Rocket End Burning Simulator

Generated by Doxygen 1.9.2

Chapter 1

Todo List

Namespace CPGF::Basics
: include length and area calculation.

Class CPGF::Basics::BezierStroke2d
: Finish implementation.

Member CPGF::PGFConf::dash_pattern
: Explain this better.

Member CPGF::PGFConf::dash_phase
: Explain this.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Chemistry Namespace Reference

The objects of this library are used to store constants and to compute the speed of Chemical Reactions.

Classes

- class SolidGasReaction

  This object is used to store all properties of a solid reactant and a gas product. It is also used to compute the speed of the combustion front.

6.1.1 Detailed Description

The objects of this library are used to store constants and to compute the speed of Chemical Reactions.

6.2 CPGF Namespace Reference

Every component of this library is part of the CTikZ namespace.

Namespaces

- namespace AffineSpace

  This namespace contains all objects related to a mathematical affine space, i.e., vectors and points.
- namespace Basics

Classes

- class Color

  An object used to represent an rgb color. r, g and b must all be real numbers between 0 and 1.
- class DashPatterns

  This class provides a handful of useful predefined constants to express DashPatterns.
- class LineWidth

  This class provides a handful of useful predefined constants to express line width.
- class PGFConf

  This object is used to store all information needed to draw a path.
- class Scene2d
- class Text

Enumerations

- enum class DrawType { **DRAW** , **FILL** }

  Determines whether a path has to be drawn (only the contour is drawn and the interior remains white) or to be filled (the contour is not drawn but the interior is).
- enum class LineCap { **ROUND** , **RECT** , **BUTT** }

  This determines how the extremes of an open path are drawn.
- enum class LineJoin { **ROUND** , **BEVEL** , **MITER** }

  This determines how the junction between two non-parallel lines is drawn.
- enum class TextAlignment {
  CENTER , LEFT , RIGHT , TOP ,
  BOTTOM , BASE , TOP_LEFT , TOP_RIGHT ,
  BOTTOM_LEFT , BOTTOM_RIGHT , BASE_LEFT , BASE_RIGHT }

  Part of the following comments are copied from the TikZ/PGF manual.

6.2.1 Detailed Description

Every component of this library is part of the CTikZ namespace.

6.2.2 Enumeration Type Documentation

6.2.2.1 TextAlignment

`enum class CPGF::TextAlignment [strong]`

Part of the following comments are copied from the TikZ/PGF manual.

Enumerator

CENTER	The default TextAlignment.
LEFT	The key causes the text box to be placed such that its left border is on the origin.
RIGHT	The key causes the text box to be placed such that its right border is on the origin.
TOP	This key causes the text box to be placed such that its top is on the origin.

Enumerator

BOTTOM	This key causes the text box to be placed such that its bottom is on the origin.
BASE	This key causes the text box to be placed such that its baseline is on the origin.
TOP_LEFT	This key causes the text box to be placed such that its top left corner is on the origin.
TOP_RIGHT	This key causes the text box to be placed such that its top right corner is on the origin.
BOTTOM_LEFT	This key causes the text box to be placed such that its bottom left corner is on the origin.
BOTTOM_RIGHT	This key causes the text box to be placed such that its bottom right corner is on the origin.
BASE_LEFT	This key causes the text box to be placed such that its base left corner is on the origin.
BASE_RIGHT	This key causes the text box to be placed such that its base right corner is on the origin.

6.3 CPGF::AffineSpace Namespace Reference

This namespace contains all objects related to a mathematical affine space, i.e., vectors and points.

Classes

- class Point2d

  A point of a 2D affine space.
- class Vector2d

  An object that represents a 2D vector.

Functions

- Point2d operator+ (const Point2d &P, const Vector2d &v)
- Vector2d operator- (const Point2d &P, const Point2d &Q)
- bool operator== (const Point2d &P, const Point2d &Q)
- Vector2d operator+ (const Vector2d &v, const Vector2d &w)
- Vector2d operator- (const Vector2d &v, const Vector2d &w)
- Vector2d operator* (const Vector2d &v, const Vector2d &w)
- Vector2d operator/ (const Vector2d &v, const Vector2d &w)
- double operator| (const Vector2d &v, const Vector2d &w)

6.3.1 Detailed Description

This namespace contains all objects related to a mathematical affine space, i.e., vectors and points.

6.3.2 Function Documentation

6.3.2.1 operator*()

```
Vector2d CPGF::AffineSpace::operator* (
              const Vector2d & v,
              const Vector2d & w )
```

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

6.3.2.2 operator+() [1/2]

```
Point2d CPGF::AffineSpace::operator+ (
            const Point2d & P,
            const Vector2d & v )
```

Returns the position of the end of the vector v when its start is placed at the point P.

Parameters

P	a 2D point.
v	a 2D vector.

Returns

Point2D

6.3.2.3 operator+() [2/2]

```
Vector2d CPGF::AffineSpace::operator+ (
            const Vector2d & v,
            const Vector2d & w )
```

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

6.3.2.4 operator-() [1/2]

```
Vector2d CPGF::AffineSpace::operator- (
              const Point2d & P,
              const Point2d & Q )
```

Returns the vector whose start is at point P and whose end is at point Q.

Parameters

P	a 2D point.
Q	a 2D point.

Returns

Vector2d

6.3.2.5 operator-() [2/2]

```
Vector2d CPGF::AffineSpace::operator- (
              const Vector2d & v,
              const Vector2d & w )
```

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

6.3.2.6 operator/()

```
Vector2d CPGF::AffineSpace::operator/ (
              const Vector2d & v,
              const Vector2d & w )
```

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

6.3.2.7 operator==()

```
bool CPGF::AffineSpace::operator== (
             const Point2d & P,
             const Point2d & Q )
```

Parameters

P	
Q	

Returns

true

false

6.3.2.8 operator"|()

```
double CPGF::AffineSpace::operator| (
             const Vector2d & v,
             const Vector2d & w )
```

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

double

6.4 CPGF::Basics Namespace Reference

Classes

- class BezierStroke2d
- class Path2d
- class SimpleStroke2d
- class StraightStroke2d

6.4.1 Detailed Description

Todo include length and area calculation.

6.5 Math Namespace Reference

Namespace that includes mathematical objects such as vectors, matrixes, dual numbers and useful numerical methods such as numerical integrators, ode solvers and solvers for algebraic equations.

Namespaces

- namespace AlgebraicSolvers

  Contains solvers for algebraic equations.

Classes

- class DualNumber

  Represents a dual number. A mathematical object written like x = a + b, where satisfies ^{2}*=0.*
- class Matrix
- class Rational
- class Vector

  Generic Vector over the field K with any number of components.

Functions

- template<typename K >
  DualNumber< K > operator+ (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator+ (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > operator+ (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator- (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator- (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > operator- (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator* (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator* (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > operator* (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator/ (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > operator/ (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > operator/ (const K x, const DualNumber< K > &y)

- template< typename K >
  DualNumber< K > cos (const DualNumber< K > &x)

  Returns cos(x.a) - x.b∗sin(x.a)
- template< typename K >
  DualNumber< K > sin (const DualNumber< K > &x)

  Returns sin(x.a) + x.b∗cos(x.a)
- template< typename K >
  DualNumber< K > tan (const DualNumber< K > &x)

  Returns tan(x.a) + x.b/cos(x.a)/cos(x.a)∗
- template< typename K >
  DualNumber< K > acos (const DualNumber< K > &x)

  Returns acos(x.a) - 1/sqrt(1-x.a∗x.a)∗x.b∗
- template< typename K >
  DualNumber< K > asin (const DualNumber< K > &x)

  Returns asin(x.a) + 1/sqrt(1-x.a∗x.a)∗x.b∗
- template< typename K >
  DualNumber< K > atan (const DualNumber< K > &x)

  Returns atan(x.a) + 1/(1+x.a∗x.a)∗x.b∗
- template< typename K >
  DualNumber< K > cosh (const DualNumber< K > &x)

  Returns cosh(x.a) + sinh(x.a)∗x.b∗
- template< typename K >
  DualNumber< K > sinh (const DualNumber< K > &x)

  Returns sinh(x.a) + cosh(x.a)∗x.b∗
- template< typename K >
  DualNumber< K > tanh (const DualNumber< K > &x)

  Returns tanh(x.a) + 1/cosh(x.a)/cosh(x.a)∗x.b∗
- template< typename K >
  DualNumber< K > **acosh** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **asinh** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **atanh** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **exp** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **log** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **log10** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **exp2** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **expm1** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **log1p** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **log2** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **logb** (const DualNumber< K > &x)
- template< typename K >
  DualNumber< K > **pow** (const DualNumber< K > &x, const DualNumber< K > &y)
- template< typename K >
  DualNumber< K > **pow** (const DualNumber< K > &x, const K &y)
- template< typename K >
  DualNumber< K > **pow** (const K &x, const DualNumber< K > &y)

- template<typename K >
  DualNumber< K > **sqrt** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **cbrt** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **erf** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **erfc** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **tgamma** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **lgamma** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **fabs** (const DualNumber< K > &x)
- template Vector< std::complex< double > > **operator+** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **operator+** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template Vector< std::complex< double > > **operator+** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **operator-** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **operator-** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template Vector< std::complex< double > > **operator-** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **operator*** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **operator*** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template Vector< std::complex< double > > **operator*** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **operator/** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **operator/** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template Vector< std::complex< double > > **operator/** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template bool **operator==** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template bool **operator==** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template bool **operator==** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template bool **operator!=** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template bool **operator!=** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template bool **operator!=** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **operator&** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **operator&** (const Vector< std::complex< double > > &v, const std::complex< double > &alpha)
- template Vector< std::complex< double > > **operator&** (const std::complex< double > &alpha, const Vector< std::complex< double > > &v)

- template std::complex< double > **operator|** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template std::complex< double > **vector_product_2d** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template Vector< std::complex< double > > **vector_product_3d** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &w)
- template std::ostream & **Math::operator<< < std::complex< double > >** (std::ostream &os, const Vector< std::complex< double > > &v)
- template std::complex< double > **sum** (const Vector< std::complex< double > > &v)
- template std::complex< double > **multiply** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **cos** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **sin** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **tan** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **acos** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **asin** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **atan** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **cosh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **sinh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **tanh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **acosh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **asinh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **atanh** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **exp** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **log** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **log10** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > **pow** (const std::complex< double > v, const Vector< std::complex< double > > &exponent)
- template Vector< std::complex< double > > **pow** (const Vector< std::complex< double > > &v, const std::complex< double > exponent)
- template Vector< std::complex< double > > **pow** (const Vector< std::complex< double > > &v, const Vector< std::complex< double > > &exponent)
- template Vector< std::complex< double > > **sqrt** (const Vector< std::complex< double > > &v)
- template Vector< std::complex< double > > nan (const unsigned int N, const char ∗tagp)
- template Vector< double > **fabs** (const Vector< std::complex< double > > &v)
- template Matrix< std::complex< double > > **operator+** (const Matrix< std::complex< double > > &A, const Matrix< std::complex< double > > &B)
- template Matrix< std::complex< double > > **operator+** (const Matrix< std::complex< double > > &A, const std::complex< double > &alpha)
- template Matrix< std::complex< double > > **operator+** (const std::complex< double > &alpha, const Matrix< std::complex< double > > &A)
- template Matrix< std::complex< double > > **operator-** (const Matrix< std::complex< double > > &A, const Matrix< std::complex< double > > &B)
- template Matrix< std::complex< double > > **operator-** (const Matrix< std::complex< double > > &A, const std::complex< double > &alpha)
- template Matrix< std::complex< double > > **operator-** (const std::complex< double > &alpha, const Matrix< std::complex< double > > &A)
- template Matrix< std::complex< double > > **operator∗** (const Matrix< std::complex< double > > &A, const std::complex< double > &alpha)
- template Matrix< std::complex< double > > **operator∗** (const std::complex< double > &alpha, const Matrix< std::complex< double > > &A)
- template Matrix< std::complex< double > > **operator/** (const Matrix< std::complex< double > > &A, const std::complex< double > &alpha)
- template Matrix< std::complex< double > > **operator|** (const Matrix< std::complex< double > > &A, const Matrix< std::complex< double > > &B)
- template Vector< std::complex< double > > **operator|** (const Matrix< std::complex< double > > &A, const Vector< std::complex< double > > &v)

- template Vector< std::complex< double > > **operator|** (const Vector< std::complex< double > > &v, const Matrix< std::complex< double > > &A)
- template Vector< double > **operator+** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **operator+** (const Vector< double > &v, const double &alpha)
- template Vector< double > **operator+** (const double &alpha, const Vector< double > &v)
- template Vector< double > **operator-** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **operator-** (const Vector< double > &v, const double &alpha)
- template Vector< double > **operator-** (const double &alpha, const Vector< double > &v)
- template Vector< double > **operator*** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **operator*** (const Vector< double > &v, const double &alpha)
- template Vector< double > **operator*** (const double &alpha, const Vector< double > &v)
- template Vector< double > **operator/** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **operator/** (const Vector< double > &v, const double &alpha)
- template Vector< double > **operator/** (const double &alpha, const Vector< double > &v)
- template bool **operator<** (const Vector< double > &v, const double &alpha)
- template bool **operator<** (const double &alpha, const Vector< double > &v)
- template bool **operator<** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator<=** (const Vector< double > &v, const double &alpha)
- template bool **operator<=** (const double &alpha, const Vector< double > &v)
- template bool **operator<=** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator>** (const Vector< double > &v, const double &alpha)
- template bool **operator>** (const double &alpha, const Vector< double > &v)
- template bool **operator>** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator>=** (const Vector< double > &v, const double &alpha)
- template bool **operator>=** (const double &alpha, const Vector< double > &v)
- template bool **operator>=** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator==** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator==** (const Vector< double > &v, const double &alpha)
- template bool **operator==** (const double &alpha, const Vector< double > &v)
- template bool **operator!=** (const Vector< double > &v, const Vector< double > &w)
- template bool **operator!=** (const Vector< double > &v, const double &alpha)
- template bool **operator!=** (const double &alpha, const Vector< double > &v)
- template Vector< double > **operator&** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **operator&** (const Vector< double > &v, const double &alpha)
- template Vector< double > **operator&** (const double &alpha, const Vector< double > &v)
- template double **operator|** (const Vector< double > &v, const Vector< double > &w)
- template double **vector_product_2d** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **vector_product_3d** (const Vector< double > &v, const Vector< double > &w)
- template double **min** (const Vector< double > &v)
- template double **max** (const Vector< double > &v)
- template double **sum** (const Vector< double > &v)
- template double **multiply** (const Vector< double > &v)
- template Vector< double > **cos** (const Vector< double > &v)
- template Vector< double > **sin** (const Vector< double > &v)
- template Vector< double > **tan** (const Vector< double > &v)
- template Vector< double > **acos** (const Vector< double > &v)
- template Vector< double > **asin** (const Vector< double > &v)
- template Vector< double > **atan** (const Vector< double > &v)
- template Vector< double > **atan2** (const double v, const Vector< double > &w)
- template Vector< double > **atan2** (const Vector< double > &v, const double w)
- template Vector< double > **atan2** (const Vector< double > &v, const Vector< double > &w)
- template Vector< double > **cosh** (const Vector< double > &v)
- template Vector< double > **sinh** (const Vector< double > &v)
- template Vector< double > **tanh** (const Vector< double > &v)
- template Vector< double > **acosh** (const Vector< double > &v)

- template Vector< double > **asinh** (const Vector< double > &v)
- template Vector< double > **atanh** (const Vector< double > &v)
- template Vector< double > **exp** (const Vector< double > &v)
- template Vector< double > **frexp** (const Vector< double > &v, Vector< int > *exp)
- template Vector< double > **ldexp** (const Vector< double > &v, const int exp)
- template Vector< double > **ldexp** (const Vector< double > &v, const Vector< int > &exp)
- template Vector< double > **log** (const Vector< double > &v)
- template Vector< double > **log10** (const Vector< double > &v)
- template Vector< double > **modf** (const Vector< double > &v, Vector< double > *intpart)
- template Vector< double > **exp2** (const Vector< double > &v)
- template Vector< double > **expm1** (const Vector< double > &v)
- template Vector< double > **ilogb** (const Vector< double > &v)
- template Vector< double > **log1p** (const Vector< double > &v)
- template Vector< double > **log2** (const Vector< double > &v)
- template Vector< double > **logb** (const Vector< double > &v)
- template Vector< double > **scalbn** (const Vector< double > &v, const int n)
- template Vector< double > **scalbn** (const Vector< double > &v, const Vector< int > &n)
- template Vector< double > **scalbln** (const Vector< double > &v, const long int n)
- template Vector< double > **pow** (const double v, const Vector< double > &exponent)
- template Vector< double > **pow** (const Vector< double > &v, const double exponent)
- template Vector< double > **pow** (const Vector< double > &v, const Vector< double > &exponent)
- template Vector< double > **sqrt** (const Vector< double > &v)
- template Vector< double > **cbrt** (const Vector< double > &v)
- template Vector< double > **hypot** (const double x, const Vector< double > &y)
- template Vector< double > **hypot** (const Vector< double > &x, const double y)
- template Vector< double > **hypot** (const Vector< double > &x, const Vector< double > &y)
- template Vector< double > **erf** (const Vector< double > &v)
- template Vector< double > **erfc** (const Vector< double > &v)
- template Vector< double > **tgamma** (const Vector< double > &v)
- template Vector< double > **lgamma** (const Vector< double > &v)
- template Vector< double > **ceil** (const Vector< double > &v)
- template Vector< double > **floor** (const Vector< double > &v)
- template Vector< double > **fmod** (const double numer, const Vector< double > &denom)
- template Vector< double > **fmod** (const Vector< double > &numer, const double denom)
- template Vector< double > **fmod** (const Vector< double > &numer, const Vector< double > &denom)
- template Vector< double > **trunc** (const Vector< double > &v)
- template Vector< double > **round** (const Vector< double > &v)
- template Vector< long int > **lround** (const Vector< double > &v)
- template Vector< long long int > **llround** (const Vector< double > &v)
- template Vector< double > **rint** (const Vector< double > &v)
- template Vector< long int > **lrint** (const Vector< double > &v)
- template Vector< long long int > **llrint** (const Vector< double > &v)
- template Vector< double > **nearbyint** (const Vector< double > &v)
- template Vector< double > **remainder** (const double numer, const Vector< double > &denom)
- template Vector< double > **remainder** (const Vector< double > &numer, const double denom)
- template Vector< double > **remainder** (const Vector< double > &numer, const Vector< double > &denom)
- template Vector< double > **remquo** (const double numer, const Vector< double > &denom, Vector< int > *quot)
- template Vector< double > **remquo** (const Vector< double > &numer, const double denom, Vector< int > *quot)
- template Vector< double > **remquo** (const Vector< double > &numer, const Vector< double > &denom, Vector< int > *quot)
- template Vector< double > **copysign** (const Vector< double > &x, const double y)
- template Vector< double > **copysign** (const Vector< double > &x, const Vector< double > &y)
- template Vector< double > **nextafter** (const Vector< double > &x, const Vector< double > &y)

- template Vector< double > **fdim** (const double x, const Vector< double > &y)
- template Vector< double > **fdim** (const Vector< double > &x, double y)
- template Vector< double > **fdim** (const Vector< double > &x, const Vector< double > &y)
- template Vector< double > **fabs** (const Vector< double > &v)
- template Vector< double > **abs** (const Vector< double > &v)
- template Vector< double > **fma** (const double x, const Vector< double > &y, const Vector< double > &z)
- template Vector< double > **fma** (const Vector< double > &x, const double y, const Vector< double > &z)
- template Vector< double > **fma** (const Vector< double > &x, const Vector< double > &y, const double z)
- template Vector< double > **fma** (const double x, const double y, const Vector< double > &z)
- template Vector< double > **fma** (const double x, const Vector< double > &y, const double z)
- template Vector< double > **fma** (const Vector< double > &x, const double y, const double z)
- template Vector< double > **fma** (const Vector< double > &x, const Vector< double > &y, const Vector< double > &z)
- template Vector< int > **fpclassify** (const Vector< double > &v)
- template bool **isfinite** (const Vector< double > &v)
- template bool **isinf** (const Vector< double > &v)
- template bool **isnan** (const Vector< double > &v)
- template bool **isnormal** (const Vector< double > &v)
- template Matrix< double > **operator+** (const Matrix< double > &A, const Matrix< double > &B)
- template Matrix< double > **operator+** (const Matrix< double > &A, const double &alpha)
- template Matrix< double > **operator+** (const double &alpha, const Matrix< double > &A)
- template Matrix< double > **operator-** (const Matrix< double > &A, const Matrix< double > &B)
- template Matrix< double > **operator-** (const Matrix< double > &A, const double &alpha)
- template Matrix< double > **operator-** (const double &alpha, const Matrix< double > &A)
- template Matrix< double > **operator*** (const Matrix< double > &A, const double &alpha)
- template Matrix< double > **operator*** (const double &alpha, const Matrix< double > &A)
- template Matrix< double > **operator/** (const Matrix< double > &A, const double &alpha)
- template Matrix< double > **operator|** (const Matrix< double > &A, const Matrix< double > &B)
- template Vector< double > **operator|** (const Matrix< double > &A, const Vector< double > &v)
- template Vector< double > **operator|** (const Vector< double > &v, const Matrix< double > &A)
- template DualNumber< double > **operator+** (const DualNumber< double > &x, const DualNumber< double > &y)
- template DualNumber< double > **operator+** (const DualNumber< double > &x, const double y)
- template DualNumber< double > **operator+** (const double x, const DualNumber< double > &y)
- template DualNumber< double > **operator-** (const DualNumber< double > &x, const DualNumber< double > &y)
- template DualNumber< double > **operator-** (const DualNumber< double > &x, const double y)
- template DualNumber< double > **operator-** (const double x, const DualNumber< double > &y)
- template DualNumber< double > **operator*** (const DualNumber< double > &x, const DualNumber< double > &y)
- template DualNumber< double > **operator*** (const DualNumber< double > &x, const double y)
- template DualNumber< double > **operator*** (const double x, const DualNumber< double > &y)
- template DualNumber< double > **operator/** (const DualNumber< double > &x, const DualNumber< double > &y)
- template DualNumber< double > **operator/** (const DualNumber< double > &x, const double y)
- template DualNumber< double > **operator/** (const double x, const DualNumber< double > &y)
- template DualNumber< double > **cos** (const DualNumber< double > &x)
- template DualNumber< double > **sin** (const DualNumber< double > &x)
- template DualNumber< double > **tan** (const DualNumber< double > &x)
- template DualNumber< double > **acos** (const DualNumber< double > &x)
- template DualNumber< double > **asin** (const DualNumber< double > &x)
- template DualNumber< double > **atan** (const DualNumber< double > &x)
- template DualNumber< double > **cosh** (const DualNumber< double > &x)
- template DualNumber< double > **sinh** (const DualNumber< double > &x)
- template DualNumber< double > **tanh** (const DualNumber< double > &x)

- template DualNumber< double > **acosh** (const DualNumber< double > &x)
- template DualNumber< double > **asinh** (const DualNumber< double > &x)
- template DualNumber< double > **atanh** (const DualNumber< double > &x)
- template DualNumber< double > **exp** (const DualNumber< double > &x)
- template DualNumber< double > **log** (const DualNumber< double > &x)
- template DualNumber< double > **log10** (const DualNumber< double > &x)
- template DualNumber< double > **exp2** (const DualNumber< double > &x)
- template DualNumber< double > **expm1** (const DualNumber< double > &x)
- template DualNumber< double > **log1p** (const DualNumber< double > &x)
- template DualNumber< double > **log2** (const DualNumber< double > &x)
- template DualNumber< double > **logb** (const DualNumber< double > &x)
- template DualNumber< double > **pow** (const DualNumber< double > &x, const DualNumber< double > &y)
- template DualNumber< double > **pow** (const DualNumber< double > &x, const double &y)
- template DualNumber< double > **pow** (const double &x, const DualNumber< double > &y)
- template DualNumber< double > **sqrt** (const DualNumber< double > &x)
- template DualNumber< double > **cbrt** (const DualNumber< double > &x)
- template DualNumber< double > **erf** (const DualNumber< double > &x)
- template DualNumber< double > **erfc** (const DualNumber< double > &x)
- template DualNumber< double > **fabs** (const DualNumber< double > &x)
- template Vector< float > **operator+** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **operator+** (const Vector< float > &v, const float &alpha)
- template Vector< float > **operator+** (const float &alpha, const Vector< float > &v)
- template Vector< float > **operator-** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **operator-** (const Vector< float > &v, const float &alpha)
- template Vector< float > **operator-** (const float &alpha, const Vector< float > &v)
- template Vector< float > **operator*** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **operator*** (const Vector< float > &v, const float &alpha)
- template Vector< float > **operator*** (const float &alpha, const Vector< float > &v)
- template Vector< float > **operator/** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **operator/** (const Vector< float > &v, const float &alpha)
- template Vector< float > **operator/** (const float &alpha, const Vector< float > &v)
- template bool **operator<** (const Vector< float > &v, const float &alpha)
- template bool **operator<** (const float &alpha, const Vector< float > &v)
- template bool **operator<** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator<=** (const Vector< float > &v, const float &alpha)
- template bool **operator<=** (const float &alpha, const Vector< float > &v)
- template bool **operator<=** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator>** (const Vector< float > &v, const float &alpha)
- template bool **operator>** (const float &alpha, const Vector< float > &v)
- template bool **operator>** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator>=** (const Vector< float > &v, const float &alpha)
- template bool **operator>=** (const float &alpha, const Vector< float > &v)
- template bool **operator>=** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator==** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator==** (const Vector< float > &v, const float &alpha)
- template bool **operator==** (const float &alpha, const Vector< float > &v)
- template bool **operator!=** (const Vector< float > &v, const Vector< float > &w)
- template bool **operator!=** (const Vector< float > &v, const float &alpha)
- template bool **operator!=** (const float &alpha, const Vector< float > &v)
- template Vector< float > **operator&** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **operator&** (const Vector< float > &v, const float &alpha)
- template Vector< float > **operator&** (const float &alpha, const Vector< float > &v)
- template float **operator|** (const Vector< float > &v, const Vector< float > &w)
- template float **vector_product_2d** (const Vector< float > &v, const Vector< float > &w)

- template Vector< float > **vector_product_3d** (const Vector< float > &v, const Vector< float > &w)
- template float **min** (const Vector< float > &v)
- template float **max** (const Vector< float > &v)
- template float **sum** (const Vector< float > &v)
- template float **multiply** (const Vector< float > &v)
- template Vector< float > **cos** (const Vector< float > &v)
- template Vector< float > **sin** (const Vector< float > &v)
- template Vector< float > **tan** (const Vector< float > &v)
- template Vector< float > **acos** (const Vector< float > &v)
- template Vector< float > **asin** (const Vector< float > &v)
- template Vector< float > **atan** (const Vector< float > &v)
- template Vector< float > **atan2** (const float v, const Vector< float > &w)
- template Vector< float > **atan2** (const Vector< float > &v, const float w)
- template Vector< float > **atan2** (const Vector< float > &v, const Vector< float > &w)
- template Vector< float > **cosh** (const Vector< float > &v)
- template Vector< float > **sinh** (const Vector< float > &v)
- template Vector< float > **tanh** (const Vector< float > &v)
- template Vector< float > **acosh** (const Vector< float > &v)
- template Vector< float > **asinh** (const Vector< float > &v)
- template Vector< float > **atanh** (const Vector< float > &v)
- template Vector< float > **exp** (const Vector< float > &v)
- template Vector< float > **frexp** (const Vector< float > &v, Vector< int > ∗exp)
- template Vector< float > **ldexp** (const Vector< float > &v, const int exp)
- template Vector< float > **ldexp** (const Vector< float > &v, const Vector< int > &exp)
- template Vector< float > **log** (const Vector< float > &v)
- template Vector< float > **log10** (const Vector< float > &v)
- template Vector< float > **exp2** (const Vector< float > &v)
- template Vector< float > **expm1** (const Vector< float > &v)
- template Vector< float > **ilogb** (const Vector< float > &v)
- template Vector< float > **log1p** (const Vector< float > &v)
- template Vector< float > **log2** (const Vector< float > &v)
- template Vector< float > **logb** (const Vector< float > &v)
- template Vector< float > **scalbn** (const Vector< float > &v, const int n)
- template Vector< float > **scalbn** (const Vector< float > &v, const Vector< int > &n)
- template Vector< float > **scalbln** (const Vector< float > &v, const long int n)
- template Vector< float > **pow** (const float v, const Vector< float > &exponent)
- template Vector< float > **pow** (const Vector< float > &v, const float exponent)
- template Vector< float > **pow** (const Vector< float > &v, const Vector< float > &exponent)
- template Vector< float > **sqrt** (const Vector< float > &v)
- template Vector< float > **cbrt** (const Vector< float > &v)
- template Vector< float > **hypot** (const float x, const Vector< float > &y)
- template Vector< float > **hypot** (const Vector< float > &x, const float y)
- template Vector< float > **hypot** (const Vector< float > &x, const Vector< float > &y)
- template Vector< float > **erf** (const Vector< float > &v)
- template Vector< float > **erfc** (const Vector< float > &v)
- template Vector< float > **tgamma** (const Vector< float > &v)
- template Vector< float > **lgamma** (const Vector< float > &v)
- template Vector< float > **ceil** (const Vector< float > &v)
- template Vector< float > **floor** (const Vector< float > &v)
- template Vector< float > **fmod** (const float numer, const Vector< float > &denom)
- template Vector< float > **fmod** (const Vector< float > &numer, const float denom)
- template Vector< float > **fmod** (const Vector< float > &numer, const Vector< float > &denom)
- template Vector< float > **trunc** (const Vector< float > &v)
- template Vector< float > **round** (const Vector< float > &v)
- template Vector< long int > **lround** (const Vector< float > &v)

- template Vector< long long int > **llround** (const Vector< float > &v)
- template Vector< float > **rint** (const Vector< float > &v)
- template Vector< long int > **lrint** (const Vector< float > &v)
- template Vector< long long int > **llrint** (const Vector< float > &v)
- template Vector< float > **nearbyint** (const Vector< float > &v)
- template Vector< float > **remainder** (const float numer, const Vector< float > &denom)
- template Vector< float > **remainder** (const Vector< float > &numer, const float denom)
- template Vector< float > **remainder** (const Vector< float > &numer, const Vector< float > &denom)
- template Vector< float > **remquo** (const float numer, const Vector< float > &denom, Vector< int > *quot)
- template Vector< float > **remquo** (const Vector< float > &numer, const float denom, Vector< int > *quot)
- template Vector< float > **remquo** (const Vector< float > &numer, const Vector< float > &denom, Vector< int > *quot)
- template Vector< float > **copysign** (const Vector< float > &x, const float y)
- template Vector< float > **copysign** (const Vector< float > &x, const Vector< float > &y)
- template Vector< float > **nextafter** (const Vector< float > &x, const Vector< float > &y)
- template Vector< float > **fdim** (const float x, const Vector< float > &y)
- template Vector< float > **fdim** (const Vector< float > &x, float y)
- template Vector< float > **fdim** (const Vector< float > &x, const Vector< float > &y)
- template Vector< float > **fabs** (const Vector< float > &v)
- template Vector< float > **abs** (const Vector< float > &v)
- template Vector< float > **fma** (const float x, const Vector< float > &y, const Vector< float > &z)
- template Vector< float > **fma** (const Vector< float > &x, const float y, const Vector< float > &z)
- template Vector< float > **fma** (const Vector< float > &x, const Vector< float > &y, const float z)
- template Vector< float > **fma** (const float x, const float y, const Vector< float > &z)
- template Vector< float > **fma** (const float x, const Vector< float > &y, const float z)
- template Vector< float > **fma** (const Vector< float > &x, const float y, const float z)
- template Vector< float > **fma** (const Vector< float > &x, const Vector< float > &y, const Vector< float > &z)
- template Vector< int > **fpclassify** (const Vector< float > &v)
- template bool **isfinite** (const Vector< float > &v)
- template bool **isinf** (const Vector< float > &v)
- template bool **isnan** (const Vector< float > &v)
- template bool **isnormal** (const Vector< float > &v)
- template Vector< int > **operator+** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **operator+** (const Vector< int > &v, const int &alpha)
- template Vector< int > **operator+** (const int &alpha, const Vector< int > &v)
- template Vector< int > **operator-** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **operator-** (const Vector< int > &v, const int &alpha)
- template Vector< int > **operator-** (const int &alpha, const Vector< int > &v)
- template Vector< int > **operator*** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **operator*** (const Vector< int > &v, const int &alpha)
- template Vector< int > **operator*** (const int &alpha, const Vector< int > &v)
- template Vector< int > **operator/** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **operator/** (const Vector< int > &v, const int &alpha)
- template Vector< int > **operator/** (const int &alpha, const Vector< int > &v)
- template bool **operator<** (const Vector< int > &v, const int &alpha)
- template bool **operator<** (const int &alpha, const Vector< int > &v)
- template bool **operator<** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator<=** (const Vector< int > &v, const int &alpha)
- template bool **operator<=** (const int &alpha, const Vector< int > &v)
- template bool **operator<=** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator>** (const Vector< int > &v, const int &alpha)
- template bool **operator>** (const int &alpha, const Vector< int > &v)
- template bool **operator>** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator>=** (const Vector< int > &v, const int &alpha)

- template bool **operator>=** (const int &alpha, const Vector< int > &v)
- template bool **operator>=** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator==** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator==** (const Vector< int > &v, const int &alpha)
- template bool **operator==** (const int &alpha, const Vector< int > &v)
- template bool **operator!=** (const Vector< int > &v, const Vector< int > &w)
- template bool **operator!=** (const Vector< int > &v, const int &alpha)
- template bool **operator!=** (const int &alpha, const Vector< int > &v)
- template Vector< int > **operator&** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **operator&** (const Vector< int > &v, const int &alpha)
- template Vector< int > **operator&** (const int &alpha, const Vector< int > &v)
- template int **operator|** (const Vector< int > &v, const Vector< int > &w)
- template int **vector_product_2d** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **vector_product_3d** (const Vector< int > &v, const Vector< int > &w)
- template int **min** (const Vector< int > &v)
- template int **max** (const Vector< int > &v)
- template int **sum** (const Vector< int > &v)
- template int **multiply** (const Vector< int > &v)
- template Vector< int > **cos** (const Vector< int > &v)
- template Vector< int > **sin** (const Vector< int > &v)
- template Vector< int > **tan** (const Vector< int > &v)
- template Vector< int > **acos** (const Vector< int > &v)
- template Vector< int > **asin** (const Vector< int > &v)
- template Vector< int > **atan** (const Vector< int > &v)
- template Vector< int > **atan2** (const int v, const Vector< int > &w)
- template Vector< int > **atan2** (const Vector< int > &v, const int w)
- template Vector< int > **atan2** (const Vector< int > &v, const Vector< int > &w)
- template Vector< int > **cosh** (const Vector< int > &v)
- template Vector< int > **sinh** (const Vector< int > &v)
- template Vector< int > **tanh** (const Vector< int > &v)
- template Vector< int > **acosh** (const Vector< int > &v)
- template Vector< int > **asinh** (const Vector< int > &v)
- template Vector< int > **atanh** (const Vector< int > &v)
- template Vector< int > **exp** (const Vector< int > &v)
- template Vector< int > **frexp** (const Vector< int > &v, Vector< int > *exp)
- template Vector< int > **ldexp** (const Vector< int > &v, const int exp)
- template Vector< int > **ldexp** (const Vector< int > &v, const Vector< int > &exp)
- template Vector< int > **log** (const Vector< int > &v)
- template Vector< int > **log10** (const Vector< int > &v)
- template Vector< int > **exp2** (const Vector< int > &v)
- template Vector< int > **expm1** (const Vector< int > &v)
- template Vector< int > **ilogb** (const Vector< int > &v)
- template Vector< int > **log1p** (const Vector< int > &v)
- template Vector< int > **log2** (const Vector< int > &v)
- template Vector< int > **logb** (const Vector< int > &v)
- template Vector< int > **scalbn** (const Vector< int > &v, const int n)
- template Vector< int > **scalbn** (const Vector< int > &v, const Vector< int > &n)
- template Vector< int > **scalbln** (const Vector< int > &v, const long int n)
- template Vector< int > **pow** (const int v, const Vector< int > &exponent)
- template Vector< int > **pow** (const Vector< int > &v, const int exponent)
- template Vector< int > **pow** (const Vector< int > &v, const Vector< int > &exponent)
- template Vector< int > **sqrt** (const Vector< int > &v)
- template Vector< int > **cbrt** (const Vector< int > &v)
- template Vector< int > **hypot** (const int x, const Vector< int > &y)
- template Vector< int > **hypot** (const Vector< int > &x, const int y)

- template Vector< int > **hypot** (const Vector< int > &x, const Vector< int > &y)
- template Vector< int > **erf** (const Vector< int > &v)
- template Vector< int > **erfc** (const Vector< int > &v)
- template Vector< int > **tgamma** (const Vector< int > &v)
- template Vector< int > **lgamma** (const Vector< int > &v)
- template Vector< int > **ceil** (const Vector< int > &v)
- template Vector< int > **floor** (const Vector< int > &v)
- template Vector< int > **fmod** (const int numer, const Vector< int > &denom)
- template Vector< int > **fmod** (const Vector< int > &numer, const int denom)
- template Vector< int > **fmod** (const Vector< int > &numer, const Vector< int > &denom)
- template Vector< int > **trunc** (const Vector< int > &v)
- template Vector< int > **round** (const Vector< int > &v)
- template Vector< long int > **lround** (const Vector< int > &v)
- template Vector< long long int > **llround** (const Vector< int > &v)
- template Vector< int > **rint** (const Vector< int > &v)
- template Vector< long int > **lrint** (const Vector< int > &v)
- template Vector< long long int > **llrint** (const Vector< int > &v)
- template Vector< int > **nearbyint** (const Vector< int > &v)
- template Vector< int > **remainder** (const int numer, const Vector< int > &denom)
- template Vector< int > **remainder** (const Vector< int > &numer, const int denom)
- template Vector< int > **remainder** (const Vector< int > &numer, const Vector< int > &denom)
- template Vector< int > **remquo** (const int numer, const Vector< int > &denom, Vector< int > ∗quot)
- template Vector< int > **remquo** (const Vector< int > &numer, const int denom, Vector< int > ∗quot)
- template Vector< int > **remquo** (const Vector< int > &numer, const Vector< int > &denom, Vector< int > ∗quot)
- template Vector< int > **copysign** (const Vector< int > &x, const int y)
- template Vector< int > **copysign** (const Vector< int > &x, const Vector< int > &y)
- template Vector< int > **nextafter** (const Vector< int > &x, const Vector< int > &y)
- template Vector< int > **fdim** (const int x, const Vector< int > &y)
- template Vector< int > **fdim** (const Vector< int > &x, int y)
- template Vector< int > **fdim** (const Vector< int > &x, const Vector< int > &y)
- template Vector< int > **fabs** (const Vector< int > &v)
- template Vector< int > **fma** (const int x, const Vector< int > &y, const Vector< int > &z)
- template Vector< int > **fma** (const Vector< int > &x, const int y, const Vector< int > &z)
- template Vector< int > **fma** (const Vector< int > &x, const Vector< int > &y, const int z)
- template Vector< int > **fma** (const int x, const int y, const Vector< int > &z)
- template Vector< int > **fma** (const int x, const Vector< int > &y, const int z)
- template Vector< int > **fma** (const Vector< int > &x, const int y, const int z)
- template Vector< int > **fma** (const Vector< int > &x, const Vector< int > &y, const Vector< int > &z)
- template Vector< int > **fpclassify** (const Vector< int > &v)
- template bool **isfinite** (const Vector< int > &v)
- template bool **isinf** (const Vector< int > &v)
- template bool **isnan** (const Vector< int > &v)
- template bool **isnormal** (const Vector< int > &v)
- template Vector< unsigned int > **operator+** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **operator+** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template Vector< unsigned int > **operator+** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template Vector< unsigned int > **operator-** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **operator-** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template Vector< unsigned int > **operator-** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template Vector< unsigned int > **operator∗** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **operator∗** (const Vector< unsigned int > &v, const unsigned int &alpha)

