitCachedBuiltins(void) {
  __pyx_builtin_IndexError = __Pyx_GetBuiltinName(__pyx_n_s_IndexError); if (!__pyx_builtin_IndexError) __PYX_ERR(0, 103, __pyx_L1_error)
  __pyx_builtin_RuntimeError = __Pyx_GetBuiltinName(__pyx_n_s_RuntimeError); if (!__pyx_builtin_RuntimeError) __PYX_ERR(0, 105, __pyx_L1_error)
  __pyx_builtin_TypeError = __Pyx_GetBuiltinName(__pyx_n_s_TypeError); if (!__pyx_builtin_TypeError) __PYX_ERR(0, 105, __pyx_L1_error)
  __pyx_builtin_ValueError = __Pyx_GetBuiltinName(__pyx_n_s_ValueError); if (!__pyx_builtin_ValueError) __PYX_ERR(0, 106, __pyx_L1_error)
  __pyx_builtin_NotImplemented = __Pyx_GetBuiltinName(__pyx_n_s_NotImplemented); if (!__pyx_builtin_NotImplemented) __PYX_ERR(0, 159, __pyx_L1_error)
  __pyx_builtin_range = __Pyx_GetBuiltinName(__pyx_n_s_range); if (!__pyx_builtin_range) __PYX_ERR(0, 236, __pyx_L1_error)
  #if PY_MAJOR_VERSION >= 3
  __pyx_builtin_xrange = __Pyx_GetBuiltinName(__pyx_n_s_range); if (!__pyx_builtin_xrange) __PYX_ERR(0, 1099, __pyx_L1_error)
  #else
  __pyx_builtin_xrange = __Pyx_GetBuiltinName(__pyx_n_s_xrange); if (!__pyx_builtin_xrange) __PYX_ERR(0, 1099, __pyx_L1_error)
  #endif
  return 0;
  __pyx_L1_error:;
  return -1;
}

static CYTHON_SMALL_CODE int __Pyx_InitCachedConstants(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_InitCachedConstants", 0);

  /* "(tree fragment)":2
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # <<<<<<<<<<<<<<
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */
  __pyx_tuple__3 = PyTuple_Pack(1, __pyx_kp_s_no_default___reduce___due_to_non); if (unlikely(!__pyx_tuple__3)) __PYX_ERR(1, 2, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__3);
  __Pyx_GIVEREF(__pyx_tuple__3);

  /* "(tree fragment)":4
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # <<<<<<<<<<<<<<
 */
  __pyx_tuple__4 = PyTuple_Pack(1, __pyx_kp_s_no_default___reduce___due_to_non); if (unlikely(!__pyx_tuple__4)) __PYX_ERR(1, 4, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__4);
  __Pyx_GIVEREF(__pyx_tuple__4);

  /* "PyClical.pyx":636
 *         TypeError: Not applicable.
 *         """
 *         raise TypeError("Not applicable.")             # <<<<<<<<<<<<<<
 *
 *     def __iter__(self):
 */
  __pyx_tuple__10 = PyTuple_Pack(1, __pyx_kp_u_Not_applicable); if (unlikely(!__pyx_tuple__10)) __PYX_ERR(0, 636, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__10);
  __Pyx_GIVEREF(__pyx_tuple__10);

  /* "(tree fragment)":2
 * def __reduce_cython__(self):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # <<<<<<<<<<<<<<
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 */
  __pyx_tuple__11 = PyTuple_Pack(1, __pyx_kp_s_no_default___reduce___due_to_non); if (unlikely(!__pyx_tuple__11)) __PYX_ERR(1, 2, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__11);
  __Pyx_GIVEREF(__pyx_tuple__11);

  /* "(tree fragment)":4
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")
 * def __setstate_cython__(self, __pyx_state):
 *     raise TypeError("no default __reduce__ due to non-trivial __cinit__")             # <<<<<<<<<<<<<<
 */
  __pyx_tuple__12 = PyTuple_Pack(1, __pyx_kp_s_no_default___reduce___due_to_non); if (unlikely(!__pyx_tuple__12)) __PYX_ERR(1, 4, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__12);
  __Pyx_GIVEREF(__pyx_tuple__12);
```

```
  /* "PyClical.pyx":406
 *         return index_set_to_str( self.unwrap() ).decode()
 *
 * def index_set_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */
  __pyx_codeobj__13 = (PyObject*)__Pyx_PyCode_New(0, 0, 0, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_index_set_hidden_doctests, 406, __pyx_empty_bytes); if (unlikely(!__pyx_codeobj__13)) __PYX_ERR(0, 406, __pyx_L1_error)

  /* "PyClical.pyx":1253
 *         return clifford_to_str( self.unwrap() ).decode()
 *
 * def clifford_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */
  __pyx_codeobj__14 = (PyObject*)__Pyx_PyCode_New(0, 0, 0, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_clifford_hidden_doctests, 1253, __pyx_empty_bytes); if (unlikely(!__pyx_codeobj__14)) __PYX_ERR(0, 1253, __pyx_L1_error)

  /* "PyClical.pyx":1907
 * scalar_epsilon = epsilon
 *
 * pi = atan(clifford(1.0)) * 4.0             # «««««««
 * tau = atan(clifford(1.0)) * 8.0
 *
 */
  __pyx_tuple__15 = PyTuple_Pack(1, __pyx_float_1_0); if (unlikely(!__pyx_tuple__15)) __PYX_ERR(0, 1907, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__15);
  __Pyx_GIVEREF(__pyx_tuple__15);

  /* "PyClical.pyx":1936
 * """
 *
 * def e(obj):             # «««««««
 *     """
 *     Abbreviation for clifford(index_set(obj)).
 */
  __pyx_tuple__16 = PyTuple_Pack(1, __pyx_n_s_obj); if (unlikely(!__pyx_tuple__16)) __PYX_ERR(0, 1936, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__16);
  __Pyx_GIVEREF(__pyx_tuple__16);
  __pyx_codeobj__17 = (PyObject*)__Pyx_PyCode_New(1, 0, 1, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__16, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_e, 1936, __pyx_empty_bytes); if (unlikely(!__pyx_codeobj__17)) __PYX_ERR(0, 1936, __pyx_L1_error)

  /* "PyClical.pyx":1949
 *     return clifford(index_set(obj))
 *
 * def istpq(p, q):             # «««««««
 *     """
 *     Abbreviation for index_set({-q,...p}).
 */
  __pyx_tuple__18 = PyTuple_Pack(2, __pyx_n_s_p, __pyx_n_s_q); if (unlikely(!__pyx_tuple__18)) __PYX_ERR(0, 1949, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__18);
  __Pyx_GIVEREF(__pyx_tuple__18);
  __pyx_codeobj__19 = (PyObject*)__Pyx_PyCode_New(2, 0, 2, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__18, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_istpq, 1949, __pyx_empty_bytes); if (unlikely(!__pyx_codeobj__19)) __PYX_ERR(0, 1949, __pyx_L1_error)

  /* "PyClical.pyx":1958
 *     return index_set(set(range(-q,p+1)))
 *
 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].             # «««««««
 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].
 *
 */
  __pyx_tuple__20 = PyTuple_Pack(1, __pyx_int_4); if (unlikely(!__pyx_tuple__20)) __PYX_ERR(0, 1958, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__20);
  __Pyx_GIVEREF(__pyx_tuple__20);
  __pyx_tuple__21 = PyTuple_Pack(1, __pyx_int_neg_1); if (unlikely(!__pyx_tuple__21)) __PYX_ERR(0, 1958, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__21);
  __Pyx_GIVEREF(__pyx_tuple__21);

  /* "PyClical.pyx":1962
 *
```

```
 * # Doctest interface.
 * def _test():            # «««««««
 *     import PyClical, doctest
 *     return doctest.testmod(PyClical)
 */
  __pyx_tuple__22 = PyTuple_Pack(2, __pyx_n_s_PyClical, __pyx_n_s_doctest); if (unlikely(!__pyx_tuple__22)) __PYX_ERR(0, 1962, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__22);
  __Pyx_GIVEREF(__pyx_tuple__22);
  __pyx_codeobj__23 = (PyObject*)__Pyx_PyCode_New(0, 0, 2, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__22, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_test, 1962, __pyx_empty_bytes); if (unlikely(!__pyx_codeobj__23)) __PYX_ERR(0, 1962, __pyx_L1_error)
  __Pyx_RefNannyFinishContext();
  return 0;
  __pyx_L1_error:;
  __Pyx_RefNannyFinishContext();
  return -1;
}

static CYTHON_SMALL_CODE int __Pyx_InitGlobals(void) {
  if (__Pyx_InitStrings(__pyx_string_tab) < 0) __PYX_ERR(0, 1, __pyx_L1_error);
  __pyx_float_0_0 = PyFloat_FromDouble(0.0); if (unlikely(!__pyx_float_0_0)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_float_1_0 = PyFloat_FromDouble(1.0); if (unlikely(!__pyx_float_1_0)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_float_4_0 = PyFloat_FromDouble(4.0); if (unlikely(!__pyx_float_4_0)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_float_8_0 = PyFloat_FromDouble(8.0); if (unlikely(!__pyx_float_8_0)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_int_0 = PyInt_FromLong(0); if (unlikely(!__pyx_int_0)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_int_1 = PyInt_FromLong(1); if (unlikely(!__pyx_int_1)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_int_4 = PyInt_FromLong(4); if (unlikely(!__pyx_int_4)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_int_neg_1 = PyInt_FromLong(-1); if (unlikely(!__pyx_int_neg_1)) __PYX_ERR(0, 1, __pyx_L1_error)
  return 0;
  __pyx_L1_error:;
  return -1;
}

static CYTHON_SMALL_CODE int __Pyx_modinit_global_init_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_variable_export_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_function_export_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_type_init_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_type_import_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_variable_import_code(void); /*proto*/
static CYTHON_SMALL_CODE int __Pyx_modinit_function_import_code(void); /*proto*/

static int __Pyx_modinit_global_init_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_global_init_code", 0);
  /*--- Global init code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}

static int __Pyx_modinit_variable_export_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_variable_export_code", 0);
  /*--- Variable export code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}

static int __Pyx_modinit_function_export_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_function_export_code", 0);
  /*--- Function export code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}

static int __Pyx_modinit_type_init_code(void) {
  __Pyx_RefNannyDeclarations
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("__Pyx_modinit_type_init_code", 0);
  /*--- Type init code ---*/
  __pyx_vtabptr_8PyClical_index_set = &__pyx_vtable_8PyClical_index_set;
  __pyx_vtable_8PyClical_index_set.wrap = (PyObject *(*)(struct __pyx_obj_8PyClical_index_set *, IndexSet))__pyx_f_8PyClical_9index_set_wrap;
  __pyx_vtable_8PyClical_index_set.unwrap = (IndexSet (*)(struct __pyx_obj_8PyClical_index_set *))__pyx_f_8PyClical_9index_set_unwrap;
  __pyx_vtable_8PyClical_index_set.copy = (PyObject *(*)(struct __pyx_obj_8PyClical_index_set *, int __pyx_skip_dispatch))__pyx_f_8PyClical_9index_set_copy;
  if (PyType_Ready(&__pyx_type_8PyClical_index_set) < 0) __PYX_ERR(0, 46, __pyx_L1_error)
```

```
  #if PY_VERSION_HEX < 0x030800B1
  __pyx_type_8PyClical_index_set.tp_print = 0;
  #endif
  if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) && likely(!__pyx_type_8PyClical_index_set.tp_dictoffset && __pyx_type_8PyClical_index_set.tp_getattro == PyObject_GenericGetAttr)) {
    __pyx_type_8PyClical_index_set.tp_getattro = __Pyx_PyObject_GenericGetAttr;
  }
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__setitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_8__setitem__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_8__setitem__.doc = __pyx_doc_8PyClical_9index_set_8__setitem__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_8__setitem__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__getitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_10__getitem__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_10__getitem__.doc = __pyx_doc_8PyClical_9index_set_10__getitem__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_10__getitem__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__contains__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_12__contains__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_12__contains__.doc = __pyx_doc_8PyClical_9index_set_12__contains__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_12__contains__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__iter__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_14__iter__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_14__iter__.doc = __pyx_doc_8PyClical_9index_set_14__iter__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_14__iter__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__invert__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_17__invert__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_17__invert__.doc = __pyx_doc_8PyClical_9index_set_17__invert__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_17__invert__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__xor__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_9index_set_19__xor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_9index_set_19__xor__.doc = __pyx_doc_8PyClical_9index_set_19__xor__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_19__xor__;
    }
  }
  #endif
```

```
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__ixor__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_21__ixor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_21__ixor__.doc =
__pyx_doc_8PyClical_9index_set_21__ixor__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_21__ixor__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__and__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_23__and__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_23__and__.doc = __pyx_doc_8PyClical_9index_set_23__and__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_23__and__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__iand__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_25__iand__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_25__iand__.doc =
__pyx_doc_8PyClical_9index_set_25__iand__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_25__iand__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__or__");
if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_27__or__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_27__or__.doc = __pyx_doc_8PyClical_9index_set_27__or__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_27__or__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__ior__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_29__ior__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_29__ior__.doc = __pyx_doc_8PyClical_9index_set_29__ior__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_29__ior__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__repr__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_47__repr__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_47__repr__.doc =
__pyx_doc_8PyClical_9index_set_47__repr__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_47__repr__;
  }
}
#endif
#if CYTHON_COMPILING_IN_CPYTHON
{
  PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
"__str__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
  if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
    __pyx_wrapperbase_8PyClical_9index_set_49__str__ = *((PyWrapperDescrObject *)wrapper)->d_base;
    __pyx_wrapperbase_8PyClical_9index_set_49__str__.doc = __pyx_doc_8PyClical_9index_set_49__str__;
    ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_49__str__;
  }
}
#endif
if (__Pyx_SetVtable(__pyx_type_8PyClical_index_set.tp_dict, __pyx_vtabptr_8PyClical_index_set) < 0)
__PYX_ERR(0, 46, __pyx_L1_error)
if (PyObject_SetAttr(__pyx_m, __pyx_n_s_index_set, (PyObject *)&__pyx_type_8PyClical_index_set) < 0)
__PYX_ERR(0, 46, __pyx_L1_error)
if (__Pyx_setup_reduce((PyObject*)&__pyx_type_8PyClical_index_set) < 0) __PYX_ERR(0, 46,
__pyx_L1_error)
__pyx_ptype_8PyClical_index_set = &__pyx_type_8PyClical_index_set;
```

```
  __pyx_vtabptr_8PyClical_clifford = &__pyx_vtable_8PyClical_clifford;
  __pyx_vtable_8PyClical_clifford.wrap = (PyObject *(*)(struct __pyx_obj_8PyClical_clifford *, Clifford))__pyx_f_8PyClical_8clifford_wrap;
  __pyx_vtable_8PyClical_clifford.unwrap = (Clifford (*)(struct __pyx_obj_8PyClical_clifford *))__pyx_f_8PyClical_8clifford_unwrap;
  __pyx_vtable_8PyClical_clifford.copy = (PyObject *(*)(struct __pyx_obj_8PyClical_clifford *, int __pyx_skip_dispatch))__pyx_f_8PyClical_8clifford_copy;
  if (PyType_Ready(&__pyx_type_8PyClical_clifford) < 0) __PYX_ERR(0, 537, __pyx_L1_error)
  #if PY_VERSION_HEX < 0x030800B1
  __pyx_type_8PyClical_clifford.tp_print = 0;
  #endif
  if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) && likely(!__pyx_type_8PyClical_clifford.tp_dictoffset && __pyx_type_8PyClical_clifford.tp_getattro == PyObject_GenericGetAttr)) {
    __pyx_type_8PyClical_clifford.tp_getattro = __Pyx_PyObject_GenericGetAttr;
  }
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__contains__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_6__contains__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_6__contains__.doc = __pyx_doc_8PyClical_8clifford_6__contains__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_6__contains__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__iter__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_8__iter__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_8__iter__.doc = __pyx_doc_8PyClical_8clifford_8__iter__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_8__iter__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__getitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_14__getitem__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_14__getitem__.doc = __pyx_doc_8PyClical_8clifford_14__getitem__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_14__getitem__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__neg__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_16__neg__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_16__neg__.doc = __pyx_doc_8PyClical_8clifford_16__neg__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_16__neg__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__pos__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_18__pos__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_18__pos__.doc = __pyx_doc_8PyClical_8clifford_18__pos__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_18__pos__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__add__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_20__add__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_20__add__.doc = __pyx_doc_8PyClical_8clifford_20__add__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_20__add__;
    }
  }
```

```
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__iadd__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_22__iadd__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_22__iadd__.doc = __pyx_doc_8PyClical_8clifford_22__iadd__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_22__iadd__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__sub__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_24__sub__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_24__sub__.doc = __pyx_doc_8PyClical_8clifford_24__sub__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_24__sub__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__isub__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_26__isub__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_26__isub__.doc = __pyx_doc_8PyClical_8clifford_26__isub__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_26__isub__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__mul__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_28__mul__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_28__mul__.doc = __pyx_doc_8PyClical_8clifford_28__mul__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_28__mul__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__imul__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_30__imul__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_30__imul__.doc = __pyx_doc_8PyClical_8clifford_30__imul__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_30__imul__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__mod__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_32__mod__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_32__mod__.doc = __pyx_doc_8PyClical_8clifford_32__mod__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_32__mod__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__imod__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_34__imod__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_34__imod__.doc = __pyx_doc_8PyClical_8clifford_34__imod__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_34__imod__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__and__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_36__and__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_36__and__.doc = __pyx_doc_8PyClical_8clifford_36__and__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_36__and__;
    }
```

```
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__iand__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_38__iand__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_38__iand__.doc = __pyx_doc_8PyClical_8clifford_38__iand__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_38__iand__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__xor__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_40__xor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_40__xor__.doc = __pyx_doc_8PyClical_8clifford_40__xor__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_40__xor__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__ixor__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_42__ixor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_42__ixor__.doc = __pyx_doc_8PyClical_8clifford_42__ixor__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_42__ixor__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__truediv__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_44__truediv__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_44__truediv__.doc =
__pyx_doc_8PyClical_8clifford_44__truediv__;
      ((PyWrapperDescrObject *)wrapper)->d_base =
&__pyx_wrapperbase_8PyClical_8clifford_44__truediv__;
    }
  }
  #endif
  #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__idiv__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_46__idiv__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_46__idiv__.doc = __pyx_doc_8PyClical_8clifford_46__idiv__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_46__idiv__;
    }
  }
  #endif
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__or__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_50__or__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_50__or__.doc = __pyx_doc_8PyClical_8clifford_50__or__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_50__or__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__ior__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_52__ior__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_52__ior__.doc = __pyx_doc_8PyClical_8clifford_52__ior__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_52__ior__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__pow__");
```

```
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_54__pow__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_54__pow__.doc = __pyx_doc_8PyClical_8clifford_54__pow__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_54__pow__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__call__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_60__call__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_60__call__.doc = __pyx_doc_8PyClical_8clifford_60__call__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_60__call__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__repr__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_94__repr__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_94__repr__.doc = __pyx_doc_8PyClical_8clifford_94__repr__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_94__repr__;
    }
  }
  #endif
  #if CYTHON_COMPILING_IN_CPYTHON
  {
    PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__str__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
    if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
      __pyx_wrapperbase_8PyClical_8clifford_96__str__ = *((PyWrapperDescrObject *)wrapper)->d_base;
      __pyx_wrapperbase_8PyClical_8clifford_96__str__.doc = __pyx_doc_8PyClical_8clifford_96__str__;
      ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_96__str__;
    }
  }
  #endif
  if (__Pyx_SetVtable(__pyx_type_8PyClical_clifford.tp_dict, __pyx_vtabptr_8PyClical_clifford) < 0) __PYX_ERR(0, 537, __pyx_L1_error)
  if (PyObject_SetAttr(__pyx_m, __pyx_n_s_clifford, (PyObject *)&__pyx_type_8PyClical_clifford) < 0) __PYX_ERR(0, 537, __pyx_L1_error)
  if (__Pyx_setup_reduce((PyObject*)&__pyx_type_8PyClical_clifford) < 0) __PYX_ERR(0, 537, __pyx_L1_error)
  __pyx_ptype_8PyClical_clifford = &__pyx_type_8PyClical_clifford;
  if (PyType_Ready(&__pyx_type_8PyClical___pyx_scope_struct____iter__) < 0) __PYX_ERR(0, 229, __pyx_L1_error)
  #if PY_VERSION_HEX < 0x030800B1
  __pyx_type_8PyClical___pyx_scope_struct____iter__.tp_print = 0;
  #endif
  if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) && likely(!__pyx_type_8PyClical___pyx_scope_struct____iter__.tp_dictoffset && __pyx_type_8PyClical___pyx_scope_struct____iter__.tp_getattro == PyObject_GenericGetAttr)) {
    __pyx_type_8PyClical___pyx_scope_struct____iter__.tp_getattro = __Pyx_PyObject_GenericGetAttrNoDict;
  }
  __pyx_ptype_8PyClical___pyx_scope_struct____iter__ = &__pyx_type_8PyClical___pyx_scope_struct____iter__;
  __Pyx_RefNannyFinishContext();
  return 0;
  __pyx_L1_error:;
  __Pyx_RefNannyFinishContext();
  return -1;
}

static int __Pyx_modinit_type_import_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_type_import_code", 0);
  /*--- Type import code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}

static int __Pyx_modinit_variable_import_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_variable_import_code", 0);
  /*--- Variable import code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}

static int __Pyx_modinit_function_import_code(void) {
  __Pyx_RefNannyDeclarations
  __Pyx_RefNannySetupContext("__Pyx_modinit_function_import_code", 0);
```

```
  /*--- Function import code ---*/
  __Pyx_RefNannyFinishContext();
  return 0;
}


#ifndef CYTHON_NO_PYINIT_EXPORT
#define __Pyx_PyMODINIT_FUNC PyMODINIT_FUNC
#elif PY_MAJOR_VERSION < 3
#ifdef __cplusplus
#define __Pyx_PyMODINIT_FUNC extern "C" void
#else
#define __Pyx_PyMODINIT_FUNC void
#endif
#else
#ifdef __cplusplus
#define __Pyx_PyMODINIT_FUNC extern "C" PyObject *
#else
#define __Pyx_PyMODINIT_FUNC PyObject *
#endif
#endif


#if PY_MAJOR_VERSION < 3
__Pyx_PyMODINIT_FUNC initPyClical(void) CYTHON_SMALL_CODE; /*proto*/
__Pyx_PyMODINIT_FUNC initPyClical(void)
#else
__Pyx_PyMODINIT_FUNC PyInit_PyClical(void) CYTHON_SMALL_CODE; /*proto*/
__Pyx_PyMODINIT_FUNC PyInit_PyClical(void)
#if CYTHON_PEP489_MULTI_PHASE_INIT
{
  return PyModuleDef_Init(&__pyx_moduledef);
}
static CYTHON_SMALL_CODE int __Pyx_check_single_interpreter(void) {
    #if PY_VERSION_HEX >= 0x030700A1
    static PY_INT64_T main_interpreter_id = -1;
    PY_INT64_T current_id = PyInterpreterState_GetID(PyThreadState_Get()->interp);
    if (main_interpreter_id == -1) {
        main_interpreter_id = current_id;
        return (unlikely(current_id == -1)) ? -1 : 0;
    } else if (unlikely(main_interpreter_id != current_id))
    #else
    static PyInterpreterState *main_interpreter = NULL;
    PyInterpreterState *current_interpreter = PyThreadState_Get()->interp;
    if (!main_interpreter) {
        main_interpreter = current_interpreter;
    } else if (unlikely(main_interpreter != current_interpreter))
    #endif
    {
        PyErr_SetString(
            PyExc_ImportError,
            "Interpreter change detected - this module can only be loaded into one interpreter per process.");
        return -1;
    }
    return 0;
}
static CYTHON_SMALL_CODE int __Pyx_copy_spec_to_module(PyObject *spec, PyObject *moddict, const char* from_name, const char* to_name, int allow_none) {
    PyObject *value = PyObject_GetAttrString(spec, from_name);
    int result = 0;
    if (likely(value)) {
        if (allow_none || value != Py_None) {
            result = PyDict_SetItemString(moddict, to_name, value);
        }
        Py_DECREF(value);
    } else if (PyErr_ExceptionMatches(PyExc_AttributeError)) {
        PyErr_Clear();
    } else {
        result = -1;
    }
    return result;
}
static CYTHON_SMALL_CODE PyObject* __pyx_pymod_create(PyObject *spec, CYTHON_UNUSED PyModuleDef *def)
{
    PyObject *module = NULL, *moddict, *modname;
    if (__Pyx_check_single_interpreter())
        return NULL;
    if (__pyx_m)
        return __Pyx_NewRef(__pyx_m);
    modname = PyObject_GetAttrString(spec, "name");
    if (unlikely(!modname)) goto bad;
    module = PyModule_NewObject(modname);
    Py_DECREF(modname);
    if (unlikely(!module)) goto bad;
    moddict = PyModule_GetDict(module);
    if (unlikely(!moddict)) goto bad;
```

```
    if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "loader", "__loader__", 1) < 0)) goto bad;
    if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "origin", "__file__", 1) < 0)) goto bad;
    if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "parent", "__package__", 1) < 0)) goto bad;
    if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "submodule_search_locations", "__path__", 0) < 0)) goto bad;
    return module;
bad:
    Py_XDECREF(module);
    return NULL;
}


static CYTHON_SMALL_CODE int __pyx_pymod_exec_PyClical(PyObject *__pyx_pyinit_module)
#endif
#endif
{
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  PyObject *__pyx_t_3 = NULL;
  int __pyx_t_4;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannyDeclarations
  #if CYTHON_PEP489_MULTI_PHASE_INIT
  if (__pyx_m) {
    if (__pyx_m == __pyx_pyinit_module) return 0;
    PyErr_SetString(PyExc_RuntimeError, "Module ’PyClical’ has already been imported. Re-initialisation is not supported.");
    return -1;
  }
  #elif PY_MAJOR_VERSION >= 3
  if (__pyx_m) return __Pyx_NewRef(__pyx_m);
  #endif
  #if CYTHON_REFNANNY
__Pyx_RefNanny = __Pyx_RefNannyImportAPI("refnanny");
if (!__Pyx_RefNanny) {
  PyErr_Clear();
  __Pyx_RefNanny = __Pyx_RefNannyImportAPI("Cython.Runtime.refnanny");
  if (!__Pyx_RefNanny)
      Py_FatalError("failed to import ’refnanny’ module");
}
#endif
  __Pyx_RefNannySetupContext("__Pyx_PyMODINIT_FUNC PyInit_PyClical(void)", 0);
  if (__Pyx_check_binary_version() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #ifdef __Pxy_PyFrame_Initialize_Offsets
  __Pxy_PyFrame_Initialize_Offsets();
  #endif
  __pyx_empty_tuple = PyTuple_New(0); if (unlikely(!__pyx_empty_tuple)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_empty_bytes = PyBytes_FromStringAndSize("", 0); if (unlikely(!__pyx_empty_bytes)) __PYX_ERR(0, 1, __pyx_L1_error)
  __pyx_empty_unicode = PyUnicode_FromStringAndSize("", 0); if (unlikely(!__pyx_empty_unicode)) __PYX_ERR(0, 1, __pyx_L1_error)
  #ifdef __Pyx_CyFunction_USED
  if (__pyx_CyFunction_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  #ifdef __Pyx_FusedFunction_USED
  if (__pyx_FusedFunction_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  #ifdef __Pyx_Coroutine_USED
  if (__pyx_Coroutine_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  #ifdef __Pyx_Generator_USED
  if (__pyx_Generator_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  #ifdef __Pyx_AsyncGen_USED
  if (__pyx_AsyncGen_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  #ifdef __Pyx_StopAsyncIteration_USED
  if (__pyx_StopAsyncIteration_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  /*--- Library function declarations ---*/
  /*--- Threads initialization code ---*/
  #if defined(WITH_THREAD) && PY_VERSION_HEX < 0x030700F0 && defined(__PYX_FORCE_INIT_THREADS) && __PYX_FORCE_INIT_THREADS
  PyEval_InitThreads();
  #endif
  /*--- Module creation code ---*/
  #if CYTHON_PEP489_MULTI_PHASE_INIT
  __pyx_m = __pyx_pyinit_module;
  Py_INCREF(__pyx_m);
  #else
  #if PY_MAJOR_VERSION < 3
  __pyx_m = Py_InitModule4("PyClical", __pyx_methods, 0, 0, PYTHON_API_VERSION); Py_XINCREF(__pyx_m);
  #else
  __pyx_m = PyModule_Create(&__pyx_moduledef);
```

```
  #endif
  if (unlikely(!__pyx_m)) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  __pyx_d = PyModule_GetDict(__pyx_m); if (unlikely(!__pyx_d)) __PYX_ERR(0, 1, __pyx_L1_error)
  Py_INCREF(__pyx_d);
  __pyx_b = PyImport_AddModule(__Pyx_BUILTIN_MODULE_NAME); if (unlikely(!__pyx_b)) __PYX_ERR(0, 1, __pyx_L1_error)
  Py_INCREF(__pyx_b);
  __pyx_cython_runtime = PyImport_AddModule((char *) "cython_runtime"); if (unlikely(!__pyx_cython_runtime)) __PYX_ERR(0, 1, __pyx_L1_error)
  Py_INCREF(__pyx_cython_runtime);
  if (PyObject_SetAttrString(__pyx_m, "__builtins__", __pyx_b) < 0) __PYX_ERR(0, 1, __pyx_L1_error);
  /*--- Initialize various global constants etc. ---*/
  if (__Pyx_InitGlobals() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #if PY_MAJOR_VERSION < 3 && (__PYX_DEFAULT_STRING_ENCODING_IS_ASCII || __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT)
  if (__Pyx_init_sys_getdefaultencoding_params() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif
  if (__pyx_module_is_main_PyClical) {
    if (PyObject_SetAttr(__pyx_m, __pyx_n_s_name, __pyx_n_s_main) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  }
  #if PY_MAJOR_VERSION >= 3
  {
    PyObject *modules = PyImport_GetModuleDict(); if (unlikely(!modules)) __PYX_ERR(0, 1, __pyx_L1_error)
    if (!PyDict_GetItemString(modules, "PyClical")) {
      if (unlikely(PyDict_SetItemString(modules, "PyClical", __pyx_m) < 0)) __PYX_ERR(0, 1, __pyx_L1_error)
    }
  }
  #endif
  /*--- Builtin init code ---*/
  if (__Pyx_InitCachedBuiltins() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  /*--- Constants init code ---*/
  if (__Pyx_InitCachedConstants() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  /*--- Global type/function init code ---*/
  (void)__Pyx_modinit_global_init_code();
  (void)__Pyx_modinit_variable_export_code();
  (void)__Pyx_modinit_function_export_code();
  if (unlikely(__Pyx_modinit_type_init_code() < 0)) __PYX_ERR(0, 1, __pyx_L1_error)
  (void)__Pyx_modinit_type_import_code();
  (void)__Pyx_modinit_variable_import_code();
  (void)__Pyx_modinit_function_import_code();
  /*--- Execution code ---*/
  #if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
  if (__Pyx_patch_abc() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  #endif

  /* "PyClical.pyx":29
 * # C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.
 *
 * import math              # <<<<<<<<<<<<<<
 * import numbers
 * import collections
 */
  __pyx_t_1 = __Pyx_Import(__pyx_n_s_math, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 29, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_math, __pyx_t_1) < 0) __PYX_ERR(0, 29, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

  /* "PyClical.pyx":30
 *
 * import math
 * import numbers              # <<<<<<<<<<<<<<
 * import collections
 *
 */
  __pyx_t_1 = __Pyx_Import(__pyx_n_s_numbers, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 30, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_numbers, __pyx_t_1) < 0) __PYX_ERR(0, 30, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

  /* "PyClical.pyx":31
 * import math
 * import numbers
 * import collections              # <<<<<<<<<<<<<<
 *
 * from PyClical cimport *
 */
  __pyx_t_1 = __Pyx_Import(__pyx_n_s_collections, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 31, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_collections, __pyx_t_1) < 0) __PYX_ERR(0, 31, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

```

```
/* "PyClical.pyx":35
 * from PyClical cimport *
 *
 * __version__ = str(glucat_package_version,'utf-8')             # «««««««
 *
 * # Forward reference
 */
  __pyx_t_1 = __pyx_convert_PyBytes_string_to_py_std__in_string(glucat_package_version); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 35, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = PyTuple_New(2); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 35, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_GIVEREF(__pyx_t_1);
  PyTuple_SET_ITEM(__pyx_t_2, 0, __pyx_t_1);
  __Pyx_INCREF(__pyx_kp_u_utf_8);
  __Pyx_GIVEREF(__pyx_kp_u_utf_8);
  PyTuple_SET_ITEM(__pyx_t_2, 1, __pyx_kp_u_utf_8);
  __pyx_t_1 = 0;
  __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)(&PyUnicode_Type)), __pyx_t_2, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 35, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_version, __pyx_t_1) < 0) __PYX_ERR(0, 35, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":406
 *         return index_set_to_str( self.unwrap() ).decode()
 *
 * def index_set_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */
  __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_1index_set_hidden_doctests, NULL, __pyx_n_s_PyClical); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 406, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_index_set_hidden_doctests, __pyx_t_1) < 0) __PYX_ERR(0, 406, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1253
 *         return clifford_to_str( self.unwrap() ).decode()
 *
 * def clifford_hidden_doctests():             # «««««««
 *     """
 *     Tests for functions that Doctest cannot see.
 */
  __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_9clifford_hidden_doctests, NULL, __pyx_n_s_PyClical); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1253, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_clifford_hidden_doctests, __pyx_t_1) < 0) __PYX_ERR(0, 1253, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1905
 *
 * # Some abbreviations.
 * scalar_epsilon = epsilon             # «««««««
 *
 * pi = atan(clifford(1.0)) * 4.0
 */
  __pyx_t_1 = PyFloat_FromDouble(epsilon); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1905, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_scalar_epsilon, __pyx_t_1) < 0) __PYX_ERR(0, 1905, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1907
 * scalar_epsilon = epsilon
 *
 * pi = atan(clifford(1.0)) * 4.0             # «««««««
 * tau = atan(clifford(1.0)) * 8.0
 *
 */
  __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_tuple__15, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1907, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_atan(__pyx_t_1, 0, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1907, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_1 = PyNumber_Multiply(__pyx_t_2, __pyx_float_4_0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1907, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_pi, __pyx_t_1) < 0) __PYX_ERR(0, 1907, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
```

```
/* "PyClical.pyx":1908
 *
 * pi = atan(clifford(1.0)) * 4.0
 * tau = atan(clifford(1.0)) * 8.0             # <<<<<<<<
 *
 * cl = clifford
 */
  __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_tuple__15,
NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1908, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __pyx_f_8PyClical_atan(__pyx_t_1, 0, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1908,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_1 = PyNumber_Multiply(__pyx_t_2, __pyx_float_8_0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1908, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_tau, __pyx_t_1) < 0) __PYX_ERR(0, 1908, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1910
 * tau = atan(clifford(1.0)) * 8.0
 *
 * cl = clifford             # <<<<<<<<
 * """
 * Abbreviation for clifford.
 */
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_cl, ((PyObject *)__pyx_ptype_8PyClical_clifford)) < 0)
__PYX_ERR(0, 1910, __pyx_L1_error)

/* "PyClical.pyx":1928
 * """
 *
 * ist = index_set             # <<<<<<<<
 * """
 * Abbreviation for index_set.
 */
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_ist, ((PyObject *)__pyx_ptype_8PyClical_index_set)) < 0)
__PYX_ERR(0, 1928, __pyx_L1_error)

/* "PyClical.pyx":1936
 * """
 *
 * def e(obj):             # <<<<<<<<
 *     """
 *     Abbreviation for clifford(index_set(obj)).
 */
  __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_89e, NULL, __pyx_n_s_PyClical); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1936, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_e, __pyx_t_1) < 0) __PYX_ERR(0, 1936, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1949
 *     return clifford(index_set(obj))
 *
 * def istpq(p, q):             # <<<<<<<<
 *     """
 *     Abbreviation for index_set({-q,...p}).
 */
  __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_91istpq, NULL, __pyx_n_s_PyClical); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1949, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_istpq, __pyx_t_1) < 0) __PYX_ERR(0, 1949, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

/* "PyClical.pyx":1958
 *     return index_set(set(range(-q,p+1)))
 *
 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].             #
<<<<<<<<
 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].
 *
 */
  __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__20, NULL); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1958, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_3 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__21, NULL); if (unlikely(!__pyx_t_3))
```

```
__PYX_ERR(0, 1958, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_1 = PyNumber_Add(__pyx_t_2, __pyx_t_3); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_ninf3, __pyx_t_1) < 0) __PYX_ERR(0, 1958, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

  /* "PyClical.pyx":1959
 *
 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].            # «««««««
 *
 * # Doctest interface.
 */
  __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_3 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__20, NULL); if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 1959, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_3);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_2 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__21, NULL); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1959, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  __pyx_t_1 = PyNumber_Subtract(__pyx_t_3, __pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_nbar3, __pyx_t_1) < 0) __PYX_ERR(0, 1959, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

  /* "PyClical.pyx":1962
 *
 * # Doctest interface.
 * def _test():             # «««««««
 *     import PyClical, doctest
 *     return doctest.testmod(PyClical)
 */
  __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_93_test, NULL, __pyx_n_s_PyClical); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1962, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_test, __pyx_t_1) < 0) __PYX_ERR(0, 1962, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

  /* "PyClical.pyx":1966
 *     return doctest.testmod(PyClical)
 *
 * if __name__ == "__main__":             # «««««««
 *     _test()
 */
  __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_name); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1966,
__pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_1);
  __pyx_t_4 = (__Pyx_PyUnicode_Equals(__pyx_t_1, __pyx_n_u_main, Py_EQ)); if (unlikely(__pyx_t_4 < 0))
__PYX_ERR(0, 1966, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
  if (__pyx_t_4) {

    /* "PyClical.pyx":1967
 *
 * if __name__ == "__main__":
 *     _test()             # «««««««
 */
    __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_test); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1967,
__pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_t_2 = __Pyx_PyObject_CallNoArg(__pyx_t_1); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1967,
__pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;

    /* "PyClical.pyx":1966
 *     return doctest.testmod(PyClical)
 *
 * if __name__ == "__main__":             # «««««««
 *     _test()
 */
```

```
  }

  /* "PyClical.pyx":1
 * # -*- coding: utf-8 -*-             # «««««««
 * # cython: language_level=3
 * # distutils: language = c++
 */
  __pyx_t_2 = __Pyx_PyDict_NewPresized(111); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_copy_line_65,
 __pyx_kp_u_Copy_this_index_set_object_s_in) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___setitem___line_179,
 __pyx_kp_u_Set_the_value_of_an_index_set_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___getitem___line_191,
 __pyx_kp_u_Get_the_value_of_an_index_set_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___iter___line_229,
 __pyx_kp_u_Iterate_over_the_indices_of_an) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___invert___line_240,
 __pyx_kp_u_Set_complement_not_print_index) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___xor___line_249,
 __pyx_kp_u_Symmetric_set_difference_exclus) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___ixor___line_260,
 __pyx_kp_u_Symmetric_set_difference_exclus_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___and___line_271,
 __pyx_kp_u_Set_intersection_and_print_inde) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___iand___line_282,
 __pyx_kp_u_Set_intersection_and_x_index_se) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___or___line_293,
 __pyx_kp_u_Set_union_or_print_index_set_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___ior___line_304,
 __pyx_kp_u_Set_union_or_x_index_set_1_x_in) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_line_315,
 __pyx_kp_u_Cardinality_Number_of_indices_i) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_neg_line_324,
 __pyx_kp_u_Number_of_negative_indices_incl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_pos_line_333,
 __pyx_kp_u_Number_of_positive_indices_incl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_min_line_342,
 __pyx_kp_u_Minimum_member_index_set_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_max_line_351,
 __pyx_kp_u_Maximum_member_index_set_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_sign_of_mult_line_366,
 __pyx_kp_u_Sign_of_geometric_product_of_tw) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_sign_of_square_line_37,
 __pyx_kp_u_Sign_of_geometric_square_of_a_C) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___repr___line_384,
 __pyx_kp_u_The_official_string_representat) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set___str___line_395,
 __pyx_kp_u_The_informal_string_representat) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_hidden_doctests_line_4,
 __pyx_kp_u_Tests_for_functions_that_Doctes) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_compare_line_492,
 __pyx_kp_u_lexicographic_compare_eg_3_4_5) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_min_neg_line_504,
 __pyx_kp_u_Minimum_negative_index_or_0_if) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_max_pos_line_513,
 __pyx_kp_u_Maximum_positive_index_or_0_if) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_copy_line_556,
 __pyx_kp_u_Copy_this_clifford_object_x_cli) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___iter___line_638,
 __pyx_kp_u_Not_applicable_for_a_in_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_reframe_line_649,
 __pyx_kp_u_Put_self_into_a_larger_frame_co) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___getitem___line_707,
 __pyx_kp_u_Subscripting_map_from_index_set) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___neg___line_722,
 __pyx_kp_u_Unary_print_clifford_1_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___pos___line_731,
 __pyx_kp_u_Unary_print_clifford_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___add___line_740,
 __pyx_kp_u_Geometric_sum_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___iadd___line_751,
 __pyx_kp_u_Geometric_sum_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___sub___line_760,
 __pyx_kp_u_Geometric_difference_print_clif) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___isub___line_771,
 __pyx_kp_u_Geometric_difference_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___mul___line_780,
 __pyx_kp_u_Geometric_product_print_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___imul___line_793,
 __pyx_kp_u_Geometric_product_x_clifford_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___mod___line_806,
 __pyx_kp_u_Contraction_print_clifford_1_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___imod___line_821,
 __pyx_kp_u_Contraction_x_clifford_1_x_clif) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___and___line_836,
 __pyx_kp_u_Inner_product_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
```

```
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___iand___line_851, __pyx_kp_u_Inner_product_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___xor___line_866, __pyx_kp_u_Outer_product_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___ixor___line_881, __pyx_kp_u_Outer_product_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___truediv___line_896, __pyx_kp_u_Geometric_quotient_print_cliffo) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___idiv___line_911, __pyx_kp_u_Geometric_quotient_x_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_inv_line_926, __pyx_kp_u_Geometric_multiplicative_invers) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___or___line_939, __pyx_kp_u_Transform_left_hand_side_using) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___ior___line_950, __pyx_kp_u_Transform_left_hand_side_using_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___pow___line_961, __pyx_kp_u_Power_self_to_the_m_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pow_line_980, __pyx_kp_u_Power_self_to_the_m_x_clifford_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_outer_pow_line_1004, __pyx_kp_u_Outer_product_power_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___call___line_1020, __pyx_kp_u_Pure_grade_vector_part_print_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_scalar_line_1039, __pyx_kp_u_Scalar_part_clifford_1_1_2_sc) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pure_line_1050, __pyx_kp_u_Pure_part_print_clifford_1_1_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_even_line_1061, __pyx_kp_u_Even_part_of_multivector_sum_of) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_odd_line_1070, __pyx_kp_u_Odd_part_of_multivector_sum_of) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_vector_part_line_1079, __pyx_kp_u_Vector_part_of_multivector_as_a) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_involute_line_1107, __pyx_kp_u_Main_involution_each_i_is_repla) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_reverse_line_1123, __pyx_kp_u_Reversion_eg_clifford_1_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_conj_line_1138, __pyx_kp_u_Conjugation_reverse_o_involute) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_quad_line_1153, __pyx_kp_u_Quadratic_form_rev_x_x_0_print) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_norm_line_1164, __pyx_kp_u_Norm_sum_of_squares_of_coordina) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_abs_line_1175, __pyx_kp_u_Absolute_value_square_root_of_n) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_max_abs_line_1184, __pyx_kp_u_Maximum_of_absolute_values_of_c) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_truncated_line_1195, __pyx_kp_u_Remove_all_terms_of_self_with_r) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_isinf_line_1206, __pyx_kp_u_Check_if_a_multivector_contains) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_isnan_line_1215, __pyx_kp_u_Check_if_a_multivector_contains_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_frame_line_1224, __pyx_kp_u_Subalgebra_generated_by_all_gen) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___repr___line_1235, __pyx_kp_u_The_official_string_representat_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford___str___line_1244, __pyx_kp_u_The_informal_string_representat_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_hidden_doctests_line_12, __pyx_kp_u_Tests_for_functions_that_Doctes_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_error_squared_tol_line_1337, __pyx_kp_u_Quadratic_norm_error_tolerance) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_error_squared_line_1346, __pyx_kp_u_Relative_or_absolute_error_usin) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_approx_equal_line_1359, __pyx_kp_u_Test_for_approximate_equality_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_inv_line_1378, __pyx_kp_u_Geometric_multiplicative_invers_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_scalar_line_1393, __pyx_kp_u_Scalar_part_scalar_clifford_1_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_real_line_1404, __pyx_kp_u_Real_part_synonym_for_scalar_pa) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_imag_line_1415, __pyx_kp_u_Imaginary_part_deprecated_alway) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_pure_line_1426, __pyx_kp_u_Pure_part_print_pure_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_even_line_1437, __pyx_kp_u_Even_part_of_multivector_sum_of_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_odd_line_1446, __pyx_kp_u_Odd_part_of_multivector_sum_of_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_involute_line_1455, __pyx_kp_u_Main_involution_each_i_is_repla_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_reverse_line_1470, __pyx_kp_u_Reversion_eg_1_2_2_1_print_reve) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_conj_line_1485,
```

```
__pyx_kp_u_Conjugation_reverse_o_involute_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_quad_line_1500,
__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_norm_line_1511, __pyx_kp_u_norm_sum_of_squares_of_coordina)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_abs_line_1522, __pyx_kp_u_Absolute_value_of_multivector_m)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_max_abs_line_1531,
__pyx_kp_u_Maximum_absolute_value_of_coord) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_pow_line_1543, __pyx_kp_u_Integer_power_of_multivector_ob)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_outer_pow_line_1567,
__pyx_kp_u_Outer_product_power_of_multivec) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_complexifier_line_1576,
__pyx_kp_u_Square_root_of_1_which_commutes) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sqrt_line_1591, __pyx_kp_u_Square_root_of_multivector_with)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_exp_line_1614, __pyx_kp_u_Exponential_of_multivector_x_cl)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_log_line_1628, __pyx_kp_u_Natural_logarithm_of_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cos_line_1651, __pyx_kp_u_Cosine_of_multivector_with_opti)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_acos_line_1668, __pyx_kp_u_Inverse_cosine_of_multivector_w)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cosh_line_1689, __pyx_kp_u_Hyperbolic_cosine_of_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_acosh_line_1705,
__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sin_line_1728, __pyx_kp_u_Sine_of_multivector_with_option)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_asin_line_1747, __pyx_kp_u_Inverse_sine_of_multivector_wit)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sinh_line_1768, __pyx_kp_u_Hyperbolic_sine_of_multivector)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_asinh_line_1782,
__pyx_kp_u_Inverse_hyperbolic_sine_of_mult) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_tan_line_1801, __pyx_kp_u_Tangent_of_multivector_with_opt)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_atan_line_1818, __pyx_kp_u_Inverse_tangent_of_multivector)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_tanh_line_1835, __pyx_kp_u_Hyperbolic_tangent_of_multivect)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_atanh_line_1847,
__pyx_kp_u_Inverse_hyperbolic_tangent_of_m) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_random_clifford_line_1864,
__pyx_kp_u_Random_multivector_within_a_fra) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cga3_line_1873, __pyx_kp_u_Convert_Euclidean_3D_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cga3std_line_1882,
__pyx_kp_u_Convert_CGA3_null_vector_to_sta) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_agc3_line_1893, __pyx_kp_u_Convert_CGA3_null_vector_to_Euc)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_e_line_1936, __pyx_kp_u_Abbreviation_for_clifford_index) <
0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_istpq_line_1949, __pyx_kp_u_Abbreviation_for_index_set_q_p)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
  if (PyDict_SetItem(__pyx_d, __pyx_n_s_test_2, __pyx_t_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
  __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;

  /* "string.to_py":55
 *
 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s):
# «««««««
 *     return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
 *
 */

  /*--- Wrapped vars code ---*/

  goto __pyx_L0;
  __pyx_L1_error:;
  __Pyx_XDECREF(__pyx_t_1);
  __Pyx_XDECREF(__pyx_t_2);
  __Pyx_XDECREF(__pyx_t_3);
  if (__pyx_m) {
    if (__pyx_d) {
      __Pyx_AddTraceback("init PyClical", __pyx_clineno, __pyx_lineno, __pyx_filename);
    }
    Py_CLEAR(__pyx_m);
  } else if (!PyErr_Occurred()) {
    PyErr_SetString(PyExc_ImportError, "init PyClical");
  }
  __pyx_L0:;
  __Pyx_RefNannyFinishContext();
  #if CYTHON_PEP489_MULTI_PHASE_INIT
```

```
    return (__pyx_m != NULL) ? 0 : -1;
    #elif PY_MAJOR_VERSION >= 3
    return __pyx_m;
    #else
    return;
    #endif
}

/* --- Runtime support code --- */
/* Refnanny */
#if CYTHON_REFNANNY
static __Pyx_RefNannyAPIStruct *__Pyx_RefNannyImportAPI(const char *modname) {
    PyObject *m = NULL, *p = NULL;
    void *r = NULL;
    m = PyImport_ImportModule(modname);
    if (!m) goto end;
    p = PyObject_GetAttrString(m, "RefNannyAPI");
    if (!p) goto end;
    r = PyLong_AsVoidPtr(p);
end:
    Py_XDECREF(p);
    Py_XDECREF(m);
    return (__Pyx_RefNannyAPIStruct *)r;
}
#endif

/* PyObjectGetAttrStr */
#if CYTHON_USE_TYPE_SLOTS
static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStr(PyObject* obj, PyObject* attr_name) {
    PyTypeObject* tp = Py_TYPE(obj);
    if (likely(tp->tp_getattro))
        return tp->tp_getattro(obj, attr_name);
#if PY_MAJOR_VERSION < 3
    if (likely(tp->tp_getattr))
        return tp->tp_getattr(obj, PyString_AS_STRING(attr_name));
#endif
    return PyObject_GetAttr(obj, attr_name);
}
#endif

/* GetBuiltinName */
static PyObject *__Pyx_GetBuiltinName(PyObject *name) {
    PyObject* result = __Pyx_PyObject_GetAttrStr(__pyx_b, name);
    if (unlikely(!result)) {
        PyErr_Format(PyExc_NameError,
#if PY_MAJOR_VERSION >= 3
            "name '%U' is not defined", name);
#else
            "name '%.200s' is not defined", PyString_AS_STRING(name));
#endif
    }
    return result;
}

/* PyCFunctionFastCall */
#if CYTHON_FAST_PYCCALL
static CYTHON_INLINE PyObject * __Pyx_PyCFunction_FastCall(PyObject *func_obj, PyObject **args,
Py_ssize_t nargs) {
    PyCFunctionObject *func = (PyCFunctionObject*)func_obj;
    PyCFunction meth = PyCFunction_GET_FUNCTION(func);
    PyObject *self = PyCFunction_GET_SELF(func);
    int flags = PyCFunction_GET_FLAGS(func);
    assert(PyCFunction_Check(func));
    assert(METH_FASTCALL == (flags & ~(METH_CLASS | METH_STATIC | METH_COEXIST | METH_KEYWORDS |
METH_STACKLESS)));
    assert(nargs >= 0);
    assert(nargs == 0 || args != NULL);
    /* _PyCFunction_FastCallDict() must not be called with an exception set,
       because it may clear it (directly or indirectly) and so the
       caller loses its exception */
    assert(!PyErr_Occurred());
    if ((PY_VERSION_HEX < 0x030700A0) || unlikely(flags & METH_KEYWORDS)) {
        return (*((__Pyx_PyCFunctionFastWithKeywords)(void*)meth)) (self, args, nargs, NULL);
    } else {
        return (*((__Pyx_PyCFunctionFast)(void*)meth)) (self, args, nargs);
    }
}
#endif

/* PyFunctionFastCall */
#if CYTHON_FAST_PYCALL
static PyObject* __Pyx_PyFunction_FastCallNoKw(PyCodeObject *co, PyObject **args, Py_ssize_t na,
                                               PyObject *globals) {
    PyFrameObject *f;
    PyThreadState *tstate = __Pyx_PyThreadState_Current;
    PyObject **fastlocals;
    Py_ssize_t i;
```

```
    PyObject *result;
    assert(globals != NULL);
    /* XXX Perhaps we should create a specialized
       PyFrame_New() that doesn't take locals, but does
       take builtins without sanity checking them.
       */
    assert(tstate != NULL);
    f = PyFrame_New(tstate, co, globals, NULL);
    if (f == NULL) {
        return NULL;
    }
    fastlocals = __Pyx_PyFrame_GetLocalsplus(f);
    for (i = 0; i < na; i++) {
        Py_INCREF(*args);
        fastlocals[i] = *args++;
    }
    result = PyEval_EvalFrameEx(f,0);
    ++tstate->recursion_depth;
    Py_DECREF(f);
    --tstate->recursion_depth;
    return result;
}
#if 1 || PY_VERSION_HEX < 0x030600B1
static PyObject *__Pyx_PyFunction_FastCallDict(PyObject *func, PyObject **args, Py_ssize_t nargs,
PyObject *kwargs) {
    PyCodeObject *co = (PyCodeObject *)PyFunction_GET_CODE(func);
    PyObject *globals = PyFunction_GET_GLOBALS(func);
    PyObject *argdefs = PyFunction_GET_DEFAULTS(func);
    PyObject *closure;
#if PY_MAJOR_VERSION >= 3
    PyObject *kwdefs;
#endif
    PyObject *kwtuple, **k;
    PyObject **d;
    Py_ssize_t nd;
    Py_ssize_t nk;
    PyObject *result;
    assert(kwargs == NULL || PyDict_Check(kwargs));
    nk = kwargs ? PyDict_Size(kwargs) : 0;
    if (Py_EnterRecursiveCall((char*)" while calling a Python object")) {
        return NULL;
    }
    if (
#if PY_MAJOR_VERSION >= 3
            co->co_kwonlyargcount == 0 &&
#endif
            likely(kwargs == NULL || nk == 0) &&
            co->co_flags == (CO_OPTIMIZED | CO_NEWLOCALS | CO_NOFREE)) {
        if (argdefs == NULL && co->co_argcount == nargs) {
            result = __Pyx_PyFunction_FastCallNoKw(co, args, nargs, globals);
            goto done;
        }
        else if (nargs == 0 && argdefs != NULL
                 && co->co_argcount == Py_SIZE(argdefs)) {
            /* function called with no arguments, but all parameters have
               a default value: use default values as arguments .*/
            args = &PyTuple_GET_ITEM(argdefs, 0);
            result =__Pyx_PyFunction_FastCallNoKw(co, args, Py_SIZE(argdefs), globals);
            goto done;
        }
    }
    if (kwargs != NULL) {
        Py_ssize_t pos, i;
        kwtuple = PyTuple_New(2 * nk);
        if (kwtuple == NULL) {
            result = NULL;
            goto done;
        }
        k = &PyTuple_GET_ITEM(kwtuple, 0);
        pos = i = 0;
        while (PyDict_Next(kwargs, &pos, &k[i], &k[i+1])) {
            Py_INCREF(k[i]);
            Py_INCREF(k[i+1]);
            i += 2;
        }
        nk = i / 2;
    }
    else {
        kwtuple = NULL;
        k = NULL;
    }
    closure = PyFunction_GET_CLOSURE(func);
#if PY_MAJOR_VERSION >= 3
    kwdefs = PyFunction_GET_KW_DEFAULTS(func);
#endif
    if (argdefs != NULL) {
        d = &PyTuple_GET_ITEM(argdefs, 0);
```

```
        nd = Py_SIZE(argdefs);
    }
    else {
        d = NULL;
        nd = 0;
    }
#if PY_MAJOR_VERSION >= 3
    result = PyEval_EvalCodeEx((PyObject*)co, globals, (PyObject *)NULL,
                               args, (int)nargs,
                               k, (int)nk,
                               d, (int)nd, kwdefs, closure);
#else
    result = PyEval_EvalCodeEx(co, globals, (PyObject *)NULL,
                               args, (int)nargs,
                               k, (int)nk,
                               d, (int)nd, closure);
#endif
    Py_XDECREF(kwtuple);
done:
    Py_LeaveRecursiveCall();
    return result;
}
#endif
#endif

/* PyObjectCall */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_Call(PyObject *func, PyObject *arg, PyObject *kw) {
    PyObject *result;
    ternaryfunc call = Py_TYPE(func)->tp_call;
    if (unlikely(!call))
        return PyObject_Call(func, arg, kw);
    if (unlikely(Py_EnterRecursiveCall((char*)" while calling a Python object")))
        return NULL;
    result = (*call)(func, arg, kw);
    Py_LeaveRecursiveCall();
    if (unlikely(!result) && unlikely(!PyErr_Occurred())) {
        PyErr_SetString(
            PyExc_SystemError,
            "NULL result without error in PyObject_Call");
    }
    return result;
}
#endif

/* PyObjectCallMethO */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallMethO(PyObject *func, PyObject *arg) {
    PyObject *self, *result;
    PyCFunction cfunc;
    cfunc = PyCFunction_GET_FUNCTION(func);
    self = PyCFunction_GET_SELF(func);
    if (unlikely(Py_EnterRecursiveCall((char*)" while calling a Python object")))
        return NULL;
    result = cfunc(self, arg);
    Py_LeaveRecursiveCall();
    if (unlikely(!result) && unlikely(!PyErr_Occurred())) {
        PyErr_SetString(
            PyExc_SystemError,
            "NULL result without error in PyObject_Call");
    }
    return result;
}
#endif

/* PyObjectCallOneArg */
#if CYTHON_COMPILING_IN_CPYTHON
static PyObject* __Pyx__PyObject_CallOneArg(PyObject *func, PyObject *arg) {
    PyObject *result;
    PyObject *args = PyTuple_New(1);
    if (unlikely(!args)) return NULL;
    Py_INCREF(arg);
    PyTuple_SET_ITEM(args, 0, arg);
    result = __Pyx_PyObject_Call(func, args, NULL);
    Py_DECREF(args);
    return result;
}
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg) {
#if CYTHON_FAST_PYCALL
    if (PyFunction_Check(func)) {
        return __Pyx_PyFunction_FastCall(func, &arg, 1);
    }
#endif
    if (likely(PyCFunction_Check(func))) {
        if (likely(PyCFunction_GET_FLAGS(func) & METH_O)) {
            return __Pyx_PyObject_CallMethO(func, arg);
#if CYTHON_FAST_PYCCALL
```

```
        } else if (__Pyx_PyFastCFunction_Check(func)) {
            return __Pyx_PyCFunction_FastCall(func, &arg, 1);
#endif
        }
    }
    return __Pyx__PyObject_CallOneArg(func, arg);
}
#else
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg) {
    PyObject *result;
    PyObject *args = PyTuple_Pack(1, arg);
    if (unlikely(!args)) return NULL;
    result = __Pyx_PyObject_Call(func, args, NULL);
    Py_DECREF(args);
    return result;
}
#endif

/* PyErrFetchRestore */
#if CYTHON_FAST_THREAD_STATE
static CYTHON_INLINE void __Pyx_ErrRestoreInState(PyThreadState *tstate, PyObject *type, PyObject *value, PyObject *tb) {
    PyObject *tmp_type, *tmp_value, *tmp_tb;
    tmp_type = tstate->curexc_type;
    tmp_value = tstate->curexc_value;
    tmp_tb = tstate->curexc_traceback;
    tstate->curexc_type = type;
    tstate->curexc_value = value;
    tstate->curexc_traceback = tb;
    Py_XDECREF(tmp_type);
    Py_XDECREF(tmp_value);
    Py_XDECREF(tmp_tb);
}
static CYTHON_INLINE void __Pyx_ErrFetchInState(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb) {
    *type = tstate->curexc_type;
    *value = tstate->curexc_value;
    *tb = tstate->curexc_traceback;
    tstate->curexc_type = 0;
    tstate->curexc_value = 0;
    tstate->curexc_traceback = 0;
}
#endif

/* WriteUnraisableException */
static void __Pyx_WriteUnraisable(const char *name, CYTHON_UNUSED int clineno,
                                  CYTHON_UNUSED int lineno, CYTHON_UNUSED const char *filename,
                                  int full_traceback, CYTHON_UNUSED int nogil) {
    PyObject *old_exc, *old_val, *old_tb;
    PyObject *ctx;
    __Pyx_PyThreadState_declare
#ifdef WITH_THREAD
    PyGILState_STATE state;
    if (nogil)
        state = PyGILState_Ensure();
#ifdef _MSC_VER
    else state = (PyGILState_STATE)-1;
#endif
#endif
    __Pyx_PyThreadState_assign
    __Pyx_ErrFetch(&old_exc, &old_val, &old_tb);
    if (full_traceback) {
        Py_XINCREF(old_exc);
        Py_XINCREF(old_val);
        Py_XINCREF(old_tb);
        __Pyx_ErrRestore(old_exc, old_val, old_tb);
        PyErr_PrintEx(1);
    }
    #if PY_MAJOR_VERSION < 3
    ctx = PyString_FromString(name);
    #else
    ctx = PyUnicode_FromString(name);
    #endif
    __Pyx_ErrRestore(old_exc, old_val, old_tb);
    if (!ctx) {
        PyErr_WriteUnraisable(Py_None);
    } else {
        PyErr_WriteUnraisable(ctx);
        Py_DECREF(ctx);
    }
#ifdef WITH_THREAD
    if (nogil)
        PyGILState_Release(state);
#endif
}

/* PyDictVersioning */
```

```
#if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_TYPE_SLOTS
static CYTHON_INLINE PY_UINT64_T __Pyx_get_tp_dict_version(PyObject *obj) {
    PyObject *dict = Py_TYPE(obj)->tp_dict;
    return likely(dict) ? __PYX_GET_DICT_VERSION(dict) : 0;
}
static CYTHON_INLINE PY_UINT64_T __Pyx_get_object_dict_version(PyObject *obj) {
    PyObject **dictptr = NULL;
    Py_ssize_t offset = Py_TYPE(obj)->tp_dictoffset;
    if (offset) {
#if CYTHON_COMPILING_IN_CPYTHON
        dictptr = (likely(offset > 0)) ? (PyObject **) ((char *)obj + offset) : _PyObject_GetDictPtr(obj);
#else
        dictptr = _PyObject_GetDictPtr(obj);
#endif
    }
    return (dictptr && *dictptr) ? __PYX_GET_DICT_VERSION(*dictptr) : 0;
}
static CYTHON_INLINE int __Pyx_object_dict_version_matches(PyObject* obj, PY_UINT64_T tp_dict_version, PY_UINT64_T obj_dict_version) {
    PyObject *dict = Py_TYPE(obj)->tp_dict;
    if (unlikely(!dict) || unlikely(tp_dict_version != __PYX_GET_DICT_VERSION(dict)))
        return 0;
    return obj_dict_version == __Pyx_get_object_dict_version(obj);
}
#endif

/* PyObjectCallNoArg */
#if CYTHON_COMPILING_IN_CPYTHON
static CYTHON_INLINE PyObject* __Pyx_PyObject_CallNoArg(PyObject *func) {
#if CYTHON_FAST_PYCALL
    if (PyFunction_Check(func)) {
        return __Pyx_PyFunction_FastCall(func, NULL, 0);
    }
#endif
#ifdef __Pyx_CyFunction_USED
    if (likely(PyCFunction_Check(func) || __Pyx_CyFunction_Check(func)))
#else
    if (likely(PyCFunction_Check(func)))
#endif
    {
        if (likely(PyCFunction_GET_FLAGS(func) & METH_NOARGS)) {
            return __Pyx_PyObject_CallMethO(func, NULL);
        }
    }
    return __Pyx_PyObject_Call(func, __pyx_empty_tuple, NULL);
}
#endif

/* RaiseDoubleKeywords */
static void __Pyx_RaiseDoubleKeywordsError(
    const char* func_name,
    PyObject* kw_name)
{
    PyErr_Format(PyExc_TypeError,
        #if PY_MAJOR_VERSION >= 3
        "%s() got multiple values for keyword argument '%U'", func_name, kw_name);
        #else
        "%s() got multiple values for keyword argument '%s'", func_name,
        PyString_AsString(kw_name));
        #endif
}

/* ParseKeywords */
static int __Pyx_ParseOptionalKeywords(
    PyObject *kwds,
    PyObject **argnames[],
    PyObject *kwds2,
    PyObject *values[],
    Py_ssize_t num_pos_args,
    const char* function_name)
{
    PyObject *key = 0, *value = 0;
    Py_ssize_t pos = 0;
    PyObject*** name;
    PyObject*** first_kw_arg = argnames + num_pos_args;
    while (PyDict_Next(kwds, &pos, &key, &value)) {
        name = first_kw_arg;
        while (*name && (**name != key)) name++;
        if (*name) {
            values[name-argnames] = value;
            continue;
        }
        name = first_kw_arg;
        #if PY_MAJOR_VERSION < 3
        if (likely(PyString_Check(key))) {
            while (*name) {
```

```
                if ((CYTHON_COMPILING_IN_PYPY || PyString_GET_SIZE(**name) == PyString_GET_SIZE(key))
                        && _PyString_Eq(**name, key)) {
                    values[name-argnames] = value;
                    break;
                }
                name++;
            }
            if (*name) continue;
            else {
                PyObject*** argname = argnames;
                while (argname != first_kw_arg) {
                    if ((**argname == key) || (
                            (CYTHON_COMPILING_IN_PYPY || PyString_GET_SIZE(**argname) == PyString_GET_SIZE(key))
                             && _PyString_Eq(**argname, key))) {
                        goto arg_passed_twice;
                    }
                    argname++;
                }
            }
        } else
        #endif
        if (likely(PyUnicode_Check(key))) {
            while (*name) {
                int cmp = (**name == key) ? 0 :
                #if !CYTHON_COMPILING_IN_PYPY && PY_MAJOR_VERSION >= 3
                    (__Pyx_PyUnicode_GET_LENGTH(**name) != __Pyx_PyUnicode_GET_LENGTH(key)) ? 1 :
                #endif
                    PyUnicode_Compare(**name, key);
                if (cmp < 0 && unlikely(PyErr_Occurred())) goto bad;
                if (cmp == 0) {
                    values[name-argnames] = value;
                    break;
                }
                name++;
            }
            if (*name) continue;
            else {
                PyObject*** argname = argnames;
                while (argname != first_kw_arg) {
                    int cmp = (**argname == key) ? 0 :
                    #if !CYTHON_COMPILING_IN_PYPY && PY_MAJOR_VERSION >= 3
                        (__Pyx_PyUnicode_GET_LENGTH(**argname) != __Pyx_PyUnicode_GET_LENGTH(key)) ? 1 :
                    #endif
                        PyUnicode_Compare(**argname, key);
                    if (cmp < 0 && unlikely(PyErr_Occurred())) goto bad;
                    if (cmp == 0) goto arg_passed_twice;
                    argname++;
                }
            }
        } else
            goto invalid_keyword_type;
        if (kwds2) {
            if (unlikely(PyDict_SetItem(kwds2, key, value))) goto bad;
        } else {
            goto invalid_keyword;
        }
    }
    return 0;
arg_passed_twice:
    __Pyx_RaiseDoubleKeywordsError(function_name, key);
    goto bad;
invalid_keyword_type:
    PyErr_Format(PyExc_TypeError,
        "%.200s() keywords must be strings", function_name);
    goto bad;
invalid_keyword:
    PyErr_Format(PyExc_TypeError,
    #if PY_MAJOR_VERSION < 3
        "%.200s() got an unexpected keyword argument '%.200s'",
        function_name, PyString_AsString(key));
    #else
        "%s() got an unexpected keyword argument '%U'",
        function_name, key);
    #endif
bad:
    return -1;
}

/* RaiseArgTupleInvalid */
static void __Pyx_RaiseArgtupleInvalid(
    const char* func_name,
    int exact,
    Py_ssize_t num_min,
    Py_ssize_t num_max,
    Py_ssize_t num_found)
```

```
{
    Py_ssize_t num_expected;
    const char *more_or_less;
    if (num_found < num_min) {
        num_expected = num_min;
        more_or_less = "at least";
    } else {
        num_expected = num_max;
        more_or_less = "at most";
    }
    if (exact) {
        more_or_less = "exactly";
    }
    PyErr_Format(PyExc_TypeError,
                 "%.200s() takes %.8s %" CYTHON_FORMAT_SSIZE_T "d positional argument%.1s (%"
CYTHON_FORMAT_SSIZE_T "d given)",
                 func_name, more_or_less, num_expected,
                 (num_expected == 1) ? "" : "s", num_found);
}

/* GetModuleGlobalName */
#if CYTHON_USE_DICT_VERSIONS
static PyObject *__Pyx__GetModuleGlobalName(PyObject *name, PY_UINT64_T *dict_version, PyObject
**dict_cached_value)
#else
static CYTHON_INLINE PyObject *__Pyx__GetModuleGlobalName(PyObject *name)
#endif
{
    PyObject *result;
#if !CYTHON_AVOID_BORROWED_REFS
#if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x030500A1
    result = _PyDict_GetItem_KnownHash(__pyx_d, name, ((PyASCIIObject *) name)->hash);
    __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
    if (likely(result)) {
        return __Pyx_NewRef(result);
    } else if (unlikely(PyErr_Occurred())) {
        return NULL;
    }
#else
    result = PyDict_GetItem(__pyx_d, name);
    __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
    if (likely(result)) {
        return __Pyx_NewRef(result);
    }
#endif
#else
    result = PyObject_GetItem(__pyx_d, name);
    __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
    if (likely(result)) {
        return __Pyx_NewRef(result);
    }
    PyErr_Clear();
#endif
    return __Pyx_GetBuiltinName(name);
}

/* GetTopmostException */
#if CYTHON_USE_EXC_INFO_STACK
static _PyErr_StackItem *
__Pyx_PyErr_GetTopmostException(PyThreadState *tstate)
{
    _PyErr_StackItem *exc_info = tstate->exc_info;
    while ((exc_info->exc_type == NULL || exc_info->exc_type == Py_None) &&
           exc_info->previous_item != NULL)
    {
        exc_info = exc_info->previous_item;
    }
    return exc_info;
}
#endif

/* SaveResetException */
#if CYTHON_FAST_THREAD_STATE
static CYTHON_INLINE void __Pyx__ExceptionSave(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb) {
    #if CYTHON_USE_EXC_INFO_STACK
    _PyErr_StackItem *exc_info = __Pyx_PyErr_GetTopmostException(tstate);
    *type = exc_info->exc_type;
    *value = exc_info->exc_value;
    *tb = exc_info->exc_traceback;
    #else
    *type = tstate->exc_type;
    *value = tstate->exc_value;
    *tb = tstate->exc_traceback;
    #endif
    Py_XINCREF(*type);
    Py_XINCREF(*value);
```

```
    Py_XINCREF(*tb);
}
static CYTHON_INLINE void __Pyx__ExceptionReset(PyThreadState *tstate, PyObject *type, PyObject
*value, PyObject *tb) {
    PyObject *tmp_type, *tmp_value, *tmp_tb;
    #if CYTHON_USE_EXC_INFO_STACK
    _PyErr_StackItem *exc_info = tstate->exc_info;
    tmp_type = exc_info->exc_type;
    tmp_value = exc_info->exc_value;
    tmp_tb = exc_info->exc_traceback;
    exc_info->exc_type = type;
    exc_info->exc_value = value;
    exc_info->exc_traceback = tb;
    #else
    tmp_type = tstate->exc_type;
    tmp_value = tstate->exc_value;
    tmp_tb = tstate->exc_traceback;
    tstate->exc_type = type;
    tstate->exc_value = value;
    tstate->exc_traceback = tb;
    #endif
    Py_XDECREF(tmp_type);
    Py_XDECREF(tmp_value);
    Py_XDECREF(tmp_tb);
}
#endif

/* PyErrExceptionMatches */
#if CYTHON_FAST_THREAD_STATE
static int __Pyx_PyErr_ExceptionMatchesTuple(PyObject *exc_type, PyObject *tuple) {
    Py_ssize_t i, n;
    n = PyTuple_GET_SIZE(tuple);
#if PY_MAJOR_VERSION >= 3
    for (i=0; i<n; i++) {
        if (exc_type == PyTuple_GET_ITEM(tuple, i)) return 1;
    }
#endif
    for (i=0; i<n; i++) {
        if (__Pyx_PyErr_GivenExceptionMatches(exc_type, PyTuple_GET_ITEM(tuple, i))) return 1;
    }
    return 0;
}
static CYTHON_INLINE int __Pyx_PyErr_ExceptionMatchesInState(PyThreadState* tstate, PyObject* err) {
    PyObject *exc_type = tstate->curexc_type;
    if (exc_type == err) return 1;
    if (unlikely(!exc_type)) return 0;
    if (unlikely(PyTuple_Check(err)))
        return __Pyx_PyErr_ExceptionMatchesTuple(exc_type, err);
    return __Pyx_PyErr_GivenExceptionMatches(exc_type, err);
}
#endif

/* GetException */
#if CYTHON_FAST_THREAD_STATE
static int __Pyx__GetException(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject
**tb)
#else
static int __Pyx_GetException(PyObject **type, PyObject **value, PyObject **tb)
#endif
{
    PyObject *local_type, *local_value, *local_tb;
#if CYTHON_FAST_THREAD_STATE
    PyObject *tmp_type, *tmp_value, *tmp_tb;
    local_type = tstate->curexc_type;
    local_value = tstate->curexc_value;
    local_tb = tstate->curexc_traceback;
    tstate->curexc_type = 0;
    tstate->curexc_value = 0;
    tstate->curexc_traceback = 0;
#else
    PyErr_Fetch(&local_type, &local_value, &local_tb);
#endif
    PyErr_NormalizeException(&local_type, &local_value, &local_tb);
#if CYTHON_FAST_THREAD_STATE
    if (unlikely(tstate->curexc_type))
#else
    if (unlikely(PyErr_Occurred()))
#endif
        goto bad;
    #if PY_MAJOR_VERSION >= 3
    if (local_tb) {
        if (unlikely(PyException_SetTraceback(local_value, local_tb) < 0))
            goto bad;
    }
    #endif
    Py_XINCREF(local_tb);
    Py_XINCREF(local_type);
```

```
    Py_XINCREF(local_value);
    *type = local_type;
    *value = local_value;
    *tb = local_tb;
#if CYTHON_FAST_THREAD_STATE
    #if CYTHON_USE_EXC_INFO_STACK
    {
        _PyErr_StackItem *exc_info = tstate->exc_info;
        tmp_type = exc_info->exc_type;
        tmp_value = exc_info->exc_value;
        tmp_tb = exc_info->exc_traceback;
        exc_info->exc_type = local_type;
        exc_info->exc_value = local_value;
        exc_info->exc_traceback = local_tb;
    }
    #else
    tmp_type = tstate->exc_type;
    tmp_value = tstate->exc_value;
    tmp_tb = tstate->exc_traceback;
    tstate->exc_type = local_type;
    tstate->exc_value = local_value;
    tstate->exc_traceback = local_tb;
    #endif
    Py_XDECREF(tmp_type);
    Py_XDECREF(tmp_value);
    Py_XDECREF(tmp_tb);
#else
    PyErr_SetExcInfo(local_type, local_value, local_tb);
#endif
    return 0;
bad:
    *type = 0;
    *value = 0;
    *tb = 0;
    Py_XDECREF(local_type);
    Py_XDECREF(local_value);
    Py_XDECREF(local_tb);
    return -1;
}

/* RaiseException */
#if PY_MAJOR_VERSION < 3
static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb,
                        CYTHON_UNUSED PyObject *cause) {
    __Pyx_PyThreadState_declare
    Py_XINCREF(type);
    if (!value || value == Py_None)
        value = NULL;
    else
        Py_INCREF(value);
    if (!tb || tb == Py_None)
        tb = NULL;
    else {
        Py_INCREF(tb);
        if (!PyTraceBack_Check(tb)) {
            PyErr_SetString(PyExc_TypeError,
                "raise: arg 3 must be a traceback or None");
            goto raise_error;
        }
    }
    if (PyType_Check(type)) {
#if CYTHON_COMPILING_IN_PYPY
        if (!value) {
            Py_INCREF(Py_None);
            value = Py_None;
        }
#endif
        PyErr_NormalizeException(&type, &value, &tb);
    } else {
        if (value) {
            PyErr_SetString(PyExc_TypeError,
                "instance exception may not have a separate value");
            goto raise_error;
        }
        value = type;
        type = (PyObject*) Py_TYPE(type);
        Py_INCREF(type);
        if (!PyType_IsSubtype((PyTypeObject *)type, (PyTypeObject *)PyExc_BaseException)) {
            PyErr_SetString(PyExc_TypeError,
                "raise: exception class must be a subclass of BaseException");
            goto raise_error;
        }
    }
    __Pyx_PyThreadState_assign
    __Pyx_ErrRestore(type, value, tb);
    return;
raise_error:
```

```
    Py_XDECREF(value);
    Py_XDECREF(type);
    Py_XDECREF(tb);
    return;
}
#else
static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb, PyObject *cause) {
    PyObject* owned_instance = NULL;
    if (tb == Py_None) {
        tb = 0;
    } else if (tb && !PyTraceBack_Check(tb)) {
        PyErr_SetString(PyExc_TypeError,
            "raise: arg 3 must be a traceback or None");
        goto bad;
    }
    if (value == Py_None)
        value = 0;
    if (PyExceptionInstance_Check(type)) {
        if (value) {
            PyErr_SetString(PyExc_TypeError,
                "instance exception may not have a separate value");
            goto bad;
        }
        value = type;
        type = (PyObject*) Py_TYPE(value);
    } else if (PyExceptionClass_Check(type)) {
        PyObject *instance_class = NULL;
        if (value && PyExceptionInstance_Check(value)) {
            instance_class = (PyObject*) Py_TYPE(value);
            if (instance_class != type) {
                int is_subclass = PyObject_IsSubclass(instance_class, type);
                if (!is_subclass) {
                    instance_class = NULL;
                } else if (unlikely(is_subclass == -1)) {
                    goto bad;
                } else {
                    type = instance_class;
                }
            }
        }
        if (!instance_class) {
            PyObject *args;
            if (!value)
                args = PyTuple_New(0);
            else if (PyTuple_Check(value)) {
                Py_INCREF(value);
                args = value;
            } else
                args = PyTuple_Pack(1, value);
            if (!args)
                goto bad;
            owned_instance = PyObject_Call(type, args, NULL);
            Py_DECREF(args);
            if (!owned_instance)
                goto bad;
            value = owned_instance;
            if (!PyExceptionInstance_Check(value)) {
                PyErr_Format(PyExc_TypeError,
                             "calling %R should have returned an instance of "
                             "BaseException, not %R",
                             type, Py_TYPE(value));
                goto bad;
            }
        }
    } else {
        PyErr_SetString(PyExc_TypeError,
            "raise: exception class must be a subclass of BaseException");
        goto bad;
    }
    if (cause) {
        PyObject *fixed_cause;
        if (cause == Py_None) {
            fixed_cause = NULL;
        } else if (PyExceptionClass_Check(cause)) {
            fixed_cause = PyObject_CallObject(cause, NULL);
            if (fixed_cause == NULL)
                goto bad;
        } else if (PyExceptionInstance_Check(cause)) {
            fixed_cause = cause;
            Py_INCREF(fixed_cause);
        } else {
            PyErr_SetString(PyExc_TypeError,
                            "exception causes must derive from "
                            "BaseException");
            goto bad;
        }
        PyException_SetCause(value, fixed_cause);
```

```
    }
    PyErr_SetObject(type, value);
    if (tb) {
#if CYTHON_COMPILING_IN_PYPY
        PyObject *tmp_type, *tmp_value, *tmp_tb;
        PyErr_Fetch(&tmp_type, &tmp_value, &tmp_tb);
        Py_INCREF(tb);
        PyErr_Restore(tmp_type, tmp_value, tb);
        Py_XDECREF(tmp_tb);
#else
        PyThreadState *tstate = __Pyx_PyThreadState_Current;
        PyObject* tmp_tb = tstate->curexc_traceback;
        if (tb != tmp_tb) {
            Py_INCREF(tb);
            tstate->curexc_traceback = tb;
            Py_XDECREF(tmp_tb);
        }
#endif
    }
bad:
    Py_XDECREF(owned_instance);
    return;
}
#endif

/* PyObjectCall2Args */
static CYTHON_UNUSED PyObject* __Pyx_PyObject_Call2Args(PyObject* function, PyObject* arg1, PyObject*
arg2) {
    PyObject *args, *result = NULL;
    #if CYTHON_FAST_PYCALL
    if (PyFunction_Check(function)) {
        PyObject *args[2] = {arg1, arg2};
        return __Pyx_PyFunction_FastCall(function, args, 2);
    }
    #endif
    #if CYTHON_FAST_PYCCALL
    if (__Pyx_PyFastCFunction_Check(function)) {
        PyObject *args[2] = {arg1, arg2};
        return __Pyx_PyCFunction_FastCall(function, args, 2);
    }
    #endif
    args = PyTuple_New(2);
    if (unlikely(!args)) goto done;
    Py_INCREF(arg1);
    PyTuple_SET_ITEM(args, 0, arg1);
    Py_INCREF(arg2);
    PyTuple_SET_ITEM(args, 1, arg2);
    Py_INCREF(function);
    result = __Pyx_PyObject_Call(function, args, NULL);
    Py_DECREF(args);
    Py_DECREF(function);
done:
    return result;
}

/* PyIntBinop */
#if !CYTHON_COMPILING_IN_PYPY
static PyObject* __Pyx_PyInt_AddObjC(PyObject *op1, PyObject *op2, CYTHON_UNUSED long intval, int
inplace, int zerodivision_check) {
    (void)inplace;
    (void)zerodivision_check;
    #if PY_MAJOR_VERSION < 3
    if (likely(PyInt_CheckExact(op1))) {
        const long b = intval;
        long x;
        long a = PyInt_AS_LONG(op1);
            x = (long)((unsigned long)a + b);
            if (likely((x^a) >= 0 || (x^b) >= 0))
                return PyInt_FromLong(x);
            return PyLong_Type.tp_as_number->nb_add(op1, op2);
    }
    #endif
    #if CYTHON_USE_PYLONG_INTERNALS
    if (likely(PyLong_CheckExact(op1))) {
        const long b = intval;
        long a, x;
#ifdef HAVE_LONG_LONG
        const PY_LONG_LONG llb = intval;
        PY_LONG_LONG lla, llx;
#endif
        const digit* digits = ((PyLongObject*)op1)->ob_digit;
        const Py_ssize_t size = Py_SIZE(op1);
        if (likely(__Pyx_sst_abs(size) <= 1)) {
            a = likely(size) ? digits[0] : 0;
            if (size == -1) a = -a;
        } else {
            switch (size) {
```

```
                case -2:
                    if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
                        a = -(long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 2 * PyLong_SHIFT) {
                        lla = -(PY_LONG_LONG) (((((unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                case 2:
                    if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
                        a = (long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 2 * PyLong_SHIFT) {
                        lla = (PY_LONG_LONG) (((((unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                case -3:
                    if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
                        a = -(long) (((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 3 * PyLong_SHIFT) {
                        lla = -(PY_LONG_LONG) (((((((unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                case 3:
                    if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
                        a = (long) (((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 3 * PyLong_SHIFT) {
                        lla = (PY_LONG_LONG) (((((((unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                case -4:
                    if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
                        a = -(long) (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 4 * PyLong_SHIFT) {
                        lla = -(PY_LONG_LONG) (((((((((unsigned PY_LONG_LONG)digits[3]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                case 4:
                    if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
                        a = (long) (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
                        break;
#ifdef HAVE_LONG_LONG
                    } else if (8 * sizeof(PY_LONG_LONG) - 1 > 4 * PyLong_SHIFT) {
                        lla = (PY_LONG_LONG) (((((((((unsigned PY_LONG_LONG)digits[3]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
                        goto long_long;
#endif
                    }
                    CYTHON_FALLTHROUGH;
                default: return PyLong_Type.tp_as_number->nb_add(op1, op2);
            }
        }
                x = a + b;
            return PyLong_FromLong(x);
```

```
#ifdef HAVE_LONG_LONG
        long_long:
                llx = lla + llb;
            return PyLong_FromLongLong(llx);
#endif


    }
    #endif
    if (PyFloat_CheckExact(op1)) {
        const long b = intval;
        double a = PyFloat_AS_DOUBLE(op1);
            double result;
            PyFPE_START_PROTECT("add", return NULL)
            result = ((double)a) + (double)b;
            PyFPE_END_PROTECT(result)
            return PyFloat_FromDouble(result);
    }
    return (inplace ? PyNumber_InPlaceAdd : PyNumber_Add)(op1, op2);
}
#endif

/* decode_c_bytes */
static CYTHON_INLINE PyObject* __Pyx_decode_c_bytes(
         const char* cstring, Py_ssize_t length, Py_ssize_t start, Py_ssize_t stop,
         const char* encoding, const char* errors,
         PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors)) {
    if (unlikely((start < 0) | (stop < 0))) {
        if (start < 0) {
            start += length;
            if (start < 0)
                start = 0;
        }
        if (stop < 0)
            stop += length;
    }
    if (stop > length)
        stop = length;
    if (unlikely(stop <= start))
        return __Pyx_NewRef(__pyx_empty_unicode);
    length = stop - start;
    cstring += start;
    if (decode_func) {
        return decode_func(cstring, length, errors);
    } else {
        return PyUnicode_Decode(cstring, length, encoding, errors);
    }
}

/* SwapException */
#if CYTHON_FAST_THREAD_STATE
static CYTHON_INLINE void __Pyx__ExceptionSwap(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject **tb) {
    PyObject *tmp_type, *tmp_value, *tmp_tb;
    #if CYTHON_USE_EXC_INFO_STACK
    _PyErr_StackItem *exc_info = tstate->exc_info;
    tmp_type = exc_info->exc_type;
    tmp_value = exc_info->exc_value;
    tmp_tb = exc_info->exc_traceback;
    exc_info->exc_type = *type;
    exc_info->exc_value = *value;
    exc_info->exc_traceback = *tb;
    #else
    tmp_type = tstate->exc_type;
    tmp_value = tstate->exc_value;
    tmp_tb = tstate->exc_traceback;
    tstate->exc_type = *type;
    tstate->exc_value = *value;
    tstate->exc_traceback = *tb;
    #endif
    *type = tmp_type;
    *value = tmp_value;
    *tb = tmp_tb;
}
#else
static CYTHON_INLINE void __Pyx_ExceptionSwap(PyObject **type, PyObject **value, PyObject **tb) {
    PyObject *tmp_type, *tmp_value, *tmp_tb;
    PyErr_GetExcInfo(&tmp_type, &tmp_value, &tmp_tb);
    PyErr_SetExcInfo(*type, *value, *tb);
    *type = tmp_type;
    *value = tmp_value;
    *tb = tmp_tb;
}
#endif

/* SetItemInt */
static int __Pyx_SetItemInt_Generic(PyObject *o, PyObject *j, PyObject *v) {
```

```
    int r;
    if (!j) return -1;
    r = PyObject_SetItem(o, j, v);
    Py_DECREF(j);
    return r;
}
static CYTHON_INLINE int __Pyx_SetItemInt_Fast(PyObject *o, Py_ssize_t i, PyObject *v, int is_list,
                                               CYTHON_NCP_UNUSED int wraparound, CYTHON_NCP_UNUSED int
    boundscheck) {
#if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS && CYTHON_USE_TYPE_SLOTS
    if (is_list || PyList_CheckExact(o)) {
        Py_ssize_t n = (!wraparound) ? i : ((likely(i >= 0)) ? i : i + PyList_GET_SIZE(o));
        if ((!boundscheck) || likely(__Pyx_is_valid_index(n, PyList_GET_SIZE(o)))) {
            PyObject* old = PyList_GET_ITEM(o, n);
            Py_INCREF(v);
            PyList_SET_ITEM(o, n, v);
            Py_DECREF(old);
            return 1;
        }
    } else {
        PySequenceMethods *m = Py_TYPE(o)->tp_as_sequence;
        if (likely(m && m->sq_ass_item)) {
            if (wraparound && unlikely(i < 0) && likely(m->sq_length)) {
                Py_ssize_t l = m->sq_length(o);
                if (likely(l >= 0)) {
                    i += l;
                } else {
                    if (!PyErr_ExceptionMatches(PyExc_OverflowError))
                        return -1;
                    PyErr_Clear();
                }
            }
            return m->sq_ass_item(o, i, v);
        }
    }
#else
#if CYTHON_COMPILING_IN_PYPY
    if (is_list || (PySequence_Check(o) && !PyDict_Check(o)))
#else
    if (is_list || PySequence_Check(o))
#endif
    {
        return PySequence_SetItem(o, i, v);
    }
#endif
    return __Pyx_SetItemInt_Generic(o, PyInt_FromSsize_t(i), v);
}

/* ArgTypeTest */
static int __Pyx__ArgTypeTest(PyObject *obj, PyTypeObject *type, const char *name, int exact)
{
    if (unlikely(!type)) {
        PyErr_SetString(PyExc_SystemError, "Missing type object");
        return 0;
    }
    else if (exact) {
        #if PY_MAJOR_VERSION == 2
        if ((type == &PyBaseString_Type) && likely(__Pyx_PyBaseString_CheckExact(obj))) return 1;
        #endif
    }
    else {
        if (likely(__Pyx_TypeCheck(obj, type))) return 1;
    }
    PyErr_Format(PyExc_TypeError,
        "Argument '%.200s' has incorrect type (expected %.200s, got %.200s)",
        name, type->tp_name, Py_TYPE(obj)->tp_name);
    return 0;
}

/* Import */
static PyObject *__Pyx_Import(PyObject *name, PyObject *from_list, int level) {
    PyObject *empty_list = 0;
    PyObject *module = 0;
    PyObject *global_dict = 0;
    PyObject *empty_dict = 0;
    PyObject *list;
    #if PY_MAJOR_VERSION < 3
    PyObject *py_import;
    py_import = __Pyx_PyObject_GetAttrStr(__pyx_b, __pyx_n_s_import);
    if (!py_import)
        goto bad;
    #endif
    if (from_list)
        list = from_list;
    else {
        empty_list = PyList_New(0);
        if (!empty_list)
```

```
            goto bad;
        list = empty_list;
    }
    global_dict = PyModule_GetDict(__pyx_m);
    if (!global_dict)
        goto bad;
    empty_dict = PyDict_New();
    if (!empty_dict)
        goto bad;
    {
        #if PY_MAJOR_VERSION >= 3
        if (level == -1) {
            if ((1) && (strchr(__Pyx_MODULE_NAME, '.'))) {
                module = PyImport_ImportModuleLevelObject(
                    name, global_dict, empty_dict, list, 1);
                if (!module) {
                    if (!PyErr_ExceptionMatches(PyExc_ImportError))
                        goto bad;
                    PyErr_Clear();
                }
            }
            level = 0;
        }
        #endif
        if (!module) {
            #if PY_MAJOR_VERSION < 3
            PyObject *py_level = PyInt_FromLong(level);
            if (!py_level)
                goto bad;
            module = PyObject_CallFunctionObjArgs(py_import,
                name, global_dict, empty_dict, list, py_level, (PyObject *)NULL);
            Py_DECREF(py_level);
            #else
            module = PyImport_ImportModuleLevelObject(
                name, global_dict, empty_dict, list, level);
            #endif
        }
    }
bad:
    #if PY_MAJOR_VERSION < 3
    Py_XDECREF(py_import);
    #endif
    Py_XDECREF(empty_list);
    Py_XDECREF(empty_dict);
    return module;
}

/* PyObject_GenericGetAttrNoDict */
#if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
static PyObject *__Pyx_RaiseGenericGetAttributeError(PyTypeObject *tp, PyObject *attr_name) {
    PyErr_Format(PyExc_AttributeError,
#if PY_MAJOR_VERSION >= 3
                 "'%.50s' object has no attribute '%U'",
                 tp->tp_name, attr_name);
#else
                 "'%.50s' object has no attribute '%.400s'",
                 tp->tp_name, PyString_AS_STRING(attr_name));
#endif
    return NULL;
}
static CYTHON_INLINE PyObject* __Pyx_PyObject_GenericGetAttrNoDict(PyObject* obj, PyObject* attr_name)
{
    PyObject *descr;
    PyTypeObject *tp = Py_TYPE(obj);
    if (unlikely(!PyString_Check(attr_name))) {
        return PyObject_GenericGetAttr(obj, attr_name);
    }
    assert(!tp->tp_dictoffset);
    descr = _PyType_Lookup(tp, attr_name);
    if (unlikely(!descr)) {
        return __Pyx_RaiseGenericGetAttributeError(tp, attr_name);
    }
    Py_INCREF(descr);
    #if PY_MAJOR_VERSION < 3
    if (likely(PyType_HasFeature(Py_TYPE(descr), Py_TPFLAGS_HAVE_CLASS)))
    #endif
    {
        descrgetfunc f = Py_TYPE(descr)->tp_descr_get;
        if (unlikely(f)) {
            PyObject *res = f(descr, obj, (PyObject *)tp);
            Py_DECREF(descr);
            return res;
        }
    }
    return descr;
}
#endif
```

```
/* PyObject_GenericGetAttr */
#if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
static PyObject* __Pyx_PyObject_GenericGetAttr(PyObject* obj, PyObject* attr_name) {
    if (unlikely(Py_TYPE(obj)->tp_dictoffset)) {
        return PyObject_GenericGetAttr(obj, attr_name);
    }
    return __Pyx_PyObject_GenericGetAttrNoDict(obj, attr_name);
}
#endif

/* SetVTable */
static int __Pyx_SetVtable(PyObject *dict, void *vtable) {
#if PY_VERSION_HEX >= 0x02070000
    PyObject *ob = PyCapsule_New(vtable, 0, 0);
#else
    PyObject *ob = PyCObject_FromVoidPtr(vtable, 0);
#endif
    if (!ob)
        goto bad;
    if (PyDict_SetItem(dict, __pyx_n_s_pyx_vtable, ob) < 0)
        goto bad;
    Py_DECREF(ob);
    return 0;
bad:
    Py_XDECREF(ob);
    return -1;
}

/* PyObjectGetAttrStrNoError */
static void __Pyx_PyObject_GetAttrStr_ClearAttributeError(void) {
    __Pyx_PyThreadState_declare
    __Pyx_PyThreadState_assign
    if (likely(__Pyx_PyErr_ExceptionMatches(PyExc_AttributeError)))
        __Pyx_PyErr_Clear();
}
static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStrNoError(PyObject* obj, PyObject* attr_name) {
    PyObject *result;
#if CYTHON_COMPILING_IN_CPYTHON && CYTHON_USE_TYPE_SLOTS && PY_VERSION_HEX >= 0x030700B1
    PyTypeObject* tp = Py_TYPE(obj);
    if (likely(tp->tp_getattro == PyObject_GenericGetAttr)) {
        return _PyObject_GenericGetAttrWithDict(obj, attr_name, NULL, 1);
    }
#endif
    result = __Pyx_PyObject_GetAttrStr(obj, attr_name);
    if (unlikely(!result)) {
        __Pyx_PyObject_GetAttrStr_ClearAttributeError();
    }
    return result;
}

/* SetupReduce */
static int __Pyx_setup_reduce_is_named(PyObject* meth, PyObject* name) {
  int ret;
  PyObject *name_attr;
  name_attr = __Pyx_PyObject_GetAttrStr(meth, __pyx_n_s_name);
  if (likely(name_attr)) {
      ret = PyObject_RichCompareBool(name_attr, name, Py_EQ);
  } else {
      ret = -1;
  }
  if (unlikely(ret < 0)) {
      PyErr_Clear();
      ret = 0;
  }
  Py_XDECREF(name_attr);
  return ret;
}
static int __Pyx_setup_reduce(PyObject* type_obj) {
    int ret = 0;
    PyObject *object_reduce = NULL;
    PyObject *object_reduce_ex = NULL;
    PyObject *reduce = NULL;
    PyObject *reduce_ex = NULL;
    PyObject *reduce_cython = NULL;
    PyObject *setstate = NULL;
    PyObject *setstate_cython = NULL;
#if CYTHON_USE_PYTYPE_LOOKUP
    if (_PyType_Lookup((PyTypeObject*)type_obj, __pyx_n_s_getstate)) goto __PYX_GOOD;
#else
    if (PyObject_HasAttr(type_obj, __pyx_n_s_getstate)) goto __PYX_GOOD;
#endif
#if CYTHON_USE_PYTYPE_LOOKUP
    object_reduce_ex = _PyType_Lookup(&PyBaseObject_Type, __pyx_n_s_reduce_ex); if (!object_reduce_ex)
goto __PYX_BAD;
#else
    object_reduce_ex = __Pyx_PyObject_GetAttrStr((PyObject*)&PyBaseObject_Type, __pyx_n_s_reduce_ex);
```

```
    if (!object_reduce_ex) goto __PYX_BAD;
#endif
    reduce_ex = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_reduce_ex); if (unlikely(!reduce_ex)) goto __PYX_BAD;
    if (reduce_ex == object_reduce_ex) {
#if CYTHON_USE_PYTYPE_LOOKUP
        object_reduce = _PyType_Lookup(&PyBaseObject_Type, __pyx_n_s_reduce); if (!object_reduce) goto __PYX_BAD;
#else
        object_reduce = __Pyx_PyObject_GetAttrStr((PyObject*)&PyBaseObject_Type, __pyx_n_s_reduce); if (!object_reduce) goto __PYX_BAD;
#endif
        reduce = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_reduce); if (unlikely(!reduce)) goto __PYX_BAD;
        if (reduce == object_reduce || __Pyx_setup_reduce_is_named(reduce, __pyx_n_s_reduce_cython)) {
            reduce_cython = __Pyx_PyObject_GetAttrStrNoError(type_obj, __pyx_n_s_reduce_cython);
            if (likely(reduce_cython)) {
                ret = PyDict_SetItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_reduce, reduce_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
                ret = PyDict_DelItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_reduce_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
            } else if (reduce == object_reduce || PyErr_Occurred()) {
                goto __PYX_BAD;
            }
            setstate = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_setstate);
            if (!setstate) PyErr_Clear();
            if (!setstate || __Pyx_setup_reduce_is_named(setstate, __pyx_n_s_setstate_cython)) {
                setstate_cython = __Pyx_PyObject_GetAttrStrNoError(type_obj, __pyx_n_s_setstate_cython);
                if (likely(setstate_cython)) {
                    ret = PyDict_SetItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_setstate, setstate_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
                    ret = PyDict_DelItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_setstate_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
                } else if (!setstate || PyErr_Occurred()) {
                    goto __PYX_BAD;
                }
            }
            PyType_Modified((PyTypeObject*)type_obj);
        }
    }
    goto __PYX_GOOD;
__PYX_BAD:
    if (!PyErr_Occurred())
        PyErr_Format(PyExc_RuntimeError, "Unable to initialize pickling for %s", ((PyTypeObject*)type_obj)->tp_name);
    ret = -1;
__PYX_GOOD:
#if !CYTHON_USE_PYTYPE_LOOKUP
    Py_XDECREF(object_reduce);
    Py_XDECREF(object_reduce_ex);
#endif
    Py_XDECREF(reduce);
    Py_XDECREF(reduce_ex);
    Py_XDECREF(reduce_cython);
    Py_XDECREF(setstate);
    Py_XDECREF(setstate_cython);
    return ret;
}

/* BytesEquals */
static CYTHON_INLINE int __Pyx_PyBytes_Equals(PyObject* s1, PyObject* s2, int equals) {
#if CYTHON_COMPILING_IN_PYPY
    return PyObject_RichCompareBool(s1, s2, equals);
#else
    if (s1 == s2) {
        return (equals == Py_EQ);
    } else if (PyBytes_CheckExact(s1) & PyBytes_CheckExact(s2)) {
        const char *ps1, *ps2;
        Py_ssize_t length = PyBytes_GET_SIZE(s1);
        if (length != PyBytes_GET_SIZE(s2))
            return (equals == Py_NE);
        ps1 = PyBytes_AS_STRING(s1);
        ps2 = PyBytes_AS_STRING(s2);
        if (ps1[0] != ps2[0]) {
            return (equals == Py_NE);
        } else if (length == 1) {
            return (equals == Py_EQ);
        } else {
            int result;
#if CYTHON_USE_UNICODE_INTERNALS
            Py_hash_t hash1, hash2;
            hash1 = ((PyBytesObject*)s1)->ob_shash;
            hash2 = ((PyBytesObject*)s2)->ob_shash;
            if (hash1 != hash2 && hash1 != -1 && hash2 != -1) {
                return (equals == Py_NE);
            }
```

```
#endif
            result = memcmp(ps1, ps2, (size_t)length);
            return (equals == Py_EQ) ? (result == 0) : (result != 0);
        }
    } else if ((s1 == Py_None) & PyBytes_CheckExact(s2)) {
        return (equals == Py_NE);
    } else if ((s2 == Py_None) & PyBytes_CheckExact(s1)) {
        return (equals == Py_NE);
    } else {
        int result;
        PyObject* py_result = PyObject_RichCompare(s1, s2, equals);
        if (!py_result)
            return -1;
        result = __Pyx_PyObject_IsTrue(py_result);
        Py_DECREF(py_result);
        return result;
    }
#endif
}

/* UnicodeEquals */
static CYTHON_INLINE int __Pyx_PyUnicode_Equals(PyObject* s1, PyObject* s2, int equals) {
#if CYTHON_COMPILING_IN_PYPY
    return PyObject_RichCompareBool(s1, s2, equals);
#else
#if PY_MAJOR_VERSION < 3
    PyObject* owned_ref = NULL;
#endif
    int s1_is_unicode, s2_is_unicode;
    if (s1 == s2) {
        goto return_eq;
    }
    s1_is_unicode = PyUnicode_CheckExact(s1);
    s2_is_unicode = PyUnicode_CheckExact(s2);
#if PY_MAJOR_VERSION < 3
    if ((s1_is_unicode & (!s2_is_unicode)) && PyString_CheckExact(s2)) {
        owned_ref = PyUnicode_FromObject(s2);
        if (unlikely(!owned_ref))
            return -1;
        s2 = owned_ref;
        s2_is_unicode = 1;
    } else if ((s2_is_unicode & (!s1_is_unicode)) && PyString_CheckExact(s1)) {
        owned_ref = PyUnicode_FromObject(s1);
        if (unlikely(!owned_ref))
            return -1;
        s1 = owned_ref;
        s1_is_unicode = 1;
    } else if (((!s2_is_unicode) & (!s1_is_unicode))) {
        return __Pyx_PyBytes_Equals(s1, s2, equals);
    }
#endif
    if (s1_is_unicode & s2_is_unicode) {
        Py_ssize_t length;
        int kind;
        void *data1, *data2;
        if (unlikely(__Pyx_PyUnicode_READY(s1) < 0) || unlikely(__Pyx_PyUnicode_READY(s2) < 0))
            return -1;
        length = __Pyx_PyUnicode_GET_LENGTH(s1);
        if (length != __Pyx_PyUnicode_GET_LENGTH(s2)) {
            goto return_ne;
        }
#if CYTHON_USE_UNICODE_INTERNALS
        {
            Py_hash_t hash1, hash2;
        #if CYTHON_PEP393_ENABLED
            hash1 = ((PyASCIIObject*)s1)->hash;
            hash2 = ((PyASCIIObject*)s2)->hash;
        #else
            hash1 = ((PyUnicodeObject*)s1)->hash;
            hash2 = ((PyUnicodeObject*)s2)->hash;
        #endif
            if (hash1 != hash2 && hash1 != -1 && hash2 != -1) {
                goto return_ne;
            }
        }
#endif
        kind = __Pyx_PyUnicode_KIND(s1);
        if (kind != __Pyx_PyUnicode_KIND(s2)) {
            goto return_ne;
        }
        data1 = __Pyx_PyUnicode_DATA(s1);
        data2 = __Pyx_PyUnicode_DATA(s2);
        if (__Pyx_PyUnicode_READ(kind, data1, 0) != __Pyx_PyUnicode_READ(kind, data2, 0)) {
            goto return_ne;
        } else if (length == 1) {
            goto return_eq;
        } else {
```

```
            int result = memcmp(data1, data2, (size_t)(length * kind));
            #if PY_MAJOR_VERSION < 3
            Py_XDECREF(owned_ref);
            #endif
            return (equals == Py_EQ) ? (result == 0) : (result != 0);
        }
    } else if ((s1 == Py_None) & s2_is_unicode) {
        goto return_ne;
    } else if ((s2 == Py_None) & s1_is_unicode) {
        goto return_ne;
    } else {
        int result;
        PyObject* py_result = PyObject_RichCompare(s1, s2, equals);
        #if PY_MAJOR_VERSION < 3
        Py_XDECREF(owned_ref);
        #endif
        if (!py_result)
            return -1;
        result = __Pyx_PyObject_IsTrue(py_result);
        Py_DECREF(py_result);
        return result;
    }
return_eq:
    #if PY_MAJOR_VERSION < 3
    Py_XDECREF(owned_ref);
    #endif
    return (equals == Py_EQ);
return_ne:
    #if PY_MAJOR_VERSION < 3
    Py_XDECREF(owned_ref);
    #endif
    return (equals == Py_NE);
#endif
}

/* CLineInTraceback */
#ifndef CYTHON_CLINE_IN_TRACEBACK
static int __Pyx_CLineForTraceback(CYTHON_NCP_UNUSED PyThreadState *tstate, int c_line) {
    PyObject *use_cline;
    PyObject *ptype, *pvalue, *ptraceback;
#if CYTHON_COMPILING_IN_CPYTHON
    PyObject **cython_runtime_dict;
#endif
    if (unlikely(!__pyx_cython_runtime)) {
        return c_line;
    }
    __Pyx_ErrFetchInState(tstate, &ptype, &pvalue, &ptraceback);
#if CYTHON_COMPILING_IN_CPYTHON
    cython_runtime_dict = _PyObject_GetDictPtr(__pyx_cython_runtime);
    if (likely(cython_runtime_dict)) {
        __PYX_PY_DICT_LOOKUP_IF_MODIFIED(
            use_cline, *cython_runtime_dict,
            __Pyx_PyDict_GetItemStr(*cython_runtime_dict, __pyx_n_s_cline_in_traceback))
    } else
#endif
    {
      PyObject *use_cline_obj = __Pyx_PyObject_GetAttrStr(__pyx_cython_runtime,
__pyx_n_s_cline_in_traceback);
      if (use_cline_obj) {
        use_cline = PyObject_Not(use_cline_obj) ? Py_False : Py_True;
        Py_DECREF(use_cline_obj);
      } else {
        PyErr_Clear();
        use_cline = NULL;
      }
    }
    if (!use_cline) {
        c_line = 0;
        (void) PyObject_SetAttr(__pyx_cython_runtime, __pyx_n_s_cline_in_traceback, Py_False);
    }
    else if (use_cline == Py_False || (use_cline != Py_True && PyObject_Not(use_cline) != 0)) {
        c_line = 0;
    }
    __Pyx_ErrRestoreInState(tstate, ptype, pvalue, ptraceback);
    return c_line;
}
#endif

/* CodeObjectCache */
static int __pyx_bisect_code_objects(__Pyx_CodeObjectCacheEntry* entries, int count, int code_line) {
    int start = 0, mid = 0, end = count - 1;
    if (end >= 0 && code_line > entries[end].code_line) {
        return count;
    }
    while (start < end) {
        mid = start + (end - start) / 2;
        if (code_line < entries[mid].code_line) {
```

```
            end = mid;
        } else if (code_line > entries[mid].code_line) {
             start = mid + 1;
        } else {
            return mid;
        }
    }
    if (code_line <= entries[mid].code_line) {
        return mid;
    } else {
        return mid + 1;
    }
}
static PyCodeObject *__pyx_find_code_object(int code_line) {
    PyCodeObject* code_object;
    int pos;
    if (unlikely(!code_line) || unlikely(!__pyx_code_cache.entries)) {
        return NULL;
    }
    pos = __pyx_bisect_code_objects(__pyx_code_cache.entries, __pyx_code_cache.count, code_line);
    if (unlikely(pos >= __pyx_code_cache.count) || unlikely(__pyx_code_cache.entries[pos].code_line != code_line)) {
        return NULL;
    }
    code_object = __pyx_code_cache.entries[pos].code_object;
    Py_INCREF(code_object);
    return code_object;
}
static void __pyx_insert_code_object(int code_line, PyCodeObject* code_object) {
    int pos, i;
    __Pyx_CodeObjectCacheEntry* entries = __pyx_code_cache.entries;
    if (unlikely(!code_line)) {
        return;
    }
    if (unlikely(!entries)) {
        entries = (__Pyx_CodeObjectCacheEntry*)PyMem_Malloc(64*sizeof(__Pyx_CodeObjectCacheEntry));
        if (likely(entries)) {
            __pyx_code_cache.entries = entries;
            __pyx_code_cache.max_count = 64;
            __pyx_code_cache.count = 1;
            entries[0].code_line = code_line;
            entries[0].code_object = code_object;
            Py_INCREF(code_object);
        }
        return;
    }
    pos = __pyx_bisect_code_objects(__pyx_code_cache.entries, __pyx_code_cache.count, code_line);
    if ((pos < __pyx_code_cache.count) && unlikely(__pyx_code_cache.entries[pos].code_line == code_line)) {
        PyCodeObject* tmp = entries[pos].code_object;
        entries[pos].code_object = code_object;
        Py_DECREF(tmp);
        return;
    }
    if (__pyx_code_cache.count == __pyx_code_cache.max_count) {
        int new_max = __pyx_code_cache.max_count + 64;
        entries = (__Pyx_CodeObjectCacheEntry*)PyMem_Realloc(
            __pyx_code_cache.entries, ((size_t)new_max) * sizeof(__Pyx_CodeObjectCacheEntry));
        if (unlikely(!entries)) {
            return;
        }
        __pyx_code_cache.entries = entries;
        __pyx_code_cache.max_count = new_max;
    }
    for (i=__pyx_code_cache.count; i>pos; i--) {
        entries[i] = entries[i-1];
    }
    entries[pos].code_line = code_line;
    entries[pos].code_object = code_object;
    __pyx_code_cache.count++;
    Py_INCREF(code_object);
}

/* AddTraceback */
#include "compile.h"
#include "frameobject.h"
#include "traceback.h"
static PyCodeObject* __Pyx_CreateCodeObjectForTraceback(
            const char *funcname, int c_line,
            int py_line, const char *filename) {
    PyCodeObject *py_code = NULL;
    PyObject *py_funcname = NULL;
    #if PY_MAJOR_VERSION < 3
    PyObject *py_srcfile = NULL;
    py_srcfile = PyString_FromString(filename);
    if (!py_srcfile) goto bad;
    #endif
```

```
    if (c_line) {
        #if PY_MAJOR_VERSION < 3
        py_funcname = PyString_FromFormat( "%s (%s:%d)", funcname, __pyx_cfilenm, c_line);
        if (!py_funcname) goto bad;
        #else
        py_funcname = PyUnicode_FromFormat( "%s (%s:%d)", funcname, __pyx_cfilenm, c_line);
        if (!py_funcname) goto bad;
        funcname = PyUnicode_AsUTF8(py_funcname);
        if (!funcname) goto bad;
        #endif
    }
    else {
        #if PY_MAJOR_VERSION < 3
        py_funcname = PyString_FromString(funcname);
        if (!py_funcname) goto bad;
        #endif
    }
    #if PY_MAJOR_VERSION < 3
    py_code = __Pyx_PyCode_New(
        0,
        0,
        0,
        0,
        0,
        __pyx_empty_bytes, /*PyObject *code,*/
        __pyx_empty_tuple, /*PyObject *consts,*/
        __pyx_empty_tuple, /*PyObject *names,*/
        __pyx_empty_tuple, /*PyObject *varnames,*/
        __pyx_empty_tuple, /*PyObject *freevars,*/
        __pyx_empty_tuple, /*PyObject *cellvars,*/
        py_srcfile,   /*PyObject *filename,*/
        py_funcname,  /*PyObject *name,*/
        py_line,
        __pyx_empty_bytes  /*PyObject *lnotab*/
    );
    Py_DECREF(py_srcfile);
    #else
    py_code = PyCode_NewEmpty(filename, funcname, py_line);
    #endif
    Py_XDECREF(py_funcname);  // XDECREF since it's only set on Py3 if cline
    return py_code;
bad:
    Py_XDECREF(py_funcname);
    #if PY_MAJOR_VERSION < 3
    Py_XDECREF(py_srcfile);
    #endif
    return NULL;
}
static void __Pyx_AddTraceback(const char *funcname, int c_line,
                               int py_line, const char *filename) {
    PyCodeObject *py_code = 0;
    PyFrameObject *py_frame = 0;
    PyThreadState *tstate = __Pyx_PyThreadState_Current;
    if (c_line) {
        c_line = __Pyx_CLineForTraceback(tstate, c_line);
    }
    py_code = __pyx_find_code_object(c_line ? -c_line : py_line);
    if (!py_code) {
        py_code = __Pyx_CreateCodeObjectForTraceback(
            funcname, c_line, py_line, filename);
        if (!py_code) goto bad;
        __pyx_insert_code_object(c_line ? -c_line : py_line, py_code);
    }
    py_frame = PyFrame_New(
        tstate,            /*PyThreadState *tstate,*/
        py_code,           /*PyCodeObject *code,*/
        __pyx_d,    /*PyObject *globals,*/
        0                  /*PyObject *locals*/
    );
    if (!py_frame) goto bad;
    __Pyx_PyFrame_SetLineNumber(py_frame, py_line);
    PyTraceBack_Here(py_frame);
bad:
    Py_XDECREF(py_code);
    Py_XDECREF(py_frame);
}

/* CIntFromPyVerify */
#define __PYX_VERIFY_RETURN_INT(target_type, func_type, func_value)\
    __PYX__VERIFY_RETURN_INT(target_type, func_type, func_value, 0)
#define __PYX_VERIFY_RETURN_INT_EXC(target_type, func_type, func_value)\
    __PYX__VERIFY_RETURN_INT(target_type, func_type, func_value, 1)
#define __PYX__VERIFY_RETURN_INT(target_type, func_type, func_value, exc)\
    {\
        func_type value = func_value;\
        if (sizeof(target_type) < sizeof(func_type)) {\
            if (unlikely(value != (func_type) (target_type) value)) {\
```

```
                func_type zero = 0;\
                if (exc && unlikely(value == (func_type)-1 && PyErr_Occurred()))\
                    return (target_type) -1;\
                if (is_unsigned && unlikely(value < zero))\
                    goto raise_neg_overflow;\
                else\
                    goto raise_overflow;\
            }\
        }\
        return (target_type) value;\
    }

/* CIntFromPy */
static CYTHON_INLINE int __Pyx_PyInt_As_int(PyObject *x) {
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wconversion"
#endif
    const int neg_one = (int) -1, const_zero = (int) 0;
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic pop
#endif
    const int is_unsigned = neg_one > const_zero;
#if PY_MAJOR_VERSION < 3
    if (likely(PyInt_Check(x))) {
        if (sizeof(int) < sizeof(long)) {
            __PYX_VERIFY_RETURN_INT(int, long, PyInt_AS_LONG(x))
        } else {
            long val = PyInt_AS_LONG(x);
            if (is_unsigned && unlikely(val < 0)) {
                goto raise_neg_overflow;
            }
            return (int) val;
        }
    } else
#endif
    if (likely(PyLong_Check(x))) {
        if (is_unsigned) {
#if CYTHON_USE_PYLONG_INTERNALS
            const digit* digits = ((PyLongObject*)x)->ob_digit;
            switch (Py_SIZE(x)) {
                case  0: return (int) 0;
                case  1: __PYX_VERIFY_RETURN_INT(int, digit, digits[0])
                case 2:
                    if (8 * sizeof(int) > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((unsigned long)digits[1])
     « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) >= 2 * PyLong_SHIFT) {
                            return (int) (((((int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
                case 3:
                    if (8 * sizeof(int) > 2 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((unsigned
     long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
     long)digits[0])))
                        } else if (8 * sizeof(int) >= 3 * PyLong_SHIFT) {
                            return (int) (((((((int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) «
     PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
                case 4:
                    if (8 * sizeof(int) > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((((unsigned
     long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
     long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) >= 4 * PyLong_SHIFT) {
                            return (int) (((((((((int)digits[3]) « PyLong_SHIFT) | (int)digits[2]) «
     PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
            }
#endif
#if CYTHON_COMPILING_IN_CPYTHON
            if (unlikely(Py_SIZE(x) < 0)) {
                goto raise_neg_overflow;
            }
#else
            {
                int result = PyObject_RichCompareBool(x, Py_False, Py_LT);
                if (unlikely(result < 0))
```

```
                    return (int) -1;
                if (unlikely(result == 1))
                    goto raise_neg_overflow;
            }
#endif
            if (sizeof(int) <= sizeof(unsigned long)) {
                __PYX_VERIFY_RETURN_INT_EXC(int, unsigned long, PyLong_AsUnsignedLong(x))
#ifdef HAVE_LONG_LONG
            } else if (sizeof(int) <= sizeof(unsigned PY_LONG_LONG)) {
                __PYX_VERIFY_RETURN_INT_EXC(int, unsigned PY_LONG_LONG, PyLong_AsUnsignedLongLong(x))
#endif
            }
        } else {
#if CYTHON_USE_PYLONG_INTERNALS
            const digit* digits = ((PyLongObject*)x)->ob_digit;
            switch (Py_SIZE(x)) {
                case  0: return (int) 0;
                case -1: __PYX_VERIFY_RETURN_INT(int, sdigit, (sdigit) (-(sdigit)digits[0]))
                case  1: __PYX_VERIFY_RETURN_INT(int,  digit, +digits[0])
                case -2:
                    if (8 * sizeof(int) - 1 > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, long, -(long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
                            return (int) (((int)-1)*(((((int)digits[1]) « PyLong_SHIFT) | (int)digits[0])));
                        }
                    }
                    break;
                case 2:
                    if (8 * sizeof(int) > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
                            return (int) (((((int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
                case -3:
                    if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, long, -(long) (((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
                            return (int) (((int)-1)*(((((((int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0])));
                        }
                    }
                    break;
                case 3:
                    if (8 * sizeof(int) > 2 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
                            return (int) (((((((int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
                case -4:
                    if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, long, -(long) (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 4 * PyLong_SHIFT) {
                            return (int) (((int)-1)*(((((((((int)digits[3]) « PyLong_SHIFT) | (int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0])));
                        }
                    }
                    break;
                case 4:
                    if (8 * sizeof(int) > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(int) - 1 > 4 * PyLong_SHIFT) {
                            return (int) (((((((((int)digits[3]) « PyLong_SHIFT) | (int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
                        }
                    }
                    break;
```

```
            }
#endif
            if (sizeof(int) <= sizeof(long)) {
                __PYX_VERIFY_RETURN_INT_EXC(int, long, PyLong_AsLong(x))
#ifdef HAVE_LONG_LONG
            } else if (sizeof(int) <= sizeof(PY_LONG_LONG)) {
                __PYX_VERIFY_RETURN_INT_EXC(int, PY_LONG_LONG, PyLong_AsLongLong(x))
#endif
            }
        }
        {
#if CYTHON_COMPILING_IN_PYPY && !defined(_PyLong_AsByteArray)
            PyErr_SetString(PyExc_RuntimeError,
                            "_PyLong_AsByteArray() not available in PyPy, cannot convert large numbers");
#else
            int val;
            PyObject *v = __Pyx_PyNumber_IntOrLong(x);
 #if PY_MAJOR_VERSION < 3
            if (likely(v) && !PyLong_Check(v)) {
                PyObject *tmp = v;
                v = PyNumber_Long(tmp);
                Py_DECREF(tmp);
            }
 #endif
            if (likely(v)) {
                int one = 1; int is_little = (int)*(unsigned char *)&one;
                unsigned char *bytes = (unsigned char *)&val;
                int ret = _PyLong_AsByteArray((PyLongObject *)v,
                                              bytes, sizeof(val),
                                              is_little, !is_unsigned);
                Py_DECREF(v);
                if (likely(!ret))
                    return val;
            }
#endif
            return (int) -1;
        }
    } else {
        int val;
        PyObject *tmp = __Pyx_PyNumber_IntOrLong(x);
        if (!tmp) return (int) -1;
        val = __Pyx_PyInt_As_int(tmp);
        Py_DECREF(tmp);
        return val;
    }
raise_overflow:
    PyErr_SetString(PyExc_OverflowError,
        "value too large to convert to int");
    return (int) -1;
raise_neg_overflow:
    PyErr_SetString(PyExc_OverflowError,
        "can't convert negative value to int");
    return (int) -1;
}

/* CIntToPy */
static CYTHON_INLINE PyObject* __Pyx_PyInt_From_int(int value) {
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wconversion"
#endif
    const int neg_one = (int) -1, const_zero = (int) 0;
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic pop
#endif
    const int is_unsigned = neg_one > const_zero;
    if (is_unsigned) {
        if (sizeof(int) < sizeof(long)) {
            return PyInt_FromLong((long) value);
        } else if (sizeof(int) <= sizeof(unsigned long)) {
            return PyLong_FromUnsignedLong((unsigned long) value);
#ifdef HAVE_LONG_LONG
        } else if (sizeof(int) <= sizeof(unsigned PY_LONG_LONG)) {
            return PyLong_FromUnsignedLongLong((unsigned PY_LONG_LONG) value);
#endif
        }
    } else {
        if (sizeof(int) <= sizeof(long)) {
            return PyInt_FromLong((long) value);
#ifdef HAVE_LONG_LONG
        } else if (sizeof(int) <= sizeof(PY_LONG_LONG)) {
            return PyLong_FromLongLong((PY_LONG_LONG) value);
#endif
        }
    }
    {
```

```
        int one = 1; int little = (int)*(unsigned char *)&one;
        unsigned char *bytes = (unsigned char *)&value;
        return _PyLong_FromByteArray(bytes, sizeof(int),
                                     little, !is_unsigned);
    }
}

/* CIntToPy */
static CYTHON_INLINE PyObject* __Pyx_PyInt_From_long(long value) {
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wconversion"
#endif
    const long neg_one = (long) -1, const_zero = (long) 0;
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic pop
#endif
    const int is_unsigned = neg_one > const_zero;
    if (is_unsigned) {
        if (sizeof(long) < sizeof(long)) {
            return PyInt_FromLong((long) value);
        } else if (sizeof(long) <= sizeof(unsigned long)) {
            return PyLong_FromUnsignedLong((unsigned long) value);
#ifdef HAVE_LONG_LONG
        } else if (sizeof(long) <= sizeof(unsigned PY_LONG_LONG)) {
            return PyLong_FromUnsignedLongLong((unsigned PY_LONG_LONG) value);
#endif
        }
    } else {
        if (sizeof(long) <= sizeof(long)) {
            return PyInt_FromLong((long) value);
#ifdef HAVE_LONG_LONG
        } else if (sizeof(long) <= sizeof(PY_LONG_LONG)) {
            return PyLong_FromLongLong((PY_LONG_LONG) value);
#endif
        }
    }
    {
        int one = 1; int little = (int)*(unsigned char *)&one;
        unsigned char *bytes = (unsigned char *)&value;
        return _PyLong_FromByteArray(bytes, sizeof(long),
                                     little, !is_unsigned);
    }
}

/* CIntFromPy */
static CYTHON_INLINE long __Pyx_PyInt_As_long(PyObject *x) {
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wconversion"
#endif
    const long neg_one = (long) -1, const_zero = (long) 0;
#ifdef __Pyx_HAS_GCC_DIAGNOSTIC
#pragma GCC diagnostic pop
#endif
    const int is_unsigned = neg_one > const_zero;
#if PY_MAJOR_VERSION < 3
    if (likely(PyInt_Check(x))) {
        if (sizeof(long) < sizeof(long)) {
            __PYX_VERIFY_RETURN_INT(long, long, PyInt_AS_LONG(x))
        } else {
            long val = PyInt_AS_LONG(x);
            if (is_unsigned && unlikely(val < 0)) {
                goto raise_neg_overflow;
            }
            return (long) val;
        }
    } else
#endif
    if (likely(PyLong_Check(x))) {
        if (is_unsigned) {
#if CYTHON_USE_PYLONG_INTERNALS
            const digit* digits = ((PyLongObject*)x)->ob_digit;
            switch (Py_SIZE(x)) {
                case  0: return (long) 0;
                case  1: __PYX_VERIFY_RETURN_INT(long, digit, digits[0])
                case 2:
                    if (8 * sizeof(long) > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((unsigned long)digits[1])
« PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) >= 2 * PyLong_SHIFT) {
                            return (long) (((((long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
                        }
                    }
                    break;
                case 3:
```

```
                    if (8 * sizeof(long) > 2 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) >= 3 * PyLong_SHIFT) {
                            return (long) (((((long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
                        }
                    }
                    break;
                case 4:
                    if (8 * sizeof(long) > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) >= 4 * PyLong_SHIFT) {
                            return (long) (((((((long)digits[3]) « PyLong_SHIFT) | (long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
                        }
                    }
                    break;
            }
#endif
#if CYTHON_COMPILING_IN_CPYTHON
            if (unlikely(Py_SIZE(x) < 0)) {
                goto raise_neg_overflow;
            }
#else
            {
                int result = PyObject_RichCompareBool(x, Py_False, Py_LT);
                if (unlikely(result < 0))
                    return (long) -1;
                if (unlikely(result == 1))
                    goto raise_neg_overflow;
            }
#endif
            if (sizeof(long) <= sizeof(unsigned long)) {
                __PYX_VERIFY_RETURN_INT_EXC(long, unsigned long, PyLong_AsUnsignedLong(x))
#ifdef HAVE_LONG_LONG
            } else if (sizeof(long) <= sizeof(unsigned PY_LONG_LONG)) {
                __PYX_VERIFY_RETURN_INT_EXC(long, unsigned PY_LONG_LONG, PyLong_AsUnsignedLongLong(x))
#endif
            }
        } else {
#if CYTHON_USE_PYLONG_INTERNALS
            const digit* digits = ((PyLongObject*)x)->ob_digit;
            switch (Py_SIZE(x)) {
                case  0: return (long) 0;
                case -1: __PYX_VERIFY_RETURN_INT(long, sdigit, (sdigit) (-(sdigit)digits[0]))
                case  1: __PYX_VERIFY_RETURN_INT(long,  digit, +digits[0])
                case -2:
                    if (8 * sizeof(long) - 1 > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
                            return (long) (((long)-1)*(((((long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
                case 2:
                    if (8 * sizeof(long) > 1 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, ((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
                            return (long) (((((long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
                case -3:
                    if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
                            return (long) (((long)-1)*(((((((long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
                case 3:
                    if (8 * sizeof(long) > 2 * PyLong_SHIFT) {
```

```
                        if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, ((((((unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
                            return (long) (((((((long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) «
PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
                case -4:
                    if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
                            return (long) (((long)-1)*(((((((((long)digits[3]) « PyLong_SHIFT) |
(long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
                case 4:
                    if (8 * sizeof(long) > 3 * PyLong_SHIFT) {
                        if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
                            __PYX_VERIFY_RETURN_INT(long, unsigned long, ((((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
                        } else if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
                            return (long) (((((((((long)digits[3]) « PyLong_SHIFT) | (long)digits[2])
« PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
                        }
                    }
                    break;
            }
#endif
            if (sizeof(long) <= sizeof(long)) {
                __PYX_VERIFY_RETURN_INT_EXC(long, long, PyLong_AsLong(x))
#ifdef HAVE_LONG_LONG
            } else if (sizeof(long) <= sizeof(PY_LONG_LONG)) {
                __PYX_VERIFY_RETURN_INT_EXC(long, PY_LONG_LONG, PyLong_AsLongLong(x))
#endif
            }
        }
        {
#if CYTHON_COMPILING_IN_PYPY && !defined(_PyLong_AsByteArray)
            PyErr_SetString(PyExc_RuntimeError,
                            "_PyLong_AsByteArray() not available in PyPy, cannot convert large
numbers");
#else
            long val;
            PyObject *v = __Pyx_PyNumber_IntOrLong(x);
 #if PY_MAJOR_VERSION < 3
            if (likely(v) && !PyLong_Check(v)) {
                PyObject *tmp = v;
                v = PyNumber_Long(tmp);
                Py_DECREF(tmp);
            }
 #endif
            if (likely(v)) {
                int one = 1; int is_little = (int)*(unsigned char *)&one;
                unsigned char *bytes = (unsigned char *)&val;
                int ret = _PyLong_AsByteArray((PyLongObject *)v,
                                              bytes, sizeof(val),
                                              is_little, !is_unsigned);
                Py_DECREF(v);
                if (likely(!ret))
                    return val;
            }
#endif
            return (long) -1;
        }
    } else {
        long val;
        PyObject *tmp = __Pyx_PyNumber_IntOrLong(x);
        if (!tmp) return (long) -1;
        val = __Pyx_PyInt_As_long(tmp);
        Py_DECREF(tmp);
        return val;
    }
raise_overflow:
    PyErr_SetString(PyExc_OverflowError,
        "value too large to convert to long");
    return (long) -1;
raise_neg_overflow:
    PyErr_SetString(PyExc_OverflowError,
        "can't convert negative value to long");
```

```
    return (long) -1;
}

/* FastTypeChecks */
#if CYTHON_COMPILING_IN_CPYTHON
static int __Pyx_InBases(PyTypeObject *a, PyTypeObject *b) {
    while (a) {
        a = a->tp_base;
        if (a == b)
            return 1;
    }
    return b == &PyBaseObject_Type;
}
static CYTHON_INLINE int __Pyx_IsSubtype(PyTypeObject *a, PyTypeObject *b) {
    PyObject *mro;
    if (a == b) return 1;
    mro = a->tp_mro;
    if (likely(mro)) {
        Py_ssize_t i, n;
        n = PyTuple_GET_SIZE(mro);
        for (i = 0; i < n; i++) {
            if (PyTuple_GET_ITEM(mro, i) == (PyObject *)b)
                return 1;
        }
        return 0;
    }
    return __Pyx_InBases(a, b);
}
#if PY_MAJOR_VERSION == 2
static int __Pyx_inner_PyErr_GivenExceptionMatches2(PyObject *err, PyObject* exc_type1, PyObject*
exc_type2) {
    PyObject *exception, *value, *tb;
    int res;
    __Pyx_PyThreadState_declare
    __Pyx_PyThreadState_assign
    __Pyx_ErrFetch(&exception, &value, &tb);
    res = exc_type1 ? PyObject_IsSubclass(err, exc_type1) : 0;
    if (unlikely(res == -1)) {
        PyErr_WriteUnraisable(err);
        res = 0;
    }
    if (!res) {
        res = PyObject_IsSubclass(err, exc_type2);
        if (unlikely(res == -1)) {
            PyErr_WriteUnraisable(err);
            res = 0;
        }
    }
    __Pyx_ErrRestore(exception, value, tb);
    return res;
}
#else
static CYTHON_INLINE int __Pyx_inner_PyErr_GivenExceptionMatches2(PyObject *err, PyObject* exc_type1,
PyObject *exc_type2) {
    int res = exc_type1 ? __Pyx_IsSubtype((PyTypeObject*)err, (PyTypeObject*)exc_type1) : 0;
    if (!res) {
        res = __Pyx_IsSubtype((PyTypeObject*)err, (PyTypeObject*)exc_type2);
    }
    return res;
}
#endif
static int __Pyx_PyErr_GivenExceptionMatchesTuple(PyObject *exc_type, PyObject *tuple) {
    Py_ssize_t i, n;
    assert(PyExceptionClass_Check(exc_type));
    n = PyTuple_GET_SIZE(tuple);
#if PY_MAJOR_VERSION >= 3
    for (i=0; i<n; i++) {
        if (exc_type == PyTuple_GET_ITEM(tuple, i)) return 1;
    }
#endif
    for (i=0; i<n; i++) {
        PyObject *t = PyTuple_GET_ITEM(tuple, i);
        #if PY_MAJOR_VERSION < 3
        if (likely(exc_type == t)) return 1;
        #endif
        if (likely(PyExceptionClass_Check(t))) {
            if (__Pyx_inner_PyErr_GivenExceptionMatches2(exc_type, NULL, t)) return 1;
        } else {
        }
    }
    return 0;
}
static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches(PyObject *err, PyObject* exc_type) {
    if (likely(err == exc_type)) return 1;
    if (likely(PyExceptionClass_Check(err))) {
        if (likely(PyExceptionClass_Check(exc_type))) {
            return __Pyx_inner_PyErr_GivenExceptionMatches2(err, NULL, exc_type);
```

```
        } else if (likely(PyTuple_Check(exc_type))) {
            return __Pyx_PyErr_GivenExceptionMatchesTuple(err, exc_type);
        } else {
        }
    }
    return PyErr_GivenExceptionMatches(err, exc_type);
}
static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches2(PyObject *err, PyObject *exc_type1, PyObject *exc_type2) {
    assert(PyExceptionClass_Check(exc_type1));
    assert(PyExceptionClass_Check(exc_type2));
    if (likely(err == exc_type1 || err == exc_type2)) return 1;
    if (likely(PyExceptionClass_Check(err))) {
        return __Pyx_inner_PyErr_GivenExceptionMatches2(err, exc_type1, exc_type2);
    }
    return (PyErr_GivenExceptionMatches(err, exc_type1) || PyErr_GivenExceptionMatches(err, exc_type2));
}
#endif

/* FetchCommonType */
static PyTypeObject* __Pyx_FetchCommonType(PyTypeObject* type) {
    PyObject* fake_module;
    PyTypeObject* cached_type = NULL;
    fake_module = PyImport_AddModule((char*) "_cython_" CYTHON_ABI);
    if (!fake_module) return NULL;
    Py_INCREF(fake_module);
    cached_type = (PyTypeObject*) PyObject_GetAttrString(fake_module, type->tp_name);
    if (cached_type) {
        if (!PyType_Check((PyObject*)cached_type)) {
            PyErr_Format(PyExc_TypeError,
                "Shared Cython type %.200s is not a type object",
                type->tp_name);
            goto bad;
        }
        if (cached_type->tp_basicsize != type->tp_basicsize) {
            PyErr_Format(PyExc_TypeError,
                "Shared Cython type %.200s has the wrong size, try recompiling",
                type->tp_name);
            goto bad;
        }
    } else {
        if (!PyErr_ExceptionMatches(PyExc_AttributeError)) goto bad;
        PyErr_Clear();
        if (PyType_Ready(type) < 0) goto bad;
        if (PyObject_SetAttrString(fake_module, type->tp_name, (PyObject*) type) < 0)
            goto bad;
        Py_INCREF(type);
        cached_type = type;
    }
done:
    Py_DECREF(fake_module);
    return cached_type;
bad:
    Py_XDECREF(cached_type);
    cached_type = NULL;
    goto done;
}

/* PyObjectGetMethod */
static int __Pyx_PyObject_GetMethod(PyObject *obj, PyObject *name, PyObject **method) {
    PyObject *attr;
#if CYTHON_UNPACK_METHODS && CYTHON_COMPILING_IN_CPYTHON && CYTHON_USE_PYTYPE_LOOKUP
    PyTypeObject *tp = Py_TYPE(obj);
    PyObject *descr;
    descrgetfunc f = NULL;
    PyObject **dictptr, *dict;
    int meth_found = 0;
    assert (*method == NULL);
    if (unlikely(tp->tp_getattro != PyObject_GenericGetAttr)) {
        attr = __Pyx_PyObject_GetAttrStr(obj, name);
        goto try_unpack;
    }
    if (unlikely(tp->tp_dict == NULL) && unlikely(PyType_Ready(tp) < 0)) {
        return 0;
    }
    descr = _PyType_Lookup(tp, name);
    if (likely(descr != NULL)) {
        Py_INCREF(descr);
#if PY_MAJOR_VERSION >= 3
        #ifdef __Pyx_CyFunction_USED
        if (likely(PyFunction_Check(descr) || (Py_TYPE(descr) == &PyMethodDescr_Type) || __Pyx_CyFunction_Check(descr)))
        #else
        if (likely(PyFunction_Check(descr) || (Py_TYPE(descr) == &PyMethodDescr_Type)))
        #endif
#else
```

```
        #ifdef __Pyx_CyFunction_USED
        if (likely(PyFunction_Check(descr) || __Pyx_CyFunction_Check(descr)))
        #else
        if (likely(PyFunction_Check(descr)))
        #endif
#endif
        {
            meth_found = 1;
        } else {
            f = Py_TYPE(descr)->tp_descr_get;
            if (f != NULL && PyDescr_IsData(descr)) {
                attr = f(descr, obj, (PyObject *)Py_TYPE(obj));
                Py_DECREF(descr);
                goto try_unpack;
            }
        }
    }
    dictptr = _PyObject_GetDictPtr(obj);
    if (dictptr != NULL && (dict = *dictptr) != NULL) {
        Py_INCREF(dict);
        attr = __Pyx_PyDict_GetItemStr(dict, name);
        if (attr != NULL) {
            Py_INCREF(attr);
            Py_DECREF(dict);
            Py_XDECREF(descr);
            goto try_unpack;
        }
        Py_DECREF(dict);
    }
    if (meth_found) {
        *method = descr;
        return 1;
    }
    if (f != NULL) {
        attr = f(descr, obj, (PyObject *)Py_TYPE(obj));
        Py_DECREF(descr);
        goto try_unpack;
    }
    if (descr != NULL) {
        *method = descr;
        return 0;
    }
    PyErr_Format(PyExc_AttributeError,
#if PY_MAJOR_VERSION >= 3
                 "'%.50s' object has no attribute '%U'",
                 tp->tp_name, name);
#else
                 "'%.50s' object has no attribute '%.400s'",
                 tp->tp_name, PyString_AS_STRING(name));
#endif
    return 0;
#else
    attr = __Pyx_PyObject_GetAttrStr(obj, name);
    goto try_unpack;
#endif
try_unpack:
#if CYTHON_UNPACK_METHODS
    if (likely(attr) && PyMethod_Check(attr) && likely(PyMethod_GET_SELF(attr) == obj)) {
        PyObject *function = PyMethod_GET_FUNCTION(attr);
        Py_INCREF(function);
        Py_DECREF(attr);
        *method = function;
        return 1;
    }
#endif
    *method = attr;
    return 0;
}

/* PyObjectCallMethod1 */
static PyObject* __Pyx__PyObject_CallMethod1(PyObject* method, PyObject* arg) {
    PyObject *result = __Pyx_PyObject_CallOneArg(method, arg);
    Py_DECREF(method);
    return result;
}
static PyObject* __Pyx_PyObject_CallMethod1(PyObject* obj, PyObject* method_name, PyObject* arg) {
    PyObject *method = NULL, *result;
    int is_method = __Pyx_PyObject_GetMethod(obj, method_name, &method);
    if (likely(is_method)) {
        result = __Pyx_PyObject_Call2Args(method, obj, arg);
        Py_DECREF(method);
        return result;
    }
    if (unlikely(!method)) return NULL;
    return __Pyx__PyObject_CallMethod1(method, arg);
}

```

```
/* CoroutineBase */
#include <structmember.h>
#include <frameobject.h>
#define __Pyx_Coroutine_Undelegate(gen) Py_CLEAR((gen)->yieldfrom)
static int __Pyx_PyGen__FetchStopIterationValue(CYTHON_UNUSED PyThreadState *__pyx_tstate, PyObject
**pvalue) {
    PyObject *et, *ev, *tb;
    PyObject *value = NULL;
    __Pyx_ErrFetch(&et, &ev, &tb);
    if (!et) {
        Py_XDECREF(tb);
        Py_XDECREF(ev);
        Py_INCREF(Py_None);
        *pvalue = Py_None;
        return 0;
    }
    if (likely(et == PyExc_StopIteration)) {
        if (!ev) {
            Py_INCREF(Py_None);
            value = Py_None;
        }
#if PY_VERSION_HEX >= 0x030300A0
        else if (Py_TYPE(ev) == (PyTypeObject*)PyExc_StopIteration) {
            value = ((PyStopIterationObject *)ev)->value;
            Py_INCREF(value);
            Py_DECREF(ev);
        }
#endif
        else if (unlikely(PyTuple_Check(ev))) {
            if (PyTuple_GET_SIZE(ev) >= 1) {
#if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
                value = PyTuple_GET_ITEM(ev, 0);
                Py_INCREF(value);
#else
                value = PySequence_ITEM(ev, 0);
#endif
            } else {
                Py_INCREF(Py_None);
                value = Py_None;
            }
            Py_DECREF(ev);
        }
        else if (!__Pyx_TypeCheck(ev, (PyTypeObject*)PyExc_StopIteration)) {
            value = ev;
        }
        if (likely(value)) {
            Py_XDECREF(tb);
            Py_DECREF(et);
            *pvalue = value;
            return 0;
        }
    } else if (!__Pyx_PyErr_GivenExceptionMatches(et, PyExc_StopIteration)) {
        __Pyx_ErrRestore(et, ev, tb);
        return -1;
    }
    PyErr_NormalizeException(&et, &ev, &tb);
    if (unlikely(!PyObject_TypeCheck(ev, (PyTypeObject*)PyExc_StopIteration))) {
        __Pyx_ErrRestore(et, ev, tb);
        return -1;
    }
    Py_XDECREF(tb);
    Py_DECREF(et);
#if PY_VERSION_HEX >= 0x030300A0
    value = ((PyStopIterationObject *)ev)->value;
    Py_INCREF(value);
    Py_DECREF(ev);
#else
    {
        PyObject* args = __Pyx_PyObject_GetAttrStr(ev, __pyx_n_s_args);
        Py_DECREF(ev);
        if (likely(args)) {
            value = PySequence_GetItem(args, 0);
            Py_DECREF(args);
        }
        if (unlikely(!value)) {
            __Pyx_ErrRestore(NULL, NULL, NULL);
            Py_INCREF(Py_None);
            value = Py_None;
        }
    }
#endif
    *pvalue = value;
    return 0;
}
static CYTHON_INLINE
void __Pyx_Coroutine_ExceptionClear(__Pyx_ExcInfoStruct *exc_state) {
    PyObject *t, *v, *tb;
```

```
    t = exc_state->exc_type;
    v = exc_state->exc_value;
    tb = exc_state->exc_traceback;
    exc_state->exc_type = NULL;
    exc_state->exc_value = NULL;
    exc_state->exc_traceback = NULL;
    Py_XDECREF(t);
    Py_XDECREF(v);
    Py_XDECREF(tb);
}
#define __Pyx_Coroutine_AlreadyRunningError(gen)  (__Pyx__Coroutine_AlreadyRunningError(gen),
(PyObject*)NULL)
static void __Pyx__Coroutine_AlreadyRunningError(CYTHON_UNUSED __pyx_CoroutineObject *gen) {
    const char *msg;
    if ((0)) {
    #ifdef __Pyx_Coroutine_USED
    } else if (__Pyx_Coroutine_Check((PyObject*)gen)) {
        msg = "coroutine already executing";
    #endif
    #ifdef __Pyx_AsyncGen_USED
    } else if (__Pyx_AsyncGen_CheckExact((PyObject*)gen)) {
        msg = "async generator already executing";
    #endif
    } else {
        msg = "generator already executing";
    }
    PyErr_SetString(PyExc_ValueError, msg);
}
#define __Pyx_Coroutine_NotStartedError(gen)  (__Pyx__Coroutine_NotStartedError(gen), (PyObject*)NULL)
static void __Pyx__Coroutine_NotStartedError(CYTHON_UNUSED PyObject *gen) {
    const char *msg;
    if ((0)) {
    #ifdef __Pyx_Coroutine_USED
    } else if (__Pyx_Coroutine_Check(gen)) {
        msg = "can't send non-None value to a just-started coroutine";
    #endif
    #ifdef __Pyx_AsyncGen_USED
    } else if (__Pyx_AsyncGen_CheckExact(gen)) {
        msg = "can't send non-None value to a just-started async generator";
    #endif
    } else {
        msg = "can't send non-None value to a just-started generator";
    }
    PyErr_SetString(PyExc_TypeError, msg);
}
#define __Pyx_Coroutine_AlreadyTerminatedError(gen, value, closing)
(__Pyx__Coroutine_AlreadyTerminatedError(gen, value, closing), (PyObject*)NULL)
static void __Pyx__Coroutine_AlreadyTerminatedError(CYTHON_UNUSED PyObject *gen, PyObject *value,
CYTHON_UNUSED int closing) {
    #ifdef __Pyx_Coroutine_USED
    if (!closing && __Pyx_Coroutine_Check(gen)) {
        PyErr_SetString(PyExc_RuntimeError, "cannot reuse already awaited coroutine");
    } else
    #endif
    if (value) {
        #ifdef __Pyx_AsyncGen_USED
        if (__Pyx_AsyncGen_CheckExact(gen))
            PyErr_SetNone(__Pyx_PyExc_StopAsyncIteration);
        else
        #endif
        PyErr_SetNone(PyExc_StopIteration);
    }
}
static
PyObject *__Pyx_Coroutine_SendEx(__pyx_CoroutineObject *self, PyObject *value, int closing) {
    __Pyx_PyThreadState_declare
    PyThreadState *tstate;
    __Pyx_ExcInfoStruct *exc_state;
    PyObject *retval;
    assert(!self->is_running);
    if (unlikely(self->resume_label == 0)) {
        if (unlikely(value && value != Py_None)) {
            return __Pyx_Coroutine_NotStartedError((PyObject*)self);
        }
    }
    if (unlikely(self->resume_label == -1)) {
        return __Pyx_Coroutine_AlreadyTerminatedError((PyObject*)self, value, closing);
    }
#if CYTHON_FAST_THREAD_STATE
    __Pyx_PyThreadState_assign
    tstate = __pyx_tstate;
#else
    tstate = __Pyx_PyThreadState_Current;
#endif
    exc_state = &self->gi_exc_state;
    if (exc_state->exc_type) {
        #if CYTHON_COMPILING_IN_PYPY || CYTHON_COMPILING_IN_PYSTON
```

```
        #else
        if (exc_state->exc_traceback) {
            PyTracebackObject *tb = (PyTracebackObject *) exc_state->exc_traceback;
            PyFrameObject *f = tb->tb_frame;
            assert(f->f_back == NULL);
            #if PY_VERSION_HEX >= 0x030B00A1
            f->f_back = PyThreadState_GetFrame(tstate);
            #else
            Py_XINCREF(tstate->frame);
            f->f_back = tstate->frame;
            #endif
        }
        #endif
    }
#if CYTHON_USE_EXC_INFO_STACK
    exc_state->previous_item = tstate->exc_info;
    tstate->exc_info = exc_state;
#else
    if (exc_state->exc_type) {
        __Pyx_ExceptionSwap(&exc_state->exc_type, &exc_state->exc_value, &exc_state->exc_traceback);
    } else {
        __Pyx_Coroutine_ExceptionClear(exc_state);
        __Pyx_ExceptionSave(&exc_state->exc_type, &exc_state->exc_value, &exc_state->exc_traceback);
    }
#endif
    self->is_running = 1;
    retval = self->body((PyObject *) self, tstate, value);
    self->is_running = 0;
#if CYTHON_USE_EXC_INFO_STACK
    exc_state = &self->gi_exc_state;
    tstate->exc_info = exc_state->previous_item;
    exc_state->previous_item = NULL;
    __Pyx_Coroutine_ResetFrameBackpointer(exc_state);
#endif
    return retval;
}
static CYTHON_INLINE void __Pyx_Coroutine_ResetFrameBackpointer(__Pyx_ExcInfoStruct *exc_state) {
    PyObject *exc_tb = exc_state->exc_traceback;
    if (likely(exc_tb)) {
#if CYTHON_COMPILING_IN_PYPY || CYTHON_COMPILING_IN_PYSTON
#else
        PyTracebackObject *tb = (PyTracebackObject *) exc_tb;
        PyFrameObject *f = tb->tb_frame;
        Py_CLEAR(f->f_back);
#endif
    }
}
static CYTHON_INLINE
PyObject *__Pyx_Coroutine_MethodReturn(CYTHON_UNUSED PyObject* gen, PyObject *retval) {
    if (unlikely(!retval)) {
        __Pyx_PyThreadState_declare
        __Pyx_PyThreadState_assign
        if (!__Pyx_PyErr_Occurred()) {
            PyObject *exc = PyExc_StopIteration;
            #ifdef __Pyx_AsyncGen_USED
            if (__Pyx_AsyncGen_CheckExact(gen))
                exc = __Pyx_PyExc_StopAsyncIteration;
            #endif
            __Pyx_PyErr_SetNone(exc);
        }
    }
    return retval;
}
#if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) ||
PY_VERSION_HEX >= 0x030600B3)
static CYTHON_INLINE
PyObject *__Pyx_PyGen_Send(PyGenObject *gen, PyObject *arg) {
#if PY_VERSION_HEX <= 0x030A00A1
    return _PyGen_Send(gen, arg);
#else
    PyObject *result;
    if (PyIter_Send((PyObject*)gen, arg ? arg : Py_None, &result) == PYGEN_RETURN) {
        if (PyAsyncGen_CheckExact(gen)) {
            assert(result == Py_None);
            PyErr_SetNone(PyExc_StopAsyncIteration);
        }
        else if (result == Py_None) {
            PyErr_SetNone(PyExc_StopIteration);
        }
        else {
            _PyGen_SetStopIterationValue(result);
        }
        Py_CLEAR(result);
    }
    return result;
#endif
}
```

```
#endif
static CYTHON_INLINE
PyObject *__Pyx_Coroutine_FinishDelegation(__pyx_CoroutineObject *gen) {
    PyObject *ret;
    PyObject *val = NULL;
    __Pyx_Coroutine_Undelegate(gen);
    __Pyx_PyGen__FetchStopIterationValue(__Pyx_PyThreadState_Current, &val);
    ret = __Pyx_Coroutine_SendEx(gen, val, 0);
    Py_XDECREF(val);
    return ret;
}
static PyObject *__Pyx_Coroutine_Send(PyObject *self, PyObject *value) {
    PyObject *retval;
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject*) self;
    PyObject *yf = gen->yieldfrom;
    if (unlikely(gen->is_running))
        return __Pyx_Coroutine_AlreadyRunningError(gen);
    if (yf) {
        PyObject *ret;
        gen->is_running = 1;
        #ifdef __Pyx_Generator_USED
        if (__Pyx_Generator_CheckExact(yf)) {
            ret = __Pyx_Coroutine_Send(yf, value);
        } else
        #endif
        #ifdef __Pyx_Coroutine_USED
        if (__Pyx_Coroutine_Check(yf)) {
            ret = __Pyx_Coroutine_Send(yf, value);
        } else
        #endif
        #ifdef __Pyx_AsyncGen_USED
        if (__pyx_PyAsyncGenASend_CheckExact(yf)) {
            ret = __Pyx_async_gen_asend_send(yf, value);
        } else
        #endif
        #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) ||
PY_VERSION_HEX >= 0x030600B3)
        if (PyGen_CheckExact(yf)) {
            ret = __Pyx_PyGen_Send((PyGenObject*)yf, value == Py_None ? NULL : value);
        } else
        #endif
        #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03050000 && defined(PyCoro_CheckExact)
&& (defined(__linux__) || PY_VERSION_HEX >= 0x030600B3)
        if (PyCoro_CheckExact(yf)) {
            ret = __Pyx_PyGen_Send((PyGenObject*)yf, value == Py_None ? NULL : value);
        } else
        #endif
        {
            if (value == Py_None)
                ret = Py_TYPE(yf)->tp_iternext(yf);
            else
                ret = __Pyx_PyObject_CallMethod1(yf, __pyx_n_s_send, value);
        }
        gen->is_running = 0;
        if (likely(ret)) {
            return ret;
        }
        retval = __Pyx_Coroutine_FinishDelegation(gen);
    } else {
        retval = __Pyx_Coroutine_SendEx(gen, value, 0);
    }
    return __Pyx_Coroutine_MethodReturn(self, retval);
}
static int __Pyx_Coroutine_CloseIter(__pyx_CoroutineObject *gen, PyObject *yf) {
    PyObject *retval = NULL;
    int err = 0;
    #ifdef __Pyx_Generator_USED
    if (__Pyx_Generator_CheckExact(yf)) {
        retval = __Pyx_Coroutine_Close(yf);
        if (!retval)
            return -1;
    } else
    #endif
    #ifdef __Pyx_Coroutine_USED
    if (__Pyx_Coroutine_Check(yf)) {
        retval = __Pyx_Coroutine_Close(yf);
        if (!retval)
            return -1;
    } else
    if (__Pyx_CoroutineAwait_CheckExact(yf)) {
        retval = __Pyx_CoroutineAwait_Close((__pyx_CoroutineAwaitObject*)yf, NULL);
        if (!retval)
            return -1;
    } else
    #endif
    #ifdef __Pyx_AsyncGen_USED
    if (__pyx_PyAsyncGenASend_CheckExact(yf)) {
```

```
        retval = __Pyx_async_gen_asend_close(yf, NULL);
    } else
    if (__pyx_PyAsyncGenAThrow_CheckExact(yf)) {
        retval = __Pyx_async_gen_athrow_close(yf, NULL);
    } else
    #endif
    {
        PyObject *meth;
        gen->is_running = 1;
        meth = __Pyx_PyObject_GetAttrStr(yf, __pyx_n_s_close);
        if (unlikely(!meth)) {
            if (!PyErr_ExceptionMatches(PyExc_AttributeError)) {
                PyErr_WriteUnraisable(yf);
            }
            PyErr_Clear();
        } else {
            retval = PyObject_CallFunction(meth, NULL);
            Py_DECREF(meth);
            if (!retval)
                err = -1;
        }
        gen->is_running = 0;
    }
    Py_XDECREF(retval);
    return err;
}
static PyObject *__Pyx_Generator_Next(PyObject *self) {
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject*) self;
    PyObject *yf = gen->yieldfrom;
    if (unlikely(gen->is_running))
        return __Pyx_Coroutine_AlreadyRunningError(gen);
    if (yf) {
        PyObject *ret;
        gen->is_running = 1;
        #ifdef __Pyx_Generator_USED
        if (__Pyx_Generator_CheckExact(yf)) {
            ret = __Pyx_Generator_Next(yf);
        } else
        #endif
        #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) || PY_VERSION_HEX >= 0x030600B3)
        if (PyGen_CheckExact(yf)) {
            ret = __Pyx_PyGen_Send((PyGenObject*)yf, NULL);
        } else
        #endif
        #ifdef __Pyx_Coroutine_USED
        if (__Pyx_Coroutine_Check(yf)) {
            ret = __Pyx_Coroutine_Send(yf, Py_None);
        } else
        #endif
            ret = Py_TYPE(yf)->tp_iternext(yf);
        gen->is_running = 0;
        if (likely(ret)) {
            return ret;
        }
        return __Pyx_Coroutine_FinishDelegation(gen);
    }
    return __Pyx_Coroutine_SendEx(gen, Py_None, 0);
}
static PyObject *__Pyx_Coroutine_Close_Method(PyObject *self, CYTHON_UNUSED PyObject *arg) {
    return __Pyx_Coroutine_Close(self);
}
static PyObject *__Pyx_Coroutine_Close(PyObject *self) {
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
    PyObject *retval, *raised_exception;
    PyObject *yf = gen->yieldfrom;
    int err = 0;
    if (unlikely(gen->is_running))
        return __Pyx_Coroutine_AlreadyRunningError(gen);
    if (yf) {
        Py_INCREF(yf);
        err = __Pyx_Coroutine_CloseIter(gen, yf);
        __Pyx_Coroutine_Undelegate(gen);
        Py_DECREF(yf);
    }
    if (err == 0)
        PyErr_SetNone(PyExc_GeneratorExit);
    retval = __Pyx_Coroutine_SendEx(gen, NULL, 1);
    if (unlikely(retval)) {
        const char *msg;
        Py_DECREF(retval);
        if ((0)) {
        #ifdef __Pyx_Coroutine_USED
        } else if (__Pyx_Coroutine_Check(self)) {
            msg = "coroutine ignored GeneratorExit";
        #endif
        #ifdef __Pyx_AsyncGen_USED
```

```
        } else if (__Pyx_AsyncGen_CheckExact(self)) {
#if PY_VERSION_HEX < 0x03060000
            msg = "async generator ignored GeneratorExit - might require Python 3.6+ finalisation (PEP
525)";
#else
            msg = "async generator ignored GeneratorExit";
#endif
        #endif
        } else {
            msg = "generator ignored GeneratorExit";
        }
        PyErr_SetString(PyExc_RuntimeError, msg);
        return NULL;
    }
    raised_exception = PyErr_Occurred();
    if (likely(!raised_exception || __Pyx_PyErr_GivenExceptionMatches2(raised_exception,
PyExc_GeneratorExit, PyExc_StopIteration))) {
        if (raised_exception) PyErr_Clear();
        Py_INCREF(Py_None);
        return Py_None;
    }
    return NULL;
}
static PyObject *__Pyx__Coroutine_Throw(PyObject *self, PyObject *typ, PyObject *val, PyObject *tb,
                                        PyObject *args, int close_on_genexit) {
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
    PyObject *yf = gen->yieldfrom;
    if (unlikely(gen->is_running))
        return __Pyx_Coroutine_AlreadyRunningError(gen);
    if (yf) {
        PyObject *ret;
        Py_INCREF(yf);
        if (__Pyx_PyErr_GivenExceptionMatches(typ, PyExc_GeneratorExit) && close_on_genexit) {
            int err = __Pyx_Coroutine_CloseIter(gen, yf);
            Py_DECREF(yf);
            __Pyx_Coroutine_Undelegate(gen);
            if (err < 0)
                return __Pyx_Coroutine_MethodReturn(self, __Pyx_Coroutine_SendEx(gen, NULL, 0));
            goto throw_here;
        }
        gen->is_running = 1;
        if (0
        #ifdef __Pyx_Generator_USED
            || __Pyx_Generator_CheckExact(yf)
        #endif
        #ifdef __Pyx_Coroutine_USED
            || __Pyx_Coroutine_Check(yf)
        #endif
            ) {
            ret = __Pyx__Coroutine_Throw(yf, typ, val, tb, args, close_on_genexit);
        #ifdef __Pyx_Coroutine_USED
        } else if (__Pyx_CoroutineAwait_CheckExact(yf)) {
            ret = __Pyx__Coroutine_Throw(((__pyx_CoroutineAwaitObject*)yf)->coroutine, typ, val, tb,
args, close_on_genexit);
        #endif
        } else {
            PyObject *meth = __Pyx_PyObject_GetAttrStr(yf, __pyx_n_s_throw);
            if (unlikely(!meth)) {
                Py_DECREF(yf);
                if (!PyErr_ExceptionMatches(PyExc_AttributeError)) {
                    gen->is_running = 0;
                    return NULL;
                }
                PyErr_Clear();
                __Pyx_Coroutine_Undelegate(gen);
                gen->is_running = 0;
                goto throw_here;
            }
            if (likely(args)) {
                ret = PyObject_CallObject(meth, args);
            } else {
                ret = PyObject_CallFunctionObjArgs(meth, typ, val, tb, NULL);
            }
            Py_DECREF(meth);
        }
        gen->is_running = 0;
        Py_DECREF(yf);
        if (!ret) {
            ret = __Pyx_Coroutine_FinishDelegation(gen);
        }
        return __Pyx_Coroutine_MethodReturn(self, ret);
    }
throw_here:
    __Pyx_Raise(typ, val, tb, NULL);
    return __Pyx_Coroutine_MethodReturn(self, __Pyx_Coroutine_SendEx(gen, NULL, 0));
}
static PyObject *__Pyx_Coroutine_Throw(PyObject *self, PyObject *args) {
```

```
    PyObject *typ;
    PyObject *val = NULL;
    PyObject *tb = NULL;
    if (!PyArg_UnpackTuple(args, (char *)"throw", 1, 3, &typ, &val, &tb))
        return NULL;
    return __Pyx__Coroutine_Throw(self, typ, val, tb, args, 1);
}
static CYTHON_INLINE int __Pyx_Coroutine_traverse_excstate(__Pyx_ExcInfoStruct *exc_state, visitproc visit, void *arg) {
    Py_VISIT(exc_state->exc_type);
    Py_VISIT(exc_state->exc_value);
    Py_VISIT(exc_state->exc_traceback);
    return 0;
}
static int __Pyx_Coroutine_traverse(__pyx_CoroutineObject *gen, visitproc visit, void *arg) {
    Py_VISIT(gen->closure);
    Py_VISIT(gen->classobj);
    Py_VISIT(gen->yieldfrom);
    return __Pyx_Coroutine_traverse_excstate(&gen->gi_exc_state, visit, arg);
}
static int __Pyx_Coroutine_clear(PyObject *self) {
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
    Py_CLEAR(gen->closure);
    Py_CLEAR(gen->classobj);
    Py_CLEAR(gen->yieldfrom);
    __Pyx_Coroutine_ExceptionClear(&gen->gi_exc_state);
#ifdef __Pyx_AsyncGen_USED
    if (__Pyx_AsyncGen_CheckExact(self)) {
        Py_CLEAR(((__pyx_PyAsyncGenObject*)gen)->ag_finalizer);
    }
#endif
    Py_CLEAR(gen->gi_code);
    Py_CLEAR(gen->gi_frame);
    Py_CLEAR(gen->gi_name);
    Py_CLEAR(gen->gi_qualname);
    Py_CLEAR(gen->gi_modulename);
    return 0;
}
static void __Pyx_Coroutine_dealloc(PyObject *self) {
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
    PyObject_GC_UnTrack(gen);
    if (gen->gi_weakreflist != NULL)
        PyObject_ClearWeakRefs(self);
    if (gen->resume_label >= 0) {
        PyObject_GC_Track(self);
#if PY_VERSION_HEX >= 0x030400a1 && CYTHON_USE_TP_FINALIZE
        if (PyObject_CallFinalizerFromDealloc(self))
#else
        Py_TYPE(gen)->tp_del(self);
        if (Py_REFCNT(self) > 0)
#endif
        {
            return;
        }
        PyObject_GC_UnTrack(self);
    }
#ifdef __Pyx_AsyncGen_USED
    if (__Pyx_AsyncGen_CheckExact(self)) {
        /* We have to handle this case for asynchronous generators
           right here, because this code has to be between UNTRACK
           and GC_Del. */
        Py_CLEAR(((__pyx_PyAsyncGenObject*)self)->ag_finalizer);
    }
#endif
    __Pyx_Coroutine_clear(self);
    PyObject_GC_Del(gen);
}
static void __Pyx_Coroutine_del(PyObject *self) {
    PyObject *error_type, *error_value, *error_traceback;
    __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
    __Pyx_PyThreadState_declare
    if (gen->resume_label < 0) {
        return;
    }
#if !CYTHON_USE_TP_FINALIZE
    assert(self->ob_refcnt == 0);
    __Pyx_SET_REFCNT(self, 1);
#endif
    __Pyx_PyThreadState_assign
    __Pyx_ErrFetch(&error_type, &error_value, &error_traceback);
#ifdef __Pyx_AsyncGen_USED
    if (__Pyx_AsyncGen_CheckExact(self)) {
        __pyx_PyAsyncGenObject *agen = (__pyx_PyAsyncGenObject*)self;
        PyObject *finalizer = agen->ag_finalizer;
        if (finalizer && !agen->ag_closed) {
            PyObject *res = __Pyx_PyObject_CallOneArg(finalizer, self);
            if (unlikely(!res)) {
```

```
                PyErr_WriteUnraisable(self);
            } else {
                Py_DECREF(res);
            }
            __Pyx_ErrRestore(error_type, error_value, error_traceback);
            return;
        }
    }
#endif
    if (unlikely(gen->resume_label == 0 && !error_value)) {
#ifdef __Pyx_Coroutine_USED
#ifdef __Pyx_Generator_USED
    if (!__Pyx_Generator_CheckExact(self))
#endif
        {
        PyObject_GC_UnTrack(self);
#if PY_MAJOR_VERSION >= 3  || defined(PyErr_WarnFormat)
        if (unlikely(PyErr_WarnFormat(PyExc_RuntimeWarning, 1, "coroutine '%.50S' was never awaited", gen->gi_qualname) < 0))
            PyErr_WriteUnraisable(self);
#else
        {PyObject *msg;
        char *cmsg;
        #if CYTHON_COMPILING_IN_PYPY
        msg = NULL;
        cmsg = (char*) "coroutine was never awaited";
        #else
        char *cname;
        PyObject *qualname;
        qualname = gen->gi_qualname;
        cname = PyString_AS_STRING(qualname);
        msg = PyString_FromFormat("coroutine '%.50s' was never awaited", cname);
        if (unlikely(!msg)) {
            PyErr_Clear();
            cmsg = (char*) "coroutine was never awaited";
        } else {
            cmsg = PyString_AS_STRING(msg);
        }
        #endif
        if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning, cmsg, 1) < 0))
            PyErr_WriteUnraisable(self);
        Py_XDECREF(msg);}
#endif
        PyObject_GC_Track(self);
        }
#endif
    } else {
        PyObject *res = __Pyx_Coroutine_Close(self);
        if (unlikely(!res)) {
            if (PyErr_Occurred())
                PyErr_WriteUnraisable(self);
        } else {
            Py_DECREF(res);
        }
    }
    __Pyx_ErrRestore(error_type, error_value, error_traceback);
#if !CYTHON_USE_TP_FINALIZE
    assert(Py_REFCNT(self) > 0);
    if (--self->ob_refcnt == 0) {
        return;
    }
    {
        Py_ssize_t refcnt = Py_REFCNT(self);
        _Py_NewReference(self);
        __Pyx_SET_REFCNT(self, refcnt);
    }
#if CYTHON_COMPILING_IN_CPYTHON
    assert(PyType_IS_GC(Py_TYPE(self)) &&
           _Py_AS_GC(self)->gc.gc_refs != _PyGC_REFS_UNTRACKED);
    _Py_DEC_REFTOTAL;
#endif
#ifdef COUNT_ALLOCS
    --Py_TYPE(self)->tp_frees;
    --Py_TYPE(self)->tp_allocs;
#endif
#endif
}
static PyObject *
__Pyx_Coroutine_get_name(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
{
    PyObject *name = self->gi_name;
    if (unlikely(!name)) name = Py_None;
    Py_INCREF(name);
    return name;
}
static int
__Pyx_Coroutine_set_name(__pyx_CoroutineObject *self, PyObject *value, CYTHON_UNUSED void *context)
```

```
{
    PyObject *tmp;
#if PY_MAJOR_VERSION >= 3
    if (unlikely(value == NULL || !PyUnicode_Check(value)))
#else
    if (unlikely(value == NULL || !PyString_Check(value)))
#endif
    {
        PyErr_SetString(PyExc_TypeError,
                        "__name__ must be set to a string object");
        return -1;
    }
    tmp = self->gi_name;
    Py_INCREF(value);
    self->gi_name = value;
    Py_XDECREF(tmp);
    return 0;
}
static PyObject *
__Pyx_Coroutine_get_qualname(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
{
    PyObject *name = self->gi_qualname;
    if (unlikely(!name)) name = Py_None;
    Py_INCREF(name);
    return name;
}
static int
__Pyx_Coroutine_set_qualname(__pyx_CoroutineObject *self, PyObject *value, CYTHON_UNUSED void
*context)
{
    PyObject *tmp;
#if PY_MAJOR_VERSION >= 3
    if (unlikely(value == NULL || !PyUnicode_Check(value)))
#else
    if (unlikely(value == NULL || !PyString_Check(value)))
#endif
    {
        PyErr_SetString(PyExc_TypeError,
                        "__qualname__ must be set to a string object");
        return -1;
    }
    tmp = self->gi_qualname;
    Py_INCREF(value);
    self->gi_qualname = value;
    Py_XDECREF(tmp);
    return 0;
}
static PyObject *
__Pyx_Coroutine_get_frame(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
{
    PyObject *frame = self->gi_frame;
    if (!frame) {
        if (unlikely(!self->gi_code)) {
            Py_RETURN_NONE;
        }
        frame = (PyObject *) PyFrame_New(
            PyThreadState_Get(),            /*PyThreadState *tstate,*/
            (PyCodeObject*) self->gi_code,  /*PyCodeObject *code,*/
            __pyx_d,                 /*PyObject *globals,*/
            0                               /*PyObject *locals*/
        );
        if (unlikely(!frame))
            return NULL;
        self->gi_frame = frame;
    }
    Py_INCREF(frame);
    return frame;
}
static __pyx_CoroutineObject *__Pyx__Coroutine_New(
            PyTypeObject* type, __pyx_coroutine_body_t body, PyObject *code, PyObject *closure,
            PyObject *name, PyObject *qualname, PyObject *module_name) {
    __pyx_CoroutineObject *gen = PyObject_GC_New(__pyx_CoroutineObject, type);
    if (unlikely(!gen))
        return NULL;
    return __Pyx__Coroutine_NewInit(gen, body, code, closure, name, qualname, module_name);
}
static __pyx_CoroutineObject *__Pyx__Coroutine_NewInit(
            __pyx_CoroutineObject *gen, __pyx_coroutine_body_t body, PyObject *code, PyObject
*closure,
            PyObject *name, PyObject *qualname, PyObject *module_name) {
    gen->body = body;
    gen->closure = closure;
    Py_XINCREF(closure);
    gen->is_running = 0;
    gen->resume_label = 0;
    gen->classobj = NULL;
    gen->yieldfrom = NULL;
```

```
    gen->gi_exc_state.exc_type = NULL;
    gen->gi_exc_state.exc_value = NULL;
    gen->gi_exc_state.exc_traceback = NULL;
#if CYTHON_USE_EXC_INFO_STACK
    gen->gi_exc_state.previous_item = NULL;
#endif
    gen->gi_weakreflist = NULL;
    Py_XINCREF(qualname);
    gen->gi_qualname = qualname;
    Py_XINCREF(name);
    gen->gi_name = name;
    Py_XINCREF(module_name);
    gen->gi_modulename = module_name;
    Py_XINCREF(code);
    gen->gi_code = code;
    gen->gi_frame = NULL;
    PyObject_GC_Track(gen);
    return gen;
}

/* PatchModuleWithCoroutine */
static PyObject* __Pyx_Coroutine_patch_module(PyObject* module, const char* py_code) {
#if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
    int result;
    PyObject *globals, *result_obj;
    globals = PyDict_New();  if (unlikely(!globals)) goto ignore;
    result = PyDict_SetItemString(globals, "_cython_coroutine_type",
    #ifdef __Pyx_Coroutine_USED
        (PyObject*)__pyx_CoroutineType);
    #else
        Py_None);
    #endif
    if (unlikely(result < 0)) goto ignore;
    result = PyDict_SetItemString(globals, "_cython_generator_type",
    #ifdef __Pyx_Generator_USED
        (PyObject*)__pyx_GeneratorType);
    #else
        Py_None);
    #endif
    if (unlikely(result < 0)) goto ignore;
    if (unlikely(PyDict_SetItemString(globals, "_module", module) < 0)) goto ignore;
    if (unlikely(PyDict_SetItemString(globals, "__builtins__", __pyx_b) < 0)) goto ignore;
    result_obj = PyRun_String(py_code, Py_file_input, globals, globals);
    if (unlikely(!result_obj)) goto ignore;
    Py_DECREF(result_obj);
    Py_DECREF(globals);
    return module;
ignore:
    Py_XDECREF(globals);
    PyErr_WriteUnraisable(module);
    if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning, "Cython module failed to patch module with custom
type", 1) < 0)) {
        Py_DECREF(module);
        module = NULL;
    }
#else
    py_code++;
#endif
    return module;
}

/* PatchGeneratorABC */
#ifndef CYTHON_REGISTER_ABCS
#define CYTHON_REGISTER_ABCS 1
#endif
#if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
static PyObject* __Pyx_patch_abc_module(PyObject *module);
static PyObject* __Pyx_patch_abc_module(PyObject *module) {
    module = __Pyx_Coroutine_patch_module(
        module, ""
"if _cython_generator_type is not None:\n"
"    try: Generator = _module.Generator\n"
"    except AttributeError: pass\n"
"    else: Generator.register(_cython_generator_type)\n"
"if _cython_coroutine_type is not None:\n"
"    try: Coroutine = _module.Coroutine\n"
"    except AttributeError: pass\n"
"    else: Coroutine.register(_cython_coroutine_type)\n"
    );
    return module;
}
#endif
static int __Pyx_patch_abc(void) {
#if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
    static int abc_patched = 0;
    if (CYTHON_REGISTER_ABCS && !abc_patched) {
        PyObject *module;
```

```
        module = PyImport_ImportModule((PY_MAJOR_VERSION >= 3) ? "collections.abc" : "collections");
        if (!module) {
            PyErr_WriteUnraisable(NULL);
            if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning,
                    ((PY_MAJOR_VERSION >= 3) ?
                        "Cython module failed to register with collections.abc module" :
                        "Cython module failed to register with collections module"), 1) < 0)) {
                return -1;
            }
        } else {
            module = __Pyx_patch_abc_module(module);
            abc_patched = 1;
            if (unlikely(!module))
                return -1;
            Py_DECREF(module);
        }
        module = PyImport_ImportModule("backports_abc");
        if (module) {
            module = __Pyx_patch_abc_module(module);
            Py_XDECREF(module);
        }
        if (!module) {
            PyErr_Clear();
        }
    }
#else
    if ((0)) __Pyx_Coroutine_patch_module(NULL, NULL);
#endif
    return 0;
}

/* Generator */
static PyMethodDef __pyx_Generator_methods[] = {
    {"send", (PyCFunction) __Pyx_Coroutine_Send, METH_O,
     (char*) PyDoc_STR("send(arg) -> send 'arg' into generator,\nreturn next yielded value or raise
StopIteration.")},
    {"throw", (PyCFunction) __Pyx_Coroutine_Throw, METH_VARARGS,
     (char*) PyDoc_STR("throw(typ[,val[,tb]]) -> raise exception in generator,\nreturn next yielded
value or raise StopIteration.")},
    {"close", (PyCFunction) __Pyx_Coroutine_Close_Method, METH_NOARGS,
     (char*) PyDoc_STR("close() -> raise GeneratorExit inside generator.")},
    {0, 0, 0, 0}
};
static PyMemberDef __pyx_Generator_memberlist[] = {
    {(char *) "gi_running", T_BOOL, offsetof(__pyx_CoroutineObject, is_running), READONLY, NULL},
    {(char*) "gi_yieldfrom", T_OBJECT, offsetof(__pyx_CoroutineObject, yieldfrom), READONLY,
     (char*) PyDoc_STR("object being iterated by 'yield from', or None")},
    {(char*) "gi_code", T_OBJECT, offsetof(__pyx_CoroutineObject, gi_code), READONLY, NULL},
    {0, 0, 0, 0, 0}
};
static PyGetSetDef __pyx_Generator_getsets[] = {
    {(char *) "__name__", (getter)__Pyx_Coroutine_get_name, (setter)__Pyx_Coroutine_set_name,
     (char*) PyDoc_STR("name of the generator"), 0},
    {(char *) "__qualname__", (getter)__Pyx_Coroutine_get_qualname,
(setter)__Pyx_Coroutine_set_qualname,
     (char*) PyDoc_STR("qualified name of the generator"), 0},
    {(char *) "gi_frame", (getter)__Pyx_Coroutine_get_frame, NULL,
     (char*) PyDoc_STR("Frame of the generator"), 0},
    {0, 0, 0, 0, 0}
};
static PyTypeObject __pyx_GeneratorType_type = {
    PyVarObject_HEAD_INIT(0, 0)
    "generator",
    sizeof(__pyx_CoroutineObject),
    0,
    (destructor) __Pyx_Coroutine_dealloc,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    Py_TPFLAGS_DEFAULT | Py_TPFLAGS_HAVE_GC | Py_TPFLAGS_HAVE_FINALIZE,
    0,
    (traverseproc) __Pyx_Coroutine_traverse,
    0,
    0,
    offsetof(__pyx_CoroutineObject, gi_weakreflist),
    0,
```

```
    (iternextfunc) __Pyx_Generator_Next,
    __pyx_Generator_methods,
    __pyx_Generator_memberlist,
    __pyx_Generator_getsets,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
#if CYTHON_USE_TP_FINALIZE
    0,
#else
    __Pyx_Coroutine_del,
#endif
    0,
#if CYTHON_USE_TP_FINALIZE
    __Pyx_Coroutine_del,
#elif PY_VERSION_HEX >= 0x030400a1
    0,
#endif
#if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM >= 0x07030800)
    0,
#endif
#if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
    0,
#endif
#if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
    0,
#endif
};
static int __pyx_Generator_init(void) {
    __pyx_GeneratorType_type.tp_getattro = __Pyx_PyObject_GenericGetAttrNoDict;
    __pyx_GeneratorType_type.tp_iter = PyObject_SelfIter;
    __pyx_GeneratorType = __Pyx_FetchCommonType(&__pyx_GeneratorType_type);
    if (unlikely(!__pyx_GeneratorType)) {
        return -1;
    }
    return 0;
}

/* CheckBinaryVersion */
static int __Pyx_check_binary_version(void) {
    char ctversion[4], rtversion[4];
    PyOS_snprintf(ctversion, 4, "%d.%d", PY_MAJOR_VERSION, PY_MINOR_VERSION);
    PyOS_snprintf(rtversion, 4, "%s", Py_GetVersion());
    if (ctversion[0] != rtversion[0] || ctversion[2] != rtversion[2]) {
        char message[200];
        PyOS_snprintf(message, sizeof(message),
                      "compiletime version %s of module '%.100s' "
                      "does not match runtime version %s",
                      ctversion, __Pyx_MODULE_NAME, rtversion);
        return PyErr_WarnEx(NULL, message, 1);
    }
    return 0;
}

/* InitStrings */
static int __Pyx_InitStrings(__Pyx_StringTabEntry *t) {
    while (t->p) {
        #if PY_MAJOR_VERSION < 3
        if (t->is_unicode) {
            *t->p = PyUnicode_DecodeUTF8(t->s, t->n - 1, NULL);
        } else if (t->intern) {
            *t->p = PyString_InternFromString(t->s);
        } else {
            *t->p = PyString_FromStringAndSize(t->s, t->n - 1);
        }
        #else
        if (t->is_unicode | t->is_str) {
            if (t->intern) {
                *t->p = PyUnicode_InternFromString(t->s);
            } else if (t->encoding) {
                *t->p = PyUnicode_Decode(t->s, t->n - 1, t->encoding, NULL);
            } else {
                *t->p = PyUnicode_FromStringAndSize(t->s, t->n - 1);
            }
        } else {
```

```
            *t->p = PyBytes_FromStringAndSize(t->s, t->n - 1);
        }
        #endif
        if (!*t->p)
            return -1;
        if (PyObject_Hash(*t->p) == -1)
            return -1;
        ++t;
    }
    return 0;
}

static CYTHON_INLINE PyObject* __Pyx_PyUnicode_FromString(const char* c_str) {
    return __Pyx_PyUnicode_FromStringAndSize(c_str, (Py_ssize_t)strlen(c_str));
}
static CYTHON_INLINE const char* __Pyx_PyObject_AsString(PyObject* o) {
    Py_ssize_t ignore;
    return __Pyx_PyObject_AsStringAndSize(o, &ignore);
}
#if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII || __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
#if !CYTHON_PEP393_ENABLED
static const char* __Pyx_PyUnicode_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
    char* defenc_c;
    PyObject* defenc = _PyUnicode_AsDefaultEncodedString(o, NULL);
    if (!defenc) return NULL;
    defenc_c = PyBytes_AS_STRING(defenc);
#if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
    {
        char* end = defenc_c + PyBytes_GET_SIZE(defenc);
        char* c;
        for (c = defenc_c; c < end; c++) {
            if ((unsigned char) (*c) >= 128) {
                PyUnicode_AsASCIIString(o);
                return NULL;
            }
        }
    }
#endif
    *length = PyBytes_GET_SIZE(defenc);
    return defenc_c;
}
#else
static CYTHON_INLINE const char* __Pyx_PyUnicode_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
    if (unlikely(__Pyx_PyUnicode_READY(o) == -1)) return NULL;
#if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
    if (likely(PyUnicode_IS_ASCII(o))) {
        *length = PyUnicode_GET_LENGTH(o);
        return PyUnicode_AsUTF8(o);
    } else {
        PyUnicode_AsASCIIString(o);
        return NULL;
    }
#else
    return PyUnicode_AsUTF8AndSize(o, length);
#endif
}
#endif
#endif
static CYTHON_INLINE const char* __Pyx_PyObject_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
#if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII || __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
    if (
#if PY_MAJOR_VERSION < 3 && __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
            __Pyx_sys_getdefaultencoding_not_ascii &&
#endif
            PyUnicode_Check(o)) {
        return __Pyx_PyUnicode_AsStringAndSize(o, length);
    } else
#endif
#if (!CYTHON_COMPILING_IN_PYPY) || (defined(PyByteArray_AS_STRING) && defined(PyByteArray_GET_SIZE))
    if (PyByteArray_Check(o)) {
        *length = PyByteArray_GET_SIZE(o);
        return PyByteArray_AS_STRING(o);
    } else
#endif
    {
        char* result;
        int r = PyBytes_AsStringAndSize(o, &result, length);
        if (unlikely(r < 0)) {
            return NULL;
        } else {
            return result;
        }
    }
}
static CYTHON_INLINE int __Pyx_PyObject_IsTrue(PyObject* x) {
   int is_true = x == Py_True;
   if (is_true | (x == Py_False) | (x == Py_None)) return is_true;
```

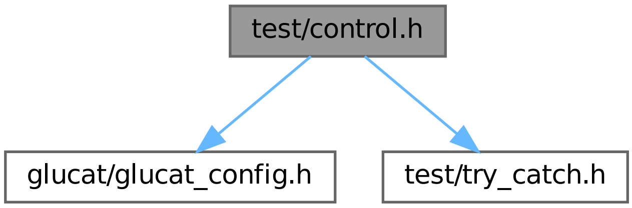
```
  else return PyObject_IsTrue(x);
}
static CYTHON_INLINE int __Pyx_PyObject_IsTrueAndDecref(PyObject* x) {
    int retval;
    if (unlikely(!x)) return -1;
    retval = __Pyx_PyObject_IsTrue(x);
    Py_DECREF(x);
    return retval;
}
static PyObject* __Pyx_PyNumber_IntOrLongWrongResultType(PyObject* result, const char* type_name) {
#if PY_MAJOR_VERSION >= 3
    if (PyLong_Check(result)) {
        if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
                "__int__ returned non-int (type %.200s).  "
                "The ability to return an instance of a strict subclass of int "
                "is deprecated, and may be removed in a future version of Python.",
                Py_TYPE(result)->tp_name)) {
            Py_DECREF(result);
            return NULL;
        }
        return result;
    }
#endif
    PyErr_Format(PyExc_TypeError,
                 "__%.4s__ returned non-%.4s (type %.200s)",
                 type_name, type_name, Py_TYPE(result)->tp_name);
    Py_DECREF(result);
    return NULL;
}
static CYTHON_INLINE PyObject* __Pyx_PyNumber_IntOrLong(PyObject* x) {
#if CYTHON_USE_TYPE_SLOTS
  PyNumberMethods *m;
#endif
  const char *name = NULL;
  PyObject *res = NULL;
#if PY_MAJOR_VERSION < 3
  if (likely(PyInt_Check(x) || PyLong_Check(x)))
#else
  if (likely(PyLong_Check(x)))
#endif
    return __Pyx_NewRef(x);
#if CYTHON_USE_TYPE_SLOTS
  m = Py_TYPE(x)->tp_as_number;
  #if PY_MAJOR_VERSION < 3
  if (m && m->nb_int) {
    name = "int";
    res = m->nb_int(x);
  }
  else if (m && m->nb_long) {
    name = "long";
    res = m->nb_long(x);
  }
  #else
  if (likely(m && m->nb_int)) {
    name = "int";
    res = m->nb_int(x);
  }
  #endif
#else
  if (!PyBytes_CheckExact(x) && !PyUnicode_CheckExact(x)) {
    res = PyNumber_Int(x);
  }
#endif
  if (likely(res)) {
#if PY_MAJOR_VERSION < 3
    if (unlikely(!PyInt_Check(res) && !PyLong_Check(res))) {
#else
    if (unlikely(!PyLong_CheckExact(res))) {
#endif
        return __Pyx_PyNumber_IntOrLongWrongResultType(res, name);
    }
  }
  else if (!PyErr_Occurred()) {
    PyErr_SetString(PyExc_TypeError,
                    "an integer is required");
  }
  return res;
}
static CYTHON_INLINE Py_ssize_t __Pyx_PyIndex_AsSsize_t(PyObject* b) {
  Py_ssize_t ival;
  PyObject *x;
#if PY_MAJOR_VERSION < 3
  if (likely(PyInt_CheckExact(b))) {
    if (sizeof(Py_ssize_t) >= sizeof(long))
        return PyInt_AS_LONG(b);
    else
        return PyInt_AsSsize_t(b);
```

```
  }
#endif
  if (likely(PyLong_CheckExact(b))) {
    #if CYTHON_USE_PYLONG_INTERNALS
    const digit* digits = ((PyLongObject*)b)->ob_digit;
    const Py_ssize_t size = Py_SIZE(b);
    if (likely(__Pyx_sst_abs(size) <= 1)) {
        ival = likely(size) ? digits[0] : 0;
        if (size == -1) ival = -ival;
        return ival;
    } else {
      switch (size) {
         case 2:
           if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {
             return (Py_ssize_t) (((((size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
         case -2:
           if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {
             return -(Py_ssize_t) (((((size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
         case 3:
           if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {
             return (Py_ssize_t) (((((((size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
         case -3:
           if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {
             return -(Py_ssize_t) (((((((size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
         case 4:
           if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {
             return (Py_ssize_t) (((((((((size_t)digits[3]) << PyLong_SHIFT) | (size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
         case -4:
           if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {
             return -(Py_ssize_t) (((((((((size_t)digits[3]) << PyLong_SHIFT) | (size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]));
           }
           break;
      }
    }
    #endif
    return PyLong_AsSsize_t(b);
  }
  x = PyNumber_Index(b);
  if (!x) return -1;
  ival = PyInt_AsSsize_t(x);
  Py_DECREF(x);
  return ival;
}
static CYTHON_INLINE Py_hash_t __Pyx_PyIndex_AsHash_t(PyObject* o) {
  if (sizeof(Py_hash_t) == sizeof(Py_ssize_t)) {
    return (Py_hash_t) __Pyx_PyIndex_AsSsize_t(o);
#if PY_MAJOR_VERSION < 3
  } else if (likely(PyInt_CheckExact(o))) {
    return PyInt_AS_LONG(o);
#endif
  } else {
    Py_ssize_t ival;
    PyObject *x;
    x = PyNumber_Index(o);
    if (!x) return -1;
    ival = PyInt_AsLong(x);
    Py_DECREF(x);
    return ival;
  }
}
static CYTHON_INLINE PyObject * __Pyx_PyBool_FromLong(long b) {
  return b ? __Pyx_NewRef(Py_True) : __Pyx_NewRef(Py_False);
}
static CYTHON_INLINE PyObject * __Pyx_PyInt_FromSize_t(size_t ival) {
    return PyInt_FromSize_t(ival);
}


#endif /* Py_PYTHON_H */
```

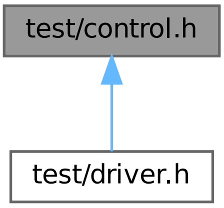
7.65 test/control.h File Reference

```
#include "glucat/glucat_config.h"
#include "test/try_catch.h"
```

Include dependency graph for control.h:



This graph shows which files directly or indirectly include this file:



Classes

- class glucat::control_t

  Parameters to control tests.

Namespaces

- namespace glucat

7.66 control.h

Go to the documentation of this file.

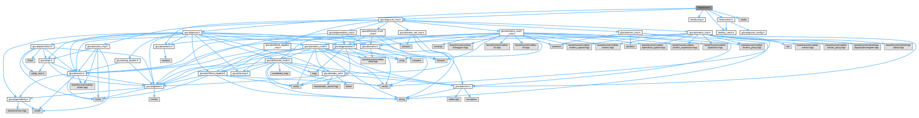
```
#ifndef _GLUCAT_CONTROL_H
#define _GLUCAT_CONTROL_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    control.h : Define and set parameters to control tests
                             -------------------
    begin                : 2010-04-21
    copyright            : (C) 2010-2016 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/
#include "glucat/glucat_config.h"
#include "test/try_catch.h"

namespace glucat
{
  class control_t
  {
  private:
    bool m_valid;
    bool valid() const
    { return m_valid; }

    bool m_catch_exceptions;
    bool catch_exceptions() const
    { return m_catch_exceptions; }

    static bool m_verbose_output;

    control_t(int argc, char ** argv);
    // Enforce singleton
    // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
    control_t() = default;
    ~control_t() = default;
    control_t(const control_t&) = delete;
    control_t& operator= (const control_t&) = delete;

    friend class friend_for_private_destructor;
  public:
    static const control_t& control(int argc, char ** argv)
    { static const control_t c(argc, argv); return c; }

    int call(intfn f) const;
    int call(intintfn f, int arg) const;

    static bool verbose()
    { return m_verbose_output; }
  };

  bool control_t::m_verbose_output = false;

  control_t::
  control_t(int argc, char ** argv)
  : m_valid(true), m_catch_exceptions(true)
  {
    bool print_help = false;
    const std::string& arg_0_str = argv[0];
    const std::string program_name = arg_0_str.substr(arg_0_str.find_last_of('/')+1);
    for (int arg_ndx = 1; arg_ndx < argc; ++arg_ndx)
    {
      const std::string& arg_str = argv[arg_ndx];
```

```
   bool valid = false;
   if (arg_str.substr(0,2) == "--")
   {
    valid = true;
    const std::string& arg_name = arg_str.substr(2);
    if (arg_name == "help")
    {
     this->m_valid = false;
     print_help = true;
    }
    else if (arg_name == "verbose")
     this->m_verbose_output = true;
    else if (arg_name == "no-catch")
     this->m_catch_exceptions = false;
    else
     valid = false;
   }
   if (!valid)
   {
    std::cout « "Invalid argument: " « arg_str « std::endl;
    this->m_valid = false;
    print_help = true;
   }
  }
  if (print_help)
  {
   std::cout « program_name « " for " « GLUCAT_PACKAGE_NAME « " version " « GLUCAT_VERSION « ":" « std::endl;
   std::cout « "Usage:  " « program_name « " [option ...]" « std::endl;
   std::cout « "Options:" « std::endl;
   std::cout « "  --help      : Print this summary." « std::endl;
   std::cout « "  --no-catch  : Do not catch exceptions." « std::endl;
   std::cout « "  --verbose   : Produce more detailed test output." « std::endl;
  }
 }

 inline
 int
 control_t::
 call(intfn f) const
 {
  if (valid())
   return (catch_exceptions())
    ? try_catch(f)
    : (*f)();
  else
   return 1;
 }

 inline
 int
 control_t::
 call(intintfn f, int arg) const
 {
  if (valid())
   return (catch_exceptions())
    ? try_catch(f, arg)
    : (*f)(arg);
  else
   return 1;
 }
}
#endif // _GLUCAT_CONTROL_H
```

7.67 test/driver.h File Reference

```
#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include "test/tuning.h"
#include "test/try_catch.h"
#include "test/control.h"
#include <cstdio>
```

Include dependency graph for driver.h:



7.68 driver.h

[Go to the documentation of this file.]

```
#ifndef GLUCAT_TEST_DRIVER_H
#define GLUCAT_TEST_DRIVER_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    driver.h : Header for example and timing test driver
                             -------------------
    begin                : Sun 2001-12-09
    copyright            : (C) 2001-2021 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include "test/tuning.h"
#include "test/try_catch.h"
#include "test/control.h"
#include <cstdio>

#endif // GLUCAT_TEST_DRIVER_H
```

7.69 test/timing.h File Reference

Namespaces

- namespace glucat
- namespace glucat::timing

Functions

- static double glucat::timing::elapsed (clock_t cpu_time)

  Elapsed time in milliseconds.

Variables

- const double glucat::timing::MS_PER_SEC = 1000.0

  Timing constant: milliseconds per second.
- const double glucat::timing::MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC)

  Timing constant: milliseconds per clock.
- const int glucat::timing::EXTRA_TRIALS = 2

  Timing constant: trial expansion factor.

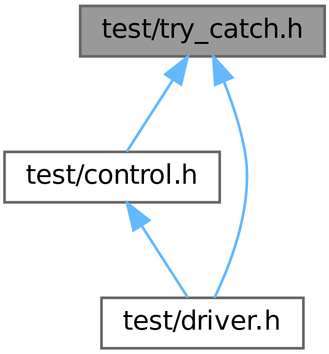
7.70 timing.h

Go to the documentation of this file.

```
#ifndef GLUCAT_TEST_TIMING_H
#define GLUCAT_TEST_TIMING_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    timing.h : Common definitions for timing tests
                             -------------------
    begin                : Tue 2012-03-27
    copyright            : (C) 2012 by Paul C. Leopardi
 ***************************************************************************

    This library is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published
    by the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

namespace glucat
{
  namespace timing
  {
    const double MS_PER_SEC = 1000.0;

    const double MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC);

    const int EXTRA_TRIALS = 2;

    inline
    static
    double
    elapsed(clock_t cpu_time)
    { return double(clock() - cpu_time) * MS_PER_CLOCK; }

  }
}
#endif // GLUCAT_TEST_TIMING_H
```

7.71 test/try_catch.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace glucat

Typedefs

- typedef int(∗ glucat::intfn) ()

  For exception catching: pointer to function returning int.
- typedef int(∗ glucat::intintfn) (int)

  For exception catching: pointer to function of int returning int.

Functions

- int glucat::try_catch (intfn f)

  Exception catching for functions returning int.
- int glucat::try_catch (intintfn f, int arg)

  Exception catching for functions of int returning int.

7.72 try_catch.h

Go to the documentation of this file.

```
#ifndef _GLUCAT_TRY_CATCH_H
#define _GLUCAT_TRY_CATCH_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    try_catch.h : Catch exceptions
                             -------------------
    begin                : Sun 2001-12-20
    copyright            : (C) 2001-2010 by Paul C. Leopardi
 ***************************************************************************
```

```
   This library is free software: you can redistribute it and/or modify
   it under the terms of the GNU Lesser General Public License as published
   by the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public License
   along with this library.  If not, see <http://www.gnu.org/licenses/>.

 ***************************************************************************
 This library is based on a prototype written by Arvind Raja and was
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
 ***************************************************************************
     See also Arvind Raja's original header comments in glucat.h
 ***************************************************************************/

namespace glucat
{
  typedef int (*intfn)();

  typedef int (*intintfn)(int);

  int try_catch(intfn f);

  int try_catch(intintfn f, int arg);

  int try_catch(intfn f)
  {
    int result = 0;
    try
      { result = (*f)(); }
    catch (const glucat_error& e)
      { e.print_error_msg(); }
    catch (const std::bad_alloc& e)
      { std::cerr « "bad_alloc" « std::endl; }
    catch (...)
      { std::cerr « "unexpected exception" « std::endl; }
    return result;
  }

  int try_catch(intintfn f, int arg)
  {
    int result = 0;
    try
      { result = (*f)(arg); }
    catch (const glucat_error& e)
      { e.print_error_msg(); }
    catch (const std::bad_alloc& e)
      { std::cerr « "bad_alloc" « std::endl; }
    catch (...)
      { std::cerr « "unexpected exception" « std::endl; }
    return result;
  }
}
#endif // _GLUCAT_TRY_CATCH_H
```

Index