glucat

0.13.0

Generated by Doxygen 1.14.0

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, BitsPerULongDoesNotMatchSetValueT) 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

- 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()

  Default for truncation.
- 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.

References BITS_PER_SET_VALUE.

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

References _GLUCAT_CTAssert(), and DEFAULT_HI.

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

Referenced by _GLUCAT_CTAssert().

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 902 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 882 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 843 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 824 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 1007 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 987 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 948 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 929 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 1107 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 1087 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 1051 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 1034 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>
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 1910 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>
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 689 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 873 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 850 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 806 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>
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 1344 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>
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>
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 1704 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>
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 1317 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>
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 1305 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 1745 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 2074 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>
auto glucat::fast (
            const Matrix_T & X,
            index_t level) -> Multivector_T  [static]
```

Inverse generalized Fast Fourier Transform.

Definition at line 1024 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>
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()

```
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()

```
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 1795 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 2033 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 798 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 790 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 1957 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 1563 of file matrix_multi_imp.h.

References abs(), approx_equal(), glucat::matrix::classify_eigenvalues(), cr_sqrt(), db_sqrt(), pade::pade_sqrt_denom< Scalar_T >::
exp(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), matrix_sqrt(), norm(), pade::pade_sqrt_numer< Sca
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.

Referenced by operator<<().

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.

Referenced by operator<<().

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.

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 579 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 560 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 500 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 1712 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 731 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 612 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 1144 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 &
```

References max_pos(), min_neg(), and sign_of_square().

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 952 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 1204 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 1243 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 963 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 541 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 756 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 714 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>
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 1242 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>
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 1890 of file matrix_multi_imp.h.

References pade::pade_log_denom< Scalar_T >::denom, glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan(), pade::pade_log_numer< Scalar_T >::numer, 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_T, ..., 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 343 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.

Referenced by operator<<().

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 978 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 955 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 910 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 1722 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 1657 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 682 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 674 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 683 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 598 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 1078 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 1059 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 1015 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.

Referenced by _GLUCAT_CTAssert().

5.2.3.2 clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::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()
```

Default for truncation.

Definition at line 74 of file clifford_algebra_imp.h.

5.2.3.3 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.

Referenced by _GLUCAT_CTAssert().

5.2.3.4 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().

5.2.3.5 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().

5.2.3.6 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.

5.2.3.7 Tuning_Fast_Basis_Max_Count

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

Definition at line 92 of file tuning.h.

5.2.3.8 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.

5.2.3.9 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.

5.2.3.10 Tuning_Fast_Div_Max_Steps

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

Definition at line 87 of file tuning.h.

5.2.3.11 Tuning_Fast_Fast_Size_Threshold

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

Definition at line 93 of file tuning.h.

5.2.3.12 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.

5.2.3.13 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.

5.2.3.14 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.

5.2.3.15 Tuning_Fast_Mult_Matrix_Threshold

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

Definition at line 86 of file tuning.h.

5.2.3.16 Tuning_Fast_Products_Size_Threshold

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

Definition at line 95 of file tuning.h.

5.2.3.17 Tuning_Int_Digits

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

Definition at line 36 of file tuning.h.

5.2.3.18 Tuning_Max_Threshold

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

Definition at line 37 of file tuning.h.

5.2.3.19 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.

5.2.3.20 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.

5.2.3.21 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.

5.2.3.22 Tuning_Naive_Mult_Matrix_Threshold

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

Definition at line 64 of file tuning.h.

5.2.3.23 Tuning_Slow_Basis_Max_Count

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

Definition at line 42 of file tuning.h.

5.2.3.24 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.

5.2.3.25 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.

5.2.3.26 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.

5.2.3.27 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_blaze (const Matrix_T &val) -> blaze::DynamicMatrix< double, blaze::rowMajor >

  Convert matrix to Blaze 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 492 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, glucat::numeric_traits< Scalar_T >::pi(), and glucat::numeric_traits< Scalar_T

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 464 of file matrix_imp.h.

References to_blaze().

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 368 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 270 of file matrix_imp.h.

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

5.4.2.5 isnan()

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

Not a Number.

Definition at line 287 of file matrix_imp.h.

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

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 78 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 114 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 315 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 253 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 177 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 147 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 390 of file matrix_imp.h.

References glucat::numeric_traits< Scalar_T >::isNaN(), and 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 356 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 223 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 345 of file matrix_imp.h.

5.4.2.16 to_blaze()

```
template<typename Matrix_T>
auto glucat::matrix::to_blaze (
            const Matrix_T & val) -> blaze::DynamicMatrix<double,blaze::rowMajor> [static]
```

Convert matrix to Blaze format.

Definition at line 435 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 411 of file matrix_imp.h.

References glucat::numeric_traits< Scalar_T >::isNaN(), and 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 305 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, LC
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()

```
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 >

Variables

- template<typename Scalar_T>
  const pade_sqrt_numer< Scalar_T >::array pade_sqrt_numer< Scalar_T >::numer
- template<typename Scalar_T>
  const pade_sqrt_denom< Scalar_T >::array pade_sqrt_denom< Scalar_T >::denom
- const pade_sqrt_numer< float >::array pade_sqrt_numer< float >::numer
- const pade_sqrt_denom< float >::array pade_sqrt_denom< float >::denom
- const pade_sqrt_numer< longdouble >::array pade_sqrt_numer< longdouble >::numer
- const pade_sqrt_denom< longdouble >::array pade_sqrt_denom< longdouble >::denom
- const pade_sqrt_numer< dd_real >::array pade_sqrt_numer< dd_real >::numer
- const pade_sqrt_denom< dd_real >::array pade_sqrt_denom< dd_real >::denom
- const pade_sqrt_numer< qd_real >::array pade_sqrt_numer< qd_real >::numer
- const pade_sqrt_denom< qd_real >::array pade_sqrt_denom< qd_real >::denom
- template<typename Scalar_T>
  const pade_log_numer< Scalar_T >::array pade_log_numer< Scalar_T >::numer
- template<typename Scalar_T>
  const pade_log_denom< Scalar_T >::array pade_log_denom< Scalar_T >::denom
- const pade_log_numer< float >::array pade_log_numer< float >::numer
- const pade_log_denom< float >::array pade_log_denom< float >::denom
- const pade_log_numer< longdouble >::array pade_log_numer< longdouble >::numer
- const pade_log_denom< longdouble >::array pade_log_denom< longdouble >::denom
- const pade_log_numer< dd_real >::array pade_log_numer< dd_real >::numer
- const pade_log_denom< dd_real >::array pade_log_denom< dd_real >::denom
- const pade_log_numer< qd_real >::array pade_log_numer< qd_real >::numer
- const pade_log_denom< qd_real >::array pade_log_denom< qd_real >::denom

5.6.1 Variable Documentation

5.6.1.1 pade_log_denom< dd_real >::denom

```
const pade_log_denom<dd_real>::array pade::pade_log_denom< dd_real >::denom
```

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 1815 of file matrix_multi_imp.h.

5.6.1.2 pade_log_denom< float >::denom

```
const pade_log_denom<float>::array pade::pade_log_denom< float >::denom
```

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 1753 of file matrix_multi_imp.h.

5.6.1.3 pade_log_denom< longdouble >::denom

```
const pade_log_denom<longdouble>::array pade::pade_log_denom< longdouble >::denom
```

Initial value:

```
=
  {
        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
  }
```

Definition at line 1780 of file matrix_multi_imp.h.

5.6.1.4 pade_log_denom< qd_real >::denom

```
const pade_log_denom<qd_real>::array pade::pade_log_denom< qd_real >::denom
```

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("627557630"),
          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")
  }
```

Definition at line 1862 of file matrix_multi_imp.h.

5.6.1.5 pade_log_denom< Scalar_T >::denom

```
template<typename Scalar_T>
const pade_log_denom<Scalar_T>::array pade::pade_log_denom< Scalar_T >::denom
```

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 1727 of file matrix_multi_imp.h.

5.6.1.6 pade_log_numer< dd_real >::numer

```
const pade_log_numer<dd_real>::array pade::pade_log_numer< dd_real >::numer
```

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 1795 of file matrix_multi_imp.h.

5.6.1.7 pade_log_numer< float >::numer

```
const pade_log_numer<float>::array pade::pade_log_numer< float >::numer
```

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 1741 of file matrix_multi_imp.h.

5.6.1.8 pade_log_numer< longdouble >::numer

```
const pade_log_numer<longdouble>::array pade::pade_log_numer< longdouble >::numer
```

Initial value:

```
=
  {
        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
  }
```

Definition at line 1766 of file matrix_multi_imp.h.

5.6.1.9 pade_log_numer< qd_real >::numer

```
const pade_log_numer<qd_real>::array pade::pade_log_numer< qd_real >::numer
```

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("3192507080522464425"),
      qd_real("236648286272519")/qd_real("14424919747503542550 0"),
   qd_real("260715545088119")/qd_real("4375558990076074573500"),
      qd_real("289596255666839")/qd_real("19287464028255336719988 0"),
     qd_real("8802625510547")/qd_real("3616399505297875634997 75"),
      qd_real("373831661521439")/qd_real("165920409303066534133696 7700"),
   qd_real("446033437968239")/qd_real("46457714604858629557435095 6000"),
      qd_real("53676090078349")/qd_real("4738686889695580214858379 7512000")
      }
```

Definition at line 1836 of file matrix_multi_imp.h.

5.6.1.10 pade_log_numer< Scalar_T >::numer

```
template<typename Scalar_T>
const pade_log_numer<Scalar_T>::array pade::pade_log_numer< Scalar_T >::numer
```

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 1710 of file matrix_multi_imp.h.

5.6.1.11 pade_sqrt_denom< dd_real >::denom

```
const pade_sqrt_denom<dd_real>::array pade::pade_sqrt_denom< dd_real >::denom
```

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 1488 of file matrix_multi_imp.h.

5.6.1.12 pade_sqrt_denom< float >::denom

```
const pade_sqrt_denom<float>::array pade::pade_sqrt_denom< float >::denom
```

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 1425 of file matrix_multi_imp.h.

5.6.1.13 pade_sqrt_denom< longdouble >::denom

```
const pade_sqrt_denom<longdouble>::array pade::pade_sqrt_denom< longdouble >::denom
```

Initial value:

```
=
{
      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
}
```

Definition at line 1452 of file matrix_multi_imp.h.

5.6.1.14 pade_sqrt_denom< qd_real >::denom

```
const pade_sqrt_denom<qd_real>::array pade::pade_sqrt_denom< qd_real >::denom
```

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 1535 of file matrix_multi_imp.h.

5.6.1.15 pade_sqrt_denom< Scalar_T >::denom

```
template<typename Scalar_T>
const pade_sqrt_denom<Scalar_T>::array pade::pade_sqrt_denom< Scalar_T >::denom
```

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 1399 of file matrix_multi_imp.h.

5.6.1.16 pade_sqrt_numer< dd_real >::numer

```
const pade_sqrt_numer<dd_real>::array pade::pade_sqrt_numer< dd_real >::numer
```

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 1468 of file matrix_multi_imp.h.

5.6.1.17 pade_sqrt_numer< float >::numer

```
const pade_sqrt_numer<float>::array pade::pade_sqrt_numer< float >::numer
```

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 1413 of file matrix_multi_imp.h.

5.6.1.18 pade_sqrt_numer< longdouble >::numer

```
const pade_sqrt_numer<longdouble>::array pade::pade_sqrt_numer< longdouble >::numer
```

Initial value:

```
=
  {
        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
  }
```

Definition at line 1438 of file matrix_multi_imp.h.

5.6.1.19 pade_sqrt_numer< qd_real >::numer

```
const pade_sqrt_numer<qd_real>::array pade::pade_sqrt_numer< qd_real >::numer
```

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 1509 of file matrix_multi_imp.h.

5.6.1.20 pade_sqrt_numer< Scalar_T >::numer

```
template<typename Scalar_T>
const pade_sqrt_numer<Scalar_T>::array pade::pade_sqrt_numer< Scalar_T >::numer
```

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 1382 of file matrix_multi_imp.h.

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 2025 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 1316 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 1999 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 2012 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 1973 of file PyClical.pyx.

5.7.2.3 fill

```
PyClical.fill
```

Definition at line 1927 of file PyClical.pyx.

5.7.2.4 i

```
PyClical.i
```

Definition at line 1654 of file PyClical.pyx.

5.7.2.5 ist

```
PyClical.ist = index_set
```

Definition at line 1991 of file PyClical.pyx.

5.7.2.6 ixt

```
PyClical.ixt
```

Definition at line 1927 of file PyClical.pyx.

5.7.2.7 lhs

```
PyClical.lhs
```

Definition at line 1422 of file PyClical.pyx.

5.7.2.8 nbar3

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

Definition at line 2022 of file PyClical.pyx.

5.7.2.9 ninf3

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

Definition at line 2021 of file PyClical.pyx.

5.7.2.10 None

```
PyClical.None
```

Definition at line 1422 of file PyClical.pyx.

5.7.2.11 obj

```
PyClical.obj
```

Definition at line 1654 of file PyClical.pyx.

5.7.2.12 pi

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

Definition at line 1970 of file PyClical.pyx.

5.7.2.13 rhs

```
PyClical.rhs
```

Definition at line 1422 of file PyClical.pyx.

5.7.2.14 scalar_epsilon

```
PyClical.scalar_epsilon = epsilon
```

Definition at line 1968 of file PyClical.pyx.

5.7.2.15 tau

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

Definition at line 1971 of file PyClical.pyx.

5.7.2.16 threshold

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

Definition at line 1422 of file PyClical.pyx.

5.7.2.17 tol

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

Definition at line 1422 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

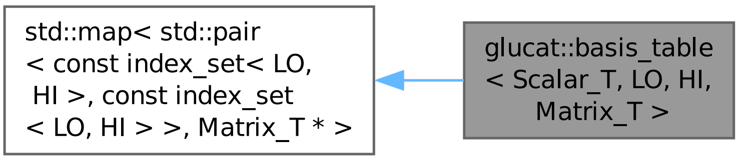
Class Documentation

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

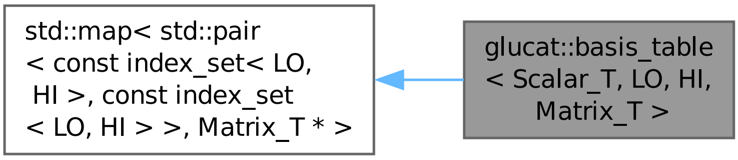
Table of basis elements used as a cache by basis_element()

`#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 1159 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]
```

Referenced by basis(), basis_table(), and operator=().

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]
```

References basis_table().

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

Single instance of basis table.

Definition at line 1165 of file matrix_multi_imp.h.

References basis_table().

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]
```

References basis_table().

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 1170 of file matrix_multi_imp.h.

References friend_for_private_destructor.

Referenced by friend_for_private_destructor.

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	

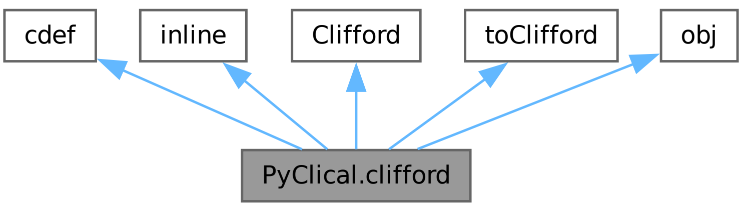
Definition at line 72 of file global.h.

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

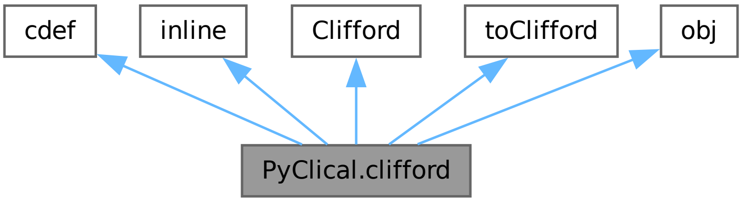
- glucat/global.h

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)
- __radd__ (rhs, lhs)
- __iadd__ (self, rhs)
- __sub__ (lhs, rhs)
- __rsub__ (rhs, lhs)
- __isub__ (self, rhs)
- __mul__ (lhs, rhs)
- __rmul__ (rhs, lhs)
- __imul__ (self, rhs)
- __mod__ (lhs, rhs)
- __rmod__ (rhs, lhs)
- __imod__ (self, rhs)
- __and__ (lhs, rhs)
- __rand__ (rhs, lhs)
- __iand__ (self, rhs)
- __xor__ (lhs, rhs)
- __rxor__ (rhs, lhs)
- __ixor__ (self, rhs)
- __truediv__ (lhs, rhs)
- __rtruediv__ (rhs, lhs)
- __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 532 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 872 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 1083 of file PyClical.pyx.

References 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 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 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 760 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 896 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 974 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 857 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 820 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 1013 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 789 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 935 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 833 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 798 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 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 1002 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 1024 of file PyClical.pyx.

References pow().

6.3.2.23 __radd__()

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

```
Geometric sum.

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

Definition at line 751 of file PyClical.pyx.

6.3.2.24 __rand__()

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

```
Inner product.

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

Definition at line 887 of file PyClical.pyx.

6.3.2.25 __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 1298 of file PyClical.pyx.

References clifford_to_repr().

6.3.2.26 __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.

6.3.2.27 __rmod__()

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

```
Contraction.

>>> print(2 % clifford("{2}"))
2{2}
```

Definition at line 848 of file PyClical.pyx.

6.3.2.28 __rmul__()

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

```
Geometric product.

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

Definition at line 811 of file PyClical.pyx.

6.3.2.29 __rsub__()

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

```
Geometric difference.

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

Definition at line 780 of file PyClical.pyx.

6.3.2.30 __rtruediv__()

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

```
Geometric quotient.

>>> print(2 / clifford("{2}"))
2{2}
```

Definition at line 965 of file PyClical.pyx.

6.3.2.31 __rxor__()

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

```
Outer product.

>>> print(2 ^ clifford("{2}"))
2{2}
```

Definition at line 926 of file PyClical.pyx.

6.3.2.32 __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 1307 of file PyClical.pyx.

References clifford_to_str().

6.3.2.33 __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 769 of file PyClical.pyx.

6.3.2.34 __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 950 of file PyClical.pyx.

6.3.2.35 __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 911 of file PyClical.pyx.

6.3.2.36 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 1238 of file PyClical.pyx.

References glucat.abs().

6.3.2.37 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 1201 of file PyClical.pyx.

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

Referenced by conj().

6.3.2.38 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 1124 of file PyClical.pyx.

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

Referenced by even().

6.3.2.39 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 1287 of file PyClical.pyx.

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

Referenced by frame().

6.3.2.40 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 989 of file PyClical.pyx.

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

Referenced by inv().

6.3.2.41 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 1170 of file PyClical.pyx.

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

Referenced by involute().

6.3.2.42 isinf()

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

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

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

Definition at line 1269 of file PyClical.pyx.

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

6.3.2.43 isnan()

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

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

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

Definition at line 1278 of file PyClical.pyx.

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

Referenced by isinf(), and isnan().

6.3.2.44 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 1247 of file PyClical.pyx.

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

Referenced by max_abs().

6.3.2.45 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 1227 of file PyClical.pyx.

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

Referenced by norm().

6.3.2.46 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 1133 of file PyClical.pyx.

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

Referenced by odd().

6.3.2.47 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 1067 of file PyClical.pyx.

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

Referenced by outer_pow().

6.3.2.48 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 1043 of file PyClical.pyx.

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

Referenced by __pow__(), and pow().

6.3.2.49 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 1113 of file PyClical.pyx.

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

Referenced by pure().

6.3.2.50 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 1216 of file PyClical.pyx.

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

Referenced by quad().

6.3.2.51 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.

6.3.2.52 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 1186 of file PyClical.pyx.

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

Referenced by reverse().

6.3.2.53 scalar()

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

```
Scalar part.

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

Definition at line 1102 of file PyClical.pyx.

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

Referenced by scalar().

6.3.2.54 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 1258 of file PyClical.pyx.

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

Referenced by truncated().

6.3.2.55 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 1142 of file PyClical.pyx.

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

Referenced by 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 __call__(), __dealloc__(), __getitem__(), __neg__(), conj(), even(), frame(), inv(), involute(), isinf(), isnan(), max_abs(), norm(), odd(), outer_pow(), pow(), pure(), quad(), reverse(), scalar(), truncated(), and vector_part().

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

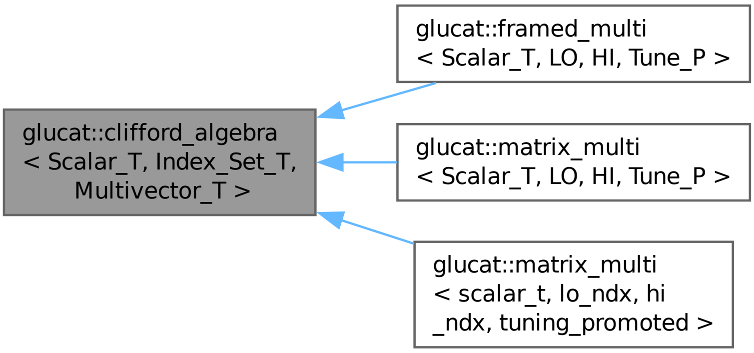
- pyclical/PyClical.pyx

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>
```

Inheritance diagram for glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >:



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

  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.

References conj().

Referenced by conj().

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.

References even().

Referenced by even().

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.

References frame().

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

References grade().

Referenced by grade(), and operator()().

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.

References inv().

Referenced by inv().

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}.

References involute().

Referenced by involute().

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.

References isinf().

Referenced by isinf().

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.

References isnan().

Referenced by glucat::cascade_log(), glucat::exp(), isnan(), 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.

References max_abs().

Referenced by max_abs().

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.

References norm().

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

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.

References odd().

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

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.

References grade().

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.

References outer_pow().

Referenced by outer_pow().

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

References pow().

Referenced by pow().

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.

References pure().

Referenced by pure().

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)

References quad().

Referenced by quad().

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}.

References reverse().

Referenced by reverse().

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.

References scalar().

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

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.

References default_truncation, and truncated().

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

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()

References vector_part().

Referenced by vector_part(), and vector_part().

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.

References vector_part().

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.

References write().

Referenced by write(), and write().

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.

References write().

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  [static]
```

Default for truncation.

Definition at line 59 of file clifford_algebra.h.

Referenced by truncated().

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 }
- enum

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

```
anonymous enum
```

Definition at line 57 of file global.h.

6.6.2.2 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().

Referenced by control(), control_t(), and operator=().

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]
```

References control_t().

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()

```
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.

References control_t().

6.7.3.5 operator=()

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

References control_t().

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()

```
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.

References friend_for_private_destructor.

Referenced by friend_for_private_destructor.

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
- 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 76 of file promotion.h.

6.10.2 Member Typedef Documentation

6.10.2.1 type [1/2]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 78 of file promotion.h.

6.10.2.2 type [2/2]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 148 of file scalar.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:

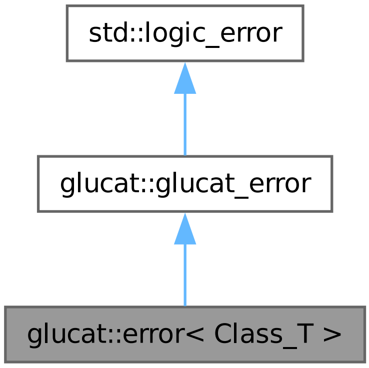
- glucat/matrix.h

6.12 glucat::error< Class_T > Class Template Reference

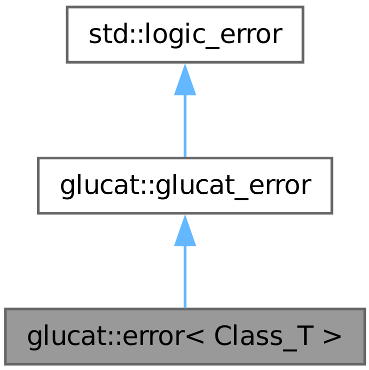
Specific exception class.

#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.

References classname(), and glucat::glucat_error::glucat_error().

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.

References glucat::glucat_error::glucat_error().

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.

References glucat::glucat_error::name.

Referenced by error(), and print_error_msg().

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.

Referenced by print_error_msg().

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.

References classname(), and heading().

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

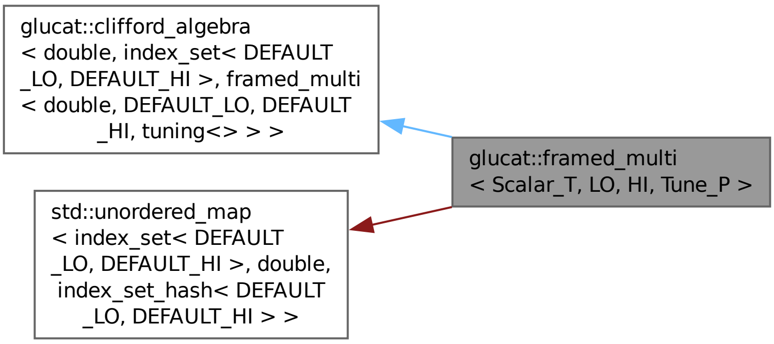
- glucat/errors.h
- glucat/errors_imp.h

6.13 glucat::framed_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference

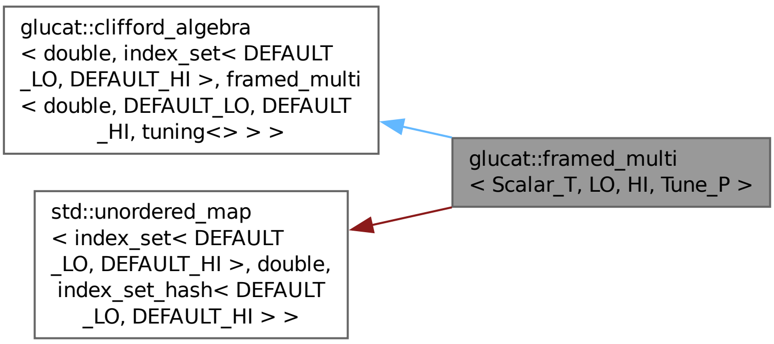
A framed_multi< Scalar_T,LO,HI,Tune_P > is a framed approximation to a multivector.

```
#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< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DE

- 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< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_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 multivector_t &rhs) -> multivector_t &=0

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

  Geometric difference.
- 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

  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

  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 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.

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< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DE

- 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< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DE

- static const index_t v_lo
- static const index_t v_hi
- static const double default_truncation

  Default for truncation.

6.13.1 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::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 126 of file framed_multi.h.

6.13.2 Member Typedef Documentation

6.13.2.1 const_iterator

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT←
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N, framed_multi, and glucat::numeric_traits< Scalar_T >::to_scalar_t().

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_HASH_N, glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame(), framed_multi, and glucat::numeric_traits< Scalar_T >::to_scalar_t().

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_HASH_N, and 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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N.

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_HASH_N, 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.

References _GLUCAT_HASH_N.

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.

References _GLUCAT_HASH_N.

6.13.3.14 framed_multi() [13/15]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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.

6.13.3.15 framed_multi() [14/15]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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.

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_N, _GLUCAT_HASH_SIZE_T, glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element(), fast_framed_multi(), glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame(), glucat::matrix::inner(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix, matrix_multi, glucat::matrix::nnz(), glucat::clifford_algebra< Scalar_T, I
and glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DEFAULT_LO, DEFAULT_H

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 1464 of file framed_multi_imp.h.

References glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DEFAULT_LO, DEF

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 1506 of file framed_multi_imp.h.

References glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DEFAULT_LO, DEF

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 1548 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 1581 of file framed_multi_imp.h.

Referenced by fast().

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 1597 of file framed_multi_imp.h.

References divide(), glucat::matrix::kron(), glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_mult
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 1695 of file framed_multi_imp.h.

Referenced by 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 1663 of file framed_multi_imp.h.

References fold(), matrix_multi, 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 1429 of file framed_multi_imp.h.

Referenced by fast_matrix_multi().

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 1351 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 1054 of file framed_multi_imp.h.

References framed_multi.

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 1446 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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
auto exp (
            const multivector_t & val) -> const multivector_t  [friend]
```

6.13.5.2 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 143 of file framed_multi.h.

Referenced by framed_multi(), framed_multi(), framed_multi(), framed_multi(), and random().

6.13.5.3 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 141 of file framed_multi.h.

Referenced by fast_matrix_multi(), and framed_multi().

6.13.5.4 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.5 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.6 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.7 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.8 operator<< [1/2]

```
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]
```

6.13.5.9 operator<< [2/2]

```
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 term_t & term) -> std::ostream &  [friend]
```

6.13.5.10 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]
```

6.13.5.11 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.12 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 multivector_t & lhs,
            const multivector_t & rhs) -> const multivector_t  [friend]
```

6.13.5.13 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 multivector_t & lhs,
            const multivector_t & rhs) -> Scalar_T  [friend]
```

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

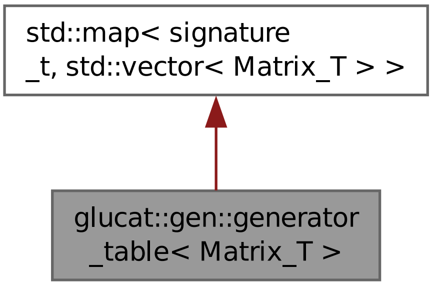
- glucat/framed_multi.h
- glucat/framed_multi_imp.h

6.14 glucat::gen::generator_table< Matrix_T > Class Template Reference

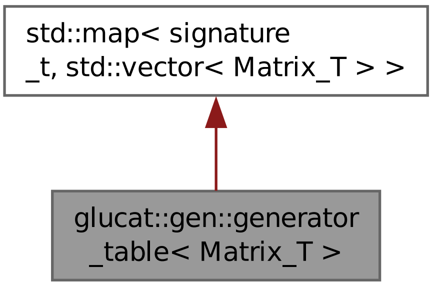
Table of generators for specific signatures.

```
#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]
```

Referenced by generator(), generator_table(), and operator=().

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]
```

References generator_table().

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().

Referenced by gen_vector().

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().

Referenced by gen_vector().

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().

Referenced by gen_vector().

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().

Referenced by gen_vector().

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 gen_from_pm1_qm1(), gen_from_pm4_qp4(), gen_from_pp4_qm4(), gen_from_qp1_pm1(), gen_vector(), glucat::pos_mod(), and glucat::matrix::unit().

Referenced by gen_vector(), and operator()().

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.

References generator_table().

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 gen_vector(), 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]
```

References generator_table().

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.

References friend_for_private_destructor.

Referenced by friend_for_private_destructor.

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

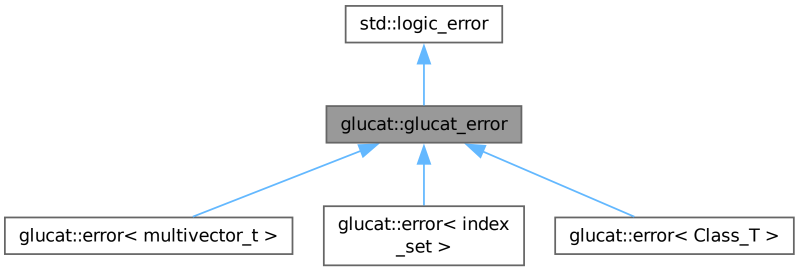
- glucat/generation.h
- glucat/generation_imp.h

6.15 glucat::glucat_error Class Reference

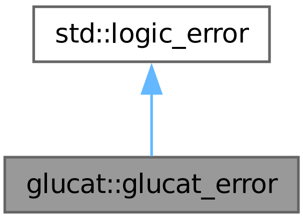
Abstract exception class.

```
#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.

References name.

Referenced by glucat::error< Class_T >::error(), and glucat::error< Class_T >::error().

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 >, glucat::error< index_set >, and glucat::error< multivector_t >.

References classname().

Referenced by classname().

6.15.3.2 heading()

```
virtual auto glucat::glucat_error::heading () const -> const std::string  [pure virtual],
[noexcept]
```

Implemented in glucat::error< Class_T >, glucat::error< index_set >, and glucat::error< multivector_t >.

References heading().

Referenced by heading().

6.15.3.3 print_error_msg()

```
virtual void glucat::glucat_error::print_error_msg () const  [pure virtual]
```

Implemented in glucat::error< Class_T >, glucat::error< index_set >, and glucat::error< multivector_t >.

References print_error_msg().

Referenced by print_error_msg().

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.

Referenced by glucat::error< Class_T >::classname(), and glucat_error().

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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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.

References n.

6.16.3 Member Function Documentation

6.16.3.1 operator()()

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 n.

6.16.4 Member Data Documentation

6.16.4.1 n

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 hash_size_t(), and operator()().

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

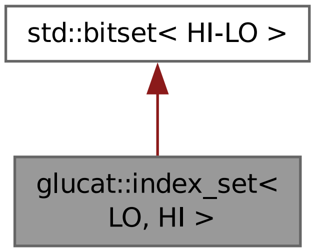
- glucat/framed_multi.h

6.17 glucat::index_set< LO, HI > Class Template Reference

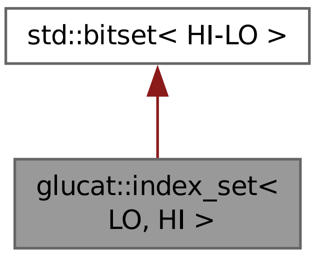
Index set class based on std::bitset<> in Gnu standard C++ library.

```
#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.

Referenced by flip(), fold(), and fold().

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.

References set().

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 count(), fold(), min(), and 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 count_neg(), count_pos(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), index_set(), is_contiguous(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), operator<(), and sign_of_square().

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 count().

Referenced by sign_of_square().

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 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.

References index_set().

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.

References fold(), and index_set().

Referenced by fold(), index_set(), and value_of_fold().

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.

References index_set(), and test().

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.

References count(), max(), and min().

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.

Referenced by operator<().

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.

References test().

Referenced by PyClical.index_set::__iter__(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), is_contiguous(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and unfold().

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.

References test().

Referenced by PyClical.index_set::__iter__(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), index_set(), is_contiguous(), glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi(), and unfold().

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.

References count(), and lex_less_than().

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.

References test().

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.

Referenced by index_set().

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.

References count(), and count_neg().

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.

Referenced by fold(), max(), min(), operator[](), and unfold().

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.

References max(), min(), and test().

Referenced by 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.

References fold().

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]
```

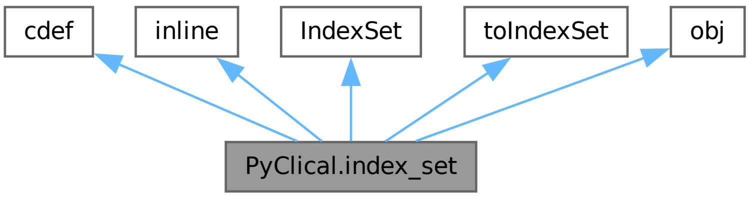
Definition at line 87 of file index_set.h.

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

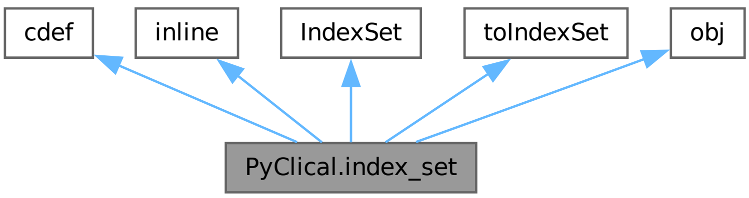
- glucat/index_set.h
- glucat/index_set_imp.h

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())

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 38 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 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 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 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 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< lo_ndx, hi_ndx >.max(), max(), glucat::index_set< LO, HI >.min(
glucat::index_set< lo_ndx, hi_ndx >.min(), and 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 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 count(), and instance.

Referenced by 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 count_neg(), and instance.

Referenced by 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 count_pos(), and instance.

Referenced by 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 hash_fn(), and instance.

Referenced by 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 instance, and max().

Referenced by __iter__(), and 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 instance, and min().

Referenced by __iter__(), and 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 instance, and sign_of_mult().

Referenced by 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 instance, and sign_of_square().

Referenced by sign_of_square().

6.18.3 Member Data Documentation

6.18.3.1 instance

```
PyClical.index_set.instance = new IndexSet((<index_set>other).unwrap())
```

Definition at line 95 of file PyClical.pyx.

Referenced by PyClical.clifford.__call__(), __contains__(), PyClical.clifford.__dealloc__(), __dealloc__(), PyClical.clifford.__getitem__(), __getitem__(), __invert__(), PyClical.clifford.__neg__(), __setitem__(), PyClical.clifford.conj(), count(), count_neg(), count_pos(), PyClical.clifford.even(), PyClical.clifford.frame(), hash_fn(), PyClical.clifford.inv(), PyClical.clifford.involute(), PyClical.clifford.isinf(), PyClical.clifford.isnan(), max(), PyClical.clifford.max_abs(), 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(), sign_of_mult(), 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:

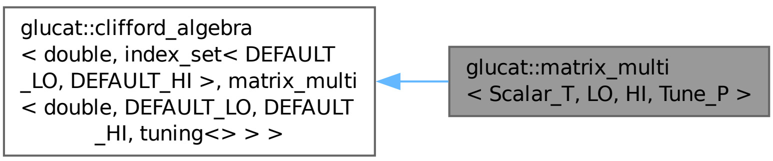
- glucat/framed_multi.h

6.20 glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference

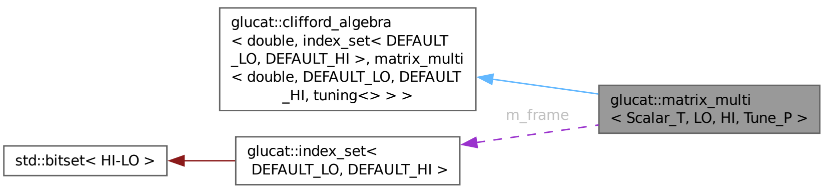
A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector.

```
#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 multivector_t &rhs) -> multivector_t &=0

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

  Geometric difference.
- 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

  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

  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 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.

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 m_frame, and 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 m_frame, m_matrix, matrix_multi, and glucat::numeric_traits< Scalar_T >::to_scalar_t().

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(), m_frame, m_matrix, matrix_multi, 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 m_frame, and m_matrix.

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(), m_frame, and m_matrix.

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(), m_frame, and 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(), m_frame, and 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(), m_frame, 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.

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.

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::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double
framed_multi, m_frame, and m_matrix.

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 276 of file matrix_multi_imp.h.

References fast_matrix_multi(), glucat::folded_dim(), framed_multi, m_frame, m_matrix, and glucat::clifford_algebra< Scalar_T, Index

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 301 of file matrix_multi_imp.h.

References m_frame, m_matrix, and glucat::numeric_traits< Scalar_T >::to_scalar_t().

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 320 of file matrix_multi_imp.h.

References m_frame, and m_matrix.

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 1183 of file matrix_multi_imp.h.

References glucat::gen::generator_table< Matrix_T >::generator(), m_frame, 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 1106 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(), framed_multi, m_frame, m_matrix, glucat::gen::offset_to_super, glucat::pos_mod(), and glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::unfold().

Referenced by fast_matrix_multi().

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 1093 of file matrix_multi_imp.h.

References fast_framed_multi(), fast_matrix_multi(), and m_frame.

Referenced by fast_matrix_multi(), and matrix_multi().

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 414 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 328 of file matrix_multi_imp.h.

References m_frame, and m_matrix.

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 923 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.

Referenced by fast_framed_multi(), matrix_multi(), and matrix_multi().

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 matrix_multi(), matrix_multi(), matrix_multi(), and matrix_multi().

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 basis_element(), fast_framed_multi(), fast_matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), and operator=().

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 fast_framed_multi(), glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), matrix_multi(), and operator=().

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

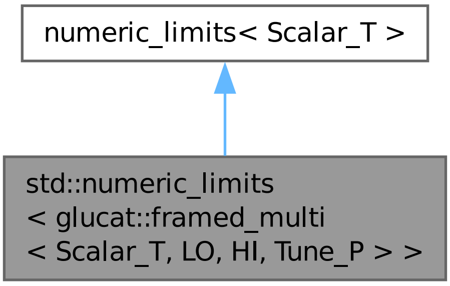
- glucat/framed_multi.h
- glucat/matrix_multi.h
- glucat/matrix_multi_imp.h

6.21 std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference

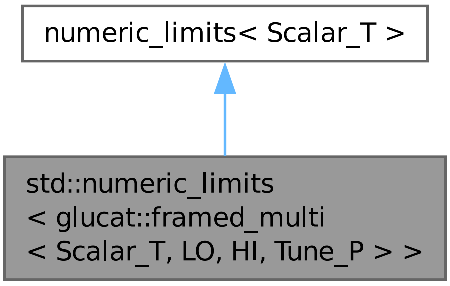
Numeric limits for framed_multi inherit limits for the corresponding scalar type.

`#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:

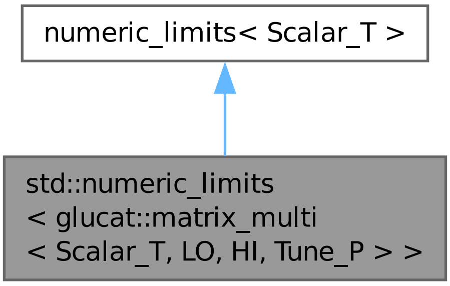
- glucat/framed_multi.h

6.22 std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference

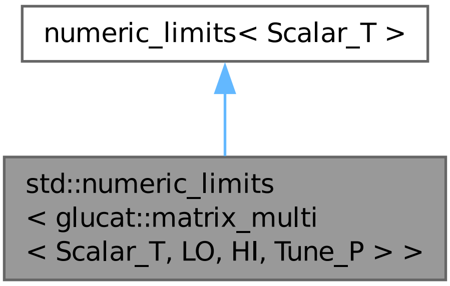
Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

```
#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>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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 isInf().

6.23.2.12 isInf() [2/3]

```
template<typename Scalar_T>
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 isInf(), glucat::matrix::isinf(), and isNaN_or_isInf().

6.23.2.13 isInf() [3/3]

```
template<typename Scalar_T>
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>
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 isNaN().

6.23.2.15 isNaN() [2/3]

```
template<typename Scalar_T>
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 isNaN(), glucat::matrix::isnan(), isNaN_or_isInf(), glucat::matrix::norm_frob2(), and glucat::matrix::trace().

6.23.2.16 isNaN() [3/3]

```
template<typename Scalar_T>
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>
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 isInf(), and 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>
auto glucat::numeric_traits< Scalar_T >::ln_2 () -> Scalar_T   [inline], [static]
```

log(2)

Definition at line 196 of file scalar.h.

Referenced by log2().

6.23.2.20 log()

```
template<typename Scalar_T>
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 log2().

6.23.2.21 log2()

```
template<typename Scalar_T>
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 ln_2(), and log().

Referenced by glucat::log2().

6.23.2.22 NaN()

```
template<typename Scalar_T>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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::matrix::classify_eigenvalues(), glucat::operator<<(), PyFloat_FromDouble(), to_scalar_t(), and to_scalar_t().

6.23.2.33 to_int()

```
template<typename Scalar_T>
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 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 to_double().

6.23.2.40 to_scalar_t() [7/9]

```
template<typename Scalar_T>
template<typename Other_Scalar_T>
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::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi(), glucat::framed_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::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 1721 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 1723 of file matrix_multi_imp.h.

6.24.3 Member Data Documentation

6.24.3.1 denom

```
template<typename Scalar_T>
const array pade::pade_log_denom< Scalar_T >::denom  [static]
```

Definition at line 1724 of file matrix_multi_imp.h.

Referenced by glucat::pade_log().

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 1810 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 1812 of file matrix_multi_imp.h.

6.25.3 Member Data Documentation

6.25.3.1 denom

```
const array pade::pade_log_denom< dd_real >::denom  [static]
```

Definition at line 1813 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 1748 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 1750 of file matrix_multi_imp.h.

6.26.3 Member Data Documentation

6.26.3.1 denom

```
const array pade::pade_log_denom< float >::denom  [static]
```

Definition at line 1751 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 1775 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 1777 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 1778 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 1857 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 1859 of file matrix_multi_imp.h.

6.28.3 Member Data Documentation

6.28.3.1 denom

```
const array pade::pade_log_denom< qd_real >::denom [static]
```

Definition at line 1860 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 1704 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 1706 of file matrix_multi_imp.h.

6.29.3 Member Data Documentation

6.29.3.1 numer

```
template<typename Scalar_T>
const array pade::pade_log_numer< Scalar_T >::numer  [static]
```

Definition at line 1707 of file matrix_multi_imp.h.

Referenced by glucat::pade_log().

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 1790 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 1792 of file matrix_multi_imp.h.

6.30.3 Member Data Documentation

6.30.3.1 numer

```
const array pade::pade_log_numer< dd_real >::numer [static]
```

Definition at line 1793 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 1736 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 1738 of file matrix_multi_imp.h.

6.31.3 Member Data Documentation

6.31.3.1 numer

```
const array pade::pade_log_numer< float >::numer  [static]
```

Definition at line 1739 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 1761 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 1763 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 1764 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 1831 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 1833 of file matrix_multi_imp.h.

6.33.3 Member Data Documentation

6.33.3.1 numer

```
const array pade::pade_log_numer< qd_real >::numer [static]
```

Definition at line 1834 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 1393 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 1395 of file matrix_multi_imp.h.

6.34.3 Member Data Documentation

6.34.3.1 denom

```
template<typename Scalar_T>
const array pade::pade_sqrt_denom< Scalar_T >::denom  [static]
```

Definition at line 1396 of file matrix_multi_imp.h.

Referenced by glucat::matrix_sqrt().

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 1483 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 1485 of file matrix_multi_imp.h.

6.35.3 Member Data Documentation

6.35.3.1 denom

```
const array pade::pade_sqrt_denom< dd_real >::denom  [static]
```

Definition at line 1486 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 1420 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 1422 of file matrix_multi_imp.h.

6.36.3 Member Data Documentation

6.36.3.1 denom

```
const array pade::pade_sqrt_denom< float >::denom  [static]
```

Definition at line 1423 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 1447 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 1449 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 1450 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 1530 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 1532 of file matrix_multi_imp.h.

6.38.3 Member Data Documentation

6.38.3.1 denom

```
const array pade::pade_sqrt_denom< qd_real >::denom  [static]
```

Definition at line 1533 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 1376 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 1378 of file matrix_multi_imp.h.

6.39.3 Member Data Documentation

6.39.3.1 numer

```
template<typename Scalar_T>
const array pade::pade_sqrt_numer< Scalar_T >::numer  [static]
```

Definition at line 1379 of file matrix_multi_imp.h.

Referenced by glucat::matrix_sqrt().

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 1463 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 1465 of file matrix_multi_imp.h.

6.40.3 Member Data Documentation

6.40.3.1 numer

```
const array pade::pade_sqrt_numer< dd_real >::numer [static]
```

Definition at line 1466 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 1408 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 1410 of file matrix_multi_imp.h.

6.41.3 Member Data Documentation

6.41.3.1 numer

```
const array pade::pade_sqrt_numer< float >::numer [static]
```

Definition at line 1411 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 1433 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 1435 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 1436 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 1504 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 1506 of file matrix_multi_imp.h.

6.43.3 Member Data Documentation

6.43.3.1 numer

```
const array pade::pade_sqrt_numer< qd_real >::numer  [static]
```

Definition at line 1507 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
- using type = 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 70 of file promotion.h.

6.44.2 Member Typedef Documentation

6.44.2.1 type [1/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = double
```

Definition at line 72 of file promotion.h.

6.44.2.2 type [2/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = long double
```

Definition at line 86 of file promotion.h.

6.44.2.3 type [3/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = double
```

Definition at line 145 of file scalar.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]
```

References random_generator().

Referenced by generator(), operator=(), and random_generator().

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 normal_dist, seed, uint_gen, and uniform_dist.

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>
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.

References random_generator().

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 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]
```

References random_generator().

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 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.

References friend_for_private_destructor.

Referenced by friend_for_private_destructor.

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 normal(), and random_generator().

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 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.

Referenced by random_generator().

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 random_generator(), and uniform().

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

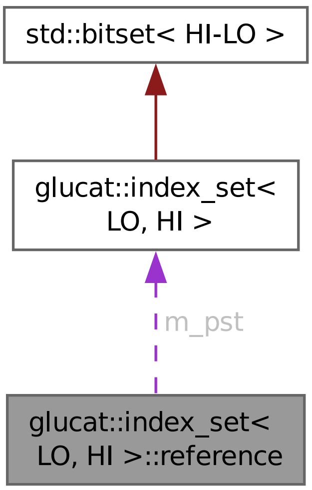
- glucat/random.h

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.

Referenced by flip(), operator=(), operator=(), operator==(), and ∼reference().

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.

References m_idx, and m_pst.

6.46.2.3 ∼reference()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::∼reference () [default]
```

References flip(), and reference().

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.

References m_idx, m_pst, and reference().

Referenced by ∼reference().

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.

References m_idx, and m_pst.

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.

References m_idx, m_pst, and reference().

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.

References m_idx, m_pst, and reference().

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.

References m_idx, m_pst, and reference().

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.

References m_idx, and m_pst.

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.

References index_set.

Referenced by index_set.

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.

Referenced by flip(), operator bool(), operator=(), operator=(), operator==(), operator∼(), and reference().

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.

Referenced by flip(), operator bool(), operator=(), operator=(), operator==(), operator∼(), and reference().

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 1108 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 1111 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 1113 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 1112 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 1115 of file framed_multi_imp.h.

References sorted_begin, and sorted_end.

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 1122 of file framed_multi_imp.h.

Referenced by 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 1123 of file framed_multi_imp.h.

Referenced by 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 1127 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 1130 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 1132 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 1131 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 1134 of file framed_multi_imp.h.

References sorted_begin, and sorted_end.

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 1138 of file framed_multi_imp.h.

Referenced by sorted_range().

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 1139 of file framed_multi_imp.h.

Referenced by sorted_range().

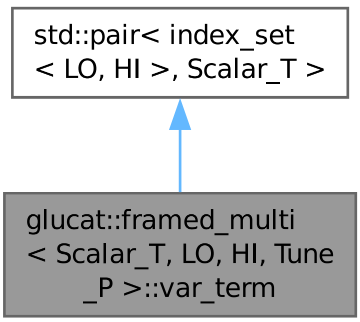
The documentation for this class was generated from the following file:

- glucat/framed_multi_imp.h

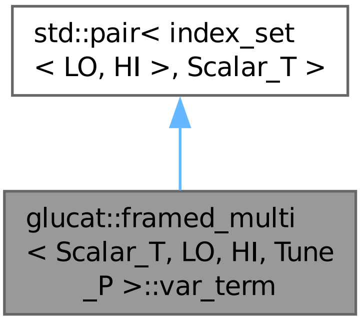
6.49 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term Class Reference

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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↩
_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT↵
_HI, typename Tune_P = tuning<>>
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

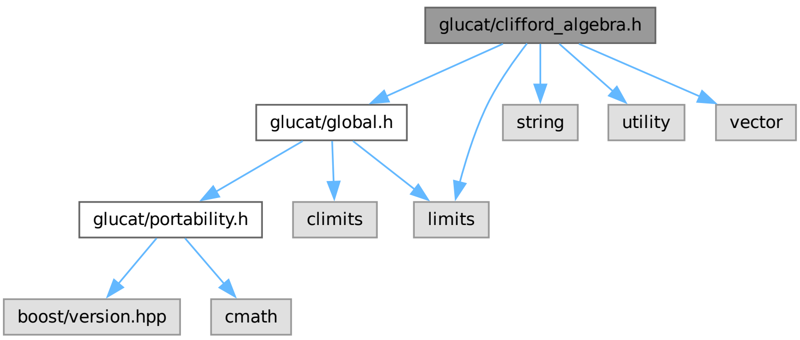
Chapter 7

File Documentation

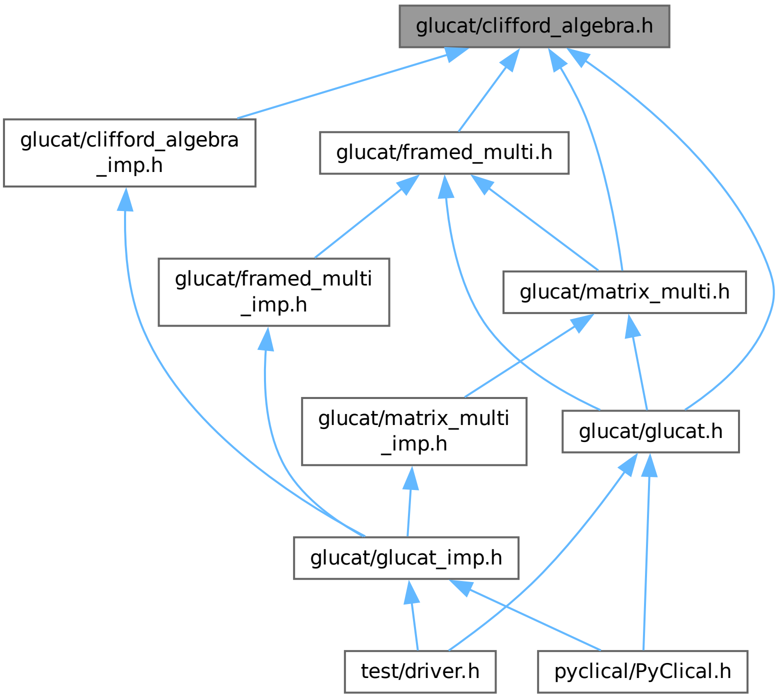
7.1 glucat/clifford_algebra.h File Reference

```
#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.

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

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
>
```

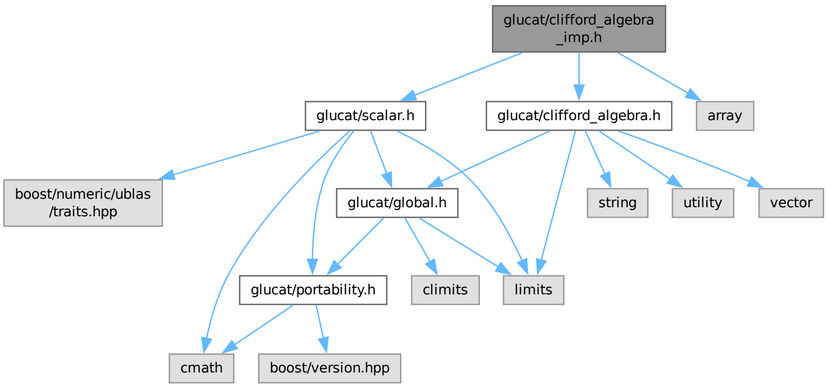
```
auto
asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
     const Multivector<Scalar_T,LO,HI,Tune_P>& i,
     const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

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
```

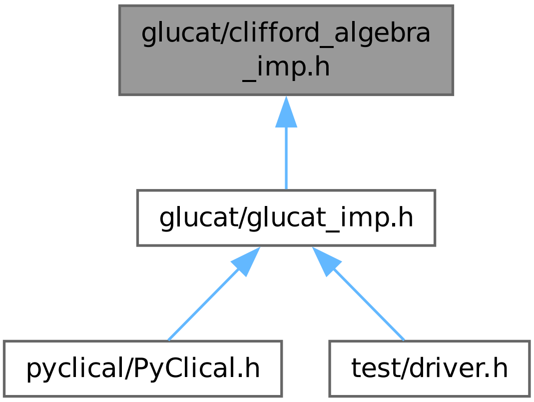
7.3 glucat/clifford_algebra_imp.h File Reference

```
#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.

Variables

- 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()

    Default for truncation.

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 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
```

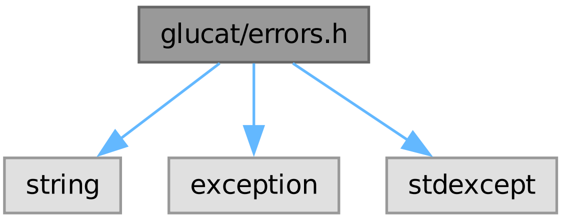
```
auto
asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{
  using traits_t = numeric_traits<Scalar_T>;
  if (val.isnan())
    return traits_t::NaN();

  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
```

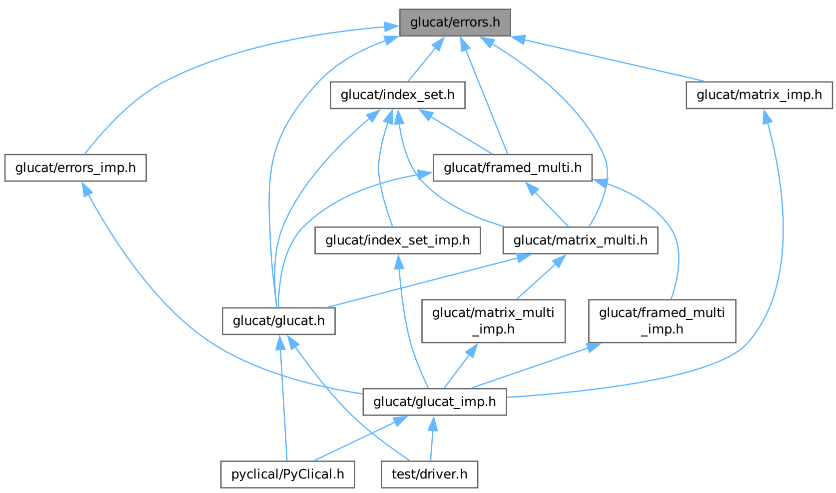
7.5 glucat/errors.h File Reference

```
#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.

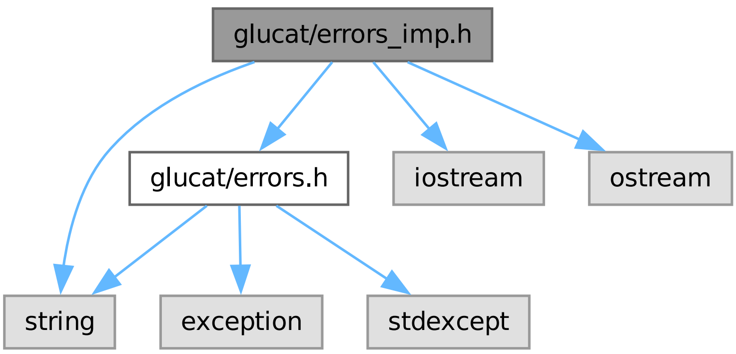
```
#ifndef _GLUCAT_ERRORS_H
#define _GLUCAT_ERRORS_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    errors.h : Declare error classes and functions
                             -------------------
    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
```

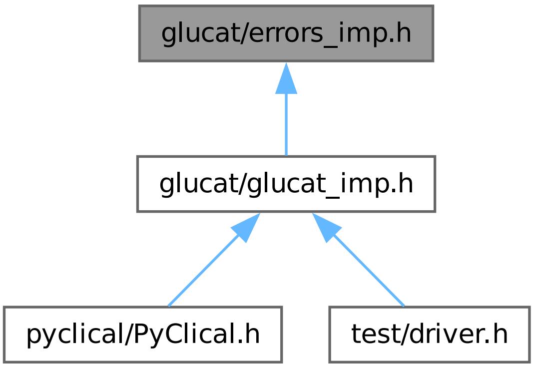
7.7 glucat/errors_imp.h File Reference

```
#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.

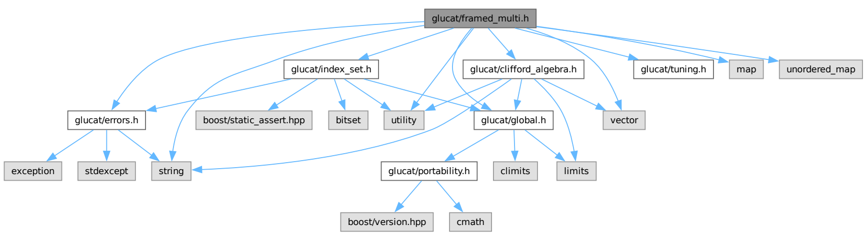
```
#ifndef _GLUCAT_ERRORS_IMP_H
#define _GLUCAT_ERRORS_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    errors_imp.h : Define error functions
                             -------------------
    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
```

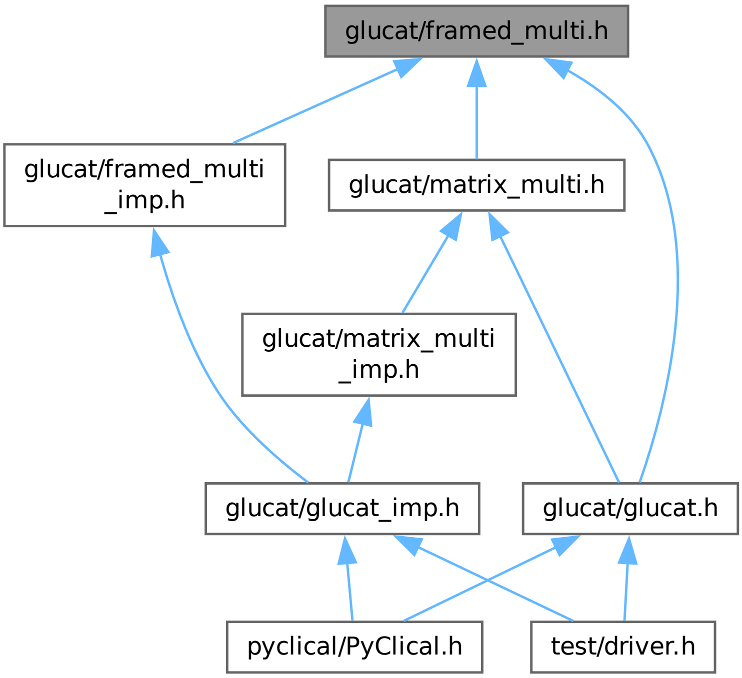
7.9 glucat/framed_multi.h File Reference

```
#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;
```

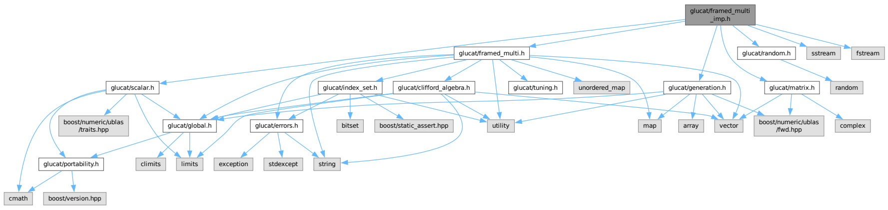
```
    ~framed_multi() override = default;
    framed_multi();

  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
```

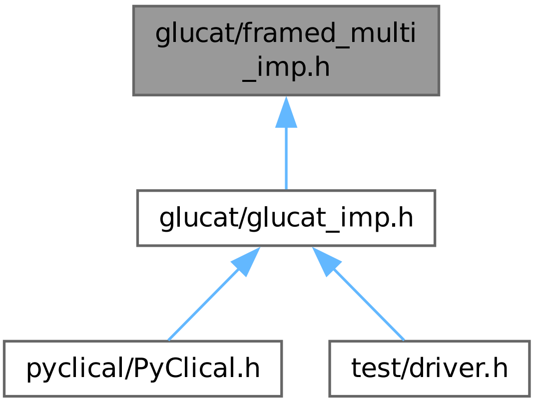
7.11 glucat/framed_multi_imp.h File Reference

```
#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.

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

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.

```
#ifndef _GLUCAT_FRAMED_MULTI_IMP_H
#define _GLUCAT_FRAMED_MULTI_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    framed_multi_imp.h : Implement the coordinate map representation of a
    Clifford algebra element
                             -------------------
    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/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;

  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
{
  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 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;
      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>
  {
```

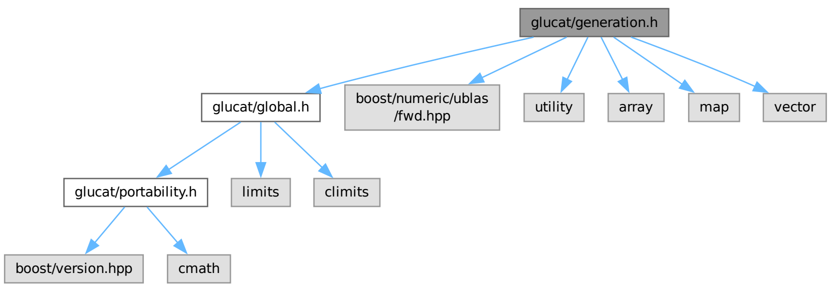
```
    using term_t = std::pair<const index_set<LO,HI>, Scalar_T>;
    return term_t(lhs.first ^ rhs.first, crd_of_mult(lhs, rhs));
  }

  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
```

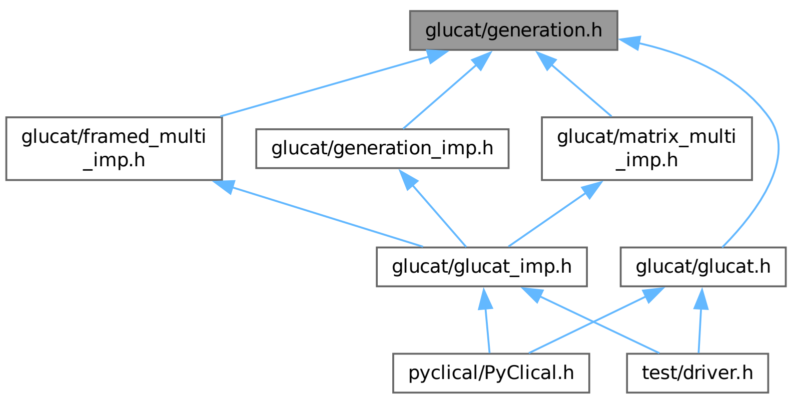
7.13 glucat/generation.h File Reference

```
#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.

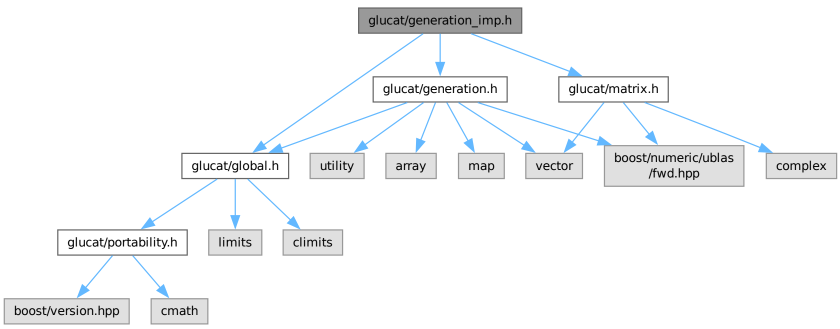
```
#ifndef _GLUCAT_GENERATION_H
#define _GLUCAT_GENERATION_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    generation.h : Declare functions for generation of the matrix representation
                             -------------------
    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
```

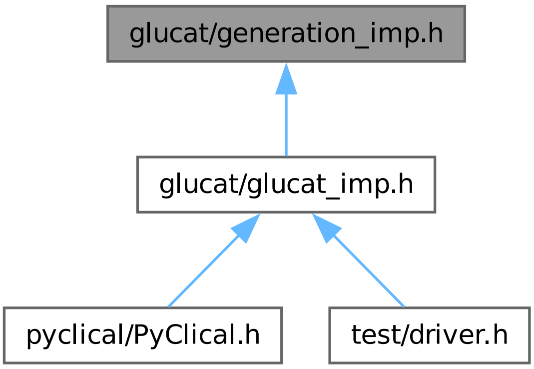
7.15 glucat/generation_imp.h File Reference

```
#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>::
```

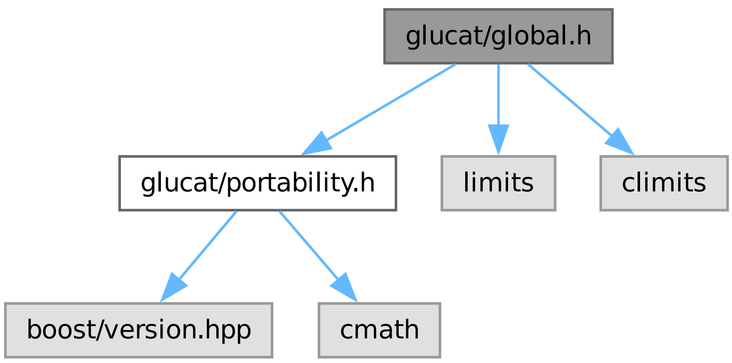
```
gen_from_pm1_qm1(const std::vector<Matrix_T>& old, const signature_t sig)
{
  const auto new_size = old.size() + 2;
  using size_t = decltype(new_size);
  using result_t = std::vector<Matrix_T>;
  auto result = result_t(new_size);

  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
```

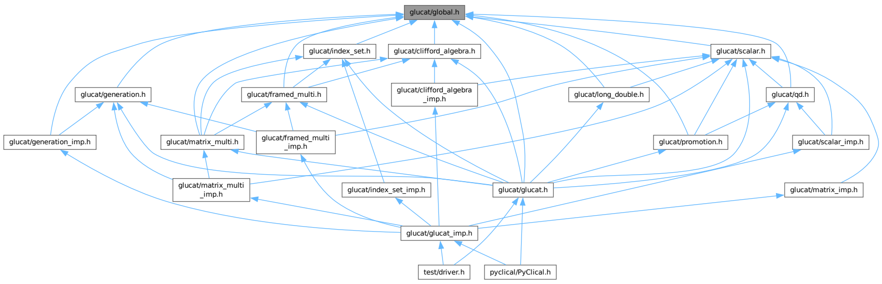
7.17 glucat/global.h File Reference

```
#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, 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::_GLUCAT_CTAssert (_GLUCAT_BITS_PER_ULONG==BITS_PER_SET_VALUE, BitsPerULongDoesNotMatchSetValueT) 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
{
```

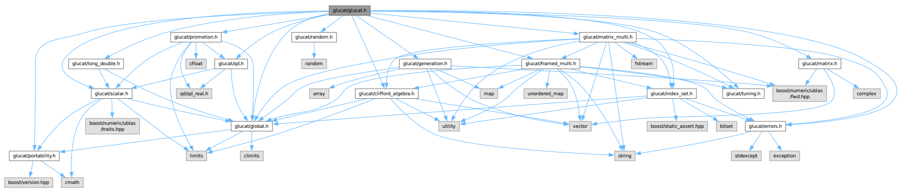
```
// References:
// [AA]: A. Alexandrescu, "Modern C++ Design", Addison-Wesley, 2001.

// 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

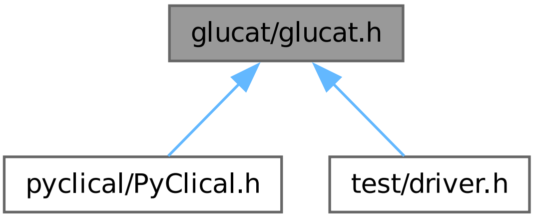
```
#include "glucat/portability.h"
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/scalar.h"
#include "glucat/long_double.h"
```

```
#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

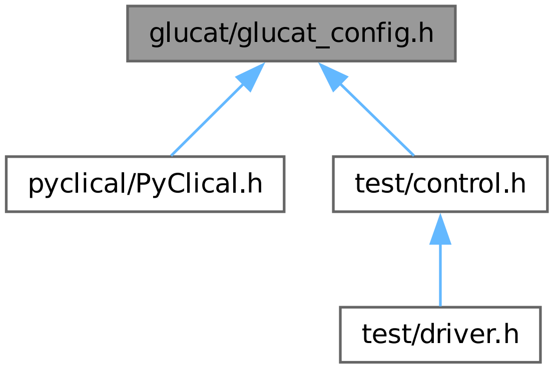
Go to the documentation of this file.

```
#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 0.13.0"
- #define GLUCAT_PACKAGE_TARNAME "glucat"
- #define GLUCAT_PACKAGE_URL ""
- #define GLUCAT_PACKAGE_VERSION "0.13.0"
- #define GLUCAT_STDC_HEADERS 1
- #define GLUCAT_VERSION "0.13.0"

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 0.13.0"
```

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 "0.13.0"
```

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 "0.13.0"
```

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.

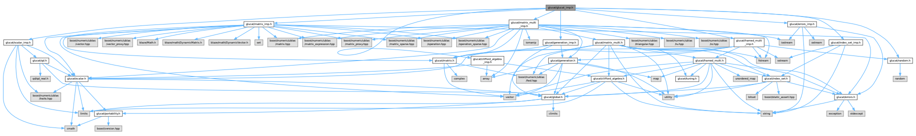
```
00001 #ifndef _GLUCAT_GLUCAT_CONFIG_H
00002 #define _GLUCAT_GLUCAT_CONFIG_H 1
00003
00004 /* glucat/glucat_config.h. Generated automatically at end of configure. */
00005 /* config.h.  Generated from config.h.in by configure.  */
00006 /* config.h.in.  Generated from configure.ac by autoheader.  */
00007
00008 /* Define to dummy 'main' function (if any) required to link to the Fortran
00009    libraries. */
00010 /* #undef F77_DUMMY_MAIN */
00011
00012 /* Define if F77 and FC dummy 'main' functions are identical. */
00013 /* #undef FC_DUMMY_MAIN_EQ_F77 */
00014
00015 /* Define if you have a BLAS library. */
00016 /* #undef HAVE_BLAS */
00017
00018 /* define if the compiler supports basic C++11 syntax */
00019 #ifndef GLUCAT_HAVE_CXX11
00020 #define GLUCAT_HAVE_CXX11 1
00021 #endif
00022
00023 /* define if the compiler supports basic C++14 syntax */
00024 /* #undef HAVE_CXX14 */
00025
00026 /* Define to 1 if you have the <inttypes.h> header file. */
00027 #ifndef GLUCAT_HAVE_INTTYPES_H
00028 #define GLUCAT_HAVE_INTTYPES_H 1
00029 #endif
00030
00031 /* Define if you have LAPACK library. */
00032 /* #undef HAVE_LAPACK */
00033
00034 /* Define to 1 if you have the 'imf' library (-limf). */
00035 /* #undef HAVE_LIBIMF */
00036
00037 /* Define to 1 if you have the <stdint.h> header file. */
00038 #ifndef GLUCAT_HAVE_STDINT_H
00039 #define GLUCAT_HAVE_STDINT_H 1
00040 #endif
00041
00042 /* Define to 1 if you have the <stdio.h> header file. */
00043 #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 0.13.0"
#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 "0.13.0"
#endif

/* Define to 1 if all of the C89 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 "0.13.0"
#endif

/* once: _GLUCAT_GLUCAT_CONFIG_H */
#endif
```

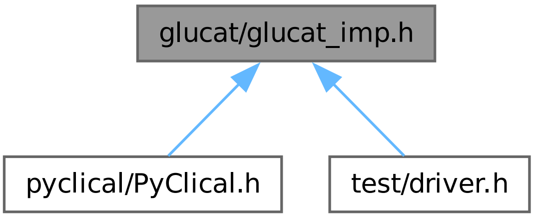
7.23 glucat/glucat_imp.h File Reference

```
#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.

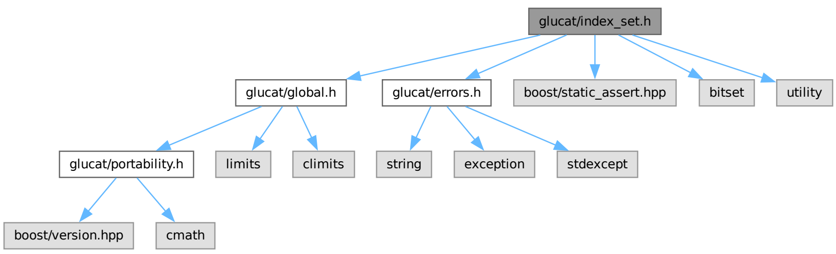
```
#ifndef _GLUCAT_GLUCAT_IMP_H
#define _GLUCAT_GLUCAT_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    glucat_imp.h : Organize GluCat template definitions which cannot be compiled separately
                             -------------------
    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
```

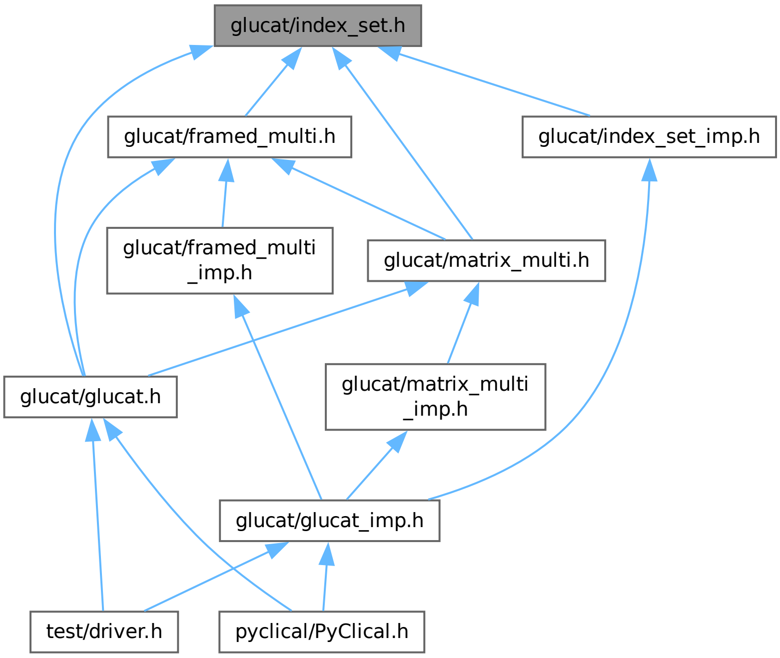
7.25 glucat/index_set.h File Reference

```
#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>
```

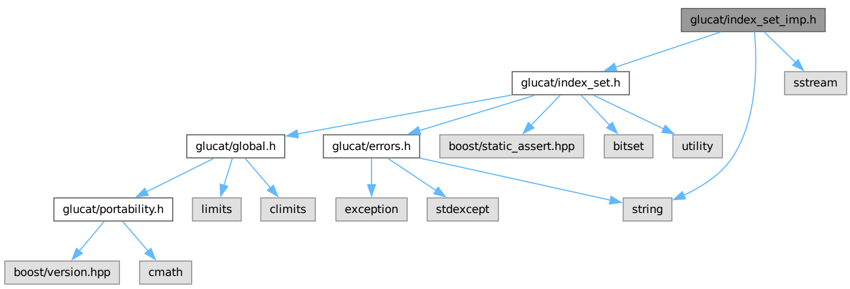
```
class index_set; // forward

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
```

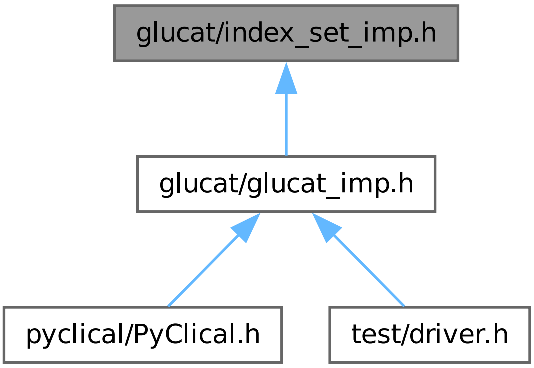
7.27 glucat/index_set_imp.h File Reference

```
#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);
```

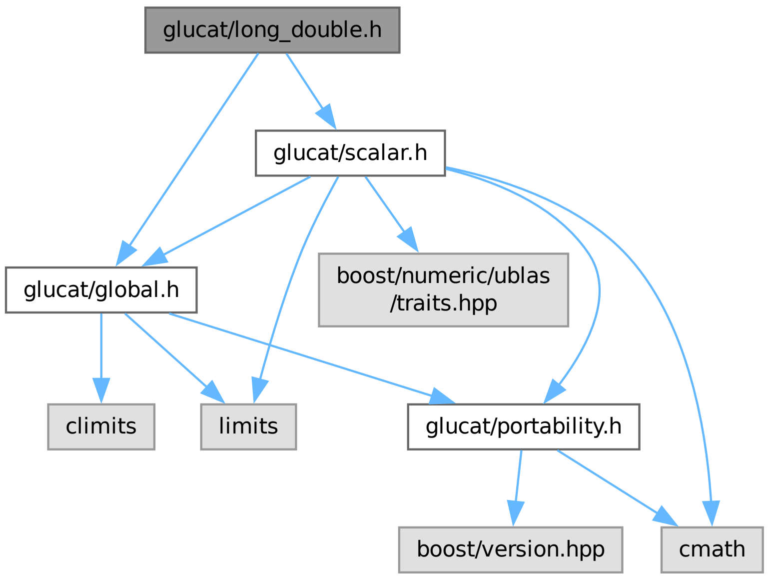
```
      // Set k to be the inverse Gray code of *this & h.
      // This sets the low bit of k to be parity(*this & h).
      const auto k = inverse_gray(uthis & h);
      // Set q to be the inverse Gray code of the positive part of *this & rhs.
      const auto q = inverse_gray((uthis & urhs) » -LO);
      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
```

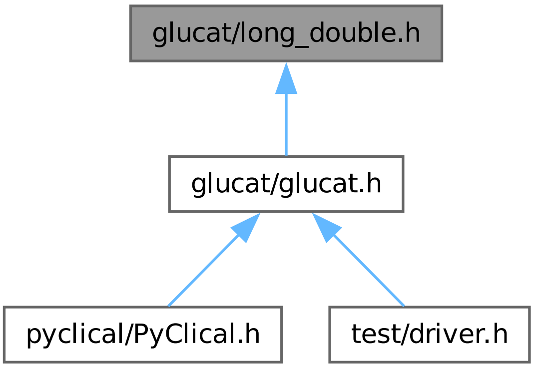
7.29 glucat/long_double.h File Reference

```
#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 = 3.1415926535897932384626433832795029L
- static const long double glucat::l_ln2 = 0.6931471805599453094172321214581766L

7.30 long_double.h

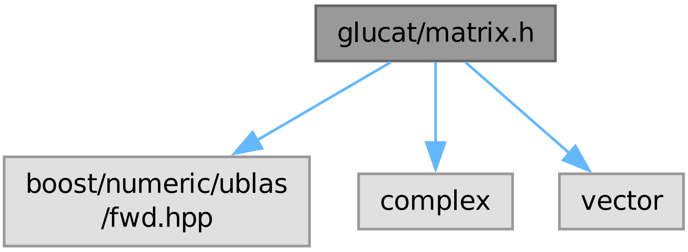
Go to the documentation of this file.

```
#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
```

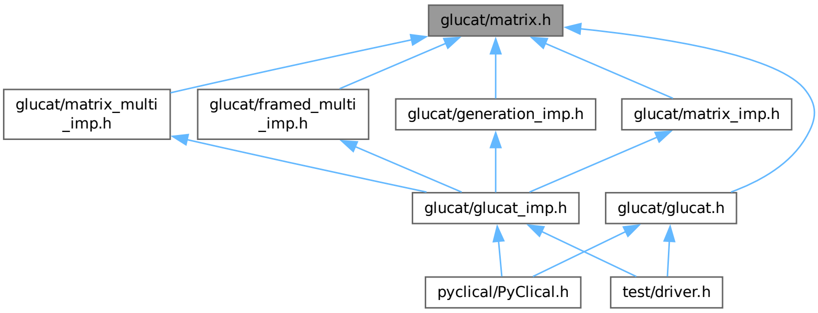
7.31 glucat/matrix.h File Reference

```
#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.

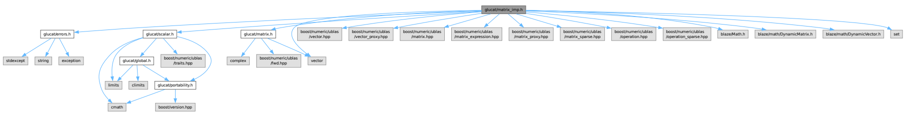
```
#ifndef _GLUCAT_MATRIX_H
#define _GLUCAT_MATRIX_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    matrix.h : Declare common matrix functions
                             -------------------
    begin                : 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
```

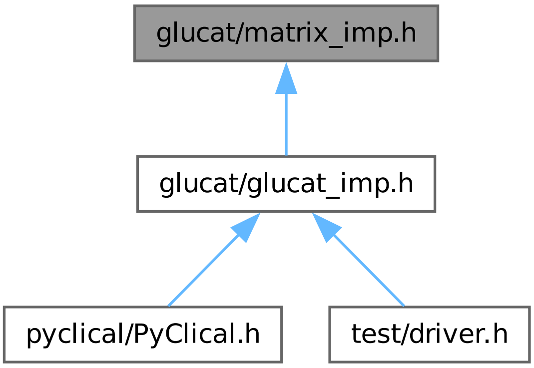
7.33 glucat/matrix_imp.h File Reference

```
#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 <blaze/Math.h>
#include <blaze/math/DynamicMatrix.h>
#include <blaze/math/DynamicVector.h>
#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_blaze (const Matrix_T &val) -> blaze::DynamicMatrix< double, blaze::rowMajor >

  Convert matrix to Blaze 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_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_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;
```

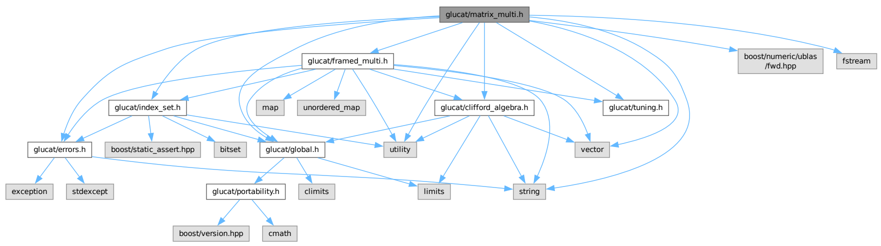
```
  }

#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_BLAZE)
    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;

    auto lambda = eigenvalues(val);

    std::set<double> arg_set;

    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
```

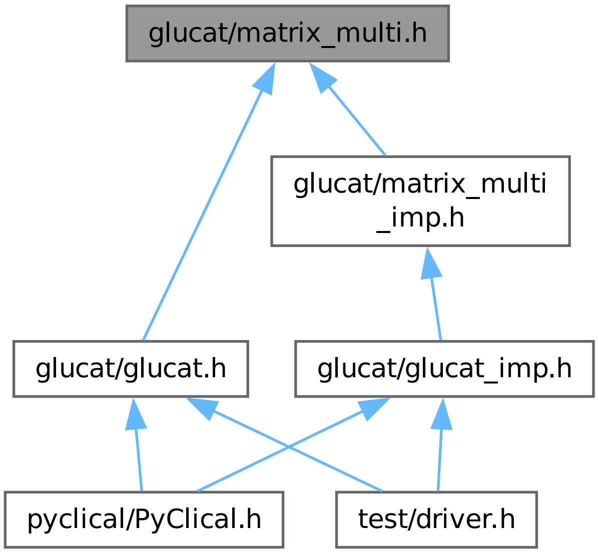
7.35 glucat/matrix_multi.h File Reference

```
#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());
```

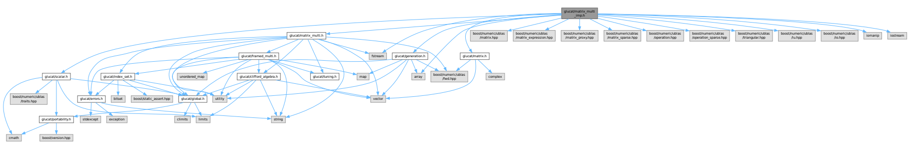
```
    matrix_multi(const int scr, const index_set_t frm = index_set_t());
    matrix_multi(const vector_t& vec,
                 const index_set_t frm, const bool prechecked = false);
    matrix_multi(const std::string& str);
    matrix_multi(const std::string& str,
                 const index_set_t 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
```

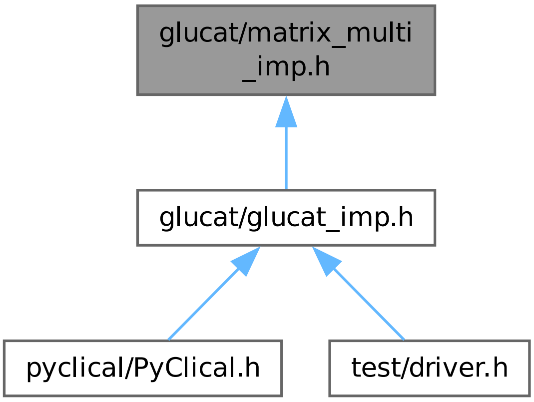
7.37 glucat/matrix_multi_imp.h File Reference

```
#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.

Variables

- template<typename Scalar_T>
  const pade_sqrt_numer< Scalar_T >::array pade::pade_sqrt_numer< Scalar_T >::numer

- template<typename Scalar_T>
  const pade_sqrt_denom< Scalar_T >::array pade::pade_sqrt_denom< Scalar_T >::denom
- const pade_sqrt_numer< float >::array pade::pade_sqrt_numer< float >::numer
- const pade_sqrt_denom< float >::array pade::pade_sqrt_denom< float >::denom
- const pade_sqrt_numer< longdouble >::array pade::pade_sqrt_numer< longdouble >::numer
- const pade_sqrt_denom< longdouble >::array pade::pade_sqrt_denom< longdouble >::denom
- const pade_sqrt_numer< dd_real >::array pade::pade_sqrt_numer< dd_real >::numer
- const pade_sqrt_denom< dd_real >::array pade::pade_sqrt_denom< dd_real >::denom
- const pade_sqrt_numer< qd_real >::array pade::pade_sqrt_numer< qd_real >::numer
- const pade_sqrt_denom< qd_real >::array pade::pade_sqrt_denom< qd_real >::denom
- template<typename Scalar_T>
  const pade_log_numer< Scalar_T >::array pade::pade_log_numer< Scalar_T >::numer
- template<typename Scalar_T>
  const pade_log_denom< Scalar_T >::array pade::pade_log_denom< Scalar_T >::denom
- const pade_log_numer< float >::array pade::pade_log_numer< float >::numer
- const pade_log_denom< float >::array pade::pade_log_denom< float >::denom
- const pade_log_numer< longdouble >::array pade::pade_log_numer< longdouble >::numer
- const pade_log_denom< longdouble >::array pade::pade_log_denom< longdouble >::denom
- const pade_log_numer< dd_real >::array pade::pade_log_numer< dd_real >::numer
- const pade_log_denom< dd_real >::array pade::pade_log_denom< dd_real >::denom
- const pade_log_numer< qd_real >::array pade::pade_log_numer< qd_real >::numer
- const pade_log_denom< qd_real >::array pade::pade_log_denom< qd_real >::denom

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();

  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();

  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>;

  // 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 >
```

```cpp
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]
  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]
    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;
    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("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")
  };
#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)
  {
    // 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("392923948371995600"),
    qd_real("10537522306718")/qd_real("319250708052246425"),
qd_real("236648286272519")/qd_real("1442491974750354255500"),
   qd_real("260715545088119")/qd_real("4375558990076074573500"),
qd_real("289596255666839")/qd_real("192874640282553367199880"),
     qd_real("8802625510547")/qd_real("361639950529787563499775"),
qd_real("373831661521439")/qd_real("16592040930306653413336967700"),
   qd_real("446033437968239")/qd_real("464577146048586295574350956000"),
qd_real("53676090078349")/qd_real("473868688969558021485837975120000")
      };
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,
```

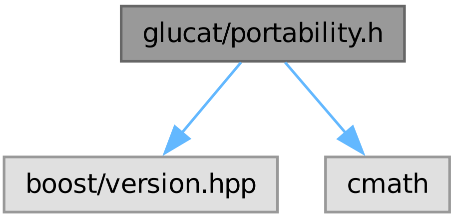
```
              const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
  {
    // Scaled incomplete square root cascade and scaled Pade' approximation of log
    // Reference: [CHKL]

    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)
    {
      // 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
```

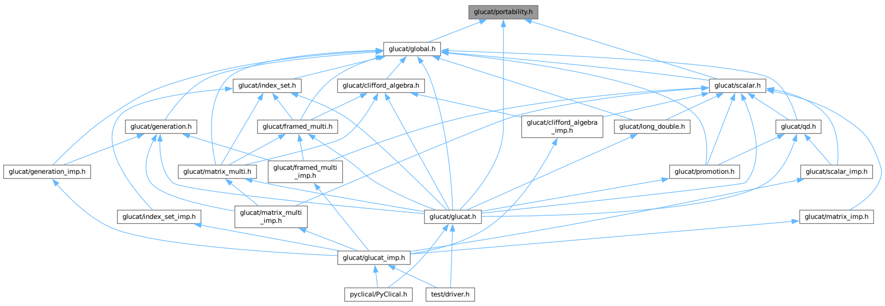
7.39 glucat/portability.h File Reference

```
#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

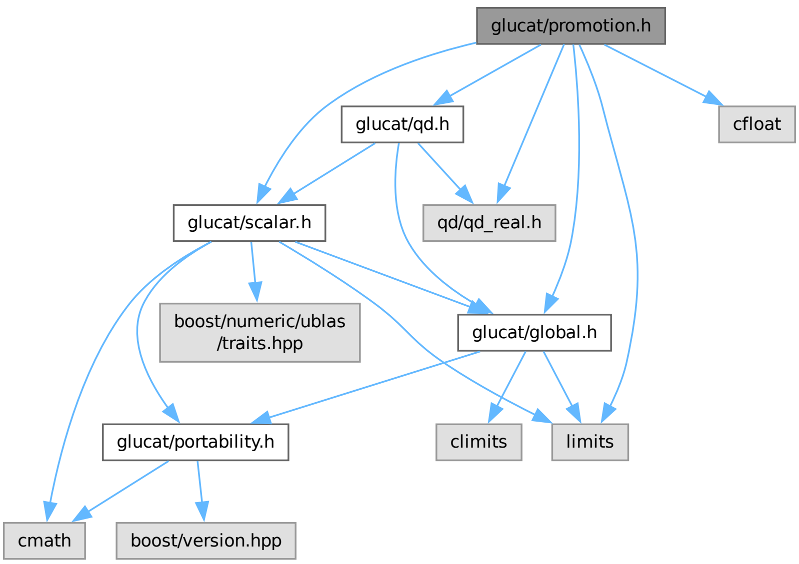
Go to the documentation of this file.

```
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
```

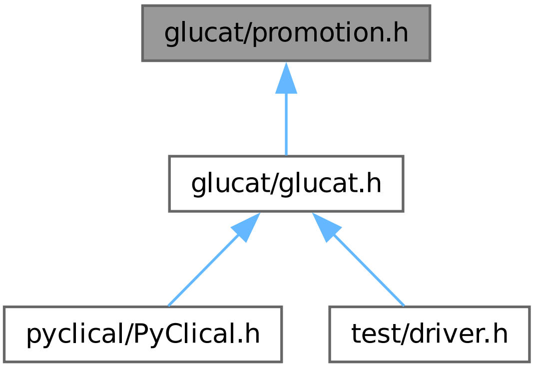
```
#if __cplusplus > 199711L
# define _GLUCAT_ISNAN(x) (std::isnan(x))
# define _GLUCAT_ISINF(x) (std::isinf(x))
#else
# define _GLUCAT_ISNAN(x) (x != x)
# define _GLUCAT_ISINF(x) (!_GLUCAT_ISNAN(x) && _GLUCAT_ISNAN(x-x))
#endif

// Workaround for abs and sqrt
#if BOOST_VERSION >= 103400
# define UBLAS_ABS  type_abs
# define UBLAS_SQRT type_sqrt
#else
# define UBLAS_ABS  abs
# define UBLAS_SQRT sqrt
#endif

// Use with Cygwin gcc to obtain __WORDSIZE
#if defined(HAVE_BITS_WORDSIZE_H)
# include <bits/wordsize.h>
#endif

#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
```

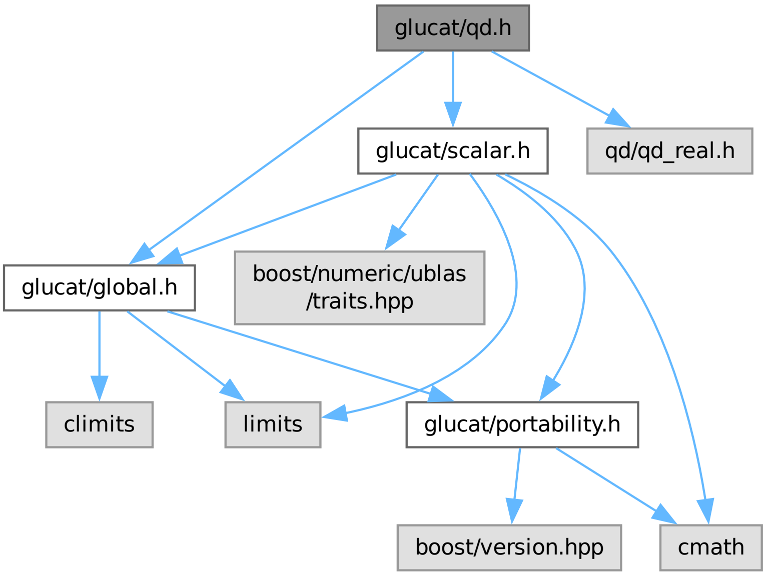
```
 licensed under the LGPL with permission of the author. See Arvind Raja,
 "Object-oriented implementations of Clifford algebras in C++: a prototype",
 in Ablamowicz, Lounesto and Parra (eds.)
 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
 ***************************************************************************
 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
```

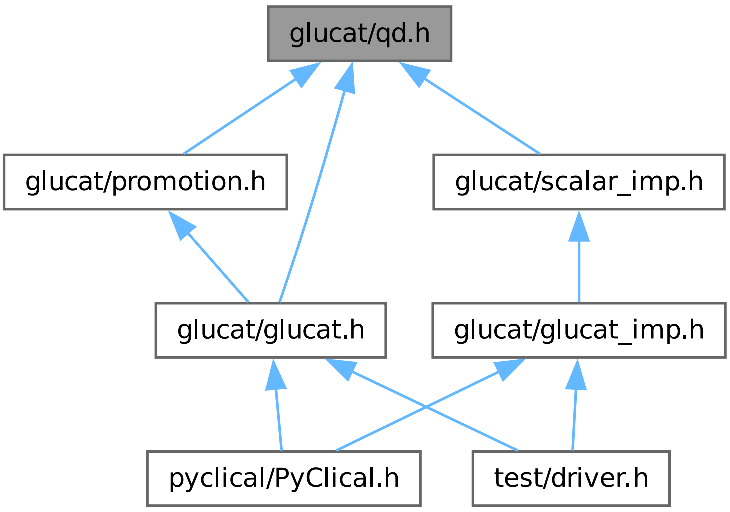
7.43 glucat/qd.h File Reference

```
#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)

```

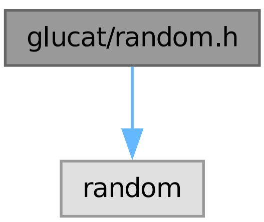
```
_GLUCAT_QD_F(dd_real, asin)


_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
```

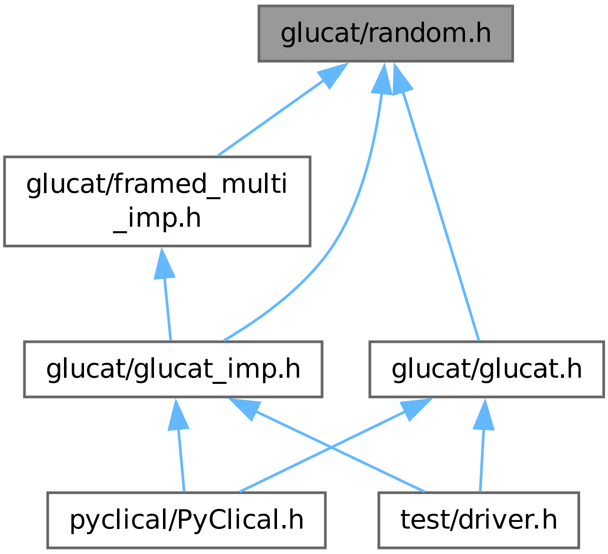
7.45 glucat/random.h File Reference

`#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.

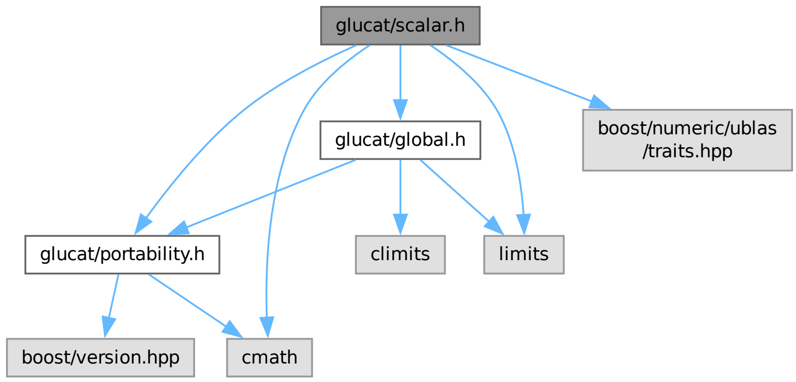
```
#ifndef _GLUCAT_RANDOM_H
#define _GLUCAT_RANDOM_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    random.h : Random number generator with single instance per Scalar_T
                             -------------------
    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
```

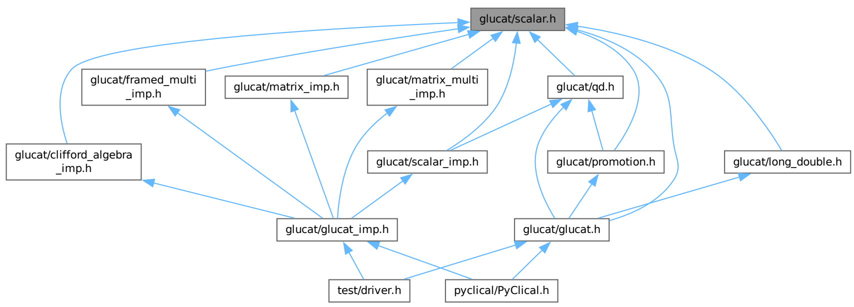
7.47 glucat/scalar.h File Reference

```
#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
```

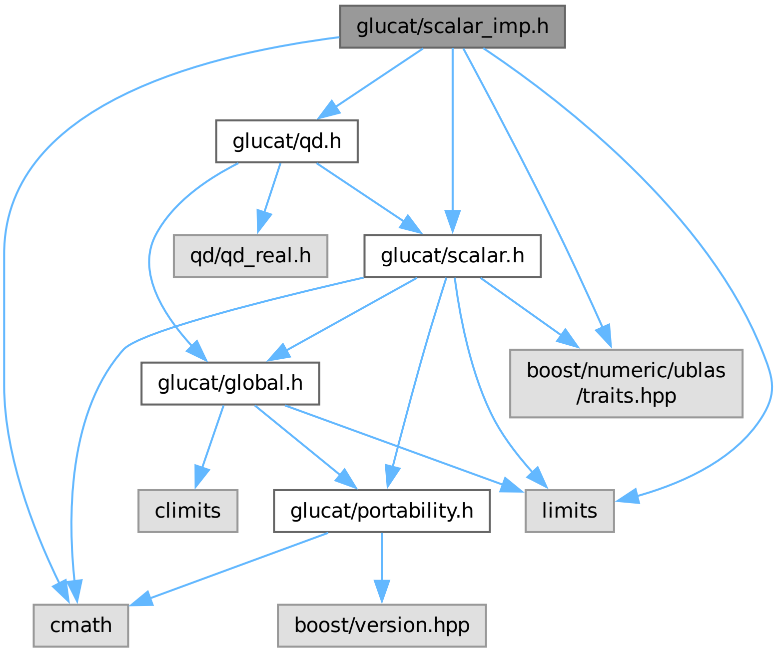
```
{ return Scalar_T(3.14159265358979323); }

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
```

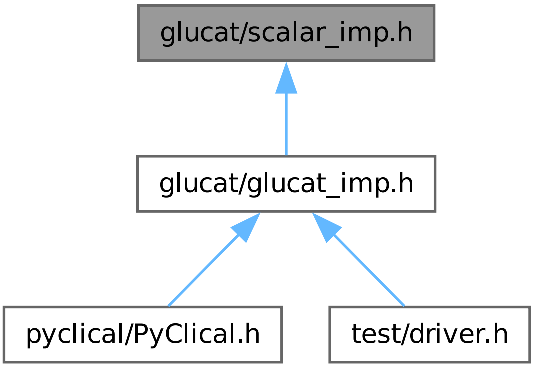
7.49 glucat/scalar_imp.h File Reference

```
#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.

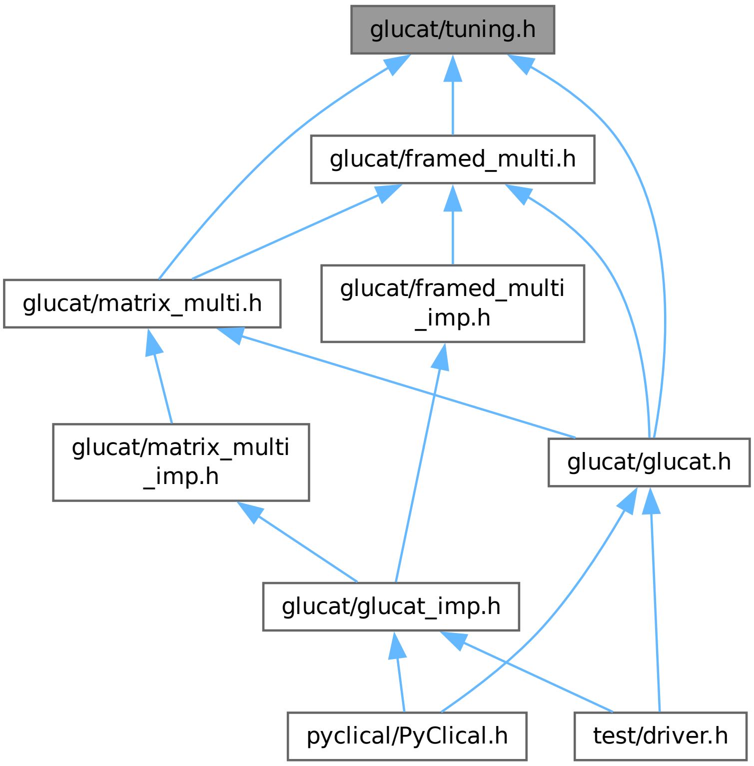
```
#ifndef _GLUCAT_SCALAR_IMP_H
#define _GLUCAT_SCALAR_IMP_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    scalar_imp.h : Define functions for scalar_t
                         -------------------
    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

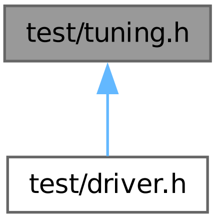
Go to the documentation of this file.

```
#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()

```

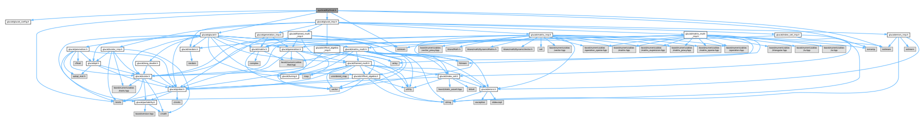
```
    cdef cppclass IndexSet:
        IndexSet ()
        IndexSet (IndexSet Ist) except+
        IndexSet (int idx) except+
        IndexSet (char* str) except+
        inline bint operator==(IndexSet Rhs)
        inline bint operator!=(IndexSet Rhs)
        inline bint operator<(IndexSet Rhs)
        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.

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

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.instance[0] = other
        return self

    cdef inline IndexSet unwrap(index_set self):
        """
        Return the wrapped C++ IndexSet instance.
        """
        return self.instance[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.instance = new IndexSet((<index_set>other).unwrap())
        elif isinstance(other, numbers.Integral):
            self.instance = new IndexSet(<int>other)
        elif isinstance(other, (set, frozenset)):
            try:
                self.instance = 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.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)) + ".")

    def __dealloc__(self):
        """
        Clean up by deallocating the instance of C++ class IndexSet.
        """
        del self.instance

    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.instance.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.instance.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.instance.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.instance.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.instance.count()

    def count_neg(self):
        """
        Number of negative indices included in set.

        >>> index_set({-1,1,2}).count_neg()
        1
        """
        return self.instance.count_neg()

    def count_pos(self):
        """
        Number of positive indices included in set.

        >>> index_set({-1,1,2}).count_pos()
        2
        """
        return self.instance.count_pos()

    def min(self):
        """
        Minimum member.

        >>> index_set({-1,1,2}).min()
        -1
        """
        return self.instance.min()

    def max(self):
        """
        Maximum member.

        >>> index_set({-1,1,2}).max()
        2
        """
        return self.instance.max()

    def hash_fn(self):
        """
        Hash function.
        """
        return self.instance.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.instance.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.instance.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 __radd__(rhs, lhs):
        """
        Geometric sum.

        >>> print(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 __rsub__(rhs, lhs):
        """
        Geometric difference.

        >>> print(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 __rmul__(rhs, lhs):
        """
        Geometric product.

        >> print(2 * clifford("{2}"))
        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 __rmod__(rhs, lhs):
        """
        Contraction.

        >> print(2 % clifford("{2}"))
        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 __rand__(rhs, lhs):
        """
        Inner product.

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

```
        """
        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 __rxor__(rhs, lhs):
        """
        Outer product.

        »> print(2 ^ clifford("{2}"))
        2{2}
        """
        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 __rtruediv__(rhs, lhs):
        """
        Geometric quotient.

        »> print(2 / clifford("{2}"))
        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):
```

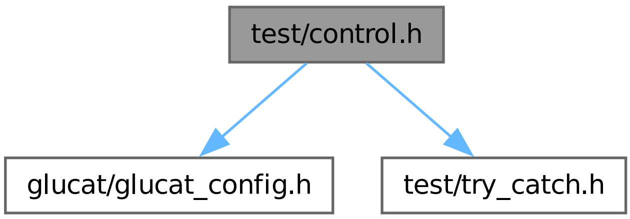
```
    """
    Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby definition.

    >>> 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].
```

```python

# Doctest interface.
def _test():
    import PyClical, doctest
    return doctest.testmod(PyClical)

if __name__ == "__main__":
    _test()
```

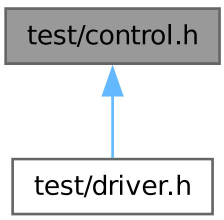
7.63 test/control.h File Reference

```
#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

7.64 control.h

Go to the documentation of this file.

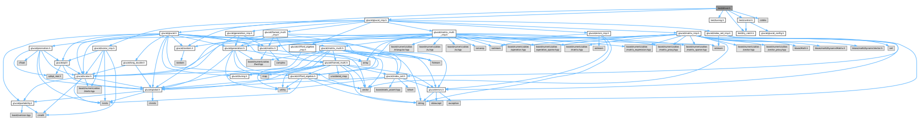
```
#ifndef _GLUCAT_CONTROL_H
#define _GLUCAT_CONTROL_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    control.h : Define and set parameters to control tests
                             -------------------
    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
```

7.65 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:



7.66 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
```

7.67 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.

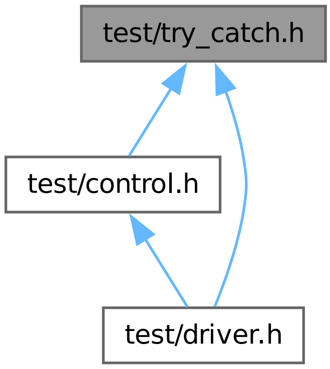
7.68 timing.h

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
```

7.69 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.

7.70 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
```

7.71 test/undefine.h File Reference

7.72 undefine.h

Go to the documentation of this file.

```
#ifndef GLUCAT_TEST_UNDEFINE_H
#define GLUCAT_TEST_UNDEFINE_H
/***************************************************************************
    GluCat : Generic library of universal Clifford algebra templates
    undefine.h : Undefine preprocessor macro names that control test tuning
                             -------------------
    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
 ***************************************************************************/

// Undefine tuning policy constants
#undef _GLUCAT_TEST_TUNING_SLOW
#undef _GLUCAT_TEST_TUNING_NAIVE
#undef _GLUCAT_TEST_TUNING_FAST
#undef _GLUCAT_TEST_TUNING_PROMOTED
#undef _GLUCAT_TEST_TUNING_DEMOTED

#endif // GLUCAT_TEST_UNDEFINE_H
```

Index