- template Vector< unsigned int > **operator∗** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template Vector< unsigned int > **operator/** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **operator/** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template Vector< unsigned int > **operator/** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator<** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator<** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator<** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator<=** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator<=** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator<=** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator>** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator>** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator>** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator>=** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator>=** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator>=** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator==** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator==** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator==** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template bool **operator!=** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template bool **operator!=** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template bool **operator!=** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template Vector< unsigned int > **operator&** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **operator&** (const Vector< unsigned int > &v, const unsigned int &alpha)
- template Vector< unsigned int > **operator&** (const unsigned int &alpha, const Vector< unsigned int > &v)
- template unsigned int **operator|** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template unsigned int **vector_product_2d** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **vector_product_3d** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template unsigned int **min** (const Vector< unsigned int > &v)
- template unsigned int **max** (const Vector< unsigned int > &v)
- template unsigned int **sum** (const Vector< unsigned int > &v)
- template unsigned int **multiply** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **cos** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **sin** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **tan** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **acos** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **asin** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **atan** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **atan2** (const unsigned int v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **atan2** (const Vector< unsigned int > &v, const unsigned int w)
- template Vector< unsigned int > **atan2** (const Vector< unsigned int > &v, const Vector< unsigned int > &w)
- template Vector< unsigned int > **cosh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **sinh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **tanh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **acosh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **asinh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **atanh** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **exp** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **frexp** (const Vector< unsigned int > &v, Vector< int > ∗exp)
- template Vector< unsigned int > **ldexp** (const Vector< unsigned int > &v, const int exp)

- template Vector< unsigned int > **ldexp** (const Vector< unsigned int > &v, const Vector< int > &exp)
- template Vector< unsigned int > **log** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **log10** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **exp2** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **expm1** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **ilogb** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **log1p** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **log2** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **logb** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **scalbn** (const Vector< unsigned int > &v, const int n)
- template Vector< unsigned int > **scalbn** (const Vector< unsigned int > &v, const Vector< int > &n)
- template Vector< unsigned int > **scalbln** (const Vector< unsigned int > &v, const long int n)
- template Vector< unsigned int > **pow** (const unsigned int v, const Vector< unsigned int > &exponent)
- template Vector< unsigned int > **pow** (const Vector< unsigned int > &v, const unsigned int exponent)
- template Vector< unsigned int > **pow** (const Vector< unsigned int > &v, const Vector< unsigned int > &exponent)
- template Vector< unsigned int > **sqrt** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **cbrt** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **hypot** (const unsigned int x, const Vector< unsigned int > &y)
- template Vector< unsigned int > **hypot** (const Vector< unsigned int > &x, const unsigned int y)
- template Vector< unsigned int > **hypot** (const Vector< unsigned int > &x, const Vector< unsigned int > &y)
- template Vector< unsigned int > **erf** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **erfc** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **tgamma** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **lgamma** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **ceil** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **floor** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **fmod** (const unsigned int numer, const Vector< unsigned int > &denom)
- template Vector< unsigned int > **fmod** (const Vector< unsigned int > &numer, const unsigned int denom)
- template Vector< unsigned int > **fmod** (const Vector< unsigned int > &numer, const Vector< unsigned int > &denom)
- template Vector< unsigned int > **trunc** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **round** (const Vector< unsigned int > &v)
- template Vector< long int > **lround** (const Vector< unsigned int > &v)
- template Vector< long long int > **llround** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **rint** (const Vector< unsigned int > &v)
- template Vector< long int > **lrint** (const Vector< unsigned int > &v)
- template Vector< long long int > **llrint** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **nearbyint** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **remainder** (const unsigned int numer, const Vector< unsigned int > &denom)
- template Vector< unsigned int > **remainder** (const Vector< unsigned int > &numer, const unsigned int denom)
- template Vector< unsigned int > **remainder** (const Vector< unsigned int > &numer, const Vector< unsigned int > &denom)
- template Vector< unsigned int > **remquo** (const unsigned int numer, const Vector< unsigned int > &denom, Vector< int > *quot)
- template Vector< unsigned int > **remquo** (const Vector< unsigned int > &numer, const unsigned int denom, Vector< int > *quot)
- template Vector< unsigned int > **remquo** (const Vector< unsigned int > &numer, const Vector< unsigned int > &denom, Vector< int > *quot)
- template Vector< unsigned int > **copysign** (const Vector< unsigned int > &x, const unsigned int y)
- template Vector< unsigned int > **copysign** (const Vector< unsigned int > &x, const Vector< unsigned int > &y)

- template Vector< unsigned int > **nextafter** (const Vector< unsigned int > &x, const Vector< unsigned int > &y)
- template Vector< unsigned int > **fdim** (const unsigned int x, const Vector< unsigned int > &y)
- template Vector< unsigned int > **fdim** (const Vector< unsigned int > &x, unsigned int y)
- template Vector< unsigned int > **fdim** (const Vector< unsigned int > &x, const Vector< unsigned int > &y)
- template Vector< unsigned int > **fabs** (const Vector< unsigned int > &v)
- template Vector< unsigned int > **fma** (const unsigned int x, const Vector< unsigned int > &y, const Vector< unsigned int > &z)
- template Vector< unsigned int > **fma** (const Vector< unsigned int > &x, const unsigned int y, const Vector< unsigned int > &z)
- template Vector< unsigned int > **fma** (const Vector< unsigned int > &x, const Vector< unsigned int > &y, const unsigned int z)
- template Vector< unsigned int > **fma** (const unsigned int x, const unsigned int y, const Vector< unsigned int > &z)
- template Vector< unsigned int > **fma** (const unsigned int x, const Vector< unsigned int > &y, const unsigned int z)
- template Vector< unsigned int > **fma** (const Vector< unsigned int > &x, const unsigned int y, const unsigned int z)
- template Vector< unsigned int > **fma** (const Vector< unsigned int > &x, const Vector< unsigned int > &y, const Vector< unsigned int > &z)
- template Vector< int > **fpclassify** (const Vector< unsigned int > &v)
- template bool **isfinite** (const Vector< unsigned int > &v)
- template bool **isinf** (const Vector< unsigned int > &v)
- template bool **isnan** (const Vector< unsigned int > &v)
- template bool **isnormal** (const Vector< unsigned int > &v)
- template<typename K >
  Matrix< K > **operator+** (const Matrix< K > &A, const Matrix< K > &B)
- template<typename K >
  Matrix< K > **operator+** (const Matrix< K > &A, const K &alpha)
- template<typename K >
  Matrix< K > **operator+** (const K &alpha, const Matrix< K > &A)
- template<typename K >
  Matrix< K > **operator-** (const Matrix< K > &A, const Matrix< K > &B)
- template<typename K >
  Matrix< K > **operator-** (const Matrix< K > &A, const K &alpha)
- template<typename K >
  Matrix< K > **operator-** (const K &alpha, const Matrix< K > &A)
- template<typename K >
  Matrix< K > **operator*** (const Matrix< K > &A, const K &alpha)
- template<typename K >
  Matrix< K > **operator*** (const K &alpha, const Matrix< K > &A)
- template<typename K >
  Matrix< K > **operator/** (const Matrix< K > &A, const K &alpha)
- template<typename K >
  Matrix< K > **operator|** (const Matrix< K > &A, const Matrix< K > &B)
- template<typename K >
  Vector< K > **operator|** (const Matrix< K > &A, const Vector< K > &v)
- template<typename K >
  Vector< K > **operator|** (const Vector< K > &v, const Matrix< K > &A)
- template<typename K >
  Rational< K > & **operator+** (const Rational< K > &p, const Rational< K > &q)
- template<typename K >
  Rational< K > & **operator-** (const Rational< K > &p, const Rational< K > &q)
- template<typename K >
  Rational< K > & **operator*** (const Rational< K > &p, const Rational< K > &q)

- template< typename K >
  Rational< K > & **operator/** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator==** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator!=** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator<** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator<=** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator>** (const Rational< K > &p, const Rational< K > &q)
- template< typename K >
  bool **operator>=** (const Rational< K > &p, const Rational< K > &q)
- char conj (const char &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned char conj (const unsigned char &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- short int conj (const short int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned short int conj (const unsigned short int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- int conj (const int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned int conj (const unsigned int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- long int conj (const long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned long int conj (const unsigned long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- long long int conj (const long long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned long long int conj (const unsigned long long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- float conj (const float &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- double conj (const double &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- long double conj (const long double &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- wchar_t conj (const wchar_t &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- template< typename K >
  std::complex< K > conj (const std::complex< K > &val)

  This function exists to make the definition of the scalar product more general. It just returns the complex conjugate of val .
- template< typename K >
  std::string **to_string** (const std::complex< K > &z)
- template< typename K >
  Vector< K > operator+ (const Vector< K > &v, const Vector< K > &w)
- template< typename K >
  Vector< K > operator+ (const Vector< K > &v, const K &alpha)

- template<typename K >
  Vector< K > operator+ (const K &alpha, const Vector< K > &w)
- template<typename K >
  Vector< K > operator- (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > operator- (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > operator- (const K &alpha, const Vector< K > &w)
- template<typename K >
  Vector< K > operator* (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > operator* (const K &alpha, const Vector< K > &v)
- template<typename K >
  Vector< K > operator* (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > operator/ (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > operator/ (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > operator/ (const K &alpha, const Vector< K > &w)
- template<typename K >
  bool **operator**< (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool **operator**< (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool **operator**< (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool **operator**<**=** (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool **operator**<**=** (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool **operator**<**=** (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool operator> (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool operator> (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool operator> (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool operator>= (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool operator>= (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool operator>= (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool operator== (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool operator== (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool operator== (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool operator!= (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool operator!= (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool operator!= (const K &alpha, const Vector< K > &v)

- template<typename K >
  Vector< K > operator& (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > operator& (const Vector< K > &v, const K &w)
- template<typename K >
  Vector< K > operator& (const K &v, const Vector< K > &w)
- template<typename K >
  K operator| (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  K vector_product_2d (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > vector_product_3d (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  std::ostream & **operator<<** (std::ostream &os, const Vector< K > &v)
- template<typename K >
  K min (const Vector< K > &v)
- template<typename K >
  K max (const Vector< K > &v)
- template<typename K >
  K sum (const Vector< K > &v)
- template<typename K >
  K multiply (const Vector< K > &v)
- template<typename K >
  Vector< K > cos (const Vector< K > &v)
- template<typename K >
  Vector< K > sin (const Vector< K > &v)
- template<typename K >
  Vector< K > tan (const Vector< K > &v)
- template<typename K >
  Vector< K > acos (const Vector< K > &v)
- template<typename K >
  Vector< K > asin (const Vector< K > &v)
- template<typename K >
  Vector< K > atan (const Vector< K > &v)
- template<typename K >
  Vector< K > atan2 (const K v, const Vector< K > &w)
- template<typename K >
  Vector< K > atan2 (const Vector< K > &v, const K w)
- template<typename K >
  Vector< K > atan2 (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > cosh (const Vector< K > &v)
- template<typename K >
  Vector< K > sinh (const Vector< K > &v)
- template<typename K >
  Vector< K > tanh (const Vector< K > &v)
- template<typename K >
  Vector< K > acosh (const Vector< K > &v)
- template<typename K >
  Vector< K > asinh (const Vector< K > &v)
- template<typename K >
  Vector< K > atanh (const Vector< K > &v)
- template<typename K >
  Vector< K > exp (const Vector< K > &v)
- template<typename K >
  Vector< K > frexp (const Vector< K > &v, Vector< int > ∗exp)

- template<typename K >
  Vector< K > ldexp (const Vector< K > &v, const int exp)
- template<typename K >
  Vector< K > ldexp (const Vector< K > &v, const Vector< int > &exp)
- template<typename K >
  Vector< K > log (const Vector< K > &v)
- template<typename K >
  Vector< K > log10 (const Vector< K > &v)
- template<typename K >
  Vector< K > modf (const Vector< K > &v, Vector< K > *intpart)
- template<typename K >
  Vector< K > exp2 (const Vector< K > &v)
- template<typename K >
  Vector< K > expm1 (const Vector< K > &v)
- template<typename K >
  Vector< K > ilogb (const Vector< K > &v)
- template<typename K >
  Vector< K > log1p (const Vector< K > &v)
- template<typename K >
  Vector< K > log2 (const Vector< K > &v)
- template<typename K >
  Vector< K > logb (const Vector< K > &v)
- template<typename K >
  Vector< K > scalbn (const Vector< K > &v, const int n)
- template<typename K >
  Vector< K > scalbn (const Vector< K > &v, const Vector< int > &n)
- template<typename K >
  Vector< K > scalbln (const Vector< K > &v, const long int n)
- template<typename K >
  Vector< K > scalbln (const Vector< K > &v, const Vector< long int > &n)
- template<typename K >
  Vector< K > pow (const K v, const Vector< K > &exponent)
- template<typename K >
  Vector< K > pow (const Vector< K > &v, const K exponent)
- template<typename K >
  Vector< K > pow (const Vector< K > &v, const Vector< K > &exponent)
- template<typename K >
  Vector< K > sqrt (const Vector< K > &v)
- template<typename K >
  Vector< K > cbrt (const Vector< K > &v)
- template<typename K >
  Vector< K > hypot (const K x, const Vector< K > &y)
- template<typename K >
  Vector< K > hypot (const Vector< K > &x, const K y)
- template<typename K >
  Vector< K > hypot (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< K > erf (const Vector< K > &v)
- template<typename K >
  Vector< K > erfc (const Vector< K > &v)
- template<typename K >
  Vector< K > tgamma (const Vector< K > &v)
- template<typename K >
  Vector< K > lgamma (const Vector< K > &v)
- template<typename K >
  Vector< K > ceil (const Vector< K > &v)

- template<typename K >
  Vector< K > floor (const Vector< K > &v)
- template<typename K >
  Vector< K > fmod (const K numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > fmod (const Vector< K > &numer, const K denom)
- template<typename K >
  Vector< K > fmod (const Vector< K > &numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > trunc (const Vector< K > &v)
- template<typename K >
  Vector< K > round (const Vector< K > &v)
- template<typename K >
  Vector< long int > lround (const Vector< K > &v)
- template<typename K >
  Vector< long long int > llround (const Vector< K > &v)
- template<typename K >
  Vector< K > rint (const Vector< K > &v)
- template<typename K >
  Vector< long int > lrint (const Vector< K > &v)
- template<typename K >
  Vector< long long int > llrint (const Vector< K > &v)
- template<typename K >
  Vector< K > nearbyint (const Vector< K > &v)
- template<typename K >
  Vector< K > remainder (const K numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > remainder (const Vector< K > &numer, const K denom)
- template<typename K >
  Vector< K > remainder (const Vector< K > &numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > remquo (const K numer, const Vector< K > &denom, Vector< int > *quot)
- template<typename K >
  Vector< K > remquo (const Vector< K > &numer, const K denom, Vector< int > *quot)
- template<typename K >
  Vector< K > remquo (const Vector< K > &numer, const Vector< K > &denom, Vector< int > *quot)
- template<typename K >
  Vector< K > copysign (const Vector< K > &x, const K y)
- template<typename K >
  Vector< K > copysign (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< K > nextafter (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< K > fdim (const K x, const Vector< K > &y)
- template<typename K >
  Vector< K > fdim (const Vector< K > &x, K y)
- template<typename K >
  Vector< K > fdim (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< double > fabs (const Vector< K > &v)
- template<typename K >
  Vector< double > abs (const Vector< K > &v)
- template<typename K >
  Vector< K > fma (const K x, const Vector< K > &y, const Vector< K > &z)
- template<typename K >
  Vector< K > fma (const Vector< K > &x, const K y, const Vector< K > &z)

- template<typename K >
  Vector< K > fma (const Vector< K > &x, const Vector< K > &y, const K z)
- template<typename K >
  Vector< K > fma (const K x, const K y, const Vector< K > &z)
- template<typename K >
  Vector< K > fma (const K x, const Vector< K > &y, const K z)
- template<typename K >
  Vector< K > fma (const Vector< K > &x, const K y, const K z)
- template<typename K >
  Vector< K > fma (const Vector< K > &x, const Vector< K > &y, const Vector< K > &z)
- template<typename K >
  Vector< int > fpclassify (const Vector< K > &v)
- template<typename K >
  bool isfinite (const Vector< K > &v)
- template<typename K >
  bool isinf (const Vector< K > &v)
- template<typename K >
  bool isnan (const Vector< K > &v)
- template<typename K >
  bool isnormal (const Vector< K > &v)
- template<typename K >
  Vector< K > **real** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > **imag** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > **arg** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > conj (const Vector< K > &v)

6.5.1 Detailed Description

Namespace that includes mathematical objects such as vectors, matrixes, dual numbers and useful numerical methods such as numerical integrators, ode solvers and solvers for algebraic equations.

6.5.2 Function Documentation

6.5.2.1 abs()

```
template<typename K >
Vector< double > Math::abs (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.2 acos() [1/2]

```
template<typename K >
DualNumber< K > Math::acos (
            const DualNumber< K > & x )
```

Returns acos(x.a) - 1/sqrt(1-x.a∗x.a)∗x.b∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.3 acos() [2/2]

```
template<typename K >
Vector< K > Math::acos (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.4 acosh()

```
template<typename K >
Vector< K > Math::acosh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.5 asin() [1/2]

```
template<typename K >
DualNumber< K > Math::asin (
            const DualNumber< K > & x )
```

Returns asin(x.a) + 1/sqrt(1-x.a∗x.a)∗x.b∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.6 asin() [2/2]

```
template<typename K >
Vector< K > Math::asin (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.7 asinh()

```
template<typename K >
Vector< K > Math::asinh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.8 atan() [1/2]

```
template<typename K >
DualNumber< K > Math::atan (
            const DualNumber< K > & x )
```

Returns atan(x.a) + 1/(1+x.a*x.a)*x.b*

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.9 atan() [2/2]

```
template<typename K >
Vector< K > Math::atan (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.10 atan2() [1/3]

```
template<typename K >
Vector< K > Math::atan2 (
            const K v,
            const Vector< K > & w )
```

Parameters

v	
w	

Returns

Vector<K>

6.5.2.11 atan2() [2/3]

```
template<typename K >
Vector< K > Math::atan2 (
            const Vector< K > & v,
            const K w )
```

Parameters

v	
w	

Returns

Vector<K>

6.5.2.12 atan2() [3/3]

```
template<typename K >
Vector< K > Math::atan2 (
            const Vector< K > & v,
            const Vector< K > & w )
```

Exceptions

Throws	an exception if *v* and *w* have different lengths.

Parameters

v	
w	

Returns

Vector<K>

6.5.2.13 atanh()

```
template<typename K >
Vector< K > Math::atanh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.14 cbrt()

```
template<typename K >
Vector< K > Math::cbrt (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.15 ceil()

```
template<typename K >
Vector< K > Math::ceil (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.16 conj() [1/16]

```
char Math::conj (
            const char & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

char

6.5.2.17 conj() [2/16]

```
double Math::conj (
            const double & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

double

6.5.2.18 conj() [3/16]

```
float Math::conj (
            const float & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

float

6.5.2.19 conj() [4/16]

```
int Math::conj (
            const int & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

int

6.5.2.20 conj() [5/16]

```
long double Math::conj (
            const long double & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

long double

6.5.2.21 conj() [6/16]

```
long int Math::conj (
             const long int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

long int

6.5.2.22 conj() [7/16]

```
long long int Math::conj (
             const long long int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

long long int

6.5.2.23 conj() [8/16]

```
short int Math::conj (
             const short int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

short int

6.5.2.24 conj() [9/16]

```
template<typename K >
std::complex< K > Math::conj (
            const std::complex< K > & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns the complex conjugate of *val* .

Template Parameters

K	

Parameters

val	

Returns

std::complex<K>

6.5.2.25 conj() [10/16]

```
unsigned char Math::conj (
            const unsigned char & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

unsigned char

6.5.2.26 conj() [11/16]

```
unsigned int Math::conj (
            const unsigned int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

unsigned int

6.5.2.27 conj() [12/16]

```
unsigned long int Math::conj (
            const unsigned long int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

unsigned long int

6.5.2.28 conj() [13/16]

```
unsigned long long int Math::conj (
            const unsigned long long int & val )  [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

unsigned long long int

6.5.2.29 conj() [14/16]

```
unsigned short int Math::conj (
            const unsigned short int & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

unsigned short int

6.5.2.30 conj() [15/16]

```
template<typename K >
Vector< K > Math::conj (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.31 conj() [16/16]

```
wchar_t Math::conj (
            const wchar_t & val ) [inline]
```

This function exists to make the definition of the scalar product more general. It just returns *val* .

Parameters

val	

Returns

wchar_t

6.5.2.32 copysign() [1/2]

```
template<typename K >
Vector< K > Math::copysign (
            const Vector< K > & x,
            const K y )
```

Parameters

x	
y	

Returns

Vector<K>

6.5.2.33 copysign() [2/2]

```
template<typename K >
Vector< K > Math::copysign (
            const Vector< K > & x,
            const Vector< K > & y )
```

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

6.5.2.34 cos() [1/2]

```
template<typename K >
DualNumber< K > Math::cos (
            const DualNumber< K > & x )
```

Returns cos(x.a) - x.b*sin(x.a)

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.35 cos() [2/2]

```
template<typename K >
Vector< K > Math::cos (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.36 cosh() [1/2]

```
template<typename K >
DualNumber< K > Math::cosh (
            const DualNumber< K > & x )
```

Returns cosh(x.a) + sinh(x.a)∗x.b∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.37 cosh() [2/2]

```
template<typename K >
Vector< K > Math::cosh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.38 erf()

```
template<typename K >
Vector< K > Math::erf (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.39 erfc()

```
template<typename K >
Vector< K > Math::erfc (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.40 exp()

```
template<typename K >
Vector< K > Math::exp (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.41 exp2()

```
template<typename K >
Vector< K > Math::exp2 (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.42 expm1()

```
template<typename K >
Vector< K > Math::expm1 (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.43 fabs()

```
template<typename K >
Vector< double > Math::fabs (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.44 fdim() [1/3]

```
template<typename K >
Vector< K > Math::fdim (
            const K x,
            const Vector< K > & y )
```

Parameters

x	
y	

Returns

Vector<K>

6.5.2.45 fdim() [2/3]

```
template<typename K >
Vector< K > Math::fdim (
            const Vector< K > & x,
            const Vector< K > & y )
```

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

6.5.2.46 fdim() [3/3]

```
template<typename K >
Vector< K > Math::fdim (
            const Vector< K > & x,
            K y )
```

Parameters

x	
y	

Returns

Vector<K>

6.5.2.47 floor()

```
template<typename K >
Vector< K > Math::floor (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.48 fma() [1/7]

```
template<typename K >
Vector< K > Math::fma (
            const K x,
            const K y,
            const Vector< K > & z )
```

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.49 fma() [2/7]

```
template<typename K >
Vector< K > Math::fma (
            const K x,
            const Vector< K > & y,
            const K z )
```

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.50 fma() [3/7]

```
template<typename K >
Vector< K > Math::fma (
            const K x,
            const Vector< K > & y,
            const Vector< K > & z )
```

Exceptions

Throws	an exception if the length of *y* does not equal the length of *z* .

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.51 fma() [4/7]

```
template<typename K >
Vector< K > Math::fma (
            const Vector< K > & x,
            const K y,
            const K z )
```

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.52 fma() [5/7]

```
template<typename K >
Vector< K > Math::fma (
            const Vector< K > & x,
            const K y,
            const Vector< K > & z )
```

Exceptions

Throws	an exception if the length of *x* does not equal the length of *z* .

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.53 fma() [6/7]

```
template<typename K >
Vector< K > Math::fma (
            const Vector< K > & x,
            const Vector< K > & y,
            const K z )
```

Exceptions

Throws	an exception if the length of *x* does not equal the length of *y* .

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.54 fma() [7/7]

```
template<typename K >
Vector< K > Math::fma (
            const Vector< K > & x,
            const Vector< K > & y,
            const Vector< K > & z )
```

Exceptions

Throws	and exception if the lenghts of *x* , *y* and *z* are not equal.

Parameters

x	
y	
z	

Returns

Vector<K>

6.5.2.55 fmod() [1/3]

```
template<typename K >
Vector< K > Math::fmod (
            const K numer,
            const Vector< K > & denom )
```

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.56 fmod() [2/3]

```
template<typename K >
Vector< K > Math::fmod (
            const Vector< K > & numer,
            const K denom )
```

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.57 fmod() [3/3]

```
template<typename K >
Vector< K > Math::fmod (
            const Vector< K > & numer,
            const Vector< K > & denom )
```

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.58 fpclassify()

```
template<typename K >
Vector< int > Math::fpclassify (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<int>

6.5.2.59 frexp()

```
template<typename K >
Vector< K > Math::frexp (
            const Vector< K > & v,
            Vector< int > * exp )
```

Parameters

v	
exp	

Returns

Vector<K>

6.5.2.60 hypot() [1/3]

```
template<typename K >
Vector< K > Math::hypot (
            const K x,
            const Vector< K > & y )
```

Parameters

x	
y	

Returns

Vector<K>

6.5.2.61 hypot() [2/3]

```
template<typename K >
Vector< K > Math::hypot (
            const Vector< K > & x,
            const K y )
```

Parameters

x	
y	

Returns

Vector<K>

6.5.2.62 hypot() [3/3]

```
template<typename K >
Vector< K > Math::hypot (
            const Vector< K > & x,
            const Vector< K > & y )
```

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

6.5.2.63 ilogb()

```
template<typename K >
Vector< K > Math::ilogb (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.64 isfinite()

```
template<typename K >
bool Math::isfinite (
            const Vector< K > & v )
```

Parameters

v	

Returns

true

false

6.5.2.65 isinf()

```
template<typename K >
bool Math::isinf (
            const Vector< K > & v )
```

Parameters

v	

Returns

true

false

6.5.2.66 isnan()

```
template<typename K >
bool Math::isnan (
            const Vector< K > & v )
```

Parameters

v	

Returns

true

false

6.5.2.67 isnormal()

```
template<typename K >
bool Math::isnormal (
            const Vector< K > & v )
```

Parameters

v	

Returns

true

false

6.5.2.68 ldexp() [1/2]

```
template<typename K >
Vector< K > Math::ldexp (
            const Vector< K > & v,
            const int exp )
```

Parameters

v	
exp	

Returns

Vector<K>

6.5.2.69 ldexp() [2/2]

```
template<typename K >
Vector< K > Math::ldexp (
            const Vector< K > & v,
            const Vector< int > & exp )
```

Parameters

v	
exp	

Returns

Vector<K>

6.5.2.70 lgamma()

```
template<typename K >
Vector< K > Math::lgamma (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.71 llrint()

```
template<typename K >
Vector< long long int > Math::llrint (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<long long int>

6.5.2.72 llround()

```
template<typename K >
Vector< long long int > Math::llround (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<long long int>

6.5.2.73 log()

```
template<typename K >
Vector< K > Math::log (
            const Vector< K > & v )
```

Exceptions

Throws	an exception if *v* and *exp* have different lengths.

Parameters

v	

Returns

Vector<K>

6.5.2.74 log10()

```
template<typename K >
Vector< K > Math::log10 (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.75 log1p()

```
template<typename K >
Vector< K > Math::log1p (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.76 log2()

```
template<typename K >
Vector< K > Math::log2 (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.77 logb()

```
template<typename K >
Vector< K > Math::logb (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.78 lrint()

```
template<typename K >
Vector< long int > Math::lrint (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<long int>

6.5.2.79 lround()

```
template<typename K >
Vector< long int > Math::lround (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<long int>

6.5.2.80 max()

```
template<typename K >
K Math::max (
            const Vector< K > & v )
```

Parameters

v	

Returns

K

6.5.2.81 min()

```
template<typename K >
K Math::min (
            const Vector< K > & v )
```

Parameters

v	

Returns

K

6.5.2.82 modf()

```
template<typename K >
Vector< K > Math::modf (
            const Vector< K > & v,
            Vector< K > * intpart )
```

Parameters

v	
intpart	

Returns

Vector<K>

6.5.2.83 multiply()

```
template<typename K >
K Math::multiply (
            const Vector< K > & v )
```

Parameters

v	

Returns

K

6.5.2.84 nan()

```
template Vector< unsigned int > Math::nan (
            const unsigned int N,
            const char * tagp )
```

Parameters

N	
tagp	

Returns

Vector<K>

6.5.2.85 nearbyint()

```
template<typename K >
Vector< K > Math::nearbyint (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.86 nextafter()

```
template<typename K >
Vector< K > Math::nextafter (
            const Vector< K > & x,
            const Vector< K > & y )
```

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

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

```
template<typename K >
bool Math::operator!= (
            const K & alpha,
            const Vector< K > & v )
```

Parameters

alpha	
v	

Returns

true

false

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

```
template<typename K >
bool Math::operator!= (
            const Vector< K > & v,
            const K & alpha )
```

Parameters

v	
alpha	

Returns

true

false

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

```
template<typename K >
bool Math::operator!= (
            const Vector< K > & v,
            const Vector< K > & w )
```

Parameters

v	
w	

Returns

true

false

6.5.2.90 operator&() [1/3]

```
template<typename K >
Vector< K > Math::operator& (
            const K & v,
            const Vector< K > & w )
```

Creates a new Vector whose components are the scalar *v* followed by the components of *w* .

Parameters

v	
w	

Returns

Vector<K>

6.5.2.91 operator&() [2/3]

```
template<typename K >
Vector< K > Math::operator& (
            const Vector< K > & v,
            const K & w )
```

Creates a new Vector whose components are the components of *v* followed by the scalar *w* .

Parameters

v	
w	

Returns

Vector<K>

6.5.2.92 operator&() [3/3]

```
template<typename K >
Vector< K > Math::operator& (
            const Vector< K > & v,
            const Vector< K > & w )
```

Creates a new Vector whose components are the components of *v* followed by the components of *w* .

Parameters

v	
w	

Returns

Vector<K>

6.5.2.93 operator*() [1/6]

```
template<typename K >
DualNumber< K > Math::operator* (
            const DualNumber< K > & x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.94 operator*() [2/6]

```
template<typename K >
DualNumber< K > Math::operator* (
            const DualNumber< K > & x,
            const K y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.95 operator*() [3/6]

```
template<typename K >
Vector< K > Math::operator* (
            const K & alpha,
            const Vector< K > & v )
```

v ∗ *alpha* is equivalent to *v[i]* ∗ *alpha* for all *i* .

Parameters

alpha	
v	

Returns

Vector<K>

6.5.2.96 operator*() [4/6]

```
template<typename K >
DualNumber< K > Math::operator* (
            const K x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.97 operator*() [5/6]

```
template<typename K >
Vector< K > Math::operator* (
            const Vector< K > & v,
            const K & alpha )
```

alpha ∗ *v* is equivalent to *alpha* ∗ *v[i]* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

6.5.2.98 operator*() [6/6]

```
template<typename K >
Vector< K > Math::operator* (
```

```
const Vector< K > & v,
const Vector< K > & w )
```

v ∗ *w* is equivalent to *v[i]* ∗ *w[i]* for all *i* .

Exceptions

An	exception is thrown if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

Vector<K>

6.5.2.99 operator+() [1/6]

```
template<typename K >
DualNumber< K > Math::operator+ (
            const DualNumber< K > & x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.100 operator+() [2/6]

```
template<typename K >
DualNumber< K > Math::operator+ (
            const DualNumber< K > & x,
            const K y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.101 operator+() [3/6]

```
template<typename K >
Vector< K > Math::operator+ (
            const K & alpha,
            const Vector< K > & w )
```

alpha + v is equivalent to *alpha + v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

6.5.2.102 operator+() [4/6]

```
template<typename K >
DualNumber< K > Math::operator+ (
            const K x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.103 operator+() [5/6]

```
template<typename K >
Vector< K > Math::operator+ (
            const Vector< K > & v,
            const K & alpha )
```

v + alpha is equivalent to *v[i] + alpha* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

6.5.2.104 operator+() [6/6]

```
template<typename K >
Vector< K > Math::operator+ (
            const Vector< K > & v,
            const Vector< K > & w )
```

v + w is equivalent to *v[i] + w[i]* for all *i* . If any of the vectors has length zero, then the other one is returned.

Exceptions

A	runtime exception is thrown if both vectors have different number of components and their lengths are both non-zero.

Parameters

v	
w	

Returns

Vector<K>

6.5.2.105 operator-() [1/6]

```
template<typename K >
DualNumber< K > Math::operator- (
            const DualNumber< K > & x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.106 operator-() [2/6]

```
template<typename K >
DualNumber< K > Math::operator- (
            const DualNumber< K > & x,
            const K y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.107 operator-() [3/6]

```
template<typename K >
Vector< K > Math::operator- (
            const K & alpha,
            const Vector< K > & w )
```

alpha - *v* is equivalent to *alpha* - *v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

6.5.2.108 operator-() [4/6]

```
template<typename K >
DualNumber< K > Math::operator- (
            const K x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.109 operator-() [5/6]

```
template<typename K >
Vector< K > Math::operator- (
            const Vector< K > & v,
            const K & alpha )
```

v - alpha is equivalent to *v[i] - alpha* for all *i*.

Parameters

v	
alpha	

Returns

Vector<K>

6.5.2.110 operator-() [6/6]

```
template<typename K >
Vector< K > Math::operator- (
            const Vector< K > & v,
            const Vector< K > & w )
```

v - w is equivalent to *v[i] - w[i]* for all *i* . If *v* has null length, then *-w* is returned. If *w* has length zero, then *v* is returned.

Exceptions

A	runtime exception is thrown if both vectors have different number of components and their lengths are both non-zero.

Parameters

v	

Parameters

w	

Returns

Vector<K>

6.5.2.111 operator/() [1/6]

```
template<typename K >
DualNumber< K > Math::operator/ (
             const DualNumber< K > & x,
             const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.112 operator/() [2/6]

```
template<typename K >
DualNumber< K > Math::operator/ (
             const DualNumber< K > & x,
             const K y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.113 operator/() [3/6]

```
template<typename K >
Vector< K > Math::operator/ (
            const K & alpha,
            const Vector< K > & w )
```

alpha / v is equivalent to *alpha / v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

6.5.2.114 operator/() [4/6]

```
template<typename K >
DualNumber< K > Math::operator/ (
            const K x,
            const DualNumber< K > & y )
```

Parameters

x	
y	

Returns

DualNumber<K>

6.5.2.115 operator/() [5/6]

```
template<typename K >
Vector< K > Math::operator/ (
            const Vector< K > & v,
            const K & alpha )
```

v / alpha is equivalent to *v[i] / alpha* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

6.5.2.116 operator/() [6/6]

```
template<typename K >
Vector< K > Math::operator/ (
            const Vector< K > & v,
            const Vector< K > & w )
```

v / *w* is equivalent to *v[i]* / *w[i]* for all *i* .

Exceptions

An	exception is thrown if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

Vector<K>

6.5.2.117 operator==() [1/3]

```
template<typename K >
bool Math::operator== (
            const K & alpha,
            const Vector< K > & v )
```

Parameters

alpha	
v	

Returns

true

false

6.5.2.118 operator==() [2/3]

```
template<typename K >
bool Math::operator== (
            const Vector< K > & v,
            const K & alpha )
```

Parameters

v	
alpha	

Returns

true

false

6.5.2.119 operator==() [3/3]

```
template<typename K >
bool Math::operator== (
            const Vector< K > & v,
            const Vector< K > & w )
```

Parameters

v	
w	

Returns

true

false

6.5.2.120 operator>() [1/3]

```
template<typename K >
bool Math::operator> (
            const K & alpha,
            const Vector< K > & v )
```

Parameters

alpha	
v	

Returns

true

false

6.5.2.121 operator>() [2/3]

```
template<typename K >
bool Math::operator> (
            const Vector< K > & v,
            const K & alpha )
```

Parameters

v	
alpha	

Returns

true

false

6.5.2.122 operator>() [3/3]

```
template<typename K >
bool Math::operator> (
            const Vector< K > & v,
            const Vector< K > & w )
```

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

6.5.2.123 operator>=() [1/3]

```
template<typename K >
bool Math::operator>= (
            const K & alpha,
            const Vector< K > & v )
```

Parameters

alpha	
v	

Returns

true

false

6.5.2.124 operator>=() [2/3]

```
template<typename K >
bool Math::operator>= (
            const Vector< K > & v,
            const K & alpha )
```

Parameters

v	
alpha	

Returns

true

false

6.5.2.125 operator>=() [3/3]

```
template<typename K >
bool Math::operator>= (
            const Vector< K > & v,
            const Vector< K > & w )
```

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

6.5.2.126 operator"|()

```
template<typename K >
K Math::operator| (
            const Vector< K > & v,
            const Vector< K > & w )
```

Performs the following operation:

$$(v|w) = \sum_{i=0}^{N-1} v_i^* w_i$$

which is the canonical scalar product of $\mathbb{K}^N$.

Parameters

v	
w	

Returns

K

6.5.2.127 pow() [1/3]

```
template<typename K >
Vector< K > Math::pow (
            const K v,
            const Vector< K > & exponent )
```

Parameters

v	
exponent	

Returns

Vector<K>

6.5.2.128 pow() [2/3]

```
template<typename K >
Vector< K > Math::pow (
            const Vector< K > & v,
            const K exponent )
```

Parameters

v	
exponent	

Returns

Vector<K>

6.5.2.129 pow() [3/3]

```
template<typename K >
Vector< K > Math::pow (
            const Vector< K > & v,
            const Vector< K > & exponent )
```

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

v	
exponent	

Returns

Vector<K>

6.5.2.130 remainder() [1/3]

```
template<typename K >
Vector< K > Math::remainder (
```

```
            const K numer,
            const Vector< K > & denom )
```

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.131 remainder() [2/3]

```
template<typename K >
Vector< K > Math::remainder (
            const Vector< K > & numer,
            const K denom )
```

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.132 remainder() [3/3]

```
template<typename K >
Vector< K > Math::remainder (
            const Vector< K > & numer,
            const Vector< K > & denom )
```

Exceptions

Throws	an exception if *numer* and *denom* have different lengths.

Parameters

numer	
denom	

Returns

Vector<K>

6.5.2.133 remquo() [1/3]

```
template<typename K >
Vector< K > Math::remquo (
            const K numer,
            const Vector< K > & denom,
            Vector< int > * quot )
```

Parameters

numer	
denom	
quot	

Returns

Vector<K>

6.5.2.134 remquo() [2/3]

```
template<typename K >
Vector< K > Math::remquo (
            const Vector< K > & numer,
            const K denom,
            Vector< int > * quot )
```

Parameters

numer	
denom	
quot	

Returns

Vector<K>

6.5.2.135 remquo() [3/3]

```
template<typename K >
Vector< K > Math::remquo (
```

```
            const Vector< K > & numer,
            const Vector< K > & denom,
            Vector< int > * quot )
```

Exceptions

Throws	an exception if *numer* and *denom* have different lengths.

Parameters

numer	
denom	
quot	

Returns

Vector<K>

6.5.2.136 rint()

```
template<typename K >
Vector< K > Math::rint (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.137 round()

```
template<typename K >
Vector< K > Math::round (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.138 scalbln() [1/2]

```
template<typename K >
Vector< K > Math::scalbln (
            const Vector< K > & v,
            const long int n )
```

Parameters

v	
n	

Returns

Vector<K>

6.5.2.139 scalbln() [2/2]

```
template<typename K >
Vector< K > Math::scalbln (
            const Vector< K > & v,
            const Vector< long int > & n )
```

Exceptions

Throws	an expception if *v* and *n* have different lenghts.

Parameters

v	
n	

Returns

Vector<K>

6.5.2.140 scalbn() [1/2]

```
template<typename K >
Vector< K > Math::scalbn (
```

```
            const Vector< K > & v,
            const int n )
```

Parameters

v	
n	

Returns

Vector<K>

6.5.2.141 scalbn() [2/2]

```
template<typename K >
Vector< K > Math::scalbn (
            const Vector< K > & v,
            const Vector< int > & n )
```

Exceptions

Throws	an expception if *v* and *n* have different lenghts.

Parameters

v	
n	

Returns

Vector<K>

6.5.2.142 sin() [1/2]

```
template<typename K >
DualNumber< K > Math::sin (
            const DualNumber< K > & x )
```

Returns sin(x.a) + x.b*cos(x.a)

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.143 sin() [2/2]

```
template<typename K >
Vector< K > Math::sin (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.144 sinh() [1/2]

```
template<typename K >
DualNumber< K > Math::sinh (
            const DualNumber< K > & x )
```

Returns sinh(x.a) + cosh(x.a)∗x.b∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.145 sinh() [2/2]

```
template<typename K >
Vector< K > Math::sinh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.146 sqrt()

```
template<typename K >
Vector< K > Math::sqrt (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.147 sum()

```
template<typename K >
K Math::sum (
            const Vector< K > & v )
```

Parameters

v	

Returns

K

6.5.2.148 tan() [1/2]

```
template<typename K >
DualNumber< K > Math::tan (
            const DualNumber< K > & x )
```

Returns tan(x.a) + x.b/cos(x.a)/cos(x.a)∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.149 tan() [2/2]

```
template<typename K >
Vector< K > Math::tan (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.150 tanh() [1/2]

```
template<typename K >
DualNumber< K > Math::tanh (
            const DualNumber< K > & x )
```

Returns tanh(x.a) + 1/cosh(x.a)/cosh(x.a)∗x.b∗

Template Parameters

K	

Parameters

x	

Returns

DualNumber<K>

6.5.2.151 tanh() [2/2]

```
template<typename K >
Vector< K > Math::tanh (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.152 tgamma()

```
template<typename K >
Vector< K > Math::tgamma (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.153 trunc()

```
template<typename K >
Vector< K > Math::trunc (
            const Vector< K > & v )
```

Parameters

v	

Returns

Vector<K>

6.5.2.154 vector_product_2d()

```
template<typename K >
K Math::vector_product_2d (
            const Vector< K > & v,
            const Vector< K > & w )
```

Exceptions

Throws	an exception if either the dimension of *v* or the dimension of *w* is different from two.

Parameters

v	
w	

Returns

K

6.5.2.155 vector_product_3d()

```
template<typename K >
Vector< K > Math::vector_product_3d (
            const Vector< K > & v,
            const Vector< K > & w )
```

Exceptions

Throws	an exception if either the dimension of *v* or the dimension of *w* is different from three.

Parameters

v	
w	

Returns

Vector<K>

6.6 Math::AlgebraicSolvers Namespace Reference

Contains solvers for algebraic equations.

Classes

- class LinearSystemSolver

    Solves linear systems of the form Ax=b through LU decomposition with partial pivoting.

Functions

- double bisection (std::function< double(const double x)> f, double a, double b, const unsigned int iter_max, const double abs_tol, const double rel_tol)

    Applies the bisection method to the function f in the interval (a,b).
- double secant (std::function< double(const double x)> f, double x1, double x2, const unsigned int iter_max, const double abs_tol, const double rel_tol)

    Applies the secant method to the function f with initial guesses x1 and x2.
- double newton_raphson (std::function< double(const double x)>f, std::function< double(const double x)>dfdx, const double x0, const unsigned int iter_max, const double abs_tol, const double rel_tol)

    Applies the Newton-Raphson method to the function f with initial guess x0.
- double newton_raphson (std::function< DualNumber< double >(const DualNumber< double > x)>f, const double x0, const unsigned int iter_max, const double abs_tol, const double rel_tol)

    Applies the Newton-Raphson method to the function f with initial guess x0.

6.6.1 Detailed Description

Contains solvers for algebraic equations.

6.6.2 Function Documentation

6.6.2.1 bisection()

```
double Math::AlgebraicSolvers::bisection (
            std::function< double(const double x)> f,
            double a,
            double b,
            const unsigned int iter_max,
            const double abs_tol,
            const double rel_tol )
```

Applies the bisection method to the function f in the interval (a,b).

Returns the current estimation to the solution as soon as one of the following things happens:

1. The maximun number of iterations is reached.
2. The absolute error is smaller then the tolerance given.
3. The relative error is smaller then the tolerance given.

To disable either the absolute or relative error check, set the corresponding tolerance to zero.

Parameters

f	the function.
a	the left side of the interval.
b	the right side of the interval.
iter_max	the maximum number of iterations.
abs_tol	the tolerance for the absolute error.
rel_tol	the tolerance for the relative error.

Returns

double

6.6.2.2 newton_raphson() [1/2]

```
double Math::AlgebraicSolvers::newton_raphson (
            std::function< double(const double x)> f,
            std::function< double(const double x)> dfdx,
            const double x0,
            const unsigned int iter_max,
            const double abs_tol,
            const double rel_tol )
```

Applies the Newton-Raphson method to the function f with initial guess x0.

Returns the current estimation to the solution as soon as one of the following things happens:

1. The maximun number of iterations is reached.
2. The absolute error is smaller then the tolerance given.
3. The relative error is smaller then the tolerance given.

To disable either the absolute or relative error check, set the corresponding tolerance to zero.

Parameters

f	the function.
dfdx	the derivative of the function.
x0	the initial estimation.
iter_max	the maximum number of iterations.
abs_tol	the tolerance for the absolute error.
rel_tol	the tolerance for the relative error.

Returns

double

6.6.2.3 newton_raphson() [2/2]

```
double Math::AlgebraicSolvers::newton_raphson (
            std::function< DualNumber< double >(const DualNumber< double > x)> f,
            const double x0,
            const unsigned int iter_max,
            const double abs_tol,
            const double rel_tol )
```

Applies the Newton-Raphson method to the function f with initial guess x0.

Returns the current estimation to the solution as soon as one of the following things happens:

1. The maximun number of iterations is reached.
2. The absolute error is smaller then the tolerance given.
3. The relative error is smaller then the tolerance given.

To disable either the absolute or relative error check, set the corresponding tolerance to zero.

Parameters

f	the function (returning a dual number).
x0	the initial estimation.
iter_max	the maximum number of iterations.
abs_tol	the tolerance for the absolute error.
rel_tol	the tolerance for the relative error.

Returns

double

6.6.2.4 secant()

```
double Math::AlgebraicSolvers::secant (
            std::function< double(const double x)> f,
            double x1,
            double x2,
            const unsigned int iter_max,
            const double abs_tol,
            const double rel_tol )
```

Applies the secant method to the function f with initial guesses x1 and x2.

Returns the current estimation to the solution as soon as one of the following things happens:

1. The maximun number of iterations is reached.
2. The absolute error is smaller then the tolerance given.
3. The relative error is smaller then the tolerance given.

To disable either the absolute or relative error check, set the corresponding tolerance to zero.

Parameters

f	the function.
x1	the first initial estimation.
x2	the second initial estimation.
iter_max	the maximum number of iterations.
abs_tol	the tolerance for the absolute error.
rel_tol	the tolerance for the relative error.

Returns

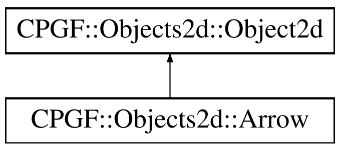
double

Chapter 7

Class Documentation

7.1 CPGF::Objects2d::Arrow Class Reference

Inheritance diagram for CPGF::Objects2d::Arrow:



Public Member Functions

- **Arrow** (const AffineSpace::Point2d &start, const AffineSpace::Point2d &end, const double arrow_head_length=0.15, const double arrow_head_width=0.3, const Color &color=Color::BLACK, const double opacity=1, const double line_width=LineWidth::SEMITHICK, const std::vector< double > &dash_pattern=DashPatterns::SOLID)

Static Protected Member Functions

- static Object2d **builder** (const AffineSpace::Point2d &start, const AffineSpace::Point2d &end, const double arrow_head_length, const double arrow_head_width, const Color &color, const double opacity, const double line_width, const std::vector< double > &dash_pattern)

Additional Inherited Members

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

- CPGF/Objects2d/BasicGeometries.hpp
- CPGF/Objects2d/BasicGeometries.cpp

7.2 Math::Interpolation::AverageLinearInterpolation Class Reference

Public Member Functions

- **AverageLinearInterpolation** (const std::vector< double > &averages, const std::vector< double > &partition)
- **AverageLinearInterpolation** (double *averages, double *partition, unsigned int N_cells)
- **AverageLinearInterpolation** (const AverageLinearInterpolation &f)
- AverageLinearInterpolation & **operator=** (const AverageLinearInterpolation &f)
- double operator() (const double x) const

  We do linear extrapolation at the ends.

Public Attributes

- unsigned int **N_cells**
- double * **x_part**
- double * **b**
- double * **c**

7.2.1 Member Function Documentation

7.2.1.1 operator()()

```
double AverageLinearInterpolation::operator() (
            const double x ) const
```

We do linear extrapolation at the ends.

Parameters

x	

Returns

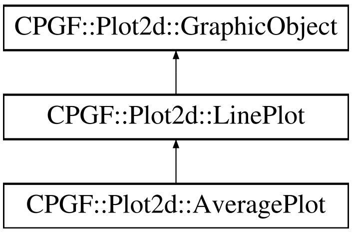
double

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

- Math/Interpolation.hpp
- Math/Interpolation.cpp

7.3 CPGF::Plot2d::AveragePlot Class Reference

Inheritance diagram for CPGF::Plot2d::AveragePlot:



Public Member Functions

- AveragePlot (const std::vector< double > &Y, const std::vector< double > &partition, const Color &color=Color::BLUE, const double line_width=LineWidth::THIN, const double opacity=1, const std::vector< double > &dash_pattern=DashPatterns::SOLID, const std::string &legend="")

  Allows to average values over a partition of the real line. Returns a LinePlot.
- **AveragePlot** (const std::vector< double > &Y, const std::vector< double > &partition, std::function< Color(unsigned int)> color, std::function< double(unsigned int)> line_width, std::function< double(unsigned int)> opacity, std::function< std::vector< double >(unsigned int)> dash_pattern, const std::string &legend="")
- **AveragePlot** (const std::vector< double > &Y, const std::vector< double > &partition, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> line_width, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::vector< double >(AffineSpace::Point2d &)> dash_pattern, const std::string &legend="")

Static Protected Member Functions

- static LinePlot **builder** (const std::vector< double > &Y, const std::vector< double > &partition, const Color &color, const double line_width, const double opacity, const std::vector< double > &dash_pattern, const std::string &legend)
- static LinePlot **builder** (const std::vector< double > &Y, const std::vector< double > &partition, std::function< Color(unsigned int)> color, std::function< double(unsigned int)> line_width, std::function< double(unsigned int)> opacity, std::function< std::vector< double >(unsigned int)> dash_pattern, const std::string &legend)
- static LinePlot **builder** (const std::vector< double > &Y, const std::vector< double > &partition, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> line_width, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::vector< double >(AffineSpace::Point2d &)> dash_pattern, const std::string &legend)

Additional Inherited Members

7.3.1 Constructor & Destructor Documentation

7.3.1.1 AveragePlot()

```
AveragePlot::AveragePlot (
            const std::vector< double > & Y,
            const std::vector< double > & partition,
            const Color & color = Color::BLUE,
            const double line_width = LineWidth::THIN,
            const double opacity = 1,
            const std::vector< double > & dash_pattern = DashPatterns::SOLID,
            const std::string & legend = "" )
```

Allows to average values over a partition of the real line. Returns a LinePlot.

Parameters

Y	
partition	
color	
line_width	
opacity	
dash_pattern	
legend	

Returns

LinePlot

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

- CPGF/Plot2d/LinePlot.hpp
- CPGF/Plot2d/LinePlot.cpp

7.4 Math::Interpolation::AverageQuadraticInterpolation Class Reference

Public Member Functions

- **AverageQuadraticInterpolation** (const std::vector< double > &averages, const std::vector< double > &partition)
- **AverageQuadraticInterpolation** (double ∗averages, double ∗partition, unsigned int N_cells)
- **AverageQuadraticInterpolation** (const AverageQuadraticInterpolation &f)
- AverageQuadraticInterpolation & **operator=** (const AverageQuadraticInterpolation &f)
- double operator() (const double x) const

  We do linear extrapolation at the ends.

Protected Attributes

- unsigned int **N_cells**
- double ∗ **x_part**
- double ∗ **a**
- double ∗ **b**
- double ∗ **c**

7.4.1 Member Function Documentation

7.4.1.1 operator()()

```
double AverageQuadraticInterpolation::operator() (
                const double x ) const
```

We do linear extrapolation at the ends.

Parameters

x	

Returns

double

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

- Math/Interpolation.hpp
- Math/Interpolation.cpp

7.5 CPGF::Plot2d::Axis Class Reference

Public Member Functions

- void **set_max_value** (const double value)
- void **reset_max_value** ()
- void **set_min_value** (const double value)
- void **reset_min_value** ()
- void **update_max_min** (const double max_value, const double min_value)
- void **calculate_transformations** ()
- double **get_min_value** () const
- double **get_max_value** () const
- std::function< double(double)> axis_transform () const

  Applied to a coordiante, it returns the coordinate on the final drawing.
- std::function< double(double)> **axis_transform_inverse** () const
- Scene2d **render_to_scene** () const
- **Axis** (const AxisType axis_type, const std::string &label="", const AxisScale axis_scale=AxisScale::LINEAR, const double scale=1, const bool visible=true, const bool inverted=false, const double position=0, const Color &color=Color::BLACK, const double line_width=LineWidth::THIN, const double opacity=1, const std::vector< double > &dash_pattern=DashPatterns::SOLID, const double aspect_ratio=0.6, const double arrow_head_length=0.3, const double arrow_head_width=0.75, const double arrow_length=0.5, std::function< Objects2d::Object2d(const AffineSpace::Point2d &start, const AffineSpace::Point2d &end, const double arrow_head_length, const double arrow_head_width, const Color &color, const double opacity, const double line_width, const std::vector< double > &dash_pattern)> arrow=[ ](const AffineSpace::Point2d &start, const AffineSpace::Point2d &end, const double arrow_head_length, const double arrow_head_width, const Color &color, const double opacity, const double line_width, const std::vector< double > &dash_pattern) {return Objects2d::Arrow(start, end, arrow_head_length, arrow_head_width, color, opacity, line_width, dash_pattern);}, const unsigned int N_major_ticks=9, const double major_tick_deviation=0.25, const double major_tick_line_width_divisor=3, const bool show_medium_ticks=true, const double medium_tick_deviation=0.20, const double medium_tick_line_width_divisor=4, const bool show_small_ticks=true, const double small_tick_deviation=0.15, const double small_tick_line_width_divisor=5, const bool show_numbers=true, const bool show_major_grid_lines=false, const std::vector< double > &major_grid_lines_dash_pattern=D Patterns::SOLID, const double major_grid_line_line_width_divisor=2, const double major_grid_line_opacity=0.75, const bool show_medium_grid_lines=false, const std::vector< double > &medium_grid_lines_dash_pattern=DashPatterns::SOLID, const double medium_grid_line_line_width_divisor=4, const double medium_grid_line_opacity=0.5, const bool show_small_grid_lines=false, const std::vector< double > &small_grid_lines_dash_pattern=DashPatterns::SOLID, const double small_grid_line_line_width_divisor=8, const double small_grid_line_opacity=0.25)

Public Attributes

- AxisType **axis_type**

    Determines whether the axis is horizontal or vertical.
- AxisScale **axis_scale**

    Represents the scale of the axis.
- double **scale**

    Determines the relative size between the graph and the labels. Use scale=1 for full-paged graphics and scale=0.5 for half-paged graphics.
- bool **visible**

    Determines whether the axis is rendered or not.
- bool **inverted**

    Controlls whether the axis is inverted, a.k.a. points in the opposite direction.
- std::string **label**

    The axis label.
- double **position**

    A real number between zero and one which represents the position of the axis on the graph.
- Color **color**

    Color of the axis.
- double **line_width**

    Axis line width.
- double **opacity**

    Axis opacity.
- std::vector< double > **dash_pattern**

    Axis dash pattern.
- double aspect_ratio

    Determines the length of the axis if it is vertical. To obtain the axis length, we have to multiply the aspect_ratio by the default length of all X axes.
- double **arrow_head_length**

    Arrow head length.
- double **arrow_head_width**

    Arrow width.
- double **arrow_length**

    Determines how much the arrow sticks out of the axis.
- std::function< Objects2d::Object2d(const AffineSpace::Point2d &start, const AffineSpace::Point2d &end, const double arrow_head_length, const double arrow_head_width, const Color &color, const double opacity, const double line_width, const std::vector< double > dash_pattern)> **arrow**

    Function that creates the arrow head.
- unsigned int N_major_ticks

    Number of major ticks.
- double **major_tick_deviation**

    Controlls how large the major ticks are in the perpendicular direction to the axis.
- double **major_tick_line_width_divisor**

    The width of the major ticks will be the axis witdh divided by this factor.
- bool **show_medium_ticks**

    Controlls whether medium ticks are displayed.
- double **medium_tick_deviation**

    Controlls how large the medium ticks are in the perpendicular direction to the axis.
- double **medium_tick_line_width_divisor**

    The width of the medium ticks will be the axis witdh divided by this factor.
- bool **show_small_ticks**

    Controlls whether small ticks are displayed.

- double **small_tick_deviation**

  Controlls how large the small ticks are in the perpendicular direction to the axis.
- double **small_tick_line_width_divisor**

  The width of the small ticks will be the axis witdh divided by this factor.
- bool **show_numbers**

  Controlls whether the number scale is displayed.
- NumberPosition **number_position**

  Determines the position of the numbers.
- bool **show_major_grid_lines**

  Determines whether the major grid lines are drawn.
- std::vector$<$ double $>$ **major_grid_lines_dash_pattern**

  The dash pattern of the major grid lines.
- double **major_grid_line_line_width_divisor**

  The width of the major grid lines will be the axis width divided by this factor.
- double **major_grid_line_opacity**

  The opacity of the major grid lines.
- bool **show_medium_grid_lines**

  Determines whether the medium grid lines are drawn.
- std::vector$<$ double $>$ **medium_grid_lines_dash_pattern**

  The dash pattern of the medium grid lines.
- double **medium_grid_line_line_width_divisor**

  The width of the medium grid lines will be the axis width divided by this factor.
- double **medium_grid_line_opacity**

  The opacity of the medium grid lines.
- bool **show_small_grid_lines**

  Determines whether the small grid lines are drawn.
- std::vector$<$ double $>$ **small_grid_lines_dash_pattern**

  The dash pattern of the medium grid lines.
- double **small_grid_line_line_width_divisor**

  The width of the small grid lines will be the axis width divided by this factor.
- double **small_grid_line_opacity**

  The opacity of the small grid lines.

Static Public Attributes

- constexpr static const double **X_MAX** = 15

  End position of the axis (if it is horizontal) on the final drawing.
- constexpr static const double **X_MIN** = 0

  Start position of the axis (if it is horizontal) on the final drawing.
- constexpr static const double **Y_MIN** = 0

  Start position of the axis (if it is vertical) on the final drawing.
- constexpr static const double **NUMBER_DISPLACEMENT** = 0.5

  Determines how much the numbers are displaced with respect to the axis.
- constexpr static const double **LABEL_DISPLACEMENT_HORIZONTAL** = 1.2

  Determines how much the label is displaced with respect to the axis.
- constexpr static const double **LABEL_DISPLACEMENT_VERTICAL** = 2

Protected Attributes

- bool **user_defined_max_value**
- double **max_value**
- bool **user_defined_min_value**
- double **min_value**
- double **x_max**
- double **_a**
- double **_b**
- int **digit_max**
- unsigned int **precision**

7.5.1 Member Function Documentation

7.5.1.1 axis_transform()

```
std::function< double(double)> Axis::axis_transform ( ) const
```

Applied to a coordiante, it returns the coordinate on the final drawing.

Returns

std::function<double(double)>

7.5.2 Member Data Documentation

7.5.2.1 aspect_ratio

```
double CPGF::Plot2d::Axis::aspect_ratio
```

Determines the length of the axis if it is vertical. To obtain the axis length, we have to multiply the aspect_ratio by the default length of all X axes.

Default value is 1/sqrt(2).

7.5.2.2 N_major_ticks

```
unsigned int CPGF::Plot2d::Axis::N_major_ticks
```

Number of major ticks.

Set to 0 to disable.

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

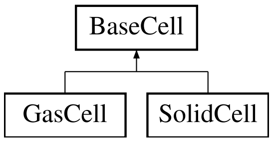
- CPGF/Plot2d/Axis.hpp
- CPGF/Plot2d/Axis.cpp

7.6 BaseCell Class Reference

The basic structure of the space discretization.

```
#include <Cell.hpp>
```

Inheritance diagram for BaseCell:



Public Member Functions

- BaseCell (const Math::Vector< double > &U, const double a, const double b, const double A, Chemistry::SolidGasReaction ∗QR)

  Construct a new Base Cell object.
- BaseCell (FILE ∗file, Chemistry::SolidGasReaction ∗QR)

  Construct a new Cell object from file.
- double x () const

  Returns cell average position.
- double len () const

  Returns cell length.
- void read_from_file (FILE ∗file)

  Reads the cell values from file.
- void write_to_file (FILE ∗file) const

  Writes the cell values to file.
- std::string to_string () const

  Returns a string representation of the cell.

Public Attributes

- Math::Vector< double > **U**

  Vector of cell conserved variables.
- double **a**

  Left limit of the cell.
- double **b**

  Right limit of the cell.
- double **A**

  Area of the cell.
- Chemistry::SolidGasReaction ∗ **QR**

  Pointer to the chemical reaction.

7.6.1 Detailed Description

The basic structure of the space discretization.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 BaseCell() [1/2]

```
BaseCell::BaseCell (
            const Math::Vector< double > & U,
            const double a,
            const double b,
            const double A,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Base Cell object.

Parameters

U	the vector of preserved variables.
a	the left limit of the cell.
b	the right limit of the cell.
A	the area of the cell.
QR	a pointer to the chemical reaction.

7.6.2.2 BaseCell() [2/2]

```
BaseCell::BaseCell (
            FILE * file,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Cell object from file.

Parameters

file	

7.6.3 Member Function Documentation

7.6.3.1 len()

```
double BaseCell::len ( ) const
```

Returns cell length.

Returns

double

7.6.3.2 read_from_file()

```
void BaseCell::read_from_file (
            FILE * file )
```

Reads the cell values from file.

Parameters

file	

7.6.3.3 to_string()

```
std::string BaseCell::to_string ( ) const
```

Returns a string representation of the cell.

Returns

std::string

7.6.3.4 write_to_file()

```
void BaseCell::write_to_file (
            FILE * file ) const
```

Writes the cell values to file.

Parameters

file	

7.6.3.5 x()

```
double BaseCell::x ( ) const
```

Returns cell average position.

Returns

double

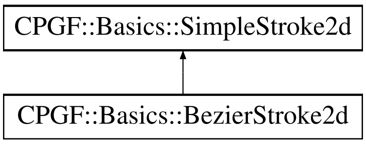
The documentation for this class was generated from the following files:

- Mesh/Cell.hpp
- Mesh/Cell.cpp

7.7 CPGF::Basics::BezierStroke2d Class Reference

```
#include <Strokes2d.hpp>
```

Inheritance diagram for CPGF::Basics::BezierStroke2d:



Public Member Functions

- **BezierStroke2d** (const AffineSpace::Point2d &P1, const AffineSpace::Point2d &P2, const AffineSpace::Point2d &Q1, const AffineSpace::Point2d &Q2)
- BezierStroke2d ∗ clone () const override
- AffineSpace::Point2d & start () override
- AffineSpace::Point2d start () const override
- AffineSpace::Point2d & end () override
- AffineSpace::Point2d end () const override
- BezierStroke2d & translate (const AffineSpace::Vector2d &v) override
- BezierStroke2d & rotate_with_respect_to (const AffineSpace::Point2d &Q, const double theta) override
- BezierStroke2d & scale_with_respect_to (const AffineSpace::Point2d &Q, const AffineSpace::Vector2d &s) override
- double length () const override
- double area () const override
- std::vector< AffineSpace::Point2d > operator/ (const SimpleStroke2d &B) override
- std::string render_to_string () const override

Public Attributes

- AffineSpace::Point2d P1
- AffineSpace::Point2d P2
- AffineSpace::Point2d Q1
- AffineSpace::Point2d Q2

7.7.1 Detailed Description

Todo Finish implementation.

7.7.2 Member Function Documentation

7.7.2.1 area()

`double BezierStroke2d::area ( ) const [override], [virtual]`

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.2 clone()

`BezierStroke2d * BezierStroke2d::clone ( ) const [override], [virtual]`

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.3 end() [1/2]

`Point2d BezierStroke2d::end ( ) const [override], [virtual]`

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.4 end() [2/2]

`Point2d & BezierStroke2d::end ( ) [override], [virtual]`

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.5 length()

`double BezierStroke2d::length ( ) const [override], [virtual]`

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.6 operator/()

```
std::vector< Point2d > BezierStroke2d::operator/ (
             const SimpleStroke2d & B ) [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.7 render_to_string()

`std::string BezierStroke2d::render_to_string ( ) const [override], [virtual]`

Parameters

alpha	a number between zero and one.

Returns

AffineSpace::Point2d

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.8 rotate_with_respect_to()

```
BezierStroke2d & BezierStroke2d::rotate_with_respect_to (
            const AffineSpace::Point2d & Q,
            const double theta )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.9 scale_with_respect_to()

```
BezierStroke2d & BezierStroke2d::scale_with_respect_to (
            const AffineSpace::Point2d & Q,
            const AffineSpace::Vector2d & s )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.10 start() [1/2]

```
Point2d BezierStroke2d::start ( ) const  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.11 start() [2/2]

```
Point2d & BezierStroke2d::start ( )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.2.12 translate()

```
BezierStroke2d & BezierStroke2d::translate (
            const AffineSpace::Vector2d & v )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.7.3 Member Data Documentation

7.7.3.1 P1

```
AffineSpace::Point2d CPGF::Basics::BezierStroke2d::P1
```

Starting point.

7.7.3.2 P2

```
AffineSpace::Point2d CPGF::Basics::BezierStroke2d::P2
```

Ending point

7.7.3.3 Q1

```
AffineSpace::Point2d CPGF::Basics::BezierStroke2d::Q1
```

First control point.

7.7.3.4 Q2

```
AffineSpace::Point2d CPGF::Basics::BezierStroke2d::Q2
```

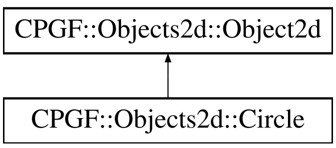
Second control point.

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

- CPGF/PGFBasics/Strokes2d.hpp
- CPGF/PGFBasics/Strokes2d.cpp

7.8 CPGF::Objects2d::Circle Class Reference

Inheritance diagram for CPGF::Objects2d::Circle:



Public Member Functions

- **Circle** (const AffineSpace::Point2d &pos, const double radius, const bool draw=true, const bool fill=false, const Color &draw_color=Color::BLACK, const Color &fill_color=Color::WHITE, const double opacity=1, const double line_width=LineWidth::SEMITHICK, const std::vector< double > &dash_pattern=DashPatterns::↩ SOLID)
- AffineSpace::Point2d **center** () const
- double **radius** () const

Static Protected Member Functions

- static Object2d **builder** (const AffineSpace::Point2d &pos, const double radius, const bool draw, const bool fill, const Color &draw_color, const Color &fill_color, const double opacity, const double line_width, const std::vector< double > &dash_pattern)

Additional Inherited Members

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

- CPGF/Objects2d/BasicGeometries.hpp
- CPGF/Objects2d/BasicGeometries.cpp

7.9 CPGF::Color Class Reference

An object used to represent an rgb color. r, g and b must all be real numbers between 0 and 1.

```
#include <PGFConf.hpp>
```

Public Member Functions

- std::string to_string ()

    Returns a string representation of the color.
- **Color** ()

    Returns the red color (1,0,0).
- Color (const double r, const double g, const double b)

    Returns the (r,g,b) color.

Static Public Member Functions

- static Color mix (const Color &A, const Color &B, const double alpha)

    Returns a mix of colors A and B in proportions alpha and 1 - alpha.
- static Color **from_RGB** (const unsigned char R, const unsigned char G, const unsigned char B)

Public Attributes

- double **r**

  The amount of red of the color. $0 \leq r \leq 1.$
- double **g**

  The amount of green of the color. $0 \leq g \leq 1.$
- double **b**

  The amount of blue of the color. $0 \leq b \leq 1.$

Static Public Attributes

- static const Color **MAROON** = Color::from_RGB(128, 0, 0)
- static const Color **DARK_RED** = Color::from_RGB(139, 0, 0)
- static const Color **BROWN** = Color::from_RGB(165, 42, 42)
- static const Color **FIREBRICK** = Color::from_RGB(178, 34, 34)
- static const Color **CRIMSON** = Color::from_RGB(220, 20, 60)
- static const Color **RED** = Color::from_RGB(255, 0 , 0)
- static const Color **TOMATO** = Color::from_RGB(255, 99, 71)
- static const Color **CORAL** = Color::from_RGB(255, 127, 80)
- static const Color **INDIAN_RED** = Color::from_RGB(205, 92, 92)
- static const Color **LIGHT_CORAL** = Color::from_RGB(240, 128, 128)
- static const Color **DARK_SALMON** = Color::from_RGB(233, 150, 122)
- static const Color **SALMON** = Color::from_RGB(250, 128, 114)
- static const Color **LIGHT_SALMON** = Color::from_RGB(255, 160, 122)
- static const Color **ORANGE_RED** = Color::from_RGB(255, 69, 0)
- static const Color **DARK_ORANGE** = Color::from_RGB(255, 140, 0)
- static const Color **ORANGE** = Color::from_RGB(255, 165, 0)
- static const Color **GOLD** = Color::from_RGB(255, 215, 0)
- static const Color **DARK_GOLDEN_ROD** = Color::from_RGB(184, 134, 11)
- static const Color **GOLDEN_ROD** = Color::from_RGB(218, 165, 32)
- static const Color **PALE_GOLDEN_ROD** = Color::from_RGB(238, 232, 170)
- static const Color **DARK_KHAKI** = Color::from_RGB(189, 183, 107)
- static const Color **KHAKI** = Color::from_RGB(240, 230, 140)
- static const Color **OLIVE** = Color::from_RGB(128, 128, 0)
- static const Color **YELLOW** = Color::from_RGB(255, 255, 0)
- static const Color **YELLOW_GREEN** = Color::from_RGB(154, 205, 50)
- static const Color **DARK_OLIVE_GREEN** = Color::from_RGB(85, 107, 47)
- static const Color **OLIVE_DRAB** = Color::from_RGB(107, 142, 35)
- static const Color **LAWN_GREEN** = Color::from_RGB(124, 252, 0)
- static const Color **CHART_REUSE** = Color::from_RGB(127, 255, 0)
- static const Color **GREEN_YELLOW** = Color::from_RGB(173, 255, 47)
- static const Color **DARK_GREEN** = Color::from_RGB(0, 100, 0)
- static const Color **GREEN** = Color::from_RGB(0, 128, 0)
- static const Color **FOREST_GREEN** = Color::from_RGB(34, 139, 34)
- static const Color **LIME** = Color::from_RGB(0, 255, 0)
- static const Color **LIME_GREEN** = Color::from_RGB(50, 205, 50)
- static const Color **LIGHT_GREEN** = Color::from_RGB(144, 238, 144)
- static const Color **PALE_GREEN** = Color::from_RGB(152, 251, 152)
- static const Color **DARK_SEA_GREEN** = Color::from_RGB(143, 188, 143)
- static const Color **MEDIUM_SPRING_GREEN** = Color::from_RGB(0, 250, 154)
- static const Color **SPRING_GREEN** = Color::from_RGB(0, 255, 127)
- static const Color **SEA_GREEN** = Color::from_RGB(46, 139, 87)
- static const Color **MEDIUM_AQUA_MARINE** = Color::from_RGB(102, 205, 170)

- static const Color **MEDIUM_SEA_GREEN** = Color::from_RGB(60, 179, 113)
- static const Color **LIGHT_SEA_GREEN** = Color::from_RGB(32, 178, 170)
- static const Color **DARK_SLATE_GRAY** = Color::from_RGB(47, 79, 79)
- static const Color **TEAL** = Color::from_RGB(0, 128, 128)
- static const Color **DARK_CYAN** = Color::from_RGB(0, 139, 139)
- static const Color **CYAN** = Color::from_RGB(0, 255, 255)
- static const Color **LIGHT_CYAN** = Color::from_RGB(244, 255, 255)
- static const Color **DARK_TURQUOISE** = Color::from_RGB(0, 206, 209)
- static const Color **TURQUOISE** = Color::from_RGB(64, 224, 208)
- static const Color **MEDIUM_TURQUOISE** = Color::from_RGB(72, 209, 204)
- static const Color **PALE_TORQUOISE** = Color::from_RGB(175, 238, 238)
- static const Color **AQUA_MARINE** = Color::from_RGB(127, 255, 212)
- static const Color **POWDER_BLUE** = Color::from_RGB(176, 224, 230)
- static const Color **CADET_BLUE** = Color::from_RGB(95, 158, 160)
- static const Color **STEEL_BLUE** = Color::from_RGB(70, 130, 180)
- static const Color **CORN_FLOWER_BLUE** = Color::from_RGB(100, 149, 237)
- static const Color **DEEP_SKY_BLUE** = Color::from_RGB(0, 191, 255)
- static const Color **DODGER_BLUE** = Color::from_RGB(30, 144, 255)
- static const Color **LIGHT_BLUE** = Color::from_RGB(173, 216, 230)
- static const Color **SKY_BLUE** = Color::from_RGB(135, 206, 235)
- static const Color **LIGHT_SKY_BLUE** = Color::from_RGB(135, 206, 250)
- static const Color **MIDNIGHT_BLUE** = Color::from_RGB(25, 25, 112)
- static const Color **NAVY** = Color::from_RGB(0, 0, 128)
- static const Color **DARK_BLUE** = Color::from_RGB(0, 0, 139)
- static const Color **MEDIUM_BLUE** = Color::from_RGB(0, 0, 205)
- static const Color **BLUE** = Color::from_RGB(0, 0, 255)
- static const Color **ROYAL_BLUE** = Color::from_RGB(65, 105, 225)
- static const Color **BLUE_VIOLET** = Color::from_RGB(138, 43, 226)
- static const Color **INDIGO** = Color::from_RGB(75, 0, 130)
- static const Color **DARK_SLATE_BLUE** = Color::from_RGB(72, 61, 139)
- static const Color **SLATE_BLUE** = Color::from_RGB(106, 90, 205)
- static const Color **MEDIUM_SLATE_BLUE** = Color::from_RGB(123, 104, 238)
- static const Color **MEDIUM_PURPLE** = Color::from_RGB(147, 112, 219)
- static const Color **DARK_MAGENTA** = Color::from_RGB(139, 0, 139)
- static const Color **DARK_VIOLET** = Color::from_RGB(148, 0, 211)
- static const Color **DARK_ORCHID** = Color::from_RGB(153, 50, 204)
- static const Color **MEDIUM_ORCHID** = Color::from_RGB(186, 85, 211)
- static const Color **PURPLE** = Color::from_RGB(128, 0, 128)
- static const Color **THISTLE** = Color::from_RGB(216, 191, 216)
- static const Color **PLUM** = Color::from_RGB(221, 160, 221)
- static const Color **VIOLET** = Color::from_RGB(238, 130, 238)
- static const Color **MAGENTA** = Color::from_RGB(255, 0, 255)
- static const Color **ORCHID** = Color::from_RGB(218, 112, 214)
- static const Color **MEDIUM_VIOLET_RED** = Color::from_RGB(199, 21, 133)
- static const Color **PALE_VIOLET_RED** = Color::from_RGB(219, 112, 147)
- static const Color **DEEP_PINK** = Color::from_RGB(255, 20, 147)
- static const Color **HOT_PINK** = Color::from_RGB(255, 105, 180)
- static const Color **LIGHT_PINK** = Color::from_RGB(255, 182, 193)
- static const Color **PINK** = Color::from_RGB(255, 192, 203)
- static const Color **ANTIQUE_WHITE** = Color::from_RGB(250, 235, 215)
- static const Color **BEIGE** = Color::from_RGB(245, 245, 220)
- static const Color **BISQUE** = Color::from_RGB(255, 228, 196)
- static const Color **BLANCHED_ALMOND** = Color::from_RGB(255, 235, 205)
- static const Color **WHEAT** = Color::from_RGB(245, 222, 179)
- static const Color **CORN_SILK** = Color::from_RGB(255, 248, 220)

- static const Color **LEMON_CHIFFON** = Color::from_RGB(255, 250, 205)
- static const Color **LIGHT_GOLDEN_ROD_YELLOW** = Color::from_RGB(250, 250, 210)
- static const Color **LIGHT_YELLOW** = Color::from_RGB(255, 255, 224)
- static const Color **SADDLE_BROWN** = Color::from_RGB(139, 69, 19)
- static const Color **SIENNA** = Color::from_RGB(160, 82, 45)
- static const Color **CHOCOLATE** = Color::from_RGB(210, 105, 30)
- static const Color **PERU** = Color::from_RGB(205, 133, 63)
- static const Color **SANDY_BROWN** = Color::from_RGB(244, 164, 96)
- static const Color **BURLY_WOOD** = Color::from_RGB(222, 184, 135)
- static const Color **TAN** = Color::from_RGB(210, 180, 140)
- static const Color **ROSY_BROWN** = Color::from_RGB(188, 143, 143)
- static const Color **MOCCASIN** = Color::from_RGB(255, 228, 181)
- static const Color **NAVAJO_WHITE** = Color::from_RGB(255, 222, 173)
- static const Color **PEACH_STUFF** = Color::from_RGB(255, 218, 185)
- static const Color **MISTY_ROSE** = Color::from_RGB(255, 228, 225)
- static const Color **LAVENDER_BLUSH** = Color::from_RGB(255, 240, 245)
- static const Color **LINEN** = Color::from_RGB(250, 240, 230)
- static const Color **OLD_LACE** = Color::from_RGB(253, 245, 230)
- static const Color **PAPAYA_WHIP** = Color::from_RGB(255, 239, 213)
- static const Color **SEA_SHELL** = Color::from_RGB(255, 245, 238)
- static const Color **MINT_CREAM** = Color::from_RGB(245, 255, 250)
- static const Color **SLATE_GRAY** = Color::from_RGB(112, 128, 144)
- static const Color **LIGHT_SLATE_GRAY** = Color::from_RGB(119, 136, 153)
- static const Color **LIGHT_STEEL_BLUE** = Color::from_RGB(176, 196, 222)
- static const Color **LAVENDER** = Color::from_RGB(230, 230, 250)
- static const Color **FLORAL_WHITE** = Color::from_RGB(255, 250, 240)
- static const Color **ALICE_BLUE** = Color::from_RGB(240, 248, 255)
- static const Color **GHOST_WHITE** = Color::from_RGB(248, 248, 255)
- static const Color **HONEYDEW** = Color::from_RGB(240, 255, 240)
- static const Color **IVORY** = Color::from_RGB(255, 255, 240)
- static const Color **AZURE** = Color::from_RGB(240, 255, 255)
- static const Color **SNOW** = Color::from_RGB(255, 250, 250)
- static const Color **BLACK** = Color::from_RGB(0, 0, 0)
- static const Color **DIM_GRAY** = Color::from_RGB(105, 105, 105)
- static const Color **GRAY** = Color::from_RGB(128, 128, 128)
- static const Color **DARK_GRAY** = Color::from_RGB(169, 169, 169)
- static const Color **SILVER** = Color::from_RGB(192, 192, 192)
- static const Color **LIGHT_GRAY** = Color::from_RGB(211, 211, 211)
- static const Color **GAINSBORO** = Color::from_RGB(220, 220, 220)
- static const Color **WHITE_SMOKE** = Color::from_RGB(245, 245, 245)
- static const Color **WHITE** = Color::from_RGB(255, 255, 255)

7.9.1 Detailed Description

An object used to represent an rgb color. r, g and b must all be real numbers between 0 and 1.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Color()

```
Color::Color (
            const double r,
            const double g,
            const double b )
```

Returns the (r,g,b) color.

Parameters

r	a real number between zero and one.
g	a real number between zero and one.
b	a real number between zero and one.

7.9.3 Member Function Documentation

7.9.3.1 mix()

```
Color Color::mix (
            const Color & A,
            const Color & B,
            const double alpha )  [static]
```

Returns a mix of colors A and B in proportions alpha and 1 - alpha.

Parameters

A	
B	
alpha	

Returns

Color

7.9.3.2 to_string()

```
std::string Color::to_string ( )
```

Returns a string representation of the color.

Returns

std::string

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

- CPGF/PGFBasics/PGFConf.hpp
- CPGF/PGFBasics/PGFConf.cpp

7.10 CPGF::DashPatterns Class Reference

This class provides a handful of useful predefined constants to express DashPatterns.

```
#include <PGFConf.hpp>
```

Static Public Attributes

- static const std::vector< double > **SOLID** = std::vector<double>()
- static const std::vector< double > **DASHED** = std::vector<double>({0.3, 0.3})
- static const std::vector< double > **DOTTED** = std::vector<double>({0.1, 0.1})

7.10.1 Detailed Description

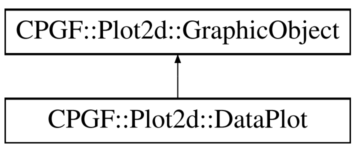
This class provides a handful of useful predefined constants to express DashPatterns.

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

- CPGF/PGFBasics/PGFConf.hpp
- CPGF/PGFBasics/PGFConf.cpp

7.11 CPGF::Plot2d::DataPlot Class Reference

Inheritance diagram for CPGF::Plot2d::DataPlot:



Public Member Functions

- **DataPlot** (const std::vector< double > &Y, const std::vector< double > &X, const Color &color=Color↩ ::BLUE, const double size=1, const double opacity=1, std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)> shape=Shapes::↩ Circle, const std::string &legend="")
- **DataPlot** (const std::vector< double > &Y, const std::vector< double > &X, std::function< Color(unsigned int)> color, std::function< double(unsigned int)> size, std::function< double(unsigned int)> opacity, std↩ ::function< std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)>(unsigned int)> shape, const std::string &legend="")
- **DataPlot** (const std::vector< double > &Y, const std::vector< double > &X, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> size, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)>(AffineSpace::Point2d &)> shape, const std::string &legend="")
- **DataPlot** (const std::vector< AffineSpace::Point2d > &data, const Color &color=Color::BLUE, const double size=1, const double opacity=1, std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)> shape=Shapes::Circle, const std::string &legend="")
- **DataPlot** (const std::vector< AffineSpace::Point2d > &data, std::function< Color(unsigned int)> color, std↩ ::function< double(unsigned int)> size, std::function< double(unsigned int)> opacity, std::function< std↩ ::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)>(unsigned int)> shape, const std::string &legend="")
- **DataPlot** (const std::vector< AffineSpace::Point2d > &data, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> size, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)>(AffineSpace::Point2d &)> shape, const std::string &legend="")
- double x_min () const override
- double x_max () const override
- double y_min () const override
- double y_max () const override
- Objects2d::Object2d miniature (const AffineSpace::Point2d &pos) const override
- Scene2d render_to_scene (std::function< AffineSpace::Point2d(const AffineSpace::Point2d &P)> transform, const double x_min, const double x_max, const double y_min, const double y_max) const override

Public Attributes

- std::vector< AffineSpace::Point2d > **points**
- std::function< Color(unsigned int)> **color**
- std::function< double(unsigned int)> **size**
- std::function< double(unsigned int)> **opacity**
- std::function< std::function< Objects2d::Object2d(const AffineSpace::Point2d pos, const Color &color, const double opacity, const double size)>(unsigned int)> **shape**

Additional Inherited Members

7.11.1 Member Function Documentation

7.11.1.1 miniature()

```
Object2d DataPlot::miniature (
            const AffineSpace::Point2d & pos ) const [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.11.1.2 render_to_scene()

```
Scene2d DataPlot::render_to_scene (
            std::function< AffineSpace::Point2d(const AffineSpace::Point2d &P)> transform,
            const double x_min,
            const double x_max,
            const double y_min,
            const double y_max ) const [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.11.1.3 x_max()

```
double DataPlot::x_max ( ) const [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.11.1.4 x_min()

```
double DataPlot::x_min ( ) const [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.11.1.5 y_max()

```
double DataPlot::y_max ( ) const [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.11.1.6 y_min()

```
double DataPlot::y_min ( ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

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

- CPGF/Plot2d/DataPlot.hpp
- CPGF/Plot2d/DataPlot.cpp

7.12 Math::DualNumber< K > Class Template Reference

Represents a dual number. A mathematical object written like x = a + b, where satisfies ^{2}=0.

`#include <DualNumbers.hpp>`

Public Member Functions

- **DualNumber** ()

  Constructs the dual number zero.
- DualNumber (const K a)

  Constructs a dual number whose real part is a and whose infinitesimal part equal to zero.
- DualNumber (const K a, const K b)

  Construct a dual number with a as real part and b as infinitesimal part.
- std::string to_string () const

  Returns a string representation of the dual number: a + b.

Public Attributes

- K **a**

  Real part.
- K **b**

  Infinitesimal part.

Static Public Attributes

- static const DualNumber< K > **epsilon** = DualNumber(0, 1)

  The pure infinitesimal dual number.

Friends

- DualNumber< K > operator+ (const DualNumber< K > &x, const DualNumber< K > &y)

    Returns the sum of two dual numbers. x + y = (x.a + y.a) + (x.b + y.b)
- DualNumber< K > operator+ (const DualNumber< K > &x, const K y)

    Returns the sum of a dual number and a real number. x + y = (x.a + y) + x.b∗
- DualNumber< K > operator+ (const K x, const DualNumber< K > &y)

    Returns the sum of a dual number and a real number. x + y = (x + y.a) + y.b∗
- DualNumber< K > operator- (const DualNumber< K > &x, const DualNumber< K > &y)

    Returns the substraction of two dual numbers. x - y = (x.a - y.a) + (x.b - y.b)
- DualNumber< K > operator- (const DualNumber< K > &x, const K y)

    Returns the substraction of a dual number and a real number. x - y = (x.a - y) + x.b∗
- DualNumber< K > operator- (const K x, const DualNumber< K > &y)

    Returns the substraction of a real number and a dual number. x - y = (x - y.a) + y.b∗
- DualNumber< K > operator∗ (const DualNumber< K > &x, const DualNumber< K > &y)

    Returns the product of two dual numbers. x∗y = (x.a∗y.a) + (x.a∗y.b + x.b∗y.a)
- DualNumber< K > operator∗ (const DualNumber< K > &x, const K y)

    Returns the product of a dual number times a real number. x∗y = (x.a∗y) + (x.b∗y)
- DualNumber< K > operator∗ (const K x, const DualNumber< K > &y)

    Returns the product of a dual number times a real number. x∗y = (x∗y.a) + (x∗y.b)
- DualNumber< K > operator/ (const DualNumber< K > &x, const DualNumber< K > &y)

    Performs the division of two dual numbers. x/y = (x.a/y.a) + (x.b∗y.a - x.a∗y.b)/(y.a∗y.a) ∗
- DualNumber< K > operator/ (const DualNumber< K > &x, const K y)

    Returns the division of a dual number and a real number. x/y = (x.a/y) + (x.b/y)
- DualNumber< K > operator/ (const K x, const DualNumber< K > &y)

    Returns the division of a real number by a dual number. x/y = (x/y.a) -x∗y.b/(y.a∗y.a) ∗

7.12.1 Detailed Description

template<typename K>
class Math::DualNumber< K >

Represents a dual number. A mathematical object written like x = a + b, where satisfies ^{2}=0.

Every mathematical function on the C++ math library has been extended to act on DualNumbers. This allows automatic differentiation.

Template Parameters

K	

7.12.2 Constructor & Destructor Documentation

7.12.2.1 DualNumber() [1/2]

```
template<typename K >
DualNumber::DualNumber (
```

```
            const K a )
```

Constructs a dual number whose real part is *a* and whose infinitesimal part equal to zero.

Parameters

a	the real part.

7.12.2.2 DualNumber() [2/2]

```
template<typename K >
DualNumber::DualNumber (
            const K a,
            const K b )
```

Construct a dual number with *a* as real part and *b* as infinitesimal part.

Parameters

a	
b	

7.12.3 Member Function Documentation

7.12.3.1 to_string()

```
template<typename K >
std::string DualNumber::to_string
```

Returns a string representation of the dual number: a + b.

Returns

std::string

7.12.4 Friends And Related Function Documentation

7.12.4.1 operator* [1/3]

```
template<typename K >
DualNumber< K > operator* (
            const DualNumber< K > & x,
            const DualNumber< K > & y )  [friend]
```

Returns the product of two dual numbers. x∗y = (x.a∗y.a) + (x.a∗y.b + x.b∗y.a)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.2 operator* [2/3]

```
template<typename K >
DualNumber< K > operator* (
            const DualNumber< K > & x,
            const K y )  [friend]
```

Returns the product of a dual number times a real number. x*y = (x.a*y) + (x.b*y)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.3 operator* [3/3]

```
template<typename K >
DualNumber< K > operator* (
            const K x,
            const DualNumber< K > & y )  [friend]
```

Returns the product of a dual number times a real number. x*y = (x*y.a) + (x*y.b)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.4 operator+ [1/3]

```
template<typename K >
DualNumber< K > operator+ (
            const DualNumber< K > & x,
            const DualNumber< K > & y )  [friend]
```

Returns the sum of two dual numbers. x + y = (x.a + y.a) + (x.b + y.b)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.5 operator+ [2/3]

```
template<typename K >
DualNumber< K > operator+ (
            const DualNumber< K > & x,
            const K y )  [friend]
```

Returns the sum of a dual number and a real number. x + y = (x.a + y) + x.b*

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.6 operator+ [3/3]

```
template<typename K >
DualNumber< K > operator+ (
            const K x,
            const DualNumber< K > & y )  [friend]
```

Returns the sum of a dual number and a real number. x + y = (x + y.a) + y.b*

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.7 operator- [1/3]

```
template<typename K >
DualNumber< K > operator- (
            const DualNumber< K > & x,
            const DualNumber< K > & y )  [friend]
```

Returns the substraction of two dual numbers. x - y = (x.a - y.a) + (x.b - y.b)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.8 operator- [2/3]

```
template<typename K >
DualNumber< K > operator- (
            const DualNumber< K > & x,
            const K y )  [friend]
```

Returns the substraction of a dual number and a real number. x - y = (x.a - y) + x.b∗

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.9 operator- [3/3]

```
template<typename K >
DualNumber< K > operator- (
            const K x,
            const DualNumber< K > & y )   [friend]
```

Returns the substraction of a real number and a dual number. x - y = (x - y.a) + y.b∗

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.10 operator/ [1/3]

```
template<typename K >
DualNumber< K > operator/ (
            const DualNumber< K > & x,
            const DualNumber< K > & y )   [friend]
```

Performs the division of two dual numbers. x/y = (x.a/y.a) + (x.b∗y.a - x.a∗y.b)/(y.a∗y.a) ∗

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.11 operator/ [2/3]

```
template<typename K >
DualNumber< K > operator/ (
            const DualNumber< K > & x,
            const K y )   [friend]
```

Returns the division of a dual number and a real number. x/y = (x.a/y) + (x.b/y)

Parameters

x	
y	

Returns

DualNumber<K>

7.12.4.12 operator/ [3/3]

```
template<typename K >
DualNumber< K > operator/ (
            const K x,
            const DualNumber< K > & y )  [friend]
```

Returns the division of a real number by a dual number. x/y = (x/y.a) -x∗y.b/(y.a∗y.a) ∗

Parameters

x	
y	

Returns

DualNumber<K>

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

- Math/DualNumbers.hpp
- Math/DualNumbers.cpp

7.13 Solvers::Gas::ExactRiemannSolver Class Reference

Solves the Riemann problem exactly for the 1D Euler Equations.

```
#include <ExactRiemannSolver.hpp>
```

Public Member Functions

- double rho (const double x, const double t) const
  Gas density as a function of space and time.
- double v (const double x, const double t) const
  Gas speed as a function of space and time.
- double P (const double x, const double t) const
  Gas pressure as a function of space and time.
- double S_max () const
  Returns the maximum absolute value of all wave speeds of the solution.
- ExactRiemannSolver (const double rho_L, const double v_L, const double P_L, const double rho_R, const double v_R, const double P_R, const double gamma, const double tol)
  Construct a new Exact Riemann Solver object.

Protected Member Functions

- double f_L (const double P) const

  Function used in the computation of the pressure equation (left part).
- double f_R (const double P) const

  Function used in the computation of the pressure equation (right part).
- double f (const double P) const

  The pressure equation. P_star is the only solution of f(P)=0.
- double df_LdP (const double P) const

  The derivative of f_L with respect to P.
- double df_RdP (const double P) const

  The derivative of f_R with respect to P.
- double dfdP (const double P) const

  The derivative of f with respecto to P.
- double rho_Lrf (const double S) const

  Returns the density in the left rarefaction region (between the head and the tail).
- double v_Lrf (const double S) const

  Returns the speed in the left rarefaction region (between the head and the tail).
- double P_Lrf (const double S) const

  Returns the pressure in the left rarefaction region (between the head and the tail).
- double rho_Rrf (const double S) const

  Returns the density in the right rarefaction region (between the head and the tail).
- double v_Rrf (const double S) const

  Returns the speed in the right rarefaction region (between the head and the tail).
- double P_Rrf (const double S) const

  Returns the pressure in the right rarefaction region (between the head and the tail).
- double rho_L0 (const double S) const

  Returns the density for the right vacuum state Riemann problem.
- double v_L0 (const double S) const

  Returns the speed for the right vacuum state Riemann problem.
- double P_L0 (const double S) const

  Returns the pressure for the right vacuum state Riemann problem.
- double rho_R0 (const double S) const

  Returns the density for the left vacuum state Riemann problem.
- double v_R0 (const double S) const

  Returns the speed for the left vacuum state Riemann problem.
- double P_R0 (const double S) const

  Returns the pressure for the left vacuum state Riemann problem.

Protected Attributes

- double **gamma**

  Gas adiabatic expansion coefficient.
- VacuumState **vacuum_state**

  Stores what type of problem we have to solve.
- double **rho_L**

  The gas density to the left of x=0.
- double **v_L**

  The gas speed to the left of x=0.
- double **P_L**

The gas pressure to the left of x=0.

- double **rho_R**

  The gas density to the right of x=0.
- double **v_R**

  The gas speed to the right of x=0.
- double **P_R**

  The gas pressure to the right of x=0.
- double **G1**

  gamma + 1
- double **G2**

  gamma - 1
- double **G3**

  (gamma - 1)/(gamma + 1)
- double **G4**

  (gamma - 1)/(2∗gamma)
- double **G5**

  (gamma + 1)/(2∗gamma)
- double **G6**

  1/gamma
- double **G7**

  2 / (gamma + 1)
- double **G8**

  2 / (gamma - 1)
- double **A_L**

  2 / (gamma + 1) / rho_L
- double **A_R**

  2 / (gamma + 1) / rho_R
- double **B_L**

  (gamma - 1)/(gamma + 1) ∗ P_L
- double **B_R**

  (gamma - 1)/(gamma + 1) ∗ P_R
- double **c_L**

  The gas sound speed to the left of x=0.
- double **c_R**

  The gas sound speed to the right of x=0.
- double **S_L**

  Speed of the left shock wave.
- double **S_HL**

  Speed of the head of the left rarefaction wave.
- double **S_TL**

  Speed of the tail of the left rarefaction wave.
- double **S_starL**

  Speed of the vacuum front to the right of x=0.
- double **rho_starL**

  Density in the star region to the left of the contact discontinuity.
- double **v_star**

  Speed in the star region.
- double **P_star**

  Pressure in the star region.
- double **S_R**

  Speed of the right shock wave.

- double **S_HR**

  Speed of the head of the right rarefaction wave.
- double **S_TR**

  Speed of the tail of the right rarefaction wave.
- double **S_starR**

  Speed of the vacuum front to the left of x=0.
- double **rho_starR**

  Density in the star region to the right of the contact discontinuity.

7.13.1 Detailed Description

Solves the Riemann problem exactly for the 1D Euler Equations.

Only valid when the area doesn't change. Computes the solution as a function of x and t.

The algorithm works by first computing the pressure in the star region numerically. Afterwards, the speed in the star region is calculated. Next, the character (shock wave or rarefaction wave) of the left and right waves is found out. Lastly, depending on the value of the quotient x/t, the correct zone is selected and the associated formulae are used to calculate the variables.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 ExactRiemannSolver()

```
ExactRiemannSolver::ExactRiemannSolver (
            const double rho_L,
            const double v_L,
            const double P_L,
            const double rho_R,
            const double v_R,
            const double P_R,
            const double gamma,
            const double tol )
```

Construct a new Exact Riemann Solver object.

Parameters

rho_L	the gas density to the left of x=0.
v_L	the gas speed to the left of x=0.
P_L	the gas pressure to the left of x=0.
rho_R	the gas density to the right of x=0.
v_R	the gas speed to the right of x=0.
P_R	the gas pressure to the right of x=0.
gamma	the gas adiabatic expansion coefficient.
tol	the precision with which the pressure in the star region is determined.

7.13.3 Member Function Documentation

7.13.3.1 df_LdP()

```
double ExactRiemannSolver::df_LdP (
            const double P ) const  [protected]
```

The derivative of f_L with respect to P.

Parameters

P	

Returns

double

7.13.3.2 df_RdP()

```
double ExactRiemannSolver::df_RdP (
            const double P ) const  [protected]
```

The derivative of f_R with respect to P.

Parameters

P	

Returns

double

7.13.3.3 dfdP()

```
double ExactRiemannSolver::dfdP (
            const double P ) const  [protected]
```

The derivative of f with respecto to P.

Parameters

P	

Returns

double

7.13.3.4 f()

```
double ExactRiemannSolver::f (
            const double P ) const  [protected]
```

The pressure equation. P_star is the only solution of f(P)=0.

Parameters

P	

Returns

double

7.13.3.5 f_L()

```
double ExactRiemannSolver::f_L (
            const double P ) const  [protected]
```

Function used in the computation of the pressure equation (left part).

Parameters

P	

Returns

double

7.13.3.6 f_R()

```
double ExactRiemannSolver::f_R (
            const double P ) const  [protected]
```

Function used in the computation of the pressure equation (right part).

Parameters

P	

Returns

double

7.13.3.7 P()

```
double ExactRiemannSolver::P (
            const double x,
            const double t ) const
```

Gas pressure as a function of space and time.

Parameters

x	spatial coordinate.
t	time coordinate.

Returns

double

7.13.3.8 P_L0()

```
double ExactRiemannSolver::P_L0 (
            const double S ) const  [protected]
```

Returns the pressure for the right vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.9 P_Lrf()

```
double ExactRiemannSolver::P_Lrf (
            const double S ) const  [protected]
```

Returns the pressure in the left rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

7.13.3.10 P_R0()

```
double ExactRiemannSolver::P_R0 (
            const double S ) const  [protected]
```

Returns the pressure for the left vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.11 P_Rrf()

```
double ExactRiemannSolver::P_Rrf (
            const double S ) const  [protected]
```

Returns the pressure in the right rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

7.13.3.12 rho()

```
double ExactRiemannSolver::rho (
            const double x,
            const double t ) const
```

Gas density as a function of space and time.

Parameters

x	spatial coordinate.
t	time coordinate.

Returns

double

7.13.3.13 rho_L0()

```
double ExactRiemannSolver::rho_L0 (
            const double S ) const  [protected]
```

Returns the density for the right vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.14 rho_Lrf()

```
double ExactRiemannSolver::rho_Lrf (
            const double S ) const  [protected]
```

Returns the density in the left rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

7.13.3.15 rho_R0()

```
double ExactRiemannSolver::rho_R0 (
            const double S ) const  [protected]
```

Returns the density for the left vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.16 rho_Rrf()

```
double ExactRiemannSolver::rho_Rrf (
            const double S ) const  [protected]
```

Returns the density in the right rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

7.13.3.17 S_max()

```
double ExactRiemannSolver::S_max ( ) const
```

Returns the maximum absolute value of all wave speeds of the solution.

Returns

double

7.13.3.18 v()

```
double ExactRiemannSolver::v (
            const double x,
            const double t ) const
```

Gas speed as a function of space and time.

Parameters

x	spatial coordinate.
t	time coordinate.

Returns

double

7.13.3.19 v_L0()

```
double ExactRiemannSolver::v_L0 (
            const double S ) const  [protected]
```

Returns the speed for the right vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.20 v_Lrf()

```
double ExactRiemannSolver::v_Lrf (
            const double S ) const  [protected]
```

Returns the speed in the left rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

7.13.3.21 v_R0()

```
double ExactRiemannSolver::v_R0 (
            const double S ) const  [protected]
```

Returns the speed for the left vacuum state Riemann problem.

Parameters

S	

Returns

double

7.13.3.22 v_Rrf()

```
double ExactRiemannSolver::v_Rrf (
            const double S ) const  [protected]
```

Returns the speed in the right rarefaction region (between the head and the tail).

Parameters

S	

Returns

double

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

- Solvers/Gas/ExactRiemannSolver.hpp
- Solvers/Gas/ExactRiemannSolver.cpp

7.14 Solvers::Gas::ExactSteadySolver Class Reference

Public Member Functions

- ExactSteadySolver (const double rho_0, const double P_0, const double T_0, const double M_x, const double A_x, const double gamma, const SolutionType solution_type)

Construct a new Exact Steady Solver object.

- double M (const double A) const

  Returns Mach number as a function of area.
- double rho (const double A) const

  Returns density as a function of area.
- double v (const double A) const

  Returns speed as a function of area.
- double P (const double A) const

  Returns pressure as a function of area.
- double T (const double A) const

  Returns temperature as a function of area.

Protected Attributes

- SolutionType **solution_type**

  Stores whether the subsonic or the supersonic solution is found.
- double **C**

  The constant f(M(x))A(x).
- double **rho_0**

  Rest density of the gas.
- double **P_0**

  Rest pressure of the gas.
- double **T_0**

  Rest temperature of the gas.
- double **R**

  Gas constant.
- double **gamma**

  Gas adiabatic expansion coefficient.
- double **G1**

  (gamma - 1) / 2
- double **G2**

  gamma / (gamma - 1)
- double **G3**

  1 / (gamma - 1)
- double **G4**

  (gamma + 1) / (2∗(gamma - 1))

7.14.1 Constructor & Destructor Documentation

7.14.1.1 ExactSteadySolver()

```
ExactSteadySolver::ExactSteadySolver (
            const double rho_0,
            const double P_0,
            const double T_0,
            const double M_x,
            const double A_x,
            const double gamma,
            const SolutionType solution_type )
```

Construct a new Exact Steady Solver object.

Parameters

rho_0	rest density.
P_0	rest pressure.
T_0	rest temperature.
M_x	Mach number at a point x.
A_x	area at a point x.
gamma	gas adiabatic expansion coefficient.
solution_type	whether to compute the subsonic or the supersonic solution.

7.14.2 Member Function Documentation

7.14.2.1 M()

```
double ExactSteadySolver::M (
            const double A ) const
```

Returns Mach number as a function of area.

Parameters

A	area.

Returns

double

7.14.2.2 P()

```
double ExactSteadySolver::P (
            const double A ) const
```

Returns pressure as a function of area.

Parameters

A	area.

Returns

double

7.14.2.3 rho()

```
double ExactSteadySolver::rho (
            const double A ) const
```

Returns density as a function of area.

Parameters

A	area.

Returns

double

7.14.2.4 T()

```
double ExactSteadySolver::T (
            const double A ) const
```

Returns temperature as a function of area.

Parameters

A	area.

Returns

double

7.14.2.5 v()

```
double ExactSteadySolver::v (
            const double A ) const
```

Returns speed as a function of area.

Parameters

A	area.

Returns

double

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

- Solvers/Gas/ExactSteadySolver.hpp
- Solvers/Gas/ExactSteadySolver.cpp

7.15 Utilities::FileArray< T > Class Template Reference

Classes

- class Iterator

Public Member Functions

- unsigned long size () const

    Returns number of elements in the array.
- T **operator[ ]** (const unsigned long i) const
- Iterator **operator[ ]** (const unsigned long i)
- T **at** (const unsigned long i) const
- Iterator **at** (const unsigned long i)
- T **front** () const
- Iterator **front** ()
- T **back** () const
- Iterator **back** ()
- Iterator **begin** ()
- Iterator **end** ()
- Iterator **rbegin** ()
- Iterator **rend** ()
- void **push_back** (const T &val)
- void **pop_back** ()
- void **clear** ()
- **FileArray** (char *file, const bool temp=false)
- **FileArray** (const std::string &file, const bool temp=false)
- **FileArray** (const std::vector< T > &vector)
- **FileArray** (char *file, const std::vector< T > &vector, const bool temp=false)
- **FileArray** (const std::string &file, const std::vector< T > &vector, const bool temp=false)
- **FileArray** (T *array, unsigned int N)
- **FileArray** (char *file, T *array, unsigned int N, const bool temp=false)
- **FileArray** (const std::string &file, T *array, unsigned int N, const bool temp=false)
- **FileArray** (const std::initializer_list< T > &list)
- **FileArray** (char *file, const std::initializer_list< T > &list, const bool temp=false)
- **FileArray** (const std::string &file, const std::initializer_list< T > &list, const bool temp=false)
- **FileArray** (const FileArray< T > &f_array)=delete
- FileArray & **operator=** (const FileArray< T > &f_array)=delete

Public Attributes

- FILE * **file**

Protected Attributes

- unsigned long **N**

  Length of the array.
- std::string **filename**
- bool **temp**
- std::vector< unsigned long > **indexes**

7.15.1 Member Function Documentation

7.15.1.1 size()

```
template<typename T >
unsigned long FileArray::size
```

Returns number of elements in the array.

Returns

unsigned long

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

- Utilities/FileArray.hpp
- Utilities/FileArray.cpp

7.16 GasBoundaryConditions Class Reference

This class is used to store the boundary conditions associated to a simulation.

```
#include <Simulation.hpp>
```

Public Member Functions

- GasBoundaryConditions (const GasBoundaryConditionsType &left=GasBoundaryConditionsType::WALL, const GasBoundaryConditionsType &right=GasBoundaryConditionsType::WALL, std::function< double(double t)> left_condition_1=[ ](double t){return 0;}, std::function< double(double t)> left_condition_2=[ ](double t){return 0;}, std::function< double(double t)> left_condition_3=[ ](double t){return 0;}, std::function< double(double t)> right_condition_1=[ ](double t){return 0;}, std::function< double(double t)> right_condition_2=[ ](double t){return 0;}, std::function< double(double t)> right_condition_3=[ ](double t){return 0;})

  Construct a new Gas Boundary Conditions object.

Public Attributes

- GasBoundaryConditionsType **left**
  The type of boundary conditions to the left of the domain.
- GasBoundaryConditionsType **right**
  The type of boundary conditions to the right of the domain.
- std::function< double(double t)> **left_condition_1**
  Function of time used to return the value of the first left boundary condition.
- std::function< double(double t)> **left_condition_2**
  Function of time used to return the value of the second left boundary condition.
- std::function< double(double t)> **left_condition_3**
  Function of time used to return the value of the third left boundary condition.
- std::function< double(double t)> **right_condition_1**
  Function of time used to return the value of the first right boundary condition.
- std::function< double(double t)> **right_condition_2**
  Function of time used to return the value of the second right boundary condition.
- std::function< double(double t)> **right_condition_3**
  Function of time used to return the value of the third right boundary condition.

7.16.1 Detailed Description

This class is used to store the boundary conditions associated to a simulation.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 GasBoundaryConditions()

```
GasBoundaryConditions::GasBoundaryConditions (
            const GasBoundaryConditionsType & left = GasBoundaryConditionsType::WALL,
            const GasBoundaryConditionsType & right = GasBoundaryConditionsType::WALL,
            std::function< double(double t)> left_condition_1 = [](double t){return 0;},
            std::function< double(double t)> left_condition_2 = [](double t){return 0;},
            std::function< double(double t)> left_condition_3 = [](double t){return 0;},
            std::function< double(double t)> right_condition_1 = [](double t){return 0;},
            std::function< double(double t)> right_condition_2 = [](double t){return 0;},
            std::function< double(double t)> right_condition_3 = [](double t){return 0;} )
```

Construct a new Gas Boundary Conditions object.

Parameters

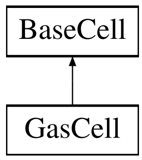
left	the type of the left boundary condition.
right	the type of the right boundary condition.
left_condition_1	function of time used for the first left boundary condition.
left_condition_2	function of time used for the second left boundary condition.
left_condition_3	function of time used for the third left boundary condition.
right_condition_1	function of time used for the first right boundary condition.
right_condition_2	function of time used for the second right boundary condition.
right_condition_3	function of time used for the third right boundary condition.

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

- Simulation.hpp
- Simulation.cpp

7.17 GasCell Class Reference

Inheritance diagram for GasCell:



Public Member Functions

- GasCell (const Math::Vector< double > &U, double a, double b, const double A, Chemistry::SolidGasReaction ∗QR)

    Construct a new Cell object.
- GasCell (FILE ∗file, Chemistry::SolidGasReaction ∗QR)

    Construct a new Cell object from file.
- Math::Vector< double > **F** () const

    Returns the flux evaluated at the conserved variables of the cell, that is, F(U).
- void **update** ()

    Calculates all cell variables (rho, T, P, ...) from the conserved quantities.
- void read_from_file (FILE ∗file)

    Reads all cell values from file and calls update()
- std::string to_string () const

    Returns a string representation of the GasCell.

Public Attributes

- GasCell ∗ **right_neighbour**

    Right neighbour of the cell.
- GasCell ∗ **left_neighbour**

    Left neighbour of the cell.
- double rho

    Gas density.
- double c

    Gas sound speed.
- double v

    Gas speed.
- double T

    Gas temperature.
- double P

    Gas pressure.
- double **H**

    Gas total enthalpy.
- double **E**

    Gas total energy.
- double **M**

    Mach number.

7.17.1 Constructor & Destructor Documentation

7.17.1.1 GasCell() [1/2]

```
GasCell::GasCell (
            const Math::Vector< double > & U,
            double a,
            double b,
            const double A,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Cell object.

Parameters

U	the vector of conserved variables.
a	the left limit of the cell.
b	the right limit of the cell.
A	the area of the cell.
QR	a reference to the quemical reaction.

7.17.1.2 GasCell() [2/2]

```
GasCell::GasCell (
            FILE * file,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Cell object from file.

Parameters

file	

7.17.2 Member Function Documentation

7.17.2.1 read_from_file()

```
void GasCell::read_from_file (
            FILE * file )
```

Reads all cell values from file and calls update()

Parameters

file	

7.17.2.2 to_string()

```
std::string GasCell::to_string ( ) const
```

Returns a string representation of the GasCell.

Returns

std::string

7.17.3 Member Data Documentation

7.17.3.1 c

```
double GasCell::c
```

Gas sound speed.

Returns

double

7.17.3.2 P

```
double GasCell::P
```

Gas pressure.

Returns

double

7.17.3.3 rho

`double GasCell::rho`

Gas density.

Returns

double

7.17.3.4 T

`double GasCell::T`

Gas temperature.

Returns

double

7.17.3.5 v

`double GasCell::v`

Gas speed.

Returns

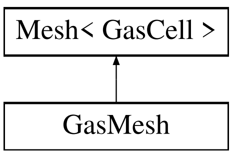
double

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

- Mesh/Cell.hpp
- Mesh/Cell.cpp

7.18 GasMesh Class Reference

Inheritance diagram for GasMesh:



Public Member Functions

- GasMesh (GasCell *first_cell=nullptr, GasCell *last_cell=nullptr, const Chemistry::SolidGasReaction &QR=Chemistry::SolidGasReaction(), std::function< double(const double x)> A_func=[ ](double x){return 0;}, const double detail_subdivide_threshold=0.01, const double detail_merge_threshold=0.0005, const double max_length_factor=1./20, const double min_length_factor=1./10000, const double boundary_cell_max_length_factor=1./100

    Construct a new Gas Mesh object.
- std::vector< double > c () const

    Returns gas sound speed of all cells in the mesh.
- std::vector< double > rho () const

    Returns the density of all cells in the mesh.
- std::vector< double > v () const

    Returns gas speed of all cells in the mesh.
- std::vector< double > T () const

    Returns the temperature of all cells in the mesh.
- std::vector< double > P () const

    Returns the pressure of all cells in the mesh.
- std::vector< double > M () const

    Returns the Mach number of all cells in the mesh.

Additional Inherited Members

7.18.1 Constructor & Destructor Documentation

7.18.1.1 GasMesh()

```
GasMesh::GasMesh (
            GasCell * first_cell = nullptr,
            GasCell * last_cell = nullptr,
            const Chemistry::SolidGasReaction & QR = Chemistry::SolidGasReaction(),
            std::function< double(const double x)> A_func = [](double x){return 0;},
            const double detail_subdivide_threshold = 0.01,
            const double detail_merge_threshold = 0.0005,
            const double max_length_factor = 1./20,
            const double min_length_factor = 1./10000,
            const double boundary_cell_max_length_factor = 1./1000 )
```

Construct a new Gas Mesh object.

Parameters

first_cell	pointer to the first cell.
last_cell	pointer to the last cell.
QR	chemical reaction.
A_func	area function.
detail_subdivide_threshold	determines when cells are subdivided.
detail_merge_threshold	determines when cells are merged.
max_length_factor	limits the maximum length of a created cell.
min_length_factor	limits the minimum length of a created cell.
boundary_cell_max_length_factor	limits the maxium length of cells near the boundary.

7.18.2 Member Function Documentation

7.18.2.1 c()

`std::vector< double > GasMesh::c ( ) const`

Returns gas sound speed of all cells in the mesh.

Returns

std::vector<double>

7.18.2.2 M()

`std::vector< double > GasMesh::M ( ) const`

Returns the Mach number of all cells in the mesh.

Returns

std::vector<double>

7.18.2.3 P()

`std::vector< double > GasMesh::P ( ) const`

Returns the pressure of all cells in the mesh.

Returns

std::vector<double>

7.18.2.4 rho()

`std::vector< double > GasMesh::rho ( ) const`

Returns the density of all cells in the mesh.

Returns

std::vector<double>

7.18.2.5 T()

```
std::vector< double > GasMesh::T ( ) const
```

Returns the temperature of all cells in the mesh.

Returns

std::vector<double>

7.18.2.6 v()

```
std::vector< double > GasMesh::v ( ) const
```

Returns gas speed of all cells in the mesh.

Returns

std::vector<double>

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

- Mesh/Mesh.hpp
- Mesh/Mesh.cpp

7.19 CPGF::Plot2d::Graphic Class Reference

Public Member Functions

- void **add** (GraphicObject *object, Axis *Y, Axis *X)
- Scene2d **render_to_scene** () const
- **Graphic** (const bool show_legend=false)

Public Attributes

- std::vector< std::tuple< GraphicObject *, Axis *, Axis * > > **graphic_objects**
- bool **show_legend**
- LegendPosition **legend_position**
- std::vector< Axis * > **axes**

Static Public Attributes

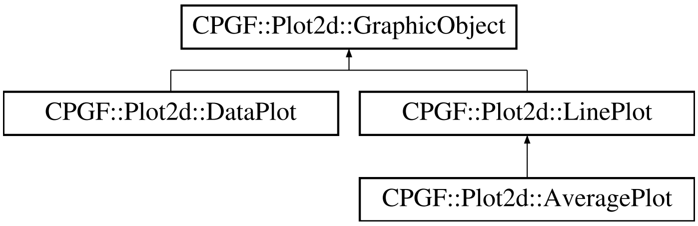
- constexpr static const double **LEGEND_MARGIN** = 0.25
- constexpr static const double **LEGEND_VERTICAL_DISPLACEMENT_PER_LINE** = 0.25
- constexpr static const double **CHARACTER_WIDTH** = 0.1

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

- CPGF/Plot2d/Graphic.hpp
- CPGF/Plot2d/Graphic.cpp

7.20 CPGF::Plot2d::GraphicObject Class Reference

Inheritance diagram for CPGF::Plot2d::GraphicObject:



Public Member Functions

- virtual double **x_min** () const =0
- virtual double **x_max** () const =0
- virtual double **y_min** () const =0
- virtual double **y_max** () const =0
- virtual Objects2d::Object2d **miniature** (const AffineSpace::Point2d &pos) const =0
- virtual Scene2d **render_to_scene** (std::function< AffineSpace::Point2d(const AffineSpace::Point2d &P)> transform, const double x_min, const double x_max, const double y_min, const double y_max) const =0

Public Attributes

- std::string **legend**

Static Public Attributes

- constexpr static const double **MINIATURE_HALF_WIDTH** = 0.5

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

- CPGF/Plot2d/GraphicObject.hpp

7.21 Mesh< Cell >::Iterator Class Reference

Iterator object to loop through all the cells of the mesh.

```
#include <Mesh.hpp>
```

Public Member Functions

- **Iterator** ()
  Construct a new Iterator object that points to nullptr.
- Iterator (Cell ∗cell)
  Construct a new Iterator object that points to cell.
- Iterator (const Iterator &J)
  Copy constructor.
- Iterator & operator= (const Iterator &J)
  Assignment operator.
- Iterator operator+ (const int i)
  Returns the cell placed i positions to the right.
- Iterator operator- (const int i)
  Returns the cell placed i positions to the left.
- Iterator operator++ (int)
  Advance one cell to the right in the mesh.
- Iterator & operator++ ()
  Advance one cell to the right in the mesh.
- Iterator operator-- (int)
  Advance one cell to the left in the mesh.
- Iterator & operator-- ()
  Advance one cell to the left in the mesh.
- Cell & operator∗ ()
  Dereference operator returns the cell it poins to.
- Cell ∗ operator-> ()
  Dereference operator returns the cell it poins to.
- operator Cell ∗ () const
  Implicit convertion into cell pointer.
- bool operator== (const Iterator &J)
  Returns true if both iterators point to the same cell and false otherwise.
- bool operator!= (const Iterator &J)
  Returns false if both iterators point to the same cell and true otherwise.

Public Attributes

- Cell ∗ **cell**
  Pointer to the current cell.

7.21.1 Detailed Description

template<class Cell>
class Mesh< Cell >::Iterator

Iterator object to loop through all the cells of the mesh.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 Iterator() [1/2]

```
template<class Cell >
Mesh< Cell >::Iterator::Iterator (
            Cell * cell )
```

Construct a new Iterator object that points to *cell*.

Parameters

cell	

7.21.2.2 Iterator() [2/2]

```
template<class Cell >
Mesh< Cell >::Iterator::Iterator (
            const Iterator & J )
```

Copy constructor.

Parameters

J	

7.21.3 Member Function Documentation

7.21.3.1 operator Cell *()

```
template<class Cell >
Mesh< Cell >::Iterator::operator Cell *
```

Implicit convertion into cell pointer.

Returns

Cell*

7.21.3.2 operator"!=()

```
template<class Cell >
bool Mesh< Cell >::Iterator::operator!= (
            const Iterator & J )
```

Returns false if both iterators point to the same cell and true otherwise.

Parameters

I	
J	

Returns

true

false

7.21.3.3 operator∗()

```
template<class Cell >
Cell & Mesh< Cell >::Iterator::operator*
```

Dereference operator returns the cell it poins to.

Returns

Cell&

7.21.3.4 operator+()

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::Iterator::operator+ (
            const int i )
```

Returns the cell placed i positions to the right.

Parameters

i	

Returns

Iterator&

7.21.3.5 operator++() [1/2]

```
template<class Cell >
Mesh< Cell >::Iterator & Mesh< Cell >::Iterator::operator++
```

Advance one cell to the right in the mesh.

Returns

Iterator&

7.21.3.6 operator++() [2/2]

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::Iterator::operator++ (
            int  )
```

Advance one cell to the right in the mesh.

Returns

Iterator&

7.21.3.7 operator-()

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::Iterator::operator- (
            const int i )
```

Returns the cell placed i positions to the left.

Parameters

i	

Returns

Iterator&

7.21.3.8 operator--() [1/2]

```
template<class Cell >
Mesh< Cell >::Iterator & Mesh< Cell >::Iterator::operator--
```

Advance one cell to the left in the mesh.

Returns

Iterator&

7.21.3.9 operator--() [2/2]

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::Iterator::operator-- (
             int )
```

Advance one cell to the left in the mesh.

Returns

Iterator&

7.21.3.10 operator->()

```
template<class Cell >
Cell * Mesh< Cell >::Iterator::operator->
```

Dereference operator returns the cell it poins to.

Returns

Cell*

7.21.3.11 operator=()

```
template<class Cell >
Mesh< Cell >::Iterator & Mesh< Cell >::Iterator::operator= (
             const Iterator & J )
```

Assignment operator.

Parameters

J	

Returns

Iterator&

7.21.3.12 operator==()

```
template<class Cell >
bool Mesh< Cell >::Iterator::operator== (
             const Iterator & J )
```

Returns true if both iterators point to the same cell and false otherwise.

Parameters

I	
J	

Returns

true

false

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

- Mesh/Mesh.hpp
- Mesh/Mesh.cpp

7.22 Utilities::FileArray< T >::Iterator Class Reference

Public Member Functions

- **Iterator** (FileArray ∗file_array, const unsigned long i=0)
- bool **operator==** (const Iterator &J)
- bool **operator!=** (const Iterator &J)
- **operator T** () const
- Iterator & **operator=** (const T &val)
- Iterator & **operator++** ()
- Iterator **operator++** (int)
- Iterator & **operator--** ()
- Iterator **operator--** (int)
- Iterator & **operator+** (const unsigned long j)
- Iterator & **operator-** (const unsigned long j)

Public Attributes

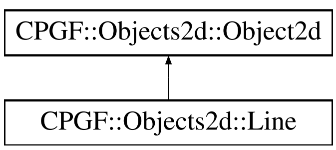
- FileArray ∗ **file_array**
- unsigned long **i**

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

- Utilities/FileArray.hpp
- Utilities/FileArray.cpp

7.23 CPGF::Objects2d::Line Class Reference

Inheritance diagram for CPGF::Objects2d::Line:



Public Member Functions

- **Line** (const AffineSpace::Point2d &A, const AffineSpace::Point2d &B, const Color &color=Color::BLACK, const double opacity=1, const double line_width=LineWidth::SEMITHICK, const std::vector< double > &dash_pattern=DashPatterns::SOLID)
- **Line** (const std::vector< AffineSpace::Point2d > &points, const Color &color=Color::BLACK, const double opacity=1, const double line_width=LineWidth::SEMITHICK, const std::vector< double > &dash_pattern=DashPatterns::SOLID)
- AffineSpace::Point2d & **start** ()
- AffineSpace::Point2d & **end** ()

Static Protected Member Functions

- static Object2d **builder** (const std::vector< AffineSpace::Point2d > &points, const Color &color, const double opacity, const double line_width, const std::vector< double > &dash_pattern)

Additional Inherited Members

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

- CPGF/Objects2d/BasicGeometries.hpp
- CPGF/Objects2d/BasicGeometries.cpp

7.24 Math::Interpolation::LinearInterpolation Class Reference

Public Member Functions

- **LinearInterpolation** (const std::vector< double > &Y, const std::vector< double > &X)
- **LinearInterpolation** (const double ∗Y, const double ∗X, const unsigned int N)
- **LinearInterpolation** (const LinearInterpolation &I)
- LinearInterpolation & **operator=** (const LinearInterpolation &I)
- double **operator()** (const double x) const

Protected Attributes

- unsigned int **N**
- double ∗ **x_part**
- double ∗ **b**
- double ∗ **c**

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

- Math/Interpolation.hpp
- Math/Interpolation.cpp

7.25 Math::AlgebraicSolvers::LinearSystemSolver Class Reference

Solves linear systems of the form Ax=b through LU decomposition with partial pivoting.

```
#include <AlgebraicSolvers.hpp>
```

Public Member Functions

- Vector< double > solve (const Vector< double > &b) const
  Returns the solution of the system Ax=b.
- LinearSystemSolver (const Matrix< double > &A)
  Construct a new Linear System Solver object for the coefficient matrix A. It computes its LU decomposition with partial pivoting.

Protected Attributes

- Matrix< double > **L**
  The lower diagonal matrix of the LU decomposition of A (PA=LU).
- Matrix< double > **U**
  The upper diagonal matrix of the LU decomposition of A (PA=LU).
- std::vector< unsigned int > P
  The permutation vector.

7.25.1 Detailed Description

Solves linear systems of the form Ax=b through LU decomposition with partial pivoting.

For efficiency reasons, the coefficient matrix must be supplied first, then the LU decomposition is done and, afterwars, we may solve any system of the form Ax=b without having to recompute the LU decomposition of A.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 LinearSystemSolver()

```
LinearSystemSolver::LinearSystemSolver (
            const Matrix< double > & A )  [explicit]
```

Construct a new Linear System Solver object for the coefficient matrix A. It computes its LU decomposition with partial pivoting.

Parameters

A	

7.25.3 Member Function Documentation

7.25.3.1 solve()

```
Vector< double > LinearSystemSolver::solve (
            const Vector< double > & b ) const
```

Returns the solution of the system Ax=b.

Parameters

b	the independent vector.

Returns

Vector<double>

7.25.4 Member Data Documentation

7.25.4.1 P

```
std::vector<unsigned int> Math::AlgebraicSolvers::LinearSystemSolver::P  [protected]
```

The permutation vector.

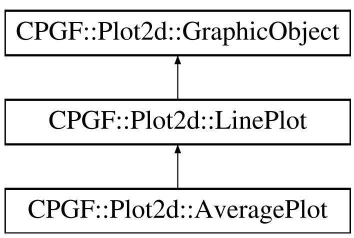
It is used to keep track of the permutations done when applying Gaussian. The i-th component of the permutation vector stores which row of the matrix A is now the i-th row of LU.

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

- Math/AlgebraicSolvers.hpp
- Math/AlgebraicSolvers.cpp

7.26 CPGF::Plot2d::LinePlot Class Reference

Inheritance diagram for CPGF::Plot2d::LinePlot:



Public Member Functions

- **LinePlot** (const std::vector< double > &Y, const std::vector< double > &X, const Color &color=Color::BLUE, const double line_width=LineWidth::THIN, const double opacity=1, const std::vector< double > &dash_pattern=DashPatterns::SOLID, const std::string &legend="")
- **LinePlot** (const std::vector< double > &Y, const std::vector< double > &X, std::function< Color(unsigned int)> color, std::function< double(unsigned int)> line_width, std::function< double(unsigned int)> opacity, std::function< std::vector< double >(unsigned int)> dash_pattern, const std::string &legend="")
- **LinePlot** (const std::vector< double > &Y, const std::vector< double > &X, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> line_width, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::vector< double >(AffineSpace::Point2d &)> dash_pattern, const std::string &legend="")
- **LinePlot** (const std::vector< AffineSpace::Point2d > &data, const Color &color=Color::BLUE, const double line_width=LineWidth::THIN, const double opacity=1, const std::vector< double > &dash_pattern=DashPatterns::SOLID, const std::string &legend="")
- **LinePlot** (const std::vector< AffineSpace::Point2d > &data, std::function< Color(unsigned int)> color, std::function< double(unsigned int)> line_width, std::function< double(unsigned int)> opacity, std::function< std::vector< double >(unsigned int)> dash_pattern, const std::string &legend="")
- **LinePlot** (const std::vector< AffineSpace::Point2d > &data, std::function< Color(AffineSpace::Point2d &)> color, std::function< double(AffineSpace::Point2d &)> line_width, std::function< double(AffineSpace::Point2d &)> opacity, std::function< std::vector< double >(AffineSpace::Point2d &)> dash_pattern, const std::string &legend="")
- double x_min () const override
- double x_max () const override
- double y_min () const override
- double y_max () const override
- Objects2d::Object2d miniature (const AffineSpace::Point2d &pos) const override
- Scene2d render_to_scene (std::function< AffineSpace::Point2d(const AffineSpace::Point2d &P)> transform, const double x_min, const double x_max, const double y_min, const double y_max) const override

Public Attributes

- std::vector< AffineSpace::Point2d > **points**
- std::function< Color(unsigned int)> **color**
- std::function< double(unsigned int)> **line_width**
- std::function< double(unsigned int)> **opacity**
- std::function< std::vector< double >(unsigned int)> **dash_pattern**

Protected Attributes

- bool **const_parameters**

Additional Inherited Members

7.26.1 Member Function Documentation

7.26.1.1 miniature()

```
Object2d LinePlot::miniature (
            const AffineSpace::Point2d & pos ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.26.1.2 render_to_scene()

```
Scene2d LinePlot::render_to_scene (
            std::function< AffineSpace::Point2d(const AffineSpace::Point2d &P)> transform,
            const double x_min,
            const double x_max,
            const double y_min,
            const double y_max ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.26.1.3 x_max()

```
double LinePlot::x_max ( ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.26.1.4 x_min()

```
double LinePlot::x_min ( ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.26.1.5 y_max()

```
double LinePlot::y_max ( ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

7.26.1.6 y_min()

```
double LinePlot::y_min ( ) const  [override], [virtual]
```

Implements CPGF::Plot2d::GraphicObject.

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

- CPGF/Plot2d/LinePlot.hpp
- CPGF/Plot2d/LinePlot.cpp

7.27 CPGF::LineWidth Class Reference

This class provides a handful of useful predefined constants to express line width.

```
#include <PGFConf.hpp>
```

Static Public Attributes

- static const double **ULTRA_THIN** = 0.025
- static const double **VERY_THIN** = 0.05
- static const double **THIN** = 0.1
- static const double **SEMITHICK** = 0.2
- static const double **THICK** = 0.4
- static const double **VERY_THICK** = 0.8
- static const double **ULTRA_THICK** = 1.2

7.27.1 Detailed Description

This class provides a handful of useful predefined constants to express line width.

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

- CPGF/PGFBasics/PGFConf.hpp
- CPGF/PGFBasics/PGFConf.cpp

7.28 Math::Matrix< K > Class Template Reference

Public Member Functions

- **Matrix** (unsigned int m=0, unsigned int n=0)
- **Matrix** (unsigned int m, unsigned int n, std::vector< K > elements)
- **Matrix** (const Matrix< K > &B)
- Matrix & **operator=** (const Matrix< K > &B)
- **operator Matrix< std::complex< K > >** () const
- Matrix< K > & **operator+=** (const K &alpha)
- Matrix< K > & **operator-=** (const K &alpha)
- Matrix< K > & **operator*=** (const K &alpha)
- Matrix< K > & **operator/=** (const K &alpha)
- Vector< K > & operator() (const unsigned int i, const std::string &j)
- Vector< K > operator() (const unsigned int i, const std::string &j) const
- Vector< K > operator() (const std::string &i, const unsigned int j) const
- K & **operator()** (const unsigned int i, const unsigned int j)
- K **operator()** (const unsigned int i, const unsigned int j) const
- unsigned int m () const

    Horizontal dimension.
- unsigned int n () const

    Vertical dimension.
- bool is_square () const

    Returns true if m==n.
- void **LU** (Matrix< K > &L, Matrix< K > &U, std::vector< unsigned int > &P) const
- K det () const

    Calculates determinant.
- K tr () const

    Calculates trace.
- std::vector< std::complex< double > > eigenvalues () const
- std::vector< Vector< std::complex< double > > > eigenvectors (std::vector< std::complex< double > > &eigenvalues, const bool calculate_eigenvalues=true) const

    Returns eigenvectors.
- std::string **to_string** () const

Static Public Member Functions

- static Matrix< K > **zero** (const unsigned int m, const unsigned int n)
- static Matrix< K > **zero** (const unsigned int n)
- static Matrix< K > **identity** (const unsigned int n)

Protected Attributes

- Vector< K > * **vec**
- unsigned int **_m**

Friends

- Matrix< K > **operator+** (const Matrix< K > &A, const Matrix< K > &B)
- Matrix< K > **operator+** (const Matrix< K > &A, const K &alpha)
- Matrix< K > **operator+** (const K &alpha, const Matrix< K > &A)
- Matrix< K > **operator-** (const Matrix< K > &A, const Matrix< K > &B)
- Matrix< K > **operator-** (const Matrix< K > &A, const K &alpha)
- Matrix< K > **operator-** (const K &alpha, const Matrix< K > &A)
- Matrix< K > **operator*** (const Matrix< K > &A, const K &alpha)
- Matrix< K > **operator*** (const K &alpha, const Matrix< K > &A)
- Matrix< K > **operator/** (const Matrix< K > &A, const K &alpha)
- Matrix< K > **operator|** (const Matrix< K > &A, const Matrix< K > &B)
- Vector< K > **operator|** (const Matrix< K > &A, const Vector< K > &v)
- Vector< K > **operator|** (const Vector< K > &v, const Matrix< K > &A)

7.28.1 Member Function Documentation

7.28.1.1 det()

```
template<typename K >
K Matrix::det
```

Calculates determinant.

Returns

double

7.28.1.2 eigenvalues()

```
template<typename K >
std::vector< std::complex< double > > Matrix::eigenvalues
```

Returns

std::vector<std::complex<double>>

7.28.1.3 eigenvectors()

```
template<typename K >
std::vector< Vector< std::complex< double > > > Matrix::eigenvectors (
            std::vector< std::complex< double > > & eigenvalues,
            const bool calculate_eigenvalues = true ) const
```

Returns eigenvectors.

May also return eigenvalues through argument.

Parameters

calculate_eigenvalues	
eigenvalues	

Returns

std::vector<Vector<std::complex<double>>>

7.28.1.4 is_square()

```
template<typename K >
bool Matrix::is_square
```

Returns true if m==n.

Returns

true

false

7.28.1.5 m()

```
template<typename K >
unsigned int Matrix::m
```

Horizontal dimension.

Returns

unsigned int

7.28.1.6 n()

```
template<typename K >
unsigned int Matrix::n
```

Vertical dimension.

Returns

unsigned int

7.28.1.7 operator()() [1/3]

```
template<typename K >
Vector< K > Matrix::operator() (
            const std::string & i,
            const unsigned int j ) const
```

Returns column j.

7.28.1.8 operator()() [2/3]

```
template<typename K >
Vector< K > & Matrix::operator() (
            const unsigned int i,
            const std::string & j )
```

Returns row i.

7.28.1.9 operator()() [3/3]

```
template<typename K >
Vector< K > Matrix::operator() (
            const unsigned int i,
            const std::string & j ) const
```

Returns row i.

7.28.1.10 tr()

```
template<typename K >
K Matrix::tr
```

Calculates trace.

Returns

double

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

- Math/Matrix.hpp
- Math/Matrix.cpp

7.29 Mesh< Cell > Class Template Reference

This object represents a collection of cells. The number of cells used can be changed. Methods for adaptive refinement are included.

```
#include <Mesh.hpp>
```

Classes

- class Iterator

  Iterator object to loop through all the cells of the mesh.

Public Member Functions

- unsigned int N_cells () const

  Returns total number of cells.
- Mesh (Cell ∗first_cell=nullptr, Cell ∗last_cell=nullptr, const Chemistry::SolidGasReaction &QR=Chemistry::SolidGasReaction(), std::function< double(const double x)> A_func=[ ](double x){return 0;}, const double detail_subdivide_threshold=0.↩ 01, const double detail_merge_threshold=0.001, const double max_length_factor=1./50, const double min_length_factor=1./2000, const double boundary_cell_max_length_factor=1./1000)

  Construct a new Mesh object.
- Mesh (const Mesh &mesh)

  Perfomrs a deep copy of mesh .
- Mesh (Mesh &&mesh)

  Move constructor.
- Mesh & operator= (const Mesh &mesh)

  Assignment operator.
- ∼**Mesh** ()

  Destroy the Mesh object.
- void **free** ()

  Frees memory associated to the mesh by deleting all cells.
- Mesh::Iterator begin () const

  Returns an iterator to the first cell.
- Mesh::Iterator rbegin () const

  Returns an iterator to the last cell.
- Mesh::Iterator end () const

  Returns an iterator to a virtual cell after the last one (or before the first one).
- Cell ∗ subdivide_at (Cell ∗C)

  Subdivides cell C. Returns a pointer to the new right cell.
- Cell ∗ merge_cells (Cell ∗L, Cell ∗R)

  Merges two cells into one. Returns a pointer to the new cell.
- void **calculate_variable_ranges** ()

  Calculates the maximum and minimum value of the norm of the vector of conserved quatities throughout the mesh. Then, substacts the maximum value and the minimum value, saving the result in the member ranges.
- double detail (Cell ∗L, Cell ∗R) const

  This is function to estimate how different are the values stored in two neighbouring cells.
- void optimize_mesh ()

  Loops through the mesh and subdivides and merges cells according to the thresholds.
- std::vector< double > x () const

  Returns the average position of all cells in the mesh.
- std::vector< double > x_partition () const

  Returns the partition of the X axis given by the boundaries of the cells.
- std::vector< double > A () const

  Returns the area of all cells in the mesh.
- std::vector< Math::Vector< double > > U () const

  Returns the vector of conserved variables of all cells in the mesh.
- void read_from_file (FILE ∗file)

  Writes the mesh to file.
- void write_to_file (FILE ∗file) const

  Reads the mesh from file.

Public Attributes

- Cell * **first_cell**

  Pointer to the first cell of the mesh.
- Cell * **last_cell**

  Pointer to the last cell of the mesh.
- Chemistry::SolidGasReaction **QR**

  Chemical reaction object.
- double **detail_subdivide_threshold**

  If a cell detail is bigger than this number, we divide the cell in two.
- double **detail_merge_threshold**

  If the details of two neighbouring cells are lower than this number, they are merged.
- double **max_length_factor**

  Determines the maximum size of a cell. A cell cannot be greater than max_length_factor times the length of the computational domain.
- double **min_length_factor**

  Determines the minimum size of a cell. A cell cannot be smaller than min_length_factor times the length of the computational domain.
- double **boundary_cell_max_length_factor**

  The max length factor for cells closer than 4 cells to the boundary.
- std::function< double(const double x)> **A_func**

  Function used to compute the area of the newly created cells when using adaptive refinement.

Protected Attributes

- double **ranges**

  The difference between the maximum and the minimum norm of the vector of preserved quantities is stored here.

7.29.1 Detailed Description

template<class Cell>
class Mesh< Cell >

This object represents a collection of cells. The number of cells used can be changed. Methods for adaptive refinement are included.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 Mesh() [1/3]

```
template<class Cell >
Mesh< Cell >::Mesh (
            Cell * first_cell = nullptr,
            Cell * last_cell = nullptr,
            const Chemistry::SolidGasReaction & QR = Chemistry::SolidGasReaction(),
            std::function< double(const double x)> A_func = [](double x){return 0;},
            const double detail_subdivide_threshold = 0.01,
            const double detail_merge_threshold = 0.001,
            const double max_length_factor = 1./50,
            const double min_length_factor = 1./2000,
            const double boundary_cell_max_length_factor = 1./1000 )
```

Construct a new Mesh object.

Parameters

first_cell	pointer to the first cell.
last_cell	pointer to the last cell.
QR	chemical reaction.
A_func	area function.
detail_subdivide_threshold	determines when cells are subdivided.
detail_merge_threshold	determines when cells are merged.
max_length_factor	limits the maximum length of a created cell.
min_length_factor	limits the minimum length of a created cell.
boundary_cell_max_length_factor	limits the maxium length of cells near the boundary.

7.29.2.2 Mesh() [2/3]

```
template<class Cell >
Mesh< Cell >::Mesh (
            const Mesh< Cell > & mesh )
```

Perfomrs a deep copy of *mesh* .

Parameters

mesh	

7.29.2.3 Mesh() [3/3]

```
template<class Cell >
Mesh< Cell >::Mesh (
            Mesh< Cell > && mesh )
```

Move constructor.

Parameters

mesh	

7.29.3 Member Function Documentation

7.29.3.1 A()

```
template<class Cell >
std::vector< double > Mesh< Cell >::A
```

Returns the area of all cells in the mesh.

Returns

std::vector<double>

7.29.3.2 begin()

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::begin
```

Returns an iterator to the first cell.

Returns

Mesh::Iterator

7.29.3.3 detail()

```
template<class Cell >
double Mesh< Cell >::detail (
            Cell * L,
            Cell * R ) const
```

This is function to estimate how different are the values stored in two neighbouring cells.

Parameters

L	
R	

Returns

double

7.29.3.4 end()

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::end
```

Returns an iterator to a virtual cell after the last one (or before the first one).

Returns

Mesh::Iterator

7.29.3.5 merge_cells()

```
template<class Cell >
Cell * Mesh< Cell >::merge_cells (
            Cell * L,
            Cell * R )
```

Merges two cells into one. Returns a pointer to the new cell.

Parameters

C1	
C2	

Returns

Mesh&

7.29.3.6 N_cells()

```
template<class Cell >
unsigned int Mesh< Cell >::N_cells
```

Returns total number of cells.

Returns

unsigned int

7.29.3.7 operator=()

```
template<class Cell >
Mesh< Cell > & Mesh< Cell >::operator= (
            const Mesh< Cell > & mesh )
```

Assignment operator.

Parameters

mesh	

Returns

Mesh&

7.29.3.8 optimize_mesh()

```
template<class Cell >
void Mesh< Cell >::optimize_mesh
```

Loops through the mesh and subdivides and merges cells according to the thresholds.

Returns

Mesh&

7.29.3.9 rbegin()

```
template<class Cell >
Mesh< Cell >::Iterator Mesh< Cell >::rbegin
```

Returns an iterator to the last cell.

Returns

Mesh::Iterator

7.29.3.10 read_from_file()

```
template<class Cell >
void Mesh< Cell >::read_from_file (
             FILE * file )
```

Writes the mesh to file.

Parameters

file	

7.29.3.11 subdivide_at()

```
template<class Cell >
Cell * Mesh< Cell >::subdivide_at (
             Cell * C )
```

Subdivides cell C. Returns a pointer to the new right cell.

Parameters

C	

7.29.3.12 U()

```
template<class Cell >
std::vector< Vector< double > > Mesh< Cell >::U
```

Returns the vector of conserved variables of all cells in the mesh.

Returns

std::vector<Math::Vector<double>>

7.29.3.13 write_to_file()

```
template<class Cell >
void Mesh< Cell >::write_to_file (
            FILE * file ) const
```

Reads the mesh from file.

Parameters

file	

7.29.3.14 x()

```
template<class Cell >
std::vector< double > Mesh< Cell >::x
```

Returns the average position of all cells in the mesh.

Returns

std::vector<double>

7.29.3.15 x_partition()

```
template<class Cell >
std::vector< double > Mesh< Cell >::x_partition
```

Returns the partition of the X axis given by the boundaries of the cells.

Returns

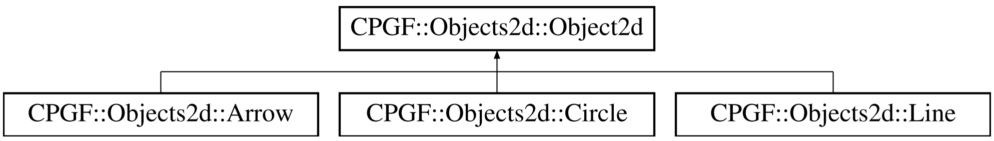
std::vector<double>

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

- Mesh/Mesh.hpp
- Mesh/Mesh.cpp

7.30 CPGF::Objects2d::Object2d Class Reference

Inheritance diagram for CPGF::Objects2d::Object2d:



Public Member Functions

- **Object2d** (const Basics::Path2d &path)
- **Object2d** (const std::vector< Basics::Path2d > &paths)
- unsigned int **size** () const
- Object2d & **translate** (const AffineSpace::Vector2d &v)
- Object2d & **rotate_with_respect_to** (const AffineSpace::Point2d &Q, const double theta)
- Object2d & **scale_with_respect_to** (const AffineSpace::Point2d &Q, const AffineSpace::Vector2d &s)
- Object2d & **operator+=** (const Object2d &B)
- Object2d & **operator+=** (const Basics::Path2d &path)
- std::string **render_to_string** () const

Public Attributes

- std::vector< Basics::Path2d > **paths**

Friends

- Object2d **operator+** (const Object2d &A, const Object2d &B)

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

- CPGF/Objects2d/Object2d.hpp
- CPGF/Objects2d/Object2d.cpp

7.31 CPGF::Basics::Path2d Class Reference

Public Member Functions

- **Path2d** (SimpleStroke2d &stroke, const PGFConf &conf=PGFConf())
- **Path2d** (const std::vector< SimpleStroke2d * > &strokes, const PGFConf &conf=PGFConf())
- **Path2d** (const Path2d &path)
- Path2d & **operator=** (const Path2d &path)
- Path2d & **translate** (const AffineSpace::Vector2d &v)
- Path2d & **rotate_with_respect_to** (const AffineSpace::Point2d &Q, const double theta)
- Path2d & **scale_with_respect_to** (const AffineSpace::Point2d &Q, const AffineSpace::Vector2d &v)
- AffineSpace::Point2d & **start** ()
- AffineSpace::Point2d **start** () const
- AffineSpace::Point2d & **end** ()
- AffineSpace::Point2d **end** () const
- double **length** () const
- double **area** () const
- Path2d & **operator+=** (SimpleStroke2d &stroke)
- Path2d & **add_stroke** (SimpleStroke2d &stroke)
- unsigned int **size** () const
- std::string **render_to_string** () const

Public Attributes

- std::vector< SimpleStroke2d * > **strokes**
- PGFConf **conf**

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

- CPGF/PGFBasics/Path2d.hpp
- CPGF/PGFBasics/Path2d.cpp

7.32 CPGF::PGFConf Class Reference

This object is used to store all information needed to draw a path.

`#include <PGFConf.hpp>`

Public Member Functions

- **PGFConf** (const DrawType draw_type=DrawType::DRAW, const Color &color=Color::BLACK, const double opacity=1, const double line_width=LineWidth::SEMITHICK, const std::vector< double > &dash_pattern=std::vector< double >(), const double dash_phase=0, const LineCap line_cap=LineCap::BUTT, const LineJoin line_join=LineJoin::BEVEL)

Public Attributes

- DrawType **draw_type**
- LineCap **line_cap**
- LineJoin **line_join**
- Color **color**
- std::vector< double > dash_pattern

  This double array is used to decide whether the line drawn is continuous or discontinuous (i.e. it has gaps).
- double dash_phase
- double **opacity**

  This determines how transparent the path drawn is. It must be a real number between zero and one. One means it is fully opaque and zero represents full transparency.
- double **line_width**

  This determines the thickness of the line drawn. For default predefined values, you can use the class LineWidth.

7.32.1 Detailed Description

This object is used to store all information needed to draw a path.

7.32.2 Member Data Documentation

7.32.2.1 dash_pattern

```
std::vector<double> CPGF::PGFConf::dash_pattern
```

This double array is used to decide whether the line drawn is continuous or discontinuous (i.e. it has gaps).

Todo Explain this better.

7.32.2.2 dash_phase

```
double CPGF::PGFConf::dash_phase
```

Todo Explain this.

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

- CPGF/PGFBasics/PGFConf.hpp
- CPGF/PGFBasics/PGFConf.cpp

7.33 CPGF::AffineSpace::Point2d Class Reference

A point of a 2D affine space.

`#include <Point2d.hpp>`

Public Member Functions

- Point2d & operator+= (const Vector2d &v)
  Traslates the point by the vector v.
- double angle_with_respect_to (const Point2d &Q)
  Calculates the angle the point Q has with respect to P.
- Point2d & rotate_with_respect_to (const Point2d &Q, const double theta)
  Rotates the point (∗this) an angle theta around the point Q.
- Point2d & rotate_to_with_respect_to (const Point2d &Q, const double theta)
  Rotates the point (∗this) to a fixed angle theta around the point Q.
- Point2d & scale_with_respect_to (const Point2d &Q, const Vector2d &s)
  The position of ∗this with respect to the point Q is scaled according to the components of the vector s.
- std::string to_string () const
  Retruns a string representation of the point.
- **Point2d** ()
  Returns the origin of coordinates.
- Point2d (double x, double y)
  Retruns the point (x,y).

Public Attributes

- double **x**
  The x coordinate of the point.
- double **y**
  The y coordinate of the point.

Friends

- Point2d operator+ (const Point2d &P, const Vector2d &v)
  Function that adds a vector v to the point P.
- Vector2d operator- (const Point2d &P, const Point2d &Q)
  Function that returns the vector that joins two points.
- bool operator== (const Point2d &P, const Point2d &Q)
  Returns whether P and Q are closer than 1e-10.

7.33.1 Detailed Description

A point of a 2D affine space.

As usual, a vector may be added to a point and two points may be "substracted" to obtain a vector.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 Point2d()

```
Point2d::Point2d (
            double x,
            double y )
```

Retruns the point (x,y).

Parameters

x	
y	

7.33.3 Member Function Documentation

7.33.3.1 angle_with_respect_to()

```
double Point2d::angle_with_respect_to (
            const Point2d & Q )
```

Calculates the angle the point Q has with respect to P.

Returns the angle between the horizontal line that passes through P and the line that joins P and Q.

Parameters

Q	a 2D point.

Returns

double. A real number between 0 and 2π.

7.33.3.2 operator+=()

```
Point2d & Point2d::operator+= (
            const Vector2d & v )
```

Traslates the point by the vector v.

Equivalent to $*$this = $*$this + v;

Parameters

v	a 2D vector.

Returns

Point2d&

7.33.3.3 rotate_to_with_respect_to()

```
Point2d & Point2d::rotate_to_with_respect_to (
            const Point2d & Q,
            const double theta )
```

Rotates the point ($*$this) to a fixed angle theta around the point Q.

Using Q as the center of rotation, the line that joins P and Q is rotated around Q until the angle between that line and the horizontal line that passes through Q is exactly theta. As a consequence, the position of P changes, although the distance PQ is preserved.

Parameters

Q	a 2D point.
theta	a double. A real number between 0 and 2π.

Returns

Point2d&

7.33.3.4 rotate_with_respect_to()

```
Point2d & Point2d::rotate_with_respect_to (
            const Point2d & Q,
            const double theta )
```

Rotates the point ($*$this) an angle theta around the point Q.

Using Q as the center of rotation, the line that joins P and Q is rotates around Q an angle theta. As a consequence, the position of P changes, although the distance PQ is preserved.

Parameters

Q	a 2D point.
theta	a double. A real number between 0 and 2π.

Returns

Point2d&

7.33.3.5 scale_with_respect_to()

```
Point2d & Point2d::scale_with_respect_to (
            const Point2d & Q,
            const Vector2d & s )
```

The position of *this with respect to the point Q is scaled according to the components of the vector s.

First, we express the point P through its coordinates with respect to the point Q. Then, those coordinates are multiplied componentwise by the vector s. Finally, the new P is expressed through its coordinates with respect to the origin.

Parameters

Q	
s	

Returns

Point2d&

7.33.3.6 to_string()

```
std::string Point2d::to_string ( ) const
```

Retruns a string representation of the point.

Returns

std::string

7.33.4 Friends And Related Function Documentation

7.33.4.1 operator+

```
Point2d operator+ (
            const Point2d & P,
            const Vector2d & v )  [friend]
```

Function that adds a vector v to the point P.

Returns the position of the end of the vector v when its start is placed at the point P.

Parameters

P	a 2D point.
v	a 2D vector.

Returns

Point2D

7.33.4.2 operator-

```
Vector2d operator- (
            const Point2d & P,
            const Point2d & Q )  [friend]
```

Function that returns the vector that joins two points.

Returns the vector whose start is at point P and whose end is at point Q.

Parameters

P	a 2D point.
Q	a 2D point.

Returns

Vector2d

7.33.4.3 operator==

```
bool operator== (
            const Point2d & P,
            const Point2d & Q )  [friend]
```

Returns whether P and Q are closer than 1e-10.

Parameters

P	
Q	

Returns

true

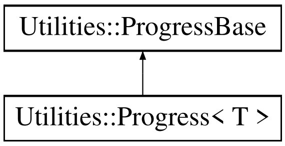
false

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

- CPGF/AffineSpace2d/Point2d.hpp
- CPGF/AffineSpace2d/Point2d.cpp

7.34 Utilities::Progress< T > Class Template Reference

Inheritance diagram for Utilities::Progress< T >:



Public Member Functions

- **Progress** (const std::string &name="", ProgressEstimation estimation=ProgressEstimation::LINEAR, const T initial=T(), const T objective=T())
- **Progress** (Progress< T > &progress)
- Progress< T > & **operator=** (const Progress< T > &progress)
- std::string report (const unsigned int level) const override
- Progress< T > & **operator=** (const T &val)
- Progress< T > & **operator+=** (const T &val)
- Progress< T > & **operator++** ()
- Progress< T > & **operator-=** (const T &val)
- Progress< T > & **operator--** ()
- Progress< T > & **operator∗=** (const T &val)
- Progress< T > & **operator/=** (const T &val)
- Progress< T > & **operator%=** (const T &val)
- **operator std::atomic< T > &** ()
- **operator T** () const

Protected Attributes

- T **initial**
- T **objective**
- std::atomic< T > **current**

Additional Inherited Members

7.34.1 Member Function Documentation

7.34.1.1 report()

```
template<class T >
std::string Progress::report (
            const unsigned int level ) const  [override], [virtual]
```

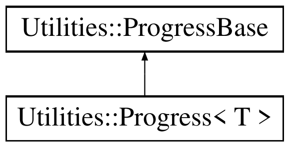
Reimplemented from Utilities::ProgressBase.

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

- Utilities/Progress.hpp
- Utilities/Progress.cpp

7.35 Utilities::ProgressBase Class Reference

Inheritance diagram for Utilities::ProgressBase:



Public Member Functions

- void **start** ()
- void **pause** ()
- void **resume** ()
- void **finish** ()
- virtual std::string **report** (const unsigned int level) const
- **ProgressBase** (const std::string &name="", const ProgressEstimation estimation=ProgressEstimation::↩ LINEAR)
- **ProgressBase** (const ProgressBase &progress)
- ProgressBase & **operator=** (const ProgressBase &progress)
- ProgressBase & **operator[ ]** (std::string name)
- void **add_child** (ProgressBase &progress)
- void **eliminate_child** (ProgressBase &progress)
- void **update_to_terminal** (unsigned int period=250)

Public Attributes

- ProgressStatus **status**

Protected Attributes

- ProgressEstimation **estimation**
- std::string **name**
- std::chrono::system_clock::time_point **t_start**
- std::chrono::system_clock::time_point **t_paused**
- std::chrono::system_clock::time_point **t_finish**
- std::chrono::system_clock::duration **inactive_time**
- std::map< std::string, ProgressBase * > **children**
- std::mutex **M**
- std::future< void > **task**

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

- Utilities/Progress.hpp
- Utilities/Progress.cpp

7.36 Math::Rational< K > Class Template Reference

Public Member Functions

- **Rational** (const K num)
- **Rational** (const K num, const K den)
- std::string **to_string** () const

Public Attributes

- K num
- K den

Friends

- Rational< K > & **operator+** (const Rational< K > &p, const Rational< K > &q)
- Rational< K > & **operator-** (const Rational< K > &p, const Rational< K > &q)
- Rational< K > & **operator*** (const Rational< K > &p, const Rational< K > &q)
- Rational< K > & **operator/** (const Rational< K > &p, const Rational< K > &q)
- bool **operator==** (const Rational< K > &p, const Rational< K > &q)
- bool **operator!=** (const Rational< K > &p, const Rational< K > &q)
- bool **operator<** (const Rational< K > &p, const Rational< K > &q)
- bool **operator<=** (const Rational< K > &p, const Rational< K > &q)
- bool **operator>** (const Rational< K > &p, const Rational< K > &q)
- bool **operator>=** (const Rational< K > &p, const Rational< K > &q)

7.36.1 Member Data Documentation

7.36.1.1 den

```
template<typename K >
K Math::Rational< K >::den
```

Denominator.

7.36.1.2 num

```
template<typename K >
K Math::Rational< K >::num
```

Numerator.

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

- Math/Rational.hpp
- Math/Rational.cpp

7.37 CPGF::Scene2d Class Reference

Public Member Functions

- **Scene2d** (const std::vector< Objects2d::Object2d * > &objects=std::vector< Objects2d::Object2d * >(), const std::vector< Text * > &texts=std::vector< Text * >())
- Scene2d & **add** (Objects2d::Object2d &object)
- Scene2d & **add** (Text &texts)
- Scene2d friend **operator+** (const Scene2d &S1, const Scene2d &S2)
- Scene2d & **operator+=** (const Scene2d &S2)
- Scene2d & **operator+=** (Objects2d::Object2d &Obj)
- Scene2d & **operator+=** (Text &text)
- std::string **render_to_string** () const
- void **render** (const std::string &filename, const unsigned int density=100) const

Public Attributes

- std::vector< Objects2d::Object2d * > **objects**
- std::vector< Text * > **texts**

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

- CPGF/Scene2d.hpp
- CPGF/Scene2d.cpp

7.38 CPGF::Plot2d::Shapes Class Reference

Static Public Member Functions

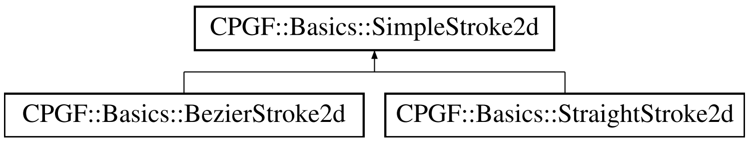
- static Objects2d::Object2d **Circle** (const AffineSpace::Point2d &pos, const Color &color, const double opacity, const double size)
- static Objects2d::Object2d **Square** (const AffineSpace::Point2d &pos, const Color &color, const double opacity, const double size)

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

- CPGF/Plot2d/DataPlot.hpp
- CPGF/Plot2d/DataPlot.cpp

7.39 CPGF::Basics::SimpleStroke2d Class Reference

Inheritance diagram for CPGF::Basics::SimpleStroke2d:



Public Member Functions

- virtual SimpleStroke2d ∗ **clone** () const =0
- virtual AffineSpace::Point2d & **start** ()=0
- virtual AffineSpace::Point2d **start** () const =0
- virtual AffineSpace::Point2d & **end** ()=0
- virtual AffineSpace::Point2d **end** () const =0
- virtual SimpleStroke2d & **translate** (const AffineSpace::Vector2d &v)=0
- virtual SimpleStroke2d & **rotate_with_respect_to** (const AffineSpace::Point2d &Q, const double theta)=0
- virtual SimpleStroke2d & **scale_with_respect_to** (const AffineSpace::Point2d &Q, const AffineSpace::Vector2d &s)=0
- virtual double **length** () const =0
- virtual double **area** () const =0
- virtual std::vector< AffineSpace::Point2d > **operator/** (const SimpleStroke2d &B)=0
- virtual std::string render_to_string () const =0

7.39.1 Member Function Documentation

7.39.1.1 render_to_string()

```
virtual std::string CPGF::Basics::SimpleStroke2d::render_to_string ( ) const  [pure virtual]
```

Parameters

alpha	a number between zero and one.

Returns

AffineSpace::Point2d

Implemented in CPGF::Basics::StraightStroke2d, and CPGF::Basics::BezierStroke2d.

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

- CPGF/PGFBasics/Strokes2d.hpp

7.40 Simulation Class Reference

This object is used to create some initial conditions and let the domain evolve in time.

```
#include <Simulation.hpp>
```

Public Member Functions

- Simulation (const std::string &name, const Chemistry::SolidGasReaction &QR, const double a, const double b, const unsigned int N_cells_solid, const unsigned int N_cells_gas, const bool adaptive_refinement_solid, const bool adaptive_refinement_gas, const unsigned int N_tasks, const double CFL, const unsigned int N_saves, const SolidBoundaryConditions &solid_BC, const GasBoundaryConditions &gas_BC, const double x_q, std::function< double(double x)> v, std::function< double(double x)> P, std::function< double(double x)> T, std::function< double(double x)> A, std::function< void(const GasCell &A, const GasCell &B, Math::Vector< double > ∗F, double ∗S_max)> convection_solver, std::function< void(SolidMesh &mesh, double &dt, const double CFL, const SolidBoundaryConditions BC, const double t)> diffusion_solver, std::function< double(std::function< double(double x)> f, const double a, const double b)> integrator=[ ](std::function< double(double x)> f, const double a, const double b) {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);}, std::function< double(const GasCell &C, const double t)> external_forces=[ ](const GasCell &C, const double t){return 0.;}, const unsigned int adaptive_refinement_period=1, const double detail_subdivide_threshold=0.01, const double detail_merge_threshold=0.001, const double max_length_factor=1./50, const double min_length_factor=1./2000, const double boundary_cell_max_length_factor=1./1000)

  Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type BOTH.
- Simulation (const std::string &name, const Chemistry::SolidGasReaction &QR, const double a, const double b, const unsigned int N_cells, const bool adaptive_refinement, const unsigned int N_tasks, const double CFL, const unsigned int N_saves, const GasBoundaryConditions &BC, std::function< double(double x)> rho, std::function< double(double x)> v, std::function< double(double x)> P, std::function< double(double x)> A, std::function< void(const GasCell &A, const GasCell &B, Math::Vector< double > ∗F, double ∗S_max)> convection_solver, std::function< double(std::function< double(double x)> f, const double a, const double b)> integrator=[ ](std::function< double(double x)> f, const double a, const double b) {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);}, std::function< double(const GasCell &C, const double t)> external_forces=[ ](const GasCell &C, const double t){return 0.;}, const unsigned int adaptive_refinement_period=1, const double detail_subdivide_threshold=0.01, const double detail_merge_threshold=0.001, const double max_length_factor=1./50, const double min_length_factor=1./2000, const double boundary_cell_max_length_factor=1./1000)

  Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type GAS.

- Simulation (const std::string &name, const Chemistry::SolidGasReaction &QR, const double a, const double b, const unsigned int N_cells, const bool adaptive_refinement, const unsigned int N_tasks, const double CFL, const unsigned int N_saves, const SolidBoundaryConditions &BC, std::function< double(double x)> T, std::function< double(double x)> A, std::function< void(SolidMesh &mesh, double &dt, const double CFL, const SolidBoundaryConditions BC, const double t)> diffusion_solver, std::function< double(std::function< double(double x)> f, const double a, const double b)> integrator=[ ](std::function< double(double x)> f, const double a, const double b) {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);}, const unsigned int adaptive_refinement_period=1, const double detail_subdivide_threshold=0.01, const double detail_merge_threshold=0.001, const double max_length_factor=1./50, const double min_length_factor=1./2000, const double boundary_cell_max_length_factor=1./1000)

    Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type SOLID.
- Simulation (const std::string &file)

    Construct a new Simulation object from a file.
- void update (double dt=std::numeric_limits< double >::max())

    Compute a new time step.
- void simulate_until (const double t)

    Runs the simulation until time t has been reached. The last computed time is exactly t.
- void write_to_file (const std::string &file) const

    Writes the simulation into a file.
- double x_q (const double t) const

    Returns the position of the combustion at a specific time t.
- double v_q (const double t) const

    Returns the burning rate at a specific time t.
- std::function< double(const double x)> A (const double t) const

    Returns the function A(x) for a certain time t.
- std::function< double(const double x)> rho (const double t) const

    Returns the function rho(x) for a certain time t.
- std::function< double(const double x)> c (const double t) const

    Returns the function c(x) for a certain time t.
- std::function< double(const double x)> v (const double t) const

    Returns the function v(x) for a certain time t.
- std::function< double(const double x)> T (const double t) const

    Returns the function T(x) for a certain time t.
- std::function< double(const double x)> P (const double t) const

    Returns the function P(x) for a certain time t.
- std::function< double(const double x)> M (const double t) const

    Returns the function M(x) for a certain time t.
- CPGF::Plot2d::Graphic * mesh_plot () const

    Returns a graphic where the cell boundaries are drawn as a function of time.

Public Attributes

- GasMesh **instant_gas_mesh**

    Gas Mesh at the current time.
- SolidMesh **instant_solid_mesh**

    Solid Mesh at the current time.
- double **instant_v_q**

    The burning rate at the current time.
- double **instant_x_q**

    The position of the combustion front at the current time.
- double **instant_t**

Current time.

- unsigned int **refine**

  The value of this variable varies from 0 to the adaptive_refinement_period - 1 and is used to determine when to refine the mesh. Its value is incremented once per iteration.

- std::string **name**

  The name of the simulation.

- SimulationType **simulation_type**

  Stores which type of simulation we have to solve.

- Utilities::FileArray< GasMesh > ∗ **gas_mesh**

  An array of gas meshes for each time instant t. It is stored in a file.

- Utilities::FileArray< SolidMesh > ∗ **solid_mesh**

  An array of solid meshes for each time instant t. It is stored in a file.

- std::vector< double > **x_q_array**

  An array which contains the position of the combustion front for each time t.

- std::vector< double > **v_q_array**

  An array which contains the burning rate for each time t.

- std::vector< double > **t_array**

  An array of all computed times.

- GasBoundaryConditions **gas_BC**

  Gas boundary conditions.

- SolidBoundaryConditions **solid_BC**

  Solid boundary conditions.

- double **CFL**

  Courant-Friedrichs-Lewy number.

- unsigned int N_saves

  Determines how many times during the simulation the current meshes and values are saved to the file.

- bool **adaptive_refinement_solid**

  Determines whether to use adaptive refinement for the solid.

- bool **adaptive_refinement_gas**

  Determines whether to use adaptive refinement for the gas.

- unsigned int adaptive_refinement_period

  Controls how often the mesh is refined. In particular, it represents how many iterations we wait after the mesh has been refined to refine it again.

- unsigned int **N_tasks**

  How many (CPU) tasks to use in other to execute the simulation.

- std::function< double(const GasCell &C, const double t)> external_forces

  Function that computes the external forces acting on each gas cell.

- std::function< void(const GasCell &A, const GasCell &B, Math::Vector< double > ∗F, double ∗S_max)> **convection_solver**

  Function that returns the numerical flux at the interface between gas cells A and B.

- std::function< void(SolidMesh &mesh, double &dt, const double CFL, const SolidBoundaryConditions BC, const double t)> **diffusion_solver**

  Function that solves the solid part.

- Utilities::Progress< double > **progress**

  Variable used to store the progress of the simulation.

Protected Attributes

- double **a**

  Coefficient that accompanies the term T_{n-1} in the energy equation of the last solid cell.
- double **b**

  Coefficient that accompanies the term T_n in the energy equation of the last solid cell.
- double **d**

  Independent term of the energy equation of the last solid cell.
- double **rho_g**

  Density of the first gas cell.
- double **v_g**

  Gas speed of the first gas cell.

7.40.1 Detailed Description

This object is used to create some initial conditions and let the domain evolve in time.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 Simulation() [1/4]

```
Simulation::Simulation (
            const std::string & name,
            const Chemistry::SolidGasReaction & QR,
            const double a,
            const double b,
            const unsigned int N_cells_solid,
            const unsigned int N_cells_gas,
            const bool adaptive_refinement_solid,
            const bool adaptive_refinement_gas,
            const unsigned int N_tasks,
            const double CFL,
            const unsigned int N_saves,
            const SolidBoundaryConditions & solid_BC,
            const GasBoundaryConditions & gas_BC,
            const double x_q,
            std::function< double(double x)> v,
            std::function< double(double x)> P,
            std::function< double(double x)> T,
            std::function< double(double x)> A,
            std::function< void(const GasCell &A, const GasCell &B, Math::Vector< double >
*F, double *S_max)> convection_solver,
            std::function< void(SolidMesh &mesh, double &dt, const double CFL, const SolidBoundaryConditions
BC, const double t)> diffusion_solver,
            std::function< double(std::function< double(double x)> f, const double a, const
double b)> integrator = [](std::function< double(double x)> f, const double a, const double b) {return Math←
::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
            std::function< double(const GasCell &C, const double t)> external_forces = [](const GasCell &C,
            const unsigned int adaptive_refinement_period = 1,
```

```
const double detail_subdivide_threshold = 0.01,
const double detail_merge_threshold = 0.001,
const double max_length_factor = 1./50,
const double min_length_factor = 1./2000,
const double boundary_cell_max_length_factor = 1./1000 )
```

Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type BOTH.

Parameters

name	the name of the simulation.
QR	the quemical reaction.
a	the left limit of the mesh.
b	the right limit of the mesh.
N_cells_solid	the initial number of cells for the solid mesh.
N_cells_gas	the initial number of cells for the gas mesh.
adaptive_refinement_solid	whether to use adaptive refinement for the solid mesh.
adaptive_refinement_gas	whether to use adaptive refinement for the gas mesh.
N_tasks	the number of CPU tasks to use in order to compute the simulation.
CFL	the Courant-Friedrichs-Lewy number.
N_saves	how many times throughout the simulation is data stored in a file.
solid_BC	solid boundary conditions for the simulation.
gas_BC	gas boundary conditions for the simulation.
x_q	the initial position of the solid-gas interface.
v	the initial gas speed as a function of space.
P	the initial gas pressure as a function of space
T	the initial temperature as a function of space.
A	the conduct area as a function of space.
convection_solver	the solver to use in order to calculate intercell fluxes.
diffusion_solver	the solver to use in order to solve the solid.
integrator	the function to use in order to integrate the initial conditions.
external_forces	a function that computes the external forces acting on a gas cell for a specific moment in time.
adaptive_refinement_period	determines how often the mesh is refined.
detail_subdivide_threshold	if a cell detail is bigger than this number, we divide the cell in two.
detail_merge_threshold	if the details of two neighbouring cells are lower than this number, they are merged.
max_length_factor	a cell cannot be bigger than max_length_factor $*$ total length of the mesh.
min_length_factor	a cell cannot be small than min_length_factor $*$ total length of the mesh.
boundary_cell_max_length_factor	max_length_factor for cells near the boundary.

7.40.2.2 Simulation() [2/4]

```
Simulation::Simulation (
        const std::string & name,
        const Chemistry::SolidGasReaction & QR,
        const double a,
        const double b,
```

```
            const unsigned int N_cells,
            const bool adaptive_refinement,
            const unsigned int N_tasks,
            const double CFL,
            const unsigned int N_saves,
            const GasBoundaryConditions & BC,
            std::function< double(double x)> rho,
            std::function< double(double x)> v,
            std::function< double(double x)> P,
            std::function< double(double x)> A,
            std::function< void(const GasCell &A, const GasCell &B, Math::Vector< double >
*F, double *S_max)> convection_solver,
            std::function< double(std::function< double(double x)> f, const double a, const
double b)> integrator = [](std::function< double(double x)> f, const double a, const double b) {return Math←
::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
            std::function< double(const GasCell &C, const double t)> external_forces = [](const GasCell &C,
            const unsigned int adaptive_refinement_period = 1,
            const double detail_subdivide_threshold = 0.01,
            const double detail_merge_threshold = 0.001,
            const double max_length_factor = 1./50,
            const double min_length_factor = 1./2000,
            const double boundary_cell_max_length_factor = 1./1000 )
```

Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type GAS.

Parameters

name	the name of the simulation.
QR	the quemical reaction object that stores the gas constants.
a	the left limit of the mesh.
b	the right limit of the mesh.
N_cells	the initial number of cells.
adaptive_refinement	whether to use adaptive refinement of the mesh.
N_tasks	the number of CPU tasks to use in order to compute the simulation.
CFL	the Courant-Friedrichs-Lewy number.
N_saves	how many times throughout the simulation is data stored in a file.
BC	boundary conditions for the simulation.
rho	the initial gas density as a function of space.
v	the initial gas speed as a function of space.
P	the initial pressure as a function of space.
A	the conduct area as a function of space.
convection_solver	the solver to use in order to solve convection.
integrator	the function to use in order to integrate the initial conditions.
external_forces	a function that computes the external forces acting on a cell for a specific moment in time.
adaptive_refinement_period	determines how often the mesh is refined.
detail_subdivide_threshold	if a cell detail is bigger than this number, we divide the cell in two.
detail_merge_threshold	if the details of two neighbouring cells are lower than this number, they are merged.
max_length_factor	a cell cannot be bigger than max_length_factor $*$ total length of the mesh.
min_length_factor	a cell cannot be small than min_length_factor $*$ total length of the mesh.
boundary_cell_max_length_factor	max_length_factor for cells near the boundary.

7.40.2.3 Simulation() [3/4]

```
Simulation::Simulation (
              const std::string & name,
              const Chemistry::SolidGasReaction & QR,
              const double a,
              const double b,
              const unsigned int N_cells,
              const bool adaptive_refinement,
              const unsigned int N_tasks,
              const double CFL,
              const unsigned int N_saves,
              const SolidBoundaryConditions & BC,
              std::function< double(double x)> T,
              std::function< double(double x)> A,
              std::function< void(SolidMesh &mesh, double &dt, const double CFL, const SolidBoundaryConditions
BC, const double t)> diffusion_solver,
              std::function< double(std::function< double(double x)> f, const double a, const
double b)> integrator = [] (std::function<double(double x)> f, const double a, const double b) {return Math←
::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
              const unsigned int adaptive_refinement_period = 1,
              const double detail_subdivide_threshold = 0.01,
              const double detail_merge_threshold = 0.001,
              const double max_length_factor = 1./50,
              const double min_length_factor = 1./2000,
              const double boundary_cell_max_length_factor = 1./1000 )
```

Constructs a Simulation object from initial conditions. It is assumed that the simulation is of type SOLID.

Parameters

name	the name of the simulation.
QR	the quemical reaction.
a	the left limit of the mesh.
b	the right limit of the mesh.
N_cells	the initial number of cells.
adaptive_refinement	whether to use adaptive refinement of the mesh.
N_tasks	the number of CPU tasks to use in order to compute the simulation.
CFL	the Courant-Friedrichs-Lewy number.
N_saves	how many times throughout the simulation is data stored in a file.
BC	boundary conditions for the simulation.
T	the initial temperature as a function of space.
A	the conduct area as a function of space.
diffusion_solver	the solver to use in order to solve diffusion.
integrator	the function to use in order to integrate the initial conditions.
adaptive_refinement_period	determines how often the mesh is refined.
detail_subdivide_threshold	if a cell detail is bigger than this number, we divide the cell in two.
detail_merge_threshold	if the details of two neighbouring cells are lower than this number, they are merged.
max_length_factor	a cell cannot be bigger than max_length_factor $*$ total length of the mesh.
min_length_factor	a cell cannot be small than min_length_factor $*$ total length of the mesh.
boundary_cell_max_length_factor	max_length_factor for cells near the boundary.

7.40.2.4 Simulation() [4/4]

```
Simulation::Simulation (
            const std::string & file )  [explicit]
```

Construct a new Simulation object from a file.

Parameters

file	the name of the file where the simulation is stored.

7.40.3 Member Function Documentation

7.40.3.1 A()

```
std::function< double(const double x)> Simulation::A (
            const double t ) const
```

Returns the function A(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.2 c()

```
std::function< double(const double x)> Simulation::c (
            const double t ) const
```

Returns the function c(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.3 M()

```
std::function< double(const double x)> Simulation::M (
            const double t ) const
```

Returns the function M(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.4 mesh_plot()

```
Graphic * Simulation::mesh_plot ( ) const
```

Returns a graphic where the cell boundaries are drawn as a function of time.

Returns

CPGF::Plot2d::Graphic

7.40.3.5 P()

```
std::function< double(const double x)> Simulation::P (
            const double t ) const
```

Returns the function P(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.6 rho()

```
std::function< double(const double x)> Simulation::rho (
            const double t ) const
```

Returns the function rho(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.7 simulate_until()

```
void Simulation::simulate_until (
            const double t )
```

Runs the simulation until time t has been reached. The last computed time is exactly t.

Parameters

t	

7.40.3.8 T()

```
std::function< double(const double x)> Simulation::T (
            const double t ) const
```

Returns the function T(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.9 update()

```
void Simulation::update (
            double dt = std::numeric_limits<double>::max() )
```

Compute a new time step.

Parameters

dt	the maximum time step allowed.

7.40.3.10 v()

```
std::function< double(const double x)> Simulation::v (
            const double t ) const
```

Returns the function v(x) for a certain time t.

A temporal and spatial linear interpolation is used.

Parameters

t	

Returns

std::function<double(const double x)>

7.40.3.11 v_q()

```
double Simulation::v_q (
            const double t ) const
```

Returns the burning rate at a specific time t.

It does a linear interpolation using t_array and v_q_array.

Parameters

t	

Returns

double

7.40.3.12 write_to_file()

```
void Simulation::write_to_file (
            const std::string & file ) const
```

Writes the simulation into a file.

Parameters

file	the name of the file to write the simulation to.

7.40.3.13 x_q()

```
double Simulation::x_q (
            const double t ) const
```

Returns the position of the combustion at a specific time t.

It does a linear interpolation using t_array and x_q_array.

Parameters

t	

Returns

double

7.40.4 Member Data Documentation

7.40.4.1 adaptive_refinement_period

```
unsigned int Simulation::adaptive_refinement_period
```

Controls how often the mesh is refined. In particular, it represents how many iterations we wait after the mesh has been refined to refine it again.

Setting this parameter to one means the mesh is refined in every iteration.

7.40.4.2 external_forces

```
std::function<double(const GasCell& C, const double t)> Simulation::external_forces
```

Function that computes the external forces acting on each gas cell.

Returns

std::function<double(const Cell& C, const double t)>

7.40.4.3 N_saves

```
unsigned int Simulation::N_saves
```

Determines how many times during the simulation the current meshes and values are saved to the file.

The number of actual saves is N_saves + 2, because the first and last state are saved by default. The saves are evenly spread throghout the simulation time.

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

- Simulation.hpp
- Simulation.cpp

7.41 SolidBoundaryConditions Class Reference

Object used to store left and right boundary conditions of the solid.

```
#include <SolidSolvers.hpp>
```

Public Member Functions

- SolidBoundaryConditions (const SolidBoundaryConditionsType left=SolidBoundaryConditionsType::FIXED_TEMPERATURE, const SolidBoundaryConditionsType right=SolidBoundaryConditionsType::FIXED_TEMPERATURE, std::function< double(double T, double t)> left_condition=[ ](double T, double t){return 0;}, std::function< double(double T, double t)> right_condition=[ ](double T, double t){return 0;})

    Construct a new Solid Boundary Conditions object.

Public Attributes

- SolidBoundaryConditionsType **left**

  Boundary condition at the left side of the mesh.
- SolidBoundaryConditionsType **right**

  Boundary condition at the right side of the mesh.
- std::function< double(double T, double t)> **left_condition**

  Function of temperature and time that returns the value for the left boundary condition.
- std::function< double(double T, double t)> **right_condition**

  Function of temperature and time that provides the value for the right boundary condition.

7.41.1 Detailed Description

Object used to store left and right boundary conditions of the solid.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 SolidBoundaryConditions()

```
SolidBoundaryConditions::SolidBoundaryConditions (
            const SolidBoundaryConditionsType left = SolidBoundaryConditionsType::FIXED_↩
TEMPERATURE,
            const SolidBoundaryConditionsType right = SolidBoundaryConditionsType::FIXED_↩
TEMPERATURE,
            std::function< double(double T, double t)> left_condition = [](double T, double t){return 0;},
            std::function< double(double T, double t)> right_condition = [](double T, double t){return 0;}
)
```

Construct a new Solid Boundary Conditions object.

Parameters

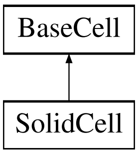
left	type of the left boundary condition.
right	type of the right boundary condition.
left_condition	function used to provide values for the left boundary condition.
right_condition	function used to provide values for the right boundary condition.

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

- Solvers/Solid/SolidSolvers.hpp
- Solvers/Solid/SolidSolvers.cpp

7.42 SolidCell Class Reference

Inheritance diagram for SolidCell:



Public Member Functions

- SolidCell (const Math::Vector< double > &U, double a, double b, const double A, Chemistry::SolidGasReaction *QR)

  Construct a new Solid Cell object.
- SolidCell (FILE *file, Chemistry::SolidGasReaction *QR)

  Construct a new Cell object from file.
- void **update** ()

  Updates the value of the temperature from the value of U.
- void read_from_file (FILE *file)

  Reads the cell values from file and calls update()
- std::string to_string () const

  Returns a string representation of the SolidCell.

Public Attributes

- SolidCell * **right_neighbour**

  Right neighbour of the cell.
- SolidCell * **left_neighbour**

  Left neighbour of the cell.
- double **T**

  Solid temperature.

7.42.1 Constructor & Destructor Documentation

7.42.1.1 SolidCell() [1/2]

```
SolidCell::SolidCell (
            const Math::Vector< double > & U,
            double a,
            double b,
            const double A,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Solid Cell object.

Parameters

U	the vector of preserved variables.
a	the left limit of the cell.
b	the right limit of the cell.
A	the area of the cell.
QR	a pointer to the chemical reaction.

7.42.1.2 SolidCell() [2/2]

```
SolidCell::SolidCell (
            FILE * file,
            Chemistry::SolidGasReaction * QR )
```

Construct a new Cell object from file.

Parameters

file	

7.42.2 Member Function Documentation

7.42.2.1 read_from_file()

```
void SolidCell::read_from_file (
            FILE * file )
```

Reads the cell values from file and calls update()

Parameters

file	

7.42.2.2 to_string()

```
std::string SolidCell::to_string ( ) const
```

Returns a string representation of the SolidCell.

Returns

std::string

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

- Mesh/Cell.hpp
- Mesh/Cell.cpp

7.43 Chemistry::SolidGasReaction Class Reference

This object is used to store all properties of a solid reactant and a gas product. It is also used to compute the speed of the combustion front.

```
#include <Reaction.hpp>
```

Public Member Functions

- double v_q (const double P) const

  Computes the burning rate at the given conditions.
- SolidGasReaction (const double rho_s=0, const double k_s=0, const double cV_s=0, const double cV_g=0, const double R_g=0, const double a=0, const double P_ref=0, const double n=0, const double delta_H=0)

  Construct a new Solid Gas Reaction object.
- SolidGasReaction (FILE ∗file)

  Construct a new Solid Gas Reaction object reading it from a file.
- void read_from_file (FILE ∗file)

  Reads the object from file.
- void write_to_file (FILE ∗file) const

  Writes the object to file.
- std::string to_string () const

  Returns a string representation of the object.

Public Attributes

- double **rho_s**

  Solid density [kg/m^3].
- double **k_s**

  Thermal conductivity of solid [W/(m·K)].
- double **cV_s**

  Specific heat capacity at constant volume of solid [J/(kg·K)].
- double **alpha**

  Thermal diffusivity of solid [m^2/s].
- double **cV_g**

  Specific heat capacity at constant volume of gas [J/(kg·K)].
- double **R_g**

  Gas constant of gas [J/(kg·K)].
- double gamma

  Adiabatic compression coefficent of gas [adimensional].
- double **a**

  The prepotential factor of Vieille's law [m/s].
- double **P_ref**

  The reference pressure for Vieille's law [Pa].
- double **n**

  The exponent of Vieille's law [adimensional].
- double **delta_H**

  The enthalpy increase of the chemical reaction [J/kg].

7.43.1 Detailed Description

This object is used to store all properties of a solid reactant and a gas product. It is also used to compute the speed of the combustion front.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 SolidGasReaction() [1/2]

```
SolidGasReaction::SolidGasReaction (
            const double rho_s = 0,
            const double k_s = 0,
            const double cV_s = 0,
            const double cV_g = 0,
            const double R_g = 0,
            const double a = 0,
            const double P_ref = 0,
            const double n = 0,
            const double delta_H = 0 )
```

Construct a new Solid Gas Reaction object.

Parameters

rho_s	the solid density [kg/m^3].
k_s	the solid thermal conductivity [W/(m·K)].
cV_s	the solid heat capacity at constant volume [J/(kg·K)].
cV_g	the gas heat capacity at constant volume [J/(kg·K)].
R_g	the gas constant [J/(kg·K)].
a	the prepotential factor of Vieille's law [m/s].
P_ref	the reference pressure for Vieille's law [Pa].
n	the exponent of Vieille's law [adimensional].
delta_H	the enthalpy increase of the chemical reaction [J/kg].

7.43.2.2 SolidGasReaction() [2/2]

```
SolidGasReaction::SolidGasReaction (
            FILE * file )  [explicit]
```

Construct a new Solid Gas Reaction object reading it from a file.

Parameters

file	

7.43.3 Member Function Documentation

7.43.3.1 read_from_file()

```
void SolidGasReaction::read_from_file (
            FILE * file )
```

Reads the object from file.

Parameters

file	

7.43.3.2 to_string()

```
std::string SolidGasReaction::to_string ( ) const
```

Returns a string representation of the object.

Returns

std::string

7.43.3.3 v_q()

```
double SolidGasReaction::v_q (
            const double P ) const
```

Computes the burning rate at the given conditions.

Parameters

P	the pressure of the gas.

Returns

double

7.43.3.4 write_to_file()

```
void SolidGasReaction::write_to_file (
            FILE * file ) const
```

Writes the object to file.

Parameters

file	

7.43.4 Member Data Documentation

7.43.4.1 gamma

```
double Chemistry::SolidGasReaction::gamma
```

Adiabatic compression coefficent of gas [adimensional].

Returns

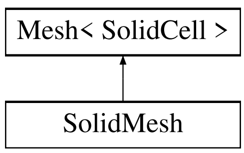
double

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

- Chemistry/Reaction.hpp
- Chemistry/Reaction.cpp

7.44 SolidMesh Class Reference

Inheritance diagram for SolidMesh:



Public Member Functions

- SolidMesh (SolidCell *first_cell=nullptr, SolidCell *last_cell=nullptr, const Chemistry::SolidGasReaction &QR=Chemistry::SolidGasReaction(), std::function< double(const double x)> A_func=[ ](double x){return 0;}, const double detail_subdivide_threshold=0.01, const double detail_merge_threshold=0.0005, const double max_length_factor=1./20, const double min_length_factor=1./2000, const double boundary_cell_max_length_factor=1./1000

  Construct a new Solid Mesh object.
- std::vector< double > T () const

  Returns the temperature of all cells in the mesh.

Additional Inherited Members

7.44.1 Constructor & Destructor Documentation

7.44.1.1 SolidMesh()

```
SolidMesh::SolidMesh (
            SolidCell * first_cell = nullptr,
            SolidCell * last_cell = nullptr,
            const Chemistry::SolidGasReaction & QR = Chemistry::SolidGasReaction(),
            std::function< double(const double x)> A_func = [](double x){return 0;},
            const double detail_subdivide_threshold = 0.01,
            const double detail_merge_threshold = 0.0005,
            const double max_length_factor = 1./20,
            const double min_length_factor = 1./2000,
            const double boundary_cell_max_length_factor = 1./1000 )
```

Construct a new Solid Mesh object.

Parameters

first_cell	pointer to the first cell.
last_cell	pointer to the last cell.
QR	chemical reaction.
A_func	area function.
detail_subdivide_threshold	determines when cells are subdivided.
detail_merge_threshold	determines when cells are merged.
max_length_factor	limits the maximum length of a created cell.
min_length_factor	limits the minimum length of a created cell.
boundary_cell_max_length_factor	limits the maxium length of cells near the boundary.

7.44.2 Member Function Documentation

7.44.2.1 T()

```
std::vector< double > SolidMesh::T ( ) const
```

Returns the temperature of all cells in the mesh.

Returns

std::vector<double>

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

- Mesh/Mesh.hpp
- Mesh/Mesh.cpp

7.45 Solvers::Rocket::SteadySolver Class Reference

This object can be used to compute chamber and exit values for a specific rocket geometry.

```
#include <RocketSolver.hpp>
```

Public Member Functions

- SteadySolver (const double T_c, const double M_catm, const double n, const double gamma, const double R, const double A_c)

  Construct a new Steady Solver object.
- void solve_for_exit_area (const double A_e)

  Determines all chamber and exit values.
- void **calculate_optimum_parameters** ()

  Calculates the chamber and exit values that provide the maximum thrust, that is, when the exit Mach number is set to one.
- std::string to_string () const

  Returns a string representation of the object.

Public Attributes

- double **A_c**

  Chamber area.
- double **A_e**

  Exit area.
- double **M_catm**

  Chamber Mach number when chamber pressure is the atmospheric pressure.
- double **gamma**

  Gas adiabatic expansion coefficient.
- double **R**

  Gas constant.
- double **n**

  Vieille's power law exponent.
- double **T_c**

  Chamber temperature. Also, the burning temperature of the propellant.
- double **v_c**

  Chamber gas speed.
- double **M_c**

  Chamber Mach number.
- double **P_c**

  Chamber pressure.
- double **rho_c**

  Chamber gas density.
- double **v_e**

  Exit speed.
- double **M_e**

  Exit Mach number.
- double **P_e**

  Exit pressure.

- double **T_e**

  Exit temperature.
- double **rho_e**

  Exist density.
- double **m_dot**

  Mass flux.
- double **thrust**

  Thrust generated by the rocket.

Protected Attributes

- double **G1**

  (gamma - 1) / 2
- double **G2**

  gamma / (gamma - 1)
- double **G3**

  (gamma + 1) / 2
- double **G4**

  (gamma + 1) / (2∗(gamma - 1))

Static Protected Attributes

- constexpr static double **P_atm** = 101325

  The atmospheric pressure.

7.45.1 Detailed Description

This object can be used to compute chamber and exit values for a specific rocket geometry.

7.45.2 Constructor & Destructor Documentation

7.45.2.1 SteadySolver()

```
SteadySolver::SteadySolver (
            const double T_c,
            const double M_catm,
            const double n,
            const double gamma,
            const double R,
            const double A_c )
```

Construct a new Steady Solver object.

Parameters

T_c	chamber temperature, also known as propellant burning temperature.
M_catm	chamber Mach number when the chamber pressure is the atmospheric pressure.
n	Vieille's power law exponent.
gamma	gas adiabatic expansion coefficient.
R	gas constant.
A_c	chamber area.

7.45.3 Member Function Documentation

7.45.3.1 solve_for_exit_area()

```
void SteadySolver::solve_for_exit_area (
            const double A_e )
```

Determines all chamber and exit values.

Parameters

A_e	exit area.

7.45.3.2 to_string()

```
std::string SteadySolver::to_string ( ) const
```

Returns a string representation of the object.

Returns

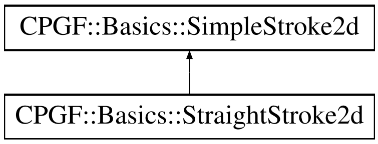
std::string

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

- Solvers/Rocket/RocketSolver.hpp
- Solvers/Rocket/RocketSolver.cpp

7.46 CPGF::Basics::StraightStroke2d Class Reference

Inheritance diagram for CPGF::Basics::StraightStroke2d:



Public Member Functions

- **StraightStroke2d** (const std::vector< AffineSpace::Point2d > points)
- StraightStroke2d ∗ clone () const override
- StraightStroke2d & **operator+=** (const AffineSpace::Point2d &P)
- StraightStroke2d & **add_point** (const AffineSpace::Point2d &P)
- unsigned int **size** () const
- AffineSpace::Point2d & start () override
- AffineSpace::Point2d start () const override
- AffineSpace::Point2d & end () override
- AffineSpace::Point2d end () const override
- StraightStroke2d & translate (const AffineSpace::Vector2d &v) override
- StraightStroke2d & rotate_with_respect_to (const AffineSpace::Point2d &Q, const double theta) override
- StraightStroke2d & scale_with_respect_to (const AffineSpace::Point2d &Q, const AffineSpace::Vector2d &s) override
- double length () const override
- double area () const override
- std::vector< AffineSpace::Point2d > operator/ (const SimpleStroke2d &B) override
- std::vector< AffineSpace::Point2d > **operator/** (const StraightStroke2d &B)
- std::string render_to_string () const override

Public Attributes

- std::vector< AffineSpace::Point2d > **points**

7.46.1 Member Function Documentation

7.46.1.1 area()

```
double StraightStroke2d::area ( ) const [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.2 clone()

```
StraightStroke2d * StraightStroke2d::clone ( ) const  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.3 end() [1/2]

```
Point2d StraightStroke2d::end ( ) const  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.4 end() [2/2]

```
Point2d & StraightStroke2d::end ( )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.5 length()

```
double StraightStroke2d::length ( ) const  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.6 operator/()

```
std::vector< Point2d > StraightStroke2d::operator/ (
             const SimpleStroke2d & B )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.7 render_to_string()

```
std::string StraightStroke2d::render_to_string ( ) const  [override], [virtual]
```

Parameters

alpha	a number between zero and one.

Returns

AffineSpace::Point2d

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.8 rotate_with_respect_to()

```
StraightStroke2d & StraightStroke2d::rotate_with_respect_to (
            const AffineSpace::Point2d & Q,
            const double theta )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.9 scale_with_respect_to()

```
StraightStroke2d & StraightStroke2d::scale_with_respect_to (
            const AffineSpace::Point2d & Q,
            const AffineSpace::Vector2d & s )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.10 start() [1/2]

```
Point2d StraightStroke2d::start ( ) const  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.11 start() [2/2]

```
Point2d & StraightStroke2d::start ( )  [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

7.46.1.12 translate()

```
StraightStroke2d & StraightStroke2d::translate (
            const AffineSpace::Vector2d & v ) [override], [virtual]
```

Implements CPGF::Basics::SimpleStroke2d.

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

- CPGF/PGFBasics/Strokes2d.hpp
- CPGF/PGFBasics/Strokes2d.cpp

7.47 CPGF::Text Class Reference

```
#include <Text.hpp>
```

Public Member Functions

- **Text** (const AffineSpace::Point2d &pos=AffineSpace::Point2d(0, 0), const std::string &text="", const Color &color=Color::BLACK, const TextAlignment text_alignment=TextAlignment::CENTER, const double rot=0)
- std::string **render_to_string** () const

Public Attributes

- std::string **text**

  The text that is going to be printed.
- AffineSpace::Point2d **pos**

  The text position.
- double **rot**

  The text rotation in degrees.
- TextAlignment **text_alignment**

  The text alignment form.
- Color **color**

  The text color.

7.47.1 Detailed Description

Warning

Text cannot be scaled or rotated, only moved.

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

- CPGF/Text/Text.hpp
- CPGF/Text/Text.cpp

7.48 Math::Vector< K > Class Template Reference

Generic Vector over the field K with any number of components.

`#include <Vector.hpp>`

Public Member Functions

- **Vector** ()

    Creates a Vector of length zero. A Vector of length zero can be safely added or substracted to another Vector of any length.
- Vector (const unsigned int N)

    Creates a Vector of length N .
- Vector (const Vector< K > &w)

    Copy constructor. Creates a deep copy of w .
- Vector (Vector< K > &&w)

    Move constructor. Copies the pointer components and N .
- Vector (const K ∗components, const unsigned int N)

    Creates a Vector of length N whose components are copied from the array components.
- Vector (std::initializer_list< K > list)

    Allows to create a Vector from a initializer_list. The dimension of the Vector is automatically deduced.
- ∼**Vector** ()

    Frees the contents of components .
- Vector< K > & operator= (const K &alpha)

    Equivalent to v[i] = alpha for all i.
- Vector< K > & operator= (const Vector< K > &w)

    Assigment operator.
- Vector< K > & operator= (Vector< K > &&w)

    Move-assigment operator.
- Vector< K > operator+ ()

    Returns the vector unaltered.
- Vector< K > operator- ()

    Returns minus the vector.
- Vector< K > & operator+= (const K &alpha)

    Equivalent to v[i] += alpha for all i .
- Vector< K > & operator+= (const Vector< K > &w)

    If this has length zero, than w is copied into this . If this and w have the same dimension, this is equivalent to this->operator[ ](i) += w[i] for all i . Otherwise, an exception is raised.
- Vector< K > & operator-= (const K &alpha)

    Equivalent to v[i] -= alpha for all i .
- Vector< K > & operator-= (const Vector< K > &w)

    If this has length zero, than -w is copied into this . If this and w have the same dimension, this is equivalent to this->operator[ ](i) -= w[i] for all i . Otherwise, an exception is raised.
- Vector< K > & operator∗= (const K &alpha)

    Equivalent to v[i] ∗= alpha for all i .
- Vector< K > & operator∗= (const Vector< K > &w)

    If this and w have the same dimension, this is equivalent to this->operator[ ](i) ∗= w[i] for all i . Otherwise, an exception is raised.
- Vector< K > & operator/= (const K &alpha)

    Equivalent to v[i] /= alpha for all i .
- Vector< K > & operator/= (const Vector< K > &w)

If this and w have the same dimension, this is equivalent to this->operator[ ](i) /= w[i] for all i . Otherwise, an exception is raised.

- Vector< K > operator+ () const

  Equivalent to v[i] = +v[i] for all i .
- Vector< K > operator- () const

  Equivalent to v[i] = -v[i] for all i .
- K & operator[ ] (const int i)

  Indexing operator that allows to use array syntax for accessing the components of a Vector. Checks for out of index errors. Allows for negative indexes.
- K operator[ ] (const int i) const

  Version of the operator[] for const objects.
- Vector< K > slice (int first, int last) const

  Slicing operator: equivalent to Python v[first: last]. Returns a subvector of this whose first component is v[first] and whose last component is v[last - 1].
- Vector< K > reverse () const

  Returns the Vector reversed: it starts with the last component and it finishes with the first.
- unsigned int size () const

  Returns the length (or dimension) of the Vector.
- double norm_2 () const

  Norm 2 (euclidean norm) of the Vector.
- double norm_1 () const

  Norm 1 (taxi-cab norm) of the Vector.
- double norm_inf () const

  Norm infinity (maximum norm) of the Vector.
- double norm_p (const double p) const

  Norm p of the Vector.
- std::string to_string () const

  Returns a string representation of the Vector. It will look similar to (1, 0, -3, 7.3).
- void read_from_file (FILE ∗file)

  Reads the Vector from a binary file. It first reads N , then deallocates and reallocates componentes and finally reads components from the file.
- void write_to_file (FILE ∗file) const

  Writes the Vector to a binary file. It first writes N and than components .

Public Attributes

- K ∗ **components**

  Array where the components of the Vector are stored.
- unsigned int **N**

  Dimension (also called length) of the Vector.

Friends

- Vector< K > operator+ (const Vector< K > &v, const Vector< K > &w)

  Componentwise addition operator.
- Vector< K > operator+ (const Vector< K > &v, const K &alpha)

  Componentwise Vector + scalar addition operator.
- Vector< K > operator+ (const K &alpha, const Vector< K > &w)

  Component wise scalar + Vector addition operator.
- Vector< K > operator- (const Vector< K > &v, const Vector< K > &w)

Componentwise substraction operator.
- Vector< K > operator- (const Vector< K > &v, const K &alpha)

  Componentwise Vector - scalar substraction operator.
- Vector< K > operator- (const K &alpha, const Vector< K > &w)

  Componentwise scalar - Vector substraction operator.
- Vector< K > operator* (const Vector< K > &v, const Vector< K > &w)

  Component wise multiplication operator.
- Vector< K > operator* (const K &alpha, const Vector< K > &v)

  Component wise Vector - scalar multiplication operator.
- Vector< K > operator* (const Vector< K > &v, const K &alpha)

  Component wise scalar - Vector multiplication operator.
- Vector< K > operator/ (const Vector< K > &v, const Vector< K > &w)

  Componentwise division operator.
- Vector< K > operator/ (const Vector< K > &v, const K &alpha)

  Componentwise Vector - scalar division operator.
- Vector< K > operator/ (const K &alpha, const Vector< K > &w)

  Componentwise scalar - Vector division operator.
- bool operator (const Vector< K > &v, const Vector< K > &w)

  Returns true if v[i] < w[i] for all i and false otherwise.
- bool operator (const Vector< K > &v, const K &alpha)

  Returns true if v[i] < alpha for all i and false otherwise.
- bool operator (const K &alpha, const Vector< K > &v)

  Returns true if alpha < v[i] for all i and false otherwise.
- bool operator (const Vector< K > &v, const Vector< K > &w)

  Returns true if v[i] <= w[i] for all i and false otherwise.
- bool operator (const Vector< K > &v, const K &alpha)

  Returns true if v[i] <= alpha for all i and false otherwise.
- bool operator (const K &alpha, const Vector< K > &v)

  Returns true if alpha <= v[i] for all i and false otherwise.
- bool operator> (const Vector< K > &v, const Vector< K > &w)

  Returns true if v[i] > w[i] for all i and false otherwise.
- bool operator> (const Vector< K > &v, const K &alpha)

  Returns true if v[i] > alpha for all i and false otherwise.
- bool operator> (const K &alpha, const Vector< K > &v)

  Returns true if alpha > v[i] for all i and false otherwise.
- bool operator>= (const Vector< K > &v, const Vector< K > &w)

  Returns true if v[i] >= w[i] for all i and false otherwise.
- bool operator>= (const Vector< K > &v, const K &alpha)

  Returns true if v[i] >= alpha for all i and false otherwise.
- bool operator>= (const K &alpha, const Vector< K > &v)

  Returns true if alpha >= v[i] for all i and false otherwise.
- bool operator== (const Vector< K > &v, const Vector< K > &w)

  If v and w have a different number of componentes, false is returned. If v and w have the same number of componenets, then true is returned if v[i] == w[i] for all i and false is returned otherwise.
- bool operator== (const Vector< K > &v, const K &alpha)

  Returns true if v[i] == alpha for all i and false otherwise.
- bool operator== (const K &alpha, const Vector< K > &v)

  Returns true if alpha == v[i] for all i and false otherwise.
- bool operator!= (const Vector< K > &v, const Vector< K > &w)

  Equivalent to !(v == w) .
- bool operator!= (const Vector< K > &v, const K &alpha)

Returns true if v[i] != alpha for all i and false otherwise.
- bool operator!= (const K &alpha, const Vector< K > &v)

  Returns true if alpha != v[i] for all i and false otherwise.
- Vector< K > operator& (const Vector< K > &v, const Vector< K > &w)

  Concatenation operator.
- Vector< K > operator& (const Vector< K > &v, const K &w)

  Concatenation operator.
- Vector< K > operator& (const K &v, const Vector< K > &w)

  Concatenation operator.
- K operator| (const Vector< K > &v, const Vector< K > &w)

  Scalar product operator.
- K vector_product_2d (const Vector< K > &v, const Vector< K > &w)

  Returns the Vector product of two 2d vectors v and w.
- Vector< K > vector_product_3d (const Vector< K > &v, const Vector< K > &w)

  Returns the Vector product of two 3d vectors v and w.
- std::ostream & operator (std::ostream &os, const Vector< K > &v)

  Allows to print a Vector v to screen with std::cout << v.
- K min (const Vector< K > &v)

  Returns the minimum element of the vector.
- K max (const Vector< K > &v)

  Returns the maximal element of the vector.
- K sum (const Vector< K > &v)

  Returns the sum of all the components of the vector.
- K multiply (const Vector< K > &v)

  Returns the product of all the components of the vector.
- Vector< K > cos (const Vector< K > &v)

  Equivalent to cos(v[i]) for all i .
- Vector< K > sin (const Vector< K > &v)

  Equivalent to sin(v[i]) for all i .
- Vector< K > tan (const Vector< K > &v)

  Equivalent to tan(v[i]) for all i .
- Vector< K > acos (const Vector< K > &v)

  Equivalent to acos(v[i]) for all i .
- Vector< K > asin (const Vector< K > &v)

  Equivalent to asin(v[i]) for all i .
- Vector< K > atan (const Vector< K > &v)

  Equivalent to atan(v[i]) for all i .
- Vector< K > atan2 (const K v, const Vector< K > &w)

  Equivalent to atan2(v,w[i]) for all i .
- Vector< K > atan2 (const Vector< K > &v, const K w)

  Equivalent to atan2(v[i],w) for all i .
- Vector< K > atan2 (const Vector< K > &v, const Vector< K > &w)

  Equivalent to atan2(v[i],w[i]) for all i .
- Vector< K > cosh (const Vector< K > &v)

  Equivalent to cosh(v[i]) for all i .
- Vector< K > sinh (const Vector< K > &v)

  Equivalent to sinh(v[i]) for all i .
- Vector< K > tanh (const Vector< K > &v)

  Equivalent to tanh(v[i]) for all i .
- Vector< K > acosh (const Vector< K > &v)

  Equivalent to acosh(v[i]) for all i .

- Vector< K > asinh (const Vector< K > &v)

  Equivalent to asinh(v[i]) for all i .
- Vector< K > atanh (const Vector< K > &v)

  Equivalent to atanh(v[i]) for all i .
- Vector< K > exp (const Vector< K > &v)

  Equivalent to exp(v[i]) for all i .
- Vector< K > frexp (const Vector< K > &v, Vector< int > *exp)

  Equivalent to frexp(v[i],exp[i]) for all i .
- Vector< K > ldexp (const Vector< K > &v, const int exp)

  Equivalent to ldexp(v[i],exp) for all i .
- Vector< K > ldexp (const Vector< K > &v, const Vector< int > &exp)

  Equivalent to ldexp(v[i],exp[i]) for all i .
- Vector< K > log (const Vector< K > &v)

  Equivalent to log(v[i]) for all i .
- Vector< K > log10 (const Vector< K > &v)

  Equivalent to log10(v[i]) for all i .
- Vector< K > modf (const Vector< K > &v, Vector< K > *intpart)

  Equivalent to modf(v[i],intpart[i]) for all i .
- Vector< K > exp2 (const Vector< K > &v)

  Equivalent to exp2(v[i]) for all i .
- Vector< K > expm1 (const Vector< K > &v)

  Equivalent to expm1(v[i]) for all i .
- Vector< K > ilogb (const Vector< K > &v)

  Equivalent to ilogb(v[i]) for all i .
- Vector< K > log1p (const Vector< K > &v)

  Equivalent to log1p(v[i]) for all i .
- Vector< K > log2 (const Vector< K > &v)

  Equivalent to log2(v[i]) for all i .
- Vector< K > logb (const Vector< K > &v)

  Equivalent to logb(v[i]) for all i .
- Vector< K > scalbn (const Vector< K > &v, const int n)

  Equivalent to sclbn(v[i],n) for all i .
- Vector< K > scalbn (const Vector< K > &v, const Vector< int > &n)

  Equivalent to sclbn(v[i],n[i]) for all i .
- Vector< K > scalbln (const Vector< K > &v, const long int n)

  Equivalent to sclbln(v[i],n[i]) for all i .
- Vector< K > scalbln (const Vector< K > &v, const Vector< long int > &n)

  Equivalent to sclbln(v[i],n[i]) for all i .
- Vector< K > pow (const K v, const Vector< K > &exponent)

  Equivalent to pow(v,exponent[i]) for all i .
- Vector< K > pow (const Vector< K > &v, const K exponent)

  Equivalent to pow(v[i],exponent) for all i .
- Vector< K > pow (const Vector< K > &v, const Vector< K > &exponent)

  Equivalent to sclbln(v[i],exponent[i]) for all i .
- Vector< K > sqrt (const Vector< K > &v)

  Equivalent to sqrt(v[i]) for all i .
- Vector< K > cbrt (const Vector< K > &v)

  Equivalent to cbrt(v[i]) for all i .
- Vector< K > hypot (const K x, const Vector< K > &y)

  Equivalent to hypot(x,y[i]) for all i .
- Vector< K > hypot (const Vector< K > &x, const K y)

Equivalent to hypot(x[i],y) for all i .

- Vector< K > hypot (const Vector< K > &x, const Vector< K > &y)

  Equivalent to hypot(x[i],y[i]) for all i .
- Vector< K > erf (const Vector< K > &v)

  Equivalent to erf(v[i]) for all i .
- Vector< K > erfc (const Vector< K > &v)

  Equivalent to erfc(v[i]) for all i .
- Vector< K > tgamma (const Vector< K > &v)

  Equivalent to tgamma(v[i]) for all i .
- Vector< K > lgamma (const Vector< K > &v)

  Equivalent to lgamma(v[i]) for all i .
- Vector< K > ceil (const Vector< K > &v)

  Equivalent to ceil(v[i]) for all i .
- Vector< K > floor (const Vector< K > &v)

  Equivalent to floor(v[i]) for all i .
- Vector< K > fmod (const K numer, const Vector< K > &denom)

  Equivalent to fmod(numer,denom[i]) for all i .
- Vector< K > fmod (const Vector< K > &numer, const K denom)

  Equivalent to fmod(numer[i],denom) for all i .
- Vector< K > fmod (const Vector< K > &numer, const Vector< K > &denom)

  Equivalent to fmod(numer[i],denom[i]) for all i .
- Vector< K > trunc (const Vector< K > &v)

  Equivalent to trunc(v[i]) for all i .
- Vector< K > round (const Vector< K > &v)

  Equivalent to round(v[i]) for all i .
- Vector< long int > lround (const Vector< K > &v)

  Equivalent to lround(v[i]) for all i .
- Vector< long long int > llround (const Vector< K > &v)

  Equivalent to llround(v[i]) for all i .
- Vector< K > rint (const Vector< K > &v)

  Equivalent to rint(v[i]) for all i .
- Vector< long int > lrint (const Vector< K > &v)

  Equivalent to lrint(v[i]) for all i .
- Vector< long long int > llrint (const Vector< K > &v)

  Equivalent to llrint(v[i]) for all i .
- Vector< K > nearbyint (const Vector< K > &v)

  Equivalent to nearbyint(v[i]) for all i .
- Vector< K > remainder (const K numer, const Vector< K > &denom)

  Equivalent to remainder(numer,denom[i]) for all i .
- Vector< K > remainder (const Vector< K > &numer, const K denom)

  Equivalent to remainder(numer[i],denom) for all i .
- Vector< K > remainder (const Vector< K > &numer, const Vector< K > &denom)

  Equivalent to remainder(numer[i],denom[i]) for all i .
- Vector< K > remquo (const K numer, const Vector< K > &denom, Vector< int > *quot)

  Equivalent to remquo(numer,denom[i],quot[i]) for all i .
- Vector< K > remquo (const Vector< K > &numer, const K denom, Vector< int > *quot)

  Equivalent to remquo(numer[i],denom,quot[i]) for all i .
- Vector< K > remquo (const Vector< K > &numer, const Vector< K > &denom, Vector< int > *quot)

  Equivalent to remquo(numer[i],denom[i],quot[i]) for all i .
- Vector< K > copysign (const Vector< K > &x, const K y)

  Equivalent to copysign(x[i],y) for all i .

- Vector< K > copysign (const Vector< K > &x, const Vector< K > &y)

  Equivalent to copysign(x[i],y[i]) for all i .
- Vector< K > nan (const unsigned int N, const char *tagp)

  Equivalent to nan(tagp) for all i .
- Vector< K > nextafter (const Vector< K > &x, const Vector< K > &y)

  Equivalent to nextafter(x[i],y[i]) for all i .
- Vector< K > fdim (const K x, const Vector< K > &y)

  Equivalent to fdim(x,y[i]) for all i .
- Vector< K > fdim (const Vector< K > &x, K y)

  Equivalent to fdim(x[i],y) for all i .
- Vector< K > fdim (const Vector< K > &x, const Vector< K > &y)

  Equivalent to fdim(x[i],y[i]) for all i .
- Vector< double > fabs (const Vector< K > &v)

  Equivalent to fabs(v[i]) for all i .
- Vector< double > abs (const Vector< K > &v)

  Equivalent to abs(v[i]) for all i .
- Vector< K > fma (const K x, const Vector< K > &y, const Vector< K > &z)

  Equivalent to fma(x,y[i],z[i]) for all i .
- Vector< K > fma (const Vector< K > &x, const K y, const Vector< K > &z)

  Equivalent to fma(x[i],y,z[i]) for all i .
- Vector< K > fma (const Vector< K > &x, const Vector< K > &y, const K z)

  Equivalent to fma(x[i],y[i],z) for all i .
- Vector< K > fma (const K x, const K y, const Vector< K > &z)

  Equivalent to fma(x,y,z[i]) for all i .
- Vector< K > fma (const K x, const Vector< K > &y, const K z)

  Equivalent to fma(x,y[i],z) for all i .
- Vector< K > fma (const Vector< K > &x, const K y, const K z)

  Equivalent to fma(x[i],y,z) for all i .
- Vector< K > fma (const Vector< K > &x, const Vector< K > &y, const Vector< K > &z)

  Equivalent to fma(x[i],y[i],z[i]) for all i .
- Vector< int > fpclassify (const Vector< K > &v)

  Equivalent to fpclassify(v[i]) for all i .
- bool isfinite (const Vector< K > &v)

  Returns true if all components of v are finite and false otherwise.
- bool isinf (const Vector< K > &v)

  Returns true if at least one components of v is infinite and false otherwise.
- bool isnan (const Vector< K > &v)

  Returns true if at least one components of v is nan and false otherwise.
- bool isnormal (const Vector< K > &v)

  Returns true if all components of v are normal and false otherwise.
- Vector< K > conj (const Vector< K > &v)

  Equivalent to conj(v[i]) for all i.

7.48.1 Detailed Description

template<typename K>
class Math::Vector< K >

Generic Vector over the field K with any number of components.

It is implemented through generic programming with a template typename K. The components of the Vector are stored in an array which is dynamically assigned.

It implements the following Vector componentwise operations: +, -, $*$, /. Boolean operators $<$, $<=$, $>$, $>=$, ==, != apply the corresponding operator component by component and "and" the results together. In other words, two vectors *v* and *w* satisfy *v* $<$ *w = true* if and only if *a[i]* $<$ *b[i]* for all *i* . Otherwise, *a* $<$ *b = false*. In addition, all previously mentioned operators allow a Vector-scalar or a scalar-Vector version, that is: *v + 5* or *5 + v*, for example. In that case, the scalar operator is applied to all components of the Vector, in other words: *v + 5* is equivalent to *v[i] + 5* for all *i* . In the same way, *v* $<$ *5 = true* is the same as *v[i]* $<$ *5 = true* for all *i*".

Moreover, a scalar product operator | is provided. Note that C++ operator precedence for | is low; so it is recommend to always write parenthesis that englobe the scalar product, obtaining a mixture of Dirac's and mathematicians' notation for the scalar product: *scalar product of v and w = (v|w)*. The object also counts with a concatenation operator &: *v & w* places the components of *v* followed by the components of *w* in a new Vector.

The index operator [] can be used to access the Vector components. Bear in mind that indexes start with zero, following the convention of the C programming language.

Template Parameters

K	Any mathematical field. Normally *double* or *float*.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 Vector() [1/5]

```
template<typename K >
Vector::Vector (
            const unsigned int N )  [explicit]
```

Creates a Vector of length *N* .

Parameters

N	Length of the Vector.

7.48.2.2 Vector() [2/5]

```
template<typename K >
```

```
Vector::Vector (
            const Vector< K > & w )
```

Copy constructor. Creates a deep copy of *w* .

Parameters

w	

7.48.2.3 Vector() [3/5]

```
template<typename K >
Vector::Vector (
            Vector< K > && w )
```

Move constructor. Copies the pointer *components* and *N* .

Warning

It leaves *w* in a corrupted state.

Parameters

w	

7.48.2.4 Vector() [4/5]

```
template<typename K >
Vector::Vector (
            const K * components,
            const unsigned int N )
```

Creates a Vector of length *N* whose components are copied from the array *components*.

Parameters

components	An array of numbers.
N	The length of the array.

Warning

May cause undefined behaviour if the array supplied has not been properly constructed.

7.48.2.5 Vector() [5/5]

```
template<typename K >
Vector::Vector (
            std::initializer_list< K > list )
```

Allows to create a Vector from a initializer_list. The dimension of the Vector is automatically deduced.

Parameters

list	

7.48.3 Member Function Documentation

7.48.3.1 norm_1()

```
template<typename K >
double Vector::norm_1
```

Norm 1 (taxi-cab norm) of the Vector.

Returns

K

7.48.3.2 norm_2()

```
template<typename K >
double Vector::norm_2
```

Norm 2 (euclidean norm) of the Vector.

Returns

K

7.48.3.3 norm_inf()

```
template<typename K >
double Vector::norm_inf
```

Norm infinity (maximum norm) of the Vector.

Returns

K

7.48.3.4 norm_p()

```
template<typename K >
double Vector::norm_p (
            const double p ) const
```

Norm p of the Vector.

Parameters

p	

Returns

K

7.48.3.5 operator*=() [1/2]

```
template<typename K >
Vector< K > & Vector::operator*= (
            const K & alpha )
```

Equivalent to *v[i] *= alpha* for all *i* .

Parameters

alpha	

Returns

Vector<K>&

7.48.3.6 operator*=() [2/2]

```
template<typename K >
Vector< K > & Vector::operator*= (
            const Vector< K > & w )
```

If *this* and *w* have the same dimension, this is equivalent to *this->operator[ ](i) *= w[i]* for all *i* . Otherwise, an exception is raised.

Exceptions

An	exception is thrown if *this* and *w* have a different number of components.

Parameters

w	

Returns

Vector<K>&

7.48.3.7 operator+() [1/2]

```
template<typename K >
Vector< K > Vector::operator+
```

Returns the vector unaltered.

Returns

Vector<K>

7.48.3.8 operator+() [2/2]

```
template<typename K >
Vector< K > Vector::operator+
```

Equivalent to *v[i] = +v[i]* for all *i* .

Returns

Vector<K>

7.48.3.9 operator+=() [1/2]

```
template<typename K >
Vector< K > & Vector::operator+= (
             const K & alpha )
```

Equivalent to *v[i] += alpha* for all *i* .

Parameters

alpha	

Returns

Vector<K>&

7.48.3.10 operator+=() [2/2]

```
template<typename K >
Vector< K > & Vector::operator+= (
            const Vector< K > & w )
```

If *this* has length zero, than *w* is copied into *this* . If *this* and *w* have the same dimension, this is equivalent to *this->operator[ ](i) += w[i]* for all *i* . Otherwise, an exception is raised.

Exceptions

An	exception is thrown if *this* and *w* have a different number of components and the dimension of *this* is non-zero.

Parameters

w	

Returns

Vector<K>&

7.48.3.11 operator-() [1/2]

```
template<typename K >
Vector< K > Vector::operator-
```

Returns minus the vector.

Returns

Vector<K>

7.48.3.12 operator-() [2/2]

```
template<typename K >
Vector< K > Vector::operator-
```

Equivalent to *v[i] = -v[i]* for all *i* .

Returns

Vector<K>

7.48.3.13 operator-=() [1/2]

```
template<typename K >
Vector< K > & Vector::operator-= (
            const K & alpha )
```

Equivalent to *v[i] -= alpha* for all *i* .

Parameters

alpha	

Returns

Vector<K>&

7.48.3.14 operator-=() [2/2]

```
template<typename K >
Vector< K > & Vector::operator-= (
            const Vector< K > & w )
```

If *this* has length zero, than *-w* is copied into *this* . If *this* and *w* have the same dimension, this is equivalent to *this->operator[ ](i) -= w[i]* for all *i* . Otherwise, an exception is raised.

Exceptions

An	exception is thrown if *this* and *w* have a different number of components and the dimension of *this* is non-zero.

Parameters

w	

Returns

Vector<K>&

7.48.3.15 operator/=() [1/2]

```
template<typename K >
Vector< K > & Vector::operator/= (
            const K & alpha )
```

Equivalent to *v[i] /= alpha* for all *i* .

Parameters

alpha	

Returns

Vector<K>&

7.48.3.16 operator/=() [2/2]

```
template<typename K >
Vector< K > & Vector::operator/= (
            const Vector< K > & w )
```

If *this* and *w* have the same dimension, this is equivalent to *this->operator[ ](i) /= w[i]* for all *i* . Otherwise, an exception is raised.

Exceptions

An	exception is thrown if *this* and *w* have a different number of components.

Parameters

w	

Returns

Vector<K>&

7.48.3.17 operator=() [1/3]

```
template<typename K >
Vector< K > & Vector::operator= (
            const K & alpha )
```

Equivalent to v[i] = alpha for all i.

Parameters

alpha	

Returns

Vector<K>&

7.48.3.18 operator=() [2/3]

```
template<typename K >
Vector< K > & Vector::operator= (
            const Vector< K > & w )
```

Assigment operator.

If needed, the *components* array is freed and dynamically reassigned to match the dimension of *w*. Afterwards, *N* and the components of the Vector *w* are copied to the current object.

Parameters

w	

Returns

Vector<K>&

7.48.3.19 operator=() [3/3]

```
template<typename K >
Vector< K > & Vector::operator= (
            Vector< K > && w )
```

Move-assigment operator.

Copies *components* and *N* instead of creating a new array. Faster than normal assignment operator.

Warning

It leaves *w* in a corrupted state.

Parameters

w	

Returns

Vector<K>&

7.48.3.20 operator[]() [1/2]

```
template<typename K >
K & Vector::operator[] (
            const int i )
```

Indexing operator that allows to use array syntax for accessing the components of a Vector. Checks for out of index errors. Allows for negative indexes.

Exceptions

Throws	an exception if the index supplied is greater than *N* .

Parameters

i	

Returns

K&

7.48.3.21 operator[]() [2/2]

```
template<typename K >
K Vector::operator[] (
            const int i ) const
```

Version of the operator[] for const objects.

Parameters

i	

Returns

K

7.48.3.22 read_from_file()

```
template<typename K >
void Vector::read_from_file (
            FILE * file )
```

Reads the Vector from a binary file. It first reads *N* , then deallocates and reallocates *componentes* and finally reads *components* from the file.

Parameters

file	

7.48.3.23 reverse()

```
template<typename K >
Vector< K > Vector::reverse
```

Returns the Vector reversed: it starts with the last component and it finishes with the first.

Returns

Vector<K>

7.48.3.24 size()

```
template<typename K >
unsigned int Vector::size
```

Returns the length (or dimension) of the Vector.

Returns

unsigned int

7.48.3.25 slice()

```
template<typename K >
Vector< K > Vector::slice (
            int first,
            int last ) const
```

Slicing operator: equivalent to Python v[first: last]. Returns a subvector of *this* whose first component is *v*[first] and whose last component is *v[last - 1]*.

Parameters

first	The first component of the slice will be *v*[first] .
last	The last component of the slice will be *v[last - 1]*.

Exceptions

Throws	an exception if *last* < *first*.

Returns

Vector<K>

7.48.3.26 to_string()

```
template<typename K >
std::string Vector::to_string
```

Returns a string representation of the Vector. It will look similar to (1, 0, -3, 7.3).

Returns

std::string

7.48.3.27 write_to_file()

```
template<typename K >
void Vector::write_to_file (
            FILE * file ) const
```

Writes the Vector to a binary file. It first writes *N* and than *components* .

Parameters

file	

7.48.4 Friends And Related Function Documentation

7.48.4.1 abs

```
template<typename K >
Vector< double > abs (
            const Vector< K > & v )  [friend]
```

Equivalent to *abs(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.2 acos

```
template<typename K >
Vector< K > acos (
            const Vector< K > & v )  [friend]
```

Equivalent to *acos(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.3 acosh

```
template<typename K >
Vector< K > acosh (
            const Vector< K > & v )  [friend]
```

Equivalent to *acosh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.4 asin

```
template<typename K >
Vector< K > asin (
            const Vector< K > & v )  [friend]
```

Equivalent to *asin(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.5 asinh

```
template<typename K >
Vector< K > asinh (
            const Vector< K > & v )  [friend]
```

Equivalent to *asinh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.6 atan

```
template<typename K >
Vector< K > atan (
            const Vector< K > & v )  [friend]
```

Equivalent to *atan(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.7 atan2 [1/3]

```
template<typename K >
Vector< K > atan2 (
            const K v,
            const Vector< K > & w )  [friend]
```

Equivalent to *atan2(v,w[i])* for all *i* .

Parameters

v	
w	

Returns

Vector<K>

7.48.4.8 atan2 [2/3]

```
template<typename K >
Vector< K > atan2 (
            const Vector< K > & v,
            const K w )  [friend]
```

Equivalent to *atan2(v[i],w)* for all *i* .

Parameters

v	
w	

Returns

Vector<K>

7.48.4.9 atan2 [3/3]

```
template<typename K >
Vector< K > atan2 (
```

```
const Vector< K > & v,
const Vector< K > & w )  [friend]
```

Equivalent to *atan2(v[i],w[i])* for all *i* .

Exceptions

Throws	an exception if *v* and *w* have different lengths.

Parameters

v	
w	

Returns

Vector<K>

7.48.4.10 atanh

```
template<typename K >
Vector< K > atanh (
    const Vector< K > & v )  [friend]
```

Equivalent to *atanh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.11 cbrt

```
template<typename K >
Vector< K > cbrt (
    const Vector< K > & v )  [friend]
```

Equivalent to *cbrt(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.12 ceil

```
template<typename K >
Vector< K > ceil (
            const Vector< K > & v )  [friend]
```

Equivalent to *ceil(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.13 conj

```
template<typename K >
Vector< K > conj (
            const Vector< K > & v )  [friend]
```

Equivalent to conj(v[i]) for all i.

Parameters

v	

Returns

Vector<K>

7.48.4.14 copysign [1/2]

```
template<typename K >
Vector< K > copysign (
            const Vector< K > & x,
            const K y )  [friend]
```

Equivalent to *copysign(x[i],y)* for all *i* .

Parameters

x	
y	

Returns

Vector<K>

7.48.4.15 copysign [2/2]

```
template<typename K >
Vector< K > copysign (
            const Vector< K > & x,
            const Vector< K > & y )  [friend]
```

Equivalent to *copysign(x[i],y[i])* for all *i* .

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

7.48.4.16 cos

```
template<typename K >
Vector< K > cos (
            const Vector< K > & v )  [friend]
```

Equivalent to *cos(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.17 cosh

```
template<typename K >
Vector< K > cosh (
            const Vector< K > & v )  [friend]
```

Equivalent to *cosh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.18 erf

```
template<typename K >
Vector< K > erf (
            const Vector< K > & v )  [friend]
```

Equivalent to *erf(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.19 erfc

```
template<typename K >
Vector< K > erfc (
            const Vector< K > & v )  [friend]
```

Equivalent to *erfc(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.20 exp

```
template<typename K >
Vector< K > exp (
            const Vector< K > & v )  [friend]
```

Equivalent to *exp(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.21 exp2

```
template<typename K >
Vector< K > exp2 (
            const Vector< K > & v )  [friend]
```

Equivalent to *exp2(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.22 expm1

```
template<typename K >
Vector< K > expm1 (
            const Vector< K > & v )  [friend]
```

Equivalent to *expm1(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.23 fabs

```
template<typename K >
Vector< double > fabs (
            const Vector< K > & v )  [friend]
```

Equivalent to *fabs(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.24 fdim [1/3]

```
template<typename K >
Vector< K > fdim (
            const K x,
            const Vector< K > & y )  [friend]
```

Equivalent to *fdim(x,y[i])* for all *i* .

Parameters

x	
y	

Returns

Vector<K>

7.48.4.25 fdim [2/3]

```
template<typename K >
Vector< K > fdim (
            const Vector< K > & x,
            const Vector< K > & y ) [friend]
```

Equivalent to *fdim(x[i],y[i])* for all *i* .

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

7.48.4.26 fdim [3/3]

```
template<typename K >
Vector< K > fdim (
            const Vector< K > & x,
            K y ) [friend]
```

Equivalent to *fdim(x[i],y)* for all *i* .

Parameters

x	
y	

Returns

Vector<K>

7.48.4.27 floor

```
template<typename K >
Vector< K > floor (
            const Vector< K > & v ) [friend]
```

Equivalent to *floor(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.28 fma [1/7]

```
template<typename K >
Vector< K > fma (
            const K x,
            const K y,
            const Vector< K > & z ) [friend]
```

Equivalent to *fma(x,y,z[i])* for all *i* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.29 fma [2/7]

```
template<typename K >
Vector< K > fma (
            const K x,
            const Vector< K > & y,
            const K z ) [friend]
```

Equivalent to *fma(x,y[i],z)* for all *i* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.30 fma [3/7]

```
template<typename K >
Vector< K > fma (
            const K x,
            const Vector< K > & y,
            const Vector< K > & z )  [friend]
```

Equivalent to *fma(x,y[i],z[i])* for all *i* .

Exceptions

Throws	an exception if the length of *y* does not equal the length of *z* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.31 fma [4/7]

```
template<typename K >
Vector< K > fma (
            const Vector< K > & x,
            const K y,
            const K z )  [friend]
```

Equivalent to *fma(x[i],y,z)* for all *i* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.32 fma [5/7]

```
template<typename K >
Vector< K > fma (
              const Vector< K > & x,
              const K y,
              const Vector< K > & z )  [friend]
```

Equivalent to *fma(x[i],y,z[i])* for all *i* .

Exceptions

Throws	an exception if the length of *x* does not equal the length of *z* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.33 fma [6/7]

```
template<typename K >
Vector< K > fma (
              const Vector< K > & x,
              const Vector< K > & y,
              const K z )  [friend]
```

Equivalent to *fma(x[i],y[i],z)* for all *i* .

Exceptions

Throws	an exception if the length of *x* does not equal the length of *y* .

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.34 fma [7/7]

```
template<typename K >
Vector< K > fma (
            const Vector< K > & x,
            const Vector< K > & y,
            const Vector< K > & z )  [friend]
```

Equivalent to *fma(x[i],y[i],z[i])* for all *i* .

Exceptions

Throws	and exception if the lenghts of *x* , *y* and *z* are not equal.

Parameters

x	
y	
z	

Returns

Vector<K>

7.48.4.35 fmod [1/3]

```
template<typename K >
Vector< K > fmod (
            const K numer,
            const Vector< K > & denom )  [friend]
```

Equivalent to *fmod(numer,denom[i])* for all *i* .

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.36 fmod [2/3]

```
template<typename K >
Vector< K > fmod (
            const Vector< K > & numer,
            const K denom )  [friend]
```

Equivalent to *fmod(numer[i],denom)* for all *i* .

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.37 fmod [3/3]

```
template<typename K >
Vector< K > fmod (
            const Vector< K > & numer,
            const Vector< K > & denom )  [friend]
```

Equivalent to *fmod(numer[i],denom[i])* for all *i* .

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.38 fpclassify

```
template<typename K >
Vector< int > fpclassify (
            const Vector< K > & v )  [friend]
```

Equivalent to *fpclassify(v[i])* for all *i* .

Parameters

v	

Returns

Vector<int>

7.48.4.39 frexp

```
template<typename K >
Vector< K > frexp (
            const Vector< K > & v,
            Vector< int > * exp )  [friend]
```

Equivalent to *frexp(v[i],exp[i])* for all *i* .

Parameters

v	
exp	

Returns

Vector<K>

7.48.4.40 hypot [1/3]

```
template<typename K >
Vector< K > hypot (
```

```
const K x,
const Vector< K > & y )  [friend]
```

Equivalent to *hypot(x,y[i])* for all *i* .

Parameters

x	
y	

Returns

Vector<K>

7.48.4.41 hypot [2/3]

```
template<typename K >
Vector< K > hypot (
            const Vector< K > & x,
            const K y ) [friend]
```

Equivalent to *hypot(x[i],y)* for all *i* .

Parameters

x	
y	

Returns

Vector<K>

7.48.4.42 hypot [3/3]

```
template<typename K >
Vector< K > hypot (
            const Vector< K > & x,
            const Vector< K > & y ) [friend]
```

Equivalent to *hypot(x[i],y[i])* for all *i* .

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

7.48.4.43 ilogb

```
template<typename K >
Vector< K > ilogb (
            const Vector< K > & v )  [friend]
```

Equivalent to *ilogb(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.44 isfinite

```
template<typename K >
bool isfinite (
            const Vector< K > & v )  [friend]
```

Returns *true* if all components of *v* are finite and *false* otherwise.

Parameters

v	

Returns

true

false

7.48.4.45 isinf

```
template<typename K >
bool isinf (
            const Vector< K > & v )  [friend]
```

Returns *true* if at least one components of *v* is infinite and *false* otherwise.

Parameters

v	

Returns

true

false

7.48.4.46 isnan

```
template<typename K >
bool isnan (
            const Vector< K > & v )  [friend]
```

Returns *true* if at least one components of *v* is nan and *false* otherwise.

Parameters

v	

Returns

true

false

7.48.4.47 isnormal

```
template<typename K >
bool isnormal (
            const Vector< K > & v )  [friend]
```

Returns *true* if all components of *v* are normal and *false* otherwise.

Parameters

v	

Returns

true

false

7.48.4.48 ldexp [1/2]

```
template<typename K >
Vector< K > ldexp (
            const Vector< K > & v,
            const int exp ) [friend]
```

Equivalent to *ldexp(v[i],exp)* for all *i* .

Parameters

v	
exp	

Returns

Vector<K>

7.48.4.49 ldexp [2/2]

```
template<typename K >
Vector< K > ldexp (
            const Vector< K > & v,
            const Vector< int > & exp ) [friend]
```

Equivalent to *ldexp(v[i],exp[i])* for all *i* .

Parameters

v	
exp	

Returns

Vector<K>

7.48.4.50 lgamma

```
template<typename K >
Vector< K > lgamma (
            const Vector< K > & v ) [friend]
```

Equivalent to *lgamma(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.51 llrint

```
template<typename K >
Vector< long long int > llrint (
            const Vector< K > & v ) [friend]
```

Equivalent to *llrint(v[i])* for all *i* .

Parameters

v	

Returns

Vector<long long int>

7.48.4.52 llround

```
template<typename K >
Vector< long long int > llround (
            const Vector< K > & v ) [friend]
```

Equivalent to *llround(v[i])* for all *i* .

Parameters

v	

Returns

Vector<long long int>

7.48.4.53 log

```
template<typename K >
Vector< K > log (
            const Vector< K > & v )  [friend]
```

Equivalent to *log(v[i])* for all *i* .

Exceptions

Throws	an exception if *v* and *exp* have different lengths.

Parameters

v	

Returns

Vector<K>

7.48.4.54 log10

```
template<typename K >
Vector< K > log10 (
            const Vector< K > & v )  [friend]
```

Equivalent to *log10(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.55 log1p

```
template<typename K >
Vector< K > log1p (
            const Vector< K > & v )  [friend]
```

Equivalent to *log1p(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.56 log2

```
template<typename K >
Vector< K > log2 (
            const Vector< K > & v )  [friend]
```

Equivalent to *log2(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.57 logb

```
template<typename K >
Vector< K > logb (
            const Vector< K > & v )  [friend]
```

Equivalent to *logb(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.58 lrint

```
template<typename K >
Vector< long int > lrint (
            const Vector< K > & v ) [friend]
```

Equivalent to *lrint(v[i])* for all *i* .

Parameters

v	

Returns

Vector<long int>

7.48.4.59 lround

```
template<typename K >
Vector< long int > lround (
            const Vector< K > & v ) [friend]
```

Equivalent to *lround(v[i])* for all *i* .

Parameters

v	

Returns

Vector<long int>

7.48.4.60 max

```
template<typename K >
K max (
            const Vector< K > & v ) [friend]
```

Returns the maximal element of the vector.

Parameters

v	

Returns

K

7.48.4.61 min

```
template<typename K >
K min (
            const Vector< K > & v ) [friend]
```

Returns the minimum element of the vector.

Parameters

v	

Returns

K

7.48.4.62 modf

```
template<typename K >
Vector< K > modf (
            const Vector< K > & v,
            Vector< K > * intpart ) [friend]
```

Equivalent to *modf(v[i],intpart[i])* for all *i* .

Parameters

v	
intpart	

Returns

Vector<K>

7.48.4.63 multiply

```
template<typename K >
K multiply (
            const Vector< K > & v ) [friend]
```

Returns the product of all the components of the vector.

Parameters

v	

Returns

K

7.48.4.64 nan

```
template<typename K >
Vector< K > nan (
            const unsigned int N,
            const char * tagp ) [friend]
```

Equivalent to *nan(tagp)* for all *i* .

Parameters

N	
tagp	

Returns

Vector<K>

7.48.4.65 nearbyint

```
template<typename K >
Vector< K > nearbyint (
            const Vector< K > & v ) [friend]
```

Equivalent to *nearbyint(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.66 nextafter

```
template<typename K >
Vector< K > nextafter (
            const Vector< K > & x,
            const Vector< K > & y )  [friend]
```

Equivalent to *nextafter(x[i],y[i])* for all *i* .

Exceptions

Throws	an exception if *x* and *y* have different lenghts.

Parameters

x	
y	

Returns

Vector<K>

7.48.4.67 operator [1/7]

```
template<typename K >
bool operator (
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Returns *true* if *alpha* < *v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.68 operator [2/7]

```
template<typename K >
bool operator (
```

```
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Returns *true* if *alpha* <= *v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.69 operator [3/7]

```
template<typename K >
bool operator (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Returns *true* if *v[i]* < *alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.70 operator [4/7]

```
template<typename K >
bool operator (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Returns *true* if *v[i]* <= *alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.71 operator [5/7]

```
template<typename K >
bool operator (
            const Vector< K > & v,
            const Vector< K > & w )   [friend]
```

Returns true if *v[i]* < *w[i]* for all *i* and false otherwise.

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

7.48.4.72 operator [6/7]

```
template<typename K >
bool operator (
            const Vector< K > & v,
            const Vector< K > & w )   [friend]
```

Returns *true* if *v[i]* <= *w[i]* for all *i* and *false* otherwise.

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

7.48.4.73 operator [7/7]

```
template<typename K >
std::ostream & operator (
            std::ostream & os,
            const Vector< K > & v )  [friend]
```

Allows to print a Vector *v* to screen with *std::cout* << *v*.

Parameters

os	
v	

Returns

std::ostream&

7.48.4.74 operator"!= [1/3]

```
template<typename K >
bool operator!= (
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Returns *true* if *alpha != v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.75 operator"!= [2/3]

```
template<typename K >
bool operator!= (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Returns *true* if *v[i] != alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.76 operator"!= [3/3]

```
template<typename K >
bool operator!= (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Equivalent to *!(v == w)* .

Parameters

v	
w	

Returns

true

false

7.48.4.77 operator& [1/3]

```
template<typename K >
Vector< K > operator& (
            const K & v,
            const Vector< K > & w )  [friend]
```

Concatenation operator.

Creates a new Vector whose components are the scalar *v* followed by the components of *w* .

Parameters

v	
w	

Returns

Vector<K>

7.48.4.78 operator& [2/3]

```
template<typename K >
Vector< K > operator& (
            const Vector< K > & v,
            const K & w )  [friend]
```

Concatenation operator.

Creates a new Vector whose components are the components of *v* followed by the scalar *w* .

Parameters

v	
w	

Returns

Vector<K>

7.48.4.79 operator& [3/3]

```
template<typename K >
Vector< K > operator& (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Concatenation operator.

Creates a new Vector whose components are the components of *v* followed by the components of *w* .

Parameters

v	
w	

Returns

Vector<K>

7.48.4.80 operator* [1/3]

```
template<typename K >
Vector< K > operator* (
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Component wise Vector - scalar multiplication operator.

v ∗ *alpha* is equivalent to *v[i]* ∗ *alpha* for all *i* .

Parameters

alpha	
v	

Returns

Vector<K>

7.48.4.81 operator* [2/3]

```
template<typename K >
Vector< K > operator* (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Component wise scalar - Vector multiplication operator.

alpha ∗ *v* is equivalent to *alpha* ∗ *v[i]* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

7.48.4.82 operator* [3/3]

```
template<typename K >
Vector< K > operator* (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Component wise multiplication operator.

v ∗ *w* is equivalent to *v[i]* ∗ *w[i]* for all *i* .

Exceptions

An	exception is thrown if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

Vector<K>

7.48.4.83 operator+ [1/3]

```
template<typename K >
Vector< K > operator+ (
            const K & alpha,
            const Vector< K > & w )  [friend]
```

Component wise scalar + Vector addition operator.

alpha + *v* is equivalent to *alpha* + *v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

7.48.4.84 operator+ [2/3]

```
template<typename K >
Vector< K > operator+ (
            const Vector< K > & v,
            const K & alpha ) [friend]
```

Componentwise Vector + scalar addition operator.

v + alpha is equivalent to *v[i] + alpha* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

7.48.4.85 operator+ [3/3]

```
template<typename K >
Vector< K > operator+ (
            const Vector< K > & v,
            const Vector< K > & w ) [friend]
```

Componentwise addition operator.

v + w is equivalent to *v[i] + w[i]* for all *i* . If any of the vectors has length zero, then the other one is returned.

Exceptions

A	runtime exception is thrown if both vectors have different number of components and their lengths are both non-zero.

Parameters

v	
w	

Returns

Vector<K>

7.48.4.86 operator- [1/3]

```
template<typename K >
Vector< K > operator- (
            const K & alpha,
            const Vector< K > & w ) [friend]
```

Componentwise scalar - Vector substraction operator.

alpha - *v* is equivalent to *alpha* - *v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

7.48.4.87 operator- [2/3]

```
template<typename K >
Vector< K > operator- (
            const Vector< K > & v,
            const K & alpha ) [friend]
```

Componentwise Vector - scalar substraction operator.

v - *alpha* is equivalent to *v[i]* - *alpha* for all *i*.

Parameters

v	
alpha	

Returns

Vector<K>

7.48.4.88 operator- [3/3]

```
template<typename K >
Vector< K > operator- (
            const Vector< K > & v,
            const Vector< K > & w ) [friend]
```

Componentwise substraction operator.

v - w is equivalent to *v[i] - w[i]* for all *i* . If *v* has null length, then *-w* is returned. If *w* has length zero, then *v* is returned.

Exceptions

A	runtime exception is thrown if both vectors have different number of components and their lengths are both non-zero.

Parameters

v	
w	

Returns

Vector<K>

7.48.4.89 operator/ [1/3]

```
template<typename K >
Vector< K > operator/ (
            const K & alpha,
            const Vector< K > & w )  [friend]
```

Componentwise scalar - Vector division operator.

alpha / v is equivalent to *alpha / v[i]* for all *i* .

Parameters

alpha	
w	

Returns

Vector<K>

7.48.4.90 operator/ [2/3]

```
template<typename K >
Vector< K > operator/ (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Componentwise Vector - scalar division operator.

v / alpha is equivalent to *v[i] / alpha* for all *i* .

Parameters

v	
alpha	

Returns

Vector<K>

7.48.4.91 operator/ [3/3]

```
template<typename K >
Vector< K > operator/ (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Componentwise division operator.

v / *w* is equivalent to *v[i]* / *w[i]* for all *i* .

Exceptions

An	exception is thrown if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

Vector<K>

7.48.4.92 operator== [1/3]

```
template<typename K >
bool operator== (
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Returns *true* if *alpha* == *v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.93 operator== [2/3]

```
template<typename K >
bool operator== (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Returns *true* if *v[i] == alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.94 operator== [3/3]

```
template<typename K >
bool operator== (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

If *v* and *w* have a different number of componentes, *false* is returned. If *v* and *w* have the same number of componenets, then *true* is returned if *v[i] == w[i]* for all *i* and *false* is returned otherwise.

Parameters

v	
w	

Returns

true

false

7.48.4.95 operator> [1/3]

```
template<typename K >
bool operator> (
            const K & alpha,
            const Vector< K > & v )  [friend]
```

Returns *true* if *alpha* > *v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.96 operator> [2/3]

```
template<typename K >
bool operator> (
            const Vector< K > & v,
            const K & alpha )  [friend]
```

Returns *true* if *v[i]* > *alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.97 operator> [3/3]

```
template<typename K >
bool operator> (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Returns *true* if *v[i]* > *w[i]* for all *i* and *false* otherwise.

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

7.48.4.98 operator>= [1/3]

```
template<typename K >
bool operator>= (
            const K & alpha,
            const Vector< K > & v )   [friend]
```

Returns *true* if *alpha >= v[i]* for all *i* and *false* otherwise.

Parameters

alpha	
v	

Returns

true

false

7.48.4.99 operator>= [2/3]

```
template<typename K >
bool operator>= (
            const Vector< K > & v,
            const K & alpha )   [friend]
```

Returns *true* if *v[i] >= alpha* for all *i* and *false* otherwise.

Parameters

v	
alpha	

Returns

true

false

7.48.4.100 operator>= [3/3]

```
template<typename K >
bool operator>= (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Returns *true* if *v[i] >= w[i]* for all *i* and *false* otherwise.

Exceptions

An	exception is raised if *v* and *w* have a different number of components.

Parameters

v	
w	

Returns

true

false

7.48.4.101 operator" |

```
template<typename K >
K operator| (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Scalar product operator.

Performs the following operation:

$$(v|w) = \sum_{i=0}^{N-1} v_i^* w_i$$

which is the canonical scalar product of $\mathbb{K}^N$.

Parameters

v	
w	

Returns

K

7.48.4.102 pow [1/3]

```
template<typename K >
Vector< K > pow (
            const K v,
            const Vector< K > & exponent )  [friend]
```

Equivalent to *pow(v,exponent[i])* for all *i* .

Parameters

v	
exponent	

Returns

Vector<K>

7.48.4.103 pow [2/3]

```
template<typename K >
Vector< K > pow (
            const Vector< K > & v,
            const K exponent )  [friend]
```

Equivalent to *pow(v[i],exponent)* for all *i* .

Parameters

v	
exponent	

Returns

Vector<K>

7.48.4.104 pow [3/3]

```
template<typename K >
Vector< K > pow (
```

```
const Vector< K > & v,
const Vector< K > & exponent )   [friend]
```

Equivalent to *sclbln(v[i],exponent[i])* for all *i* .

Exceptions

Throws	an expception if *v* and *exponent* have different lenghts.

Parameters

v	
exponent	

Returns

Vector<K>

7.48.4.105 remainder [1/3]

```
template<typename K >
Vector< K > remainder (
            const K numer,
            const Vector< K > & denom )   [friend]
```

Equivalent to *remainder(numer,denom[i])* for all *i* .

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.106 remainder [2/3]

```
template<typename K >
Vector< K > remainder (
            const Vector< K > & numer,
            const K denom )   [friend]
```

Equivalent to *remainder(numer[i],denom)* for all *i* .

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.107 remainder [3/3]

```
template<typename K >
Vector< K > remainder (
            const Vector< K > & numer,
            const Vector< K > & denom )  [friend]
```

Equivalent to *remainder(numer[i],denom[i])* for all *i* .

Exceptions

Throws	an exception if *numer* and *denom* have different lengths.

Parameters

numer	
denom	

Returns

Vector<K>

7.48.4.108 remquo [1/3]

```
template<typename K >
Vector< K > remquo (
            const K numer,
            const Vector< K > & denom,
            Vector< int > * quot )  [friend]
```

Equivalent to *remquo(numer,denom[i],quot[i])* for all *i* .

Parameters

numer	
denom	
quot	

Returns

Vector<K>

7.48.4.109 remquo [2/3]

```
template<typename K >
Vector< K > remquo (
            const Vector< K > & numer,
            const K denom,
            Vector< int > * quot ) [friend]
```

Equivalent to *remquo(numer[i],denom,quot[i])* for all *i* .

Parameters

numer	
denom	
quot	

Returns

Vector<K>

7.48.4.110 remquo [3/3]

```
template<typename K >
Vector< K > remquo (
            const Vector< K > & numer,
            const Vector< K > & denom,
            Vector< int > * quot ) [friend]
```

Equivalent to *remquo(numer[i],denom[i],quot[i])* for all *i* .

Exceptions

Throws	an exception if *numer* and *denom* have different lengths.

Parameters

numer	
denom	
quot	

Returns

Vector<K>

7.48.4.111 rint

```
template<typename K >
Vector< K > rint (
            const Vector< K > & v )  [friend]
```

Equivalent to *rint(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.112 round

```
template<typename K >
Vector< K > round (
            const Vector< K > & v )  [friend]
```

Equivalent to *round(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.113 scalbln [1/2]

```
template<typename K >
Vector< K > scalbln (
            const Vector< K > & v,
            const long int n )  [friend]
```

Equivalent to *sclbln(v[i],n[i])* for all *i* .

Parameters

v	
n	

Returns

Vector<K>

7.48.4.114 scalbln [2/2]

```
template<typename K >
Vector< K > scalbln (
            const Vector< K > & v,
            const Vector< long int > & n )  [friend]
```

Equivalent to *sclbln(v[i],n[i])* for all *i* .

Exceptions

Throws	an expception if *v* and *n* have different lenghts.

Parameters

v	
n	

Returns

Vector<K>

7.48.4.115 scalbn [1/2]

```
template<typename K >
Vector< K > scalbn (
            const Vector< K > & v,
            const int n )  [friend]
```

Equivalent to *sclbn(v[i],n)* for all *i* .

Parameters

v	
n	

Returns

Vector<K>

7.48.4.116 scalbn [2/2]

```
template<typename K >
Vector< K > scalbn (
            const Vector< K > & v,
            const Vector< int > & n ) [friend]
```

Equivalent to *sclbn(v[i],n[i])* for all *i* .

Exceptions

Throws	an expception if *v* and *n* have different lenghts.

Parameters

v	
n	

Returns

Vector<K>

7.48.4.117 sin

```
template<typename K >
Vector< K > sin (
            const Vector< K > & v ) [friend]
```

Equivalent to *sin(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.118 sinh

```
template<typename K >
Vector< K > sinh (
            const Vector< K > & v ) [friend]
```

Equivalent to *sinh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.119 sqrt

```
template<typename K >
Vector< K > sqrt (
            const Vector< K > & v ) [friend]
```

Equivalent to *sqrt(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.120 sum

```
template<typename K >
K sum (
            const Vector< K > & v ) [friend]
```

Returns the sum of all the components of the vector.

Parameters

v	

Returns

K

7.48.4.121 tan

```
template<typename K >
Vector< K > tan (
            const Vector< K > & v )  [friend]
```

Equivalent to *tan(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.122 tanh

```
template<typename K >
Vector< K > tanh (
            const Vector< K > & v )  [friend]
```

Equivalent to *tanh(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.123 tgamma

```
template<typename K >
Vector< K > tgamma (
            const Vector< K > & v )  [friend]
```

Equivalent to *tgamma(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.124 trunc

```
template<typename K >
Vector< K > trunc (
            const Vector< K > & v )  [friend]
```

Equivalent to *trunc(v[i])* for all *i* .

Parameters

v	

Returns

Vector<K>

7.48.4.125 vector_product_2d

```
template<typename K >
K vector_product_2d (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Returns the Vector product of two 2d vectors *v* and *w*.

Exceptions

Throws	an exception if either the dimension of *v* or the dimension of *w* is different from two.

Parameters

v	
w	

Returns

K

7.48.4.126 vector_product_3d

```
template<typename K >
Vector< K > vector_product_3d (
            const Vector< K > & v,
            const Vector< K > & w )  [friend]
```

Returns the Vector product of two 3d vectors *v* and *w*.

Exceptions

Throws	an exception if either the dimension of *v* or the dimension of *w* is different from three.

Parameters

v	
w	

Returns

Vector<K>

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

- Math/Vector.hpp
- Math/Vector.cpp

7.49 CPGF::AffineSpace::Vector2d Class Reference

An object that represents a 2D vector.

```
#include <Vector2d.hpp>
```

Public Member Functions

- Vector2d & operator+ ()

  Returns itself.
- Vector2d & operator- ()

  Multiplies all components by -1.
- Vector2d & operator+= (const Vector2d &w)

  Stores in ∗this the sum of ∗this and w.

- Vector2d & operator-= (const Vector2d &w)

    Stores in ∗this the substraction of ∗this and w.
- Vector2d & operator∗= (const Vector2d &w)

    Stores in ∗this the multiplication of ∗this and w.
- Vector2d & operator/= (const Vector2d &w)

    Stores in ∗this the division of ∗this and w.
- Vector2d perp () const

    Returns a perpendicular vector.
- double norm () const

    Returns the euclidean norm of the vector.
- std::string to_string () const

    Retruns a string representation of the vector.
- **Vector2d** ()

    Returns the null vector (0,0).
- Vector2d (double x)

    Returns the vector (x,x).
- Vector2d (double x, double y)

    Returns the vector (x,y).

Public Attributes

- double **x**

    The x component of the vector.
- double **y**

    The y component of the vector.

Friends

- Vector2d operator+ (const Vector2d &v, const Vector2d &w)

    Returns (v.x + w.x, v.y + w.y).
- Vector2d operator- (const Vector2d &v, const Vector2d &w)

    Retruns (v.x - w.x, v.y - w.y).
- Vector2d operator∗ (const Vector2d &v, const Vector2d &w)

    Returns (v.x ∗ w.x, v.y ∗ w.y).
- Vector2d operator/ (const Vector2d &v, const Vector2d &w)

    Retruns (v.x / w.x, v.y / w.y).
- double operator| (const Vector2d &v, const Vector2d &w)

    Retruns the euclidean scalar product of the two vectors.

7.49.1 Detailed Description

An object that represents a 2D vector.

All usual operations +,-,∗,/ are defined component wise. The | operator is used for the euclidean scalar product.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 Vector2d() [1/2]

```
Vector2d::Vector2d (
            double x )
```

Returns the vector (x,x).

Parameters

x	a real number.

7.49.2.2 Vector2d() [2/2]

```
Vector2d::Vector2d (
            double x,
            double y )
```

Returns the vector (x,y).

Parameters

x	a real number.
y	a real number.

7.49.3 Member Function Documentation

7.49.3.1 norm()

```
double Vector2d::norm ( ) const
```

Returns the euclidean norm of the vector.

Returns

double

7.49.3.2 operator∗=()

```
Vector2d & Vector2d::operator*= (
            const Vector2d & w )
```

Stores in ∗this the multiplication of ∗this and w.

Equivalent to ∗this = ∗this ∗ w;

Parameters

w	a 2D vector.

Returns

Vector2d&

7.49.3.3 operator+()

```
Vector2d & Vector2d::operator+ ( )
```

Returns itself.

Returns

Vector2d&

7.49.3.4 operator+=()

```
Vector2d & Vector2d::operator+= (
            const Vector2d & w )
```

Stores in ∗this the sum of ∗this and w.

Equivalent to ∗this = ∗this + w;

Parameters

w	a 2D vector.

Returns

Vector2d&

7.49.3.5 operator-()

```
Vector2d & Vector2d::operator- ( )
```

Multiplies all components by -1.

Returns

Vector2d&

7.49.3.6 operator-=()

```
Vector2d & Vector2d::operator-= (
            const Vector2d & w )
```

Stores in *this the substraction of *this and w.

Equivalent to *this = *this - w;

Parameters

w	a 2D vector.

Returns

Vector2d&

7.49.3.7 operator/=()

```
Vector2d & Vector2d::operator/= (
            const Vector2d & w )
```

Stores in *this the division of *this and w.

Equivalent to *this = *this / w;

Parameters

w	

Returns

Vector2d&

7.49.3.8 perp()

```
Vector2d Vector2d::perp ( ) const
```

Returns a perpendicular vector.

The returned vector is always positioned 90° degrees anticlockwise.

7.49.3.9 to_string()

```
std::string Vector2d::to_string ( ) const
```

Retruns a string representation of the vector.

Returns

std::string

7.49.4 Friends And Related Function Documentation

7.49.4.1 operator*

```
Vector2d operator* (
            const Vector2d & v,
            const Vector2d & w ) [friend]
```

Returns (v.x * w.x, v.y * w.y).

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

7.49.4.2 operator+

```
Vector2d operator+ (
            const Vector2d & v,
            const Vector2d & w ) [friend]
```

Returns (v.x + w.x, v.y + w.y).

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

7.49.4.3 operator-

```
Vector2d operator- (
            const Vector2d & v,
            const Vector2d & w )  [friend]
```

Retruns (v.x - w.x, v.y - w.y).

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

7.49.4.4 operator/

```
Vector2d operator/ (
            const Vector2d & v,
            const Vector2d & w )  [friend]
```

Retruns (v.x / w.x, v.y / w.y).

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

Vector2d

7.49.4.5 operator" |

```
double operator| (
            const Vector2d & v,
            const Vector2d & w )  [friend]
```

Retruns the euclidean scalar product of the two vectors.

Parameters

v	a 2D vector.
w	a 2D vector.

Returns

double

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

- CPGF/AffineSpace2d/Vector2d.hpp
- CPGF/AffineSpace2d/Vector2d.cpp

Chapter 8

File Documentation

8.1 Chemistry/Reaction.hpp File Reference

This file contains objects used to store constants and to compute the kinetics of chemical reactions.

```
#include <cstdio>
#include <string>
```

Classes

- class Chemistry::SolidGasReaction

  This object is used to store all properties of a solid reactant and a gas product. It is also used to compute the speed of the combustion front.

Namespaces

- namespace Chemistry

  The objects of this library are used to store constants and to compute the speed of Chemical Reactions.

8.1.1 Detailed Description

This file contains objects used to store constants and to compute the kinetics of chemical reactions.

Author

Andrés Laín Sanclemente

Version

0.2.0

Date

9th September 2021

8.2 Reaction.hpp

Go to the documentation of this file.

```
#ifndef REACTION_HPP
#define REACTION_HPP

#include <cstdio>
#include <string>

namespace Chemistry
{
    class SolidGasReaction
    {
        public:

        double rho_s;

        double k_s;

        double cV_s;

        double alpha;

        double cV_g;

        double R_g;

        double gamma;

        double a;

        double P_ref;

        double n;

        double delta_H;

         double v_q(const double P) const;

         SolidGasReaction(const double rho_s = 0, const double k_s = 0, const double cV_s = 0,
             const double cV_g = 0, const double R_g = 0, const double a = 0,
             const double P_ref = 0, const double n = 0, const double delta_H = 0);

         explicit SolidGasReaction(FILE* file);

         void read_from_file(FILE* file);

         void write_to_file(FILE* file) const;

         std::string to_string() const;
    };
}

#endif // REACTION_HPP
```

8.3 Point2d.hpp

```
#ifndef POINT2D_HPP
#define POINT2D_HPP

#include "Vector2d.hpp"

namespace CPGF
{
    namespace AffineSpace
    {
        class Point2d
        {
            public:
            double x;
            double y;

            friend Point2d operator+(const Point2d& P, const Vector2d& v);

            friend Vector2d operator-(const Point2d& P, const Point2d& Q);

            friend bool operator==(const Point2d& P, const Point2d& Q);

            Point2d& operator+= (const Vector2d& v);
```

```
            double angle_with_respect_to(const Point2d& Q);

             Point2d& rotate_with_respect_to(const Point2d& Q, const double theta);

             Point2d& rotate_to_with_respect_to(const Point2d& Q, const double theta);

             Point2d& scale_with_respect_to(const Point2d& Q, const Vector2d& s);

             std::string to_string() const;

             Point2d();

             Point2d(double x, double y);
        };
    }

}


#endif // POINT2D_HPP
```

8.4 Vector2d.hpp

```
#ifndef VECTOR2D_HPP
#define VECTOR2D_HPP

#include <string>

namespace CPGF
{
    namespace AffineSpace
    {
        class Vector2d
        {
            public:
            double x;

            double y;

            friend Vector2d operator+(const Vector2d& v, const Vector2d& w);

            friend Vector2d operator-(const Vector2d& v, const Vector2d& w);

            friend Vector2d operator*(const Vector2d& v, const Vector2d& w);

            friend Vector2d operator/(const Vector2d& v, const Vector2d& w);

            friend double operator|(const Vector2d& v, const Vector2d& w);

            Vector2d& operator+();

            Vector2d& operator-();

             Vector2d& operator+= (const Vector2d& w);

             Vector2d& operator-= (const Vector2d& w);

             Vector2d& operator*= (const Vector2d& w);

             Vector2d& operator/= (const Vector2d& w);

             Vector2d perp() const;

             double norm() const;

             std::string to_string() const;

             Vector2d();

             Vector2d(double x);

             Vector2d(double x, double y);
        };
    }
}



#endif // VECTOR2D_HPP
```

8.5 CPGF.hpp

```
#ifndef CPGF_HPP
#define CPGF_HPP

#include "AffineSpace2d/Vector2d.hpp"
#include "AffineSpace2d/Point2d.hpp"
#include "PGFBasics/PGFConf.hpp"
#include "PGFBasics/Strokes2d.hpp"
#include "PGFBasics/Path2d.hpp"
#include "Objects2d/Object2d.hpp"
#include "Objects2d/BasicGeometries.hpp"
#include "Plot2d/Axis.hpp"
#include "Plot2d/DataPlot.hpp"
#include "Plot2d/Graphic.hpp"
#include "Plot2d/LinePlot.hpp"
#include "Text/Text.hpp"
#include "Scene2d.hpp"

#endif // CPGF_HPP
```

8.6 BasicGeometries.hpp

```
#ifndef BASICGEOMETRIES_HPP
#define BASICGEOMETRIES_HPP

#include "Object2d.hpp"
#include "../PGFBasics/Path2d.hpp"

namespace CPGF
{
    namespace Objects2d
     {
        class Line: public Object2d
        {
            public:
            Line(const AffineSpace::Point2d& A, const AffineSpace::Point2d& B,
                const Color& color = Color::BLACK, const double opacity = 1,
                const double line_width = LineWidth::SEMITHICK,
                const std::vector<double>& dash_pattern = DashPatterns::SOLID);
            Line(const std::vector<AffineSpace::Point2d>& points,
                const Color& color = Color::BLACK, const double opacity = 1,
                const double line_width = LineWidth::SEMITHICK,
                const std::vector<double>& dash_pattern = DashPatterns::SOLID);

            AffineSpace::Point2d& start();
            AffineSpace::Point2d& end();

            protected:
            static Object2d builder(const std::vector<AffineSpace::Point2d>& points,
                const Color& color, const double opacity, const double line_width,
                const std::vector<double>& dash_pattern);
        };

        class Circle: public Object2d
        {
            public:
            Circle(const AffineSpace::Point2d& pos, const double radius,
                const bool draw = true, const bool fill = false,
                const Color& draw_color = Color::BLACK,
                const Color& fill_color = Color::WHITE,
                const double opacity = 1,
                const double line_width = LineWidth::SEMITHICK,
                const std::vector<double>& dash_pattern = DashPatterns::SOLID);

            AffineSpace::Point2d center() const;
            double radius() const;

            protected:
            static Object2d builder(const AffineSpace::Point2d& pos, const double radius,
                const bool draw, const bool fill, const Color& draw_color,
                const Color& fill_color, const double opacity, const double line_width,
                const std::vector<double>& dash_pattern);
        };

        class Arrow: public Object2d
        {
            public:
            Arrow(const AffineSpace::Point2d& start,
                const AffineSpace::Point2d& end,
                const double arrow_head_length = 0.15, const double arrow_head_width = 0.3,
                const Color& color = Color::BLACK, const double opacity = 1,
                const double line_width = LineWidth::SEMITHICK,
```

```
                    const std::vector<double>& dash_pattern = DashPatterns::SOLID);

            protected:
            static Object2d builder(const AffineSpace::Point2d& start,
                const AffineSpace::Point2d& end,
                const double arrow_head_length, const double arrow_head_width,
                const Color& color, const double opacity, const double line_width,
                const std::vector<double>& dash_pattern);
        };
    }
}

#endif // BASICGEOMETRIES_HPP
```

8.7 Object2d.hpp

```
#ifndef OBJECT2D_HPP
#define OBJECT2D_HPP

#include <string>
#include <vector>
#include "../PGFBasics/PGFConf.hpp"
#include "../PGFBasics/Path2d.hpp"

namespace CPGF
{
    namespace Objects2d
    {
        class Object2d
        {
            public:
            std::vector<Basics::Path2d> paths;

            Object2d();
            Object2d(const Basics::Path2d& path);
            Object2d(const std::vector<Basics::Path2d>& paths);

            unsigned int size() const;

            Object2d& translate(const AffineSpace::Vector2d& v);
            Object2d& rotate_with_respect_to(const AffineSpace::Point2d& Q, const double theta);
            Object2d& scale_with_respect_to(const AffineSpace::Point2d& Q, const AffineSpace::Vector2d& s);

            friend Object2d operator+(const Object2d& A, const Object2d& B);
            Object2d& operator+=(const Object2d& B);
            Object2d& operator+=(const Basics::Path2d& path);

            std::string render_to_string() const;
        };
    }
}

#endif // OBJECT2D_HPP
```

8.8 Path2d.hpp

```
#ifndef PATH2D_HPP
#define PATH2D_HPP

#include "PGFConf.hpp"
#include "Strokes2d.hpp"
#include <vector>

namespace CPGF
{
    namespace Basics
    {

        class Path2d
        {
            public:
            std::vector<SimpleStroke2d*> strokes;
            PGFConf conf;

            Path2d();
            Path2d(SimpleStroke2d& stroke,
                const PGFConf& conf = PGFConf());
            Path2d(const std::vector<SimpleStroke2d*>& strokes,
```

```
                const PGFConf& conf = PGFConf());
            Path2d(const Path2d& path);
            Path2d& operator=(const Path2d& path);
            ~Path2d();

            Path2d& translate(const AffineSpace::Vector2d& v);
            Path2d& rotate_with_respect_to(const AffineSpace::Point2d& Q, const double theta);
            Path2d& scale_with_respect_to(const AffineSpace::Point2d& Q, const AffineSpace::Vector2d& v);

            AffineSpace::Point2d& start();
            AffineSpace::Point2d start() const;
            AffineSpace::Point2d& end();
            AffineSpace::Point2d end() const;

            double length() const;
            double area() const;

            Path2d& operator+=(SimpleStroke2d& stroke);
            Path2d& add_stroke(SimpleStroke2d& stroke);
            unsigned int size() const;
            std::string render_to_string() const;
        };
    }
}

#endif // PATH2D_HPP
```

8.9 PGFConf.hpp

```
#ifndef PGFCONF_HPP
#define PGFCONF_HPP

#include <vector>
#include <string>

namespace CPGF
{
    enum class DrawType
    {
        DRAW,
        FILL,
    };

    class LineWidth
    {
        public:
        static const double ULTRA_THIN;
        static const double VERY_THIN;
        static const double THIN;
        static const double SEMITHICK;
        static const double THICK;
        static const double VERY_THICK;
        static const double ULTRA_THICK;
    };

    enum class LineCap
    {
        ROUND,
        RECT,
        BUTT
    };

    enum class LineJoin
    {
        ROUND,
        BEVEL,
        MITER
    };

    // Remember miter limit. Currently not available.

    class Color
    {
        public:
        double r;

        double g;

        double b;

        std::string to_string();

         Color();
```

```
        Color(const double r, const double g, const double b);

        static Color mix(const Color& A, const Color& B, const double alpha);

        static Color from_RGB(const unsigned char R, const unsigned char G, const unsigned char B);

        // Color list taken from https://www.rapidtables.com/web/color/RGB_Color.html
        static const Color MAROON;
        static const Color DARK_RED;
        static const Color BROWN;
        static const Color FIREBRICK;
        static const Color CRIMSON;
        static const Color RED;
        static const Color TOMATO;
        static const Color CORAL;
        static const Color INDIAN_RED;
        static const Color LIGHT_CORAL;
        static const Color DARK_SALMON;
        static const Color SALMON;
        static const Color LIGHT_SALMON;
        static const Color ORANGE_RED;
        static const Color DARK_ORANGE;
        static const Color ORANGE;
        static const Color GOLD;
        static const Color DARK_GOLDEN_ROD;
        static const Color GOLDEN_ROD;
        static const Color PALE_GOLDEN_ROD;
        static const Color DARK_KHAKI;
        static const Color KHAKI;
        static const Color OLIVE;
        static const Color YELLOW;
        static const Color YELLOW_GREEN;
        static const Color DARK_OLIVE_GREEN;
        static const Color OLIVE_DRAB;
        static const Color LAWN_GREEN;
        static const Color CHART_REUSE;
        static const Color GREEN_YELLOW;
        static const Color DARK_GREEN;
        static const Color GREEN;
        static const Color FOREST_GREEN;
        static const Color LIME;
        static const Color LIME_GREEN;
        static const Color LIGHT_GREEN;
        static const Color PALE_GREEN;
        static const Color DARK_SEA_GREEN;
        static const Color MEDIUM_SPRING_GREEN;
        static const Color SPRING_GREEN;
        static const Color SEA_GREEN;
        static const Color MEDIUM_AQUA_MARINE;
        static const Color MEDIUM_SEA_GREEN;
        static const Color LIGHT_SEA_GREEN;
        static const Color DARK_SLATE_GRAY;
        static const Color TEAL;
        static const Color DARK_CYAN;
        static const Color CYAN;
        static const Color LIGHT_CYAN;
        static const Color DARK_TURQUOISE;
        static const Color TURQUOISE;
        static const Color MEDIUM_TURQUOISE;
        static const Color PALE_TORQUOISE;
        static const Color AQUA_MARINE;
        static const Color POWDER_BLUE;
        static const Color CADET_BLUE;
        static const Color STEEL_BLUE;
        static const Color CORN_FLOWER_BLUE;
        static const Color DEEP_SKY_BLUE;
        static const Color DODGER_BLUE;
        static const Color LIGHT_BLUE;
        static const Color SKY_BLUE;
        static const Color LIGHT_SKY_BLUE;
        static const Color MIDNIGHT_BLUE;
        static const Color NAVY;
        static const Color DARK_BLUE;
        static const Color MEDIUM_BLUE;
        static const Color BLUE;
        static const Color ROYAL_BLUE;
        static const Color BLUE_VIOLET;
        static const Color INDIGO;
        static const Color DARK_SLATE_BLUE;
        static const Color SLATE_BLUE;
        static const Color MEDIUM_SLATE_BLUE;
        static const Color MEDIUM_PURPLE;
        static const Color DARK_MAGENTA;
        static const Color DARK_VIOLET;
        static const Color DARK_ORCHID;
        static const Color MEDIUM_ORCHID;
```

```
        static const Color PURPLE;
        static const Color THISTLE;
        static const Color PLUM;
        static const Color VIOLET;
        static const Color MAGENTA;
        static const Color ORCHID;
        static const Color MEDIUM_VIOLET_RED;
        static const Color PALE_VIOLET_RED;
        static const Color DEEP_PINK;
        static const Color HOT_PINK;
        static const Color LIGHT_PINK;
        static const Color PINK;
        static const Color ANTIQUE_WHITE;
        static const Color BEIGE;
        static const Color BISQUE;
        static const Color BLANCHED_ALMOND;
        static const Color WHEAT;
        static const Color CORN_SILK;
        static const Color LEMON_CHIFFON;
        static const Color LIGHT_GOLDEN_ROD_YELLOW;
        static const Color LIGHT_YELLOW;
        static const Color SADDLE_BROWN;
        static const Color SIENNA;
        static const Color CHOCOLATE;
        static const Color PERU;
        static const Color SANDY_BROWN;
        static const Color BURLY_WOOD;
        static const Color TAN;
        static const Color ROSY_BROWN;
        static const Color MOCCASIN;
        static const Color NAVAJO_WHITE;
        static const Color PEACH_STUFF;
        static const Color MISTY_ROSE;
        static const Color LAVENDER_BLUSH;
        static const Color LINEN;
        static const Color OLD_LACE;
        static const Color PAPAYA_WHIP;
        static const Color SEA_SHELL;
        static const Color MINT_CREAM;
        static const Color SLATE_GRAY;
        static const Color LIGHT_SLATE_GRAY;
        static const Color LIGHT_STEEL_BLUE;
        static const Color LAVENDER;
        static const Color FLORAL_WHITE;
        static const Color ALICE_BLUE;
        static const Color GHOST_WHITE;
        static const Color HONEYDEW;
        static const Color IVORY;
        static const Color AZURE;
        static const Color SNOW;
        static const Color BLACK;
        static const Color DIM_GRAY;
        static const Color GRAY;
        static const Color DARK_GRAY;
        static const Color SILVER;
        static const Color LIGHT_GRAY;
        static const Color GAINSBORO;
        static const Color WHITE_SMOKE;
        static const Color WHITE;
    };

    class DashPatterns
    {
        public:
        static const std::vector<double> SOLID;
        static const std::vector<double> DASHED;
        static const std::vector<double> DOTTED;
    };

    class PGFConf
    {
        public:
        DrawType draw_type;
        LineCap line_cap;
        LineJoin line_join;
        Color color;

        std::vector<double> dash_pattern;

        double dash_phase;

        double opacity;

        double line_width;

        PGFConf(const DrawType draw_type = DrawType::DRAW,
            const Color& color = Color::BLACK,
```

```
            const double opacity = 1,
            const double line_width = LineWidth::SEMITHICK,
            const std::vector<double>& dash_pattern = std::vector<double>(),
            const double dash_phase = 0,
            const LineCap line_cap = LineCap::BUTT,
            const LineJoin line_join = LineJoin::BEVEL
            );
    };
}


#endif // TIKZCONF_HPP
```

8.10 Strokes2d.hpp

```
#ifndef STROKES2D_HPP
#define STROKES2D_HPP

#include "../AffineSpace2d/Point2d.hpp"
#include <vector>

namespace CPGF
{
    namespace Basics
    {
        class SimpleStroke2d
        {
            public:
            // Clone function and destructor.
            virtual SimpleStroke2d* clone() const = 0;
            virtual ~SimpleStroke2d() = default;

            // Start and end of the stroke.
            virtual AffineSpace::Point2d& start() = 0;
            virtual AffineSpace::Point2d start() const = 0;
            virtual AffineSpace::Point2d& end() = 0;
            virtual AffineSpace::Point2d end() const = 0;

            // Invert stroke.
            // virtual SimpleStroke2d& invert() = 0;

            // Basic affine transformations.
            virtual SimpleStroke2d& translate(const AffineSpace::Vector2d& v) = 0;
            virtual SimpleStroke2d& rotate_with_respect_to(const AffineSpace::Point2d& Q, const double
        theta) = 0;
            virtual SimpleStroke2d& scale_with_respect_to(const AffineSpace::Point2d& Q, const
        AffineSpace::Vector2d& s) = 0;

            // Length and area.
            virtual double length() const = 0;
            virtual double area() const = 0;

            // Intersections.
            virtual std::vector<AffineSpace::Point2d> operator/(const SimpleStroke2d& B) = 0;

            // Return point in specific position. Tangent and normal unitary vectors.
            // virtual AffineSpace::Point2d point_at_position(const double alpha) const = 0;
            // virtual AffineSpace::Vector2d tangent_vector_at_position(const double alpha) const = 0;
            // virtual AffineSpace::Vector2d normal_vector_at_position(const double alpha) const = 0;

            // // Closest point.
            // virtual double closest_position(const AffineSpace::Point2d& P) const = 0;
            // virtual AffineSpace::Point2d closest_point(const AffineSpace::Point2d& P) const = 0;

            // Rendering to string.
            virtual std::string render_to_string() const = 0;
        };

        class StraightStroke2d;
        class BezierStroke2d;

        class StraightStroke2d: public SimpleStroke2d
        {
            public:
            std::vector<AffineSpace::Point2d> points;

            StraightStroke2d();
            StraightStroke2d(const std::vector<AffineSpace::Point2d> points);
            StraightStroke2d* clone() const override;

            StraightStroke2d& operator+=(const AffineSpace::Point2d& P);
            StraightStroke2d& add_point(const AffineSpace::Point2d& P);
            unsigned int size() const;
```

```
            AffineSpace::Point2d& start() override;
            AffineSpace::Point2d start() const override;
            AffineSpace::Point2d& end() override;
            AffineSpace::Point2d end() const override;
            StraightStroke2d& translate(const AffineSpace::Vector2d& v) override;
            StraightStroke2d& rotate_with_respect_to(const AffineSpace::Point2d& Q, const double theta) override;
            StraightStroke2d& scale_with_respect_to(const AffineSpace::Point2d& Q, const AffineSpace::Vector2d& s) override;

            double length() const override;
            double area() const override;

            std::vector<AffineSpace::Point2d> operator/(const SimpleStroke2d& B) override;
            std::vector<AffineSpace::Point2d> operator/(const StraightStroke2d& B);
            // std::vector<AffineSpace::Point2d> operator/(const BezierStroke2d& B);

            std::string render_to_string() const override;
        };

        class BezierStroke2d: public SimpleStroke2d
        {
            public:
             // Start and end points.
             AffineSpace::Point2d P1;
             AffineSpace::Point2d P2;

             // Control Points.
             AffineSpace::Point2d Q1;
             AffineSpace::Point2d Q2;

             BezierStroke2d();
             BezierStroke2d(const AffineSpace::Point2d& P1, const AffineSpace::Point2d& P2,
                 const AffineSpace::Point2d& Q1, const AffineSpace::Point2d& Q2);

             BezierStroke2d* clone() const override;

             AffineSpace::Point2d& start() override;
             AffineSpace::Point2d start() const override;
             AffineSpace::Point2d& end() override;
             AffineSpace::Point2d end() const override;
             BezierStroke2d& translate(const AffineSpace::Vector2d& v) override;
             BezierStroke2d& rotate_with_respect_to(const AffineSpace::Point2d& Q, const double theta) override;
             BezierStroke2d& scale_with_respect_to(const AffineSpace::Point2d& Q, const AffineSpace::Vector2d& s) override;

             double length() const override;
             double area() const override;

             std::vector<AffineSpace::Point2d> operator/(const SimpleStroke2d& B) override;
             // std::vector<AffineSpace::Point2d> operator/(const StraightStroke2d& B);
             // std::vector<AffineSpace::Point2d> operator/(const BezierStroke2d& B);

             std::string render_to_string() const override;
         };
     }
}

#endif // STROKES2D_HPP
```

8.11 Axis.hpp

```
#ifndef AXIS_HPP
#define AXIS_HPP

#include "../PGFBasics/PGFConf.hpp"
#include "../Scene2d.hpp"
#include <functional>
#include "../../Utilities/FormatNumber.hpp"
#include "../Objects2d/BasicGeometries.hpp"
#include <cmath>

namespace CPGF
{
    namespace Plot2d
    {
        enum class AxisType
        {
            HORIZONTAL,
            VERTICAL,
        };
```

```
    enum class AxisScale
    {
        LINEAR,
        LOG,
    };

    enum class NumberPosition
    {
        LEFT,
        RIGHT,
    };

    class Axis
    {
        public:

        AxisType axis_type;

        AxisScale axis_scale;

        double scale;

        bool visible;

        bool inverted;

        std::string label;

        double position;

        Color color;

        double line_width;

        double opacity;

        std::vector<double> dash_pattern;

        double aspect_ratio;

        double arrow_head_length;

        double arrow_head_width;

        double arrow_length;

        std::function<Objects2d::Object2d(const AffineSpace::Point2d& start,
            const AffineSpace::Point2d& end,
            const double arrow_head_length, const double arrow_head_width,
            const Color& color, const double opacity, const double line_width,
            const std::vector<double> dash_pattern)> arrow;

        unsigned int N_major_ticks;

        double major_tick_deviation;

        double major_tick_line_width_divisor;

        bool show_medium_ticks;

        double medium_tick_deviation;

        double medium_tick_line_width_divisor;

        bool show_small_ticks;

        double small_tick_deviation;

        double small_tick_line_width_divisor;

        bool show_numbers;

        NumberPosition number_position;

        bool show_major_grid_lines;

        std::vector<double> major_grid_lines_dash_pattern;

        double major_grid_line_line_width_divisor;

        double major_grid_line_opacity;

        bool show_medium_grid_lines;

        std::vector<double> medium_grid_lines_dash_pattern;

```

```
        double medium_grid_line_line_width_divisor;

        double medium_grid_line_opacity;

        bool show_small_grid_lines;

        std::vector<double> small_grid_lines_dash_pattern;

        double small_grid_line_line_width_divisor;

        double small_grid_line_opacity;

        void set_max_value(const double value);
        void reset_max_value();
        void set_min_value(const double value);
        void reset_min_value();

        void update_max_min(const double max_value, const double min_value);

        void calculate_transformations();

        double get_min_value() const;
        double get_max_value() const;

        std::function<double(double)> axis_transform() const;

        std::function<double(double)> axis_transform_inverse() const;

        Scene2d render_to_scene() const;


        Axis(const AxisType axis_type, const std::string& label = "",
            const AxisScale axis_scale = AxisScale::LINEAR,
            const double scale = 1,
            const bool visible = true, const bool inverted = false,
            const double position = 0, const Color& color = Color::BLACK,
            const double line_width = LineWidth::THIN,
            const double opacity = 1, const std::vector<double>& dash_pattern = DashPatterns::SOLID,
            const double aspect_ratio = 0.6,
            const double arrow_head_length = 0.3,
            const double arrow_head_width = 0.75,
            const double arrow_length = 0.5,
            std::function<Objects2d::Object2d(const AffineSpace::Point2d& start,
            const AffineSpace::Point2d& end,
            const double arrow_head_length, const double arrow_head_width,
            const Color& color, const double opacity, const double line_width,
            const std::vector<double>& dash_pattern)> arrow =
                [] (const AffineSpace::Point2d& start,
                    const AffineSpace::Point2d& end,
                    const double arrow_head_length, const double arrow_head_width,
                    const Color& color, const double opacity, const double line_width,
                    const std::vector<double>& dash_pattern)
                    {return Objects2d::Arrow(start, end, arrow_head_length, arrow_head_width, color,
                    opacity, line_width, dash_pattern);},
            const unsigned int N_major_ticks = 9,
            const double major_tick_deviation = 0.25,
            const double major_tick_line_width_divisor = 3,
            const bool show_medium_ticks = true,
            const double medium_tick_deviation = 0.20,
            const double medium_tick_line_width_divisor = 4,
            const bool show_small_ticks = true,
            const double small_tick_deviation = 0.15,
            const double small_tick_line_width_divisor = 5,
            const bool show_numbers = true,
            const bool show_major_grid_lines = false,
            const std::vector<double>& major_grid_lines_dash_pattern = DashPatterns::SOLID,
            const double major_grid_line_line_width_divisor = 2,
            const double major_grid_line_opacity = 0.75,
            const bool show_medium_grid_lines = false,
            const std::vector<double>& medium_grid_lines_dash_pattern = DashPatterns::SOLID,
            const double medium_grid_line_line_width_divisor = 4,
            const double medium_grid_line_opacity = 0.5,
            const bool show_small_grid_lines = false,
            const std::vector<double>& small_grid_lines_dash_pattern = DashPatterns::SOLID,
            const double small_grid_line_line_width_divisor = 8,
            const double small_grid_line_opacity = 0.25);

        protected:
        bool user_defined_max_value;
        double max_value;
        bool user_defined_min_value;
        double min_value;

        double x_max;

        // Transformation parameters.
        double _a;
```

```
            double _b;

            int digit_max;
            unsigned int precision;

            public:
            // Predefined constants.

            constexpr static const double X_MAX = 15;

            constexpr static const double X_MIN = 0;

            constexpr static const double Y_MIN = 0;

            constexpr static const double NUMBER_DISPLACEMENT = 0.5;

            constexpr static const double LABEL_DISPLACEMENT_HORIZONTAL = 1.2;

            constexpr static const double LABEL_DISPLACEMENT_VERTICAL = 2;
        };
    }
}
#endif // AXIS_HPP
```

8.12 DataPlot.hpp

```
#ifndef DATA_PLOT_HPP
#define DATA_PLOT_HPP

#include "../AffineSpace2d/Point2d.hpp"
#include "../Objects2d/BasicGeometries.hpp"
#include "GraphicObject.hpp"
#include <functional>

namespace CPGF
{
    namespace Plot2d
    {
        class Shapes
        {
            public:
            static Objects2d::Object2d Circle(const AffineSpace::Point2d& pos,
                const Color& color, const double opacity,
                const double size);
            static Objects2d::Object2d Square(const AffineSpace::Point2d& pos,
                const Color& color, const double opacity,
                const double size);
        };

        class DataPlot: public GraphicObject
        {
            public:
            std::vector<AffineSpace::Point2d> points;
            std::function<Color(unsigned int)> color;
            std::function<double(unsigned int)> size;
            std::function<double(unsigned int)> opacity;
            std::function<std::function<Objects2d::Object2d(const AffineSpace::Point2d pos,
                const Color& color, const double opacity, const double size)>
                (unsigned int)> shape;


            DataPlot(const std::vector<double>& Y, const std::vector<double>& X,
                const Color&color = Color::BLUE, const double size = 1,
                const double opacity = 1, std::function<Objects2d::Object2d(
                    const AffineSpace::Point2d pos,
                    const Color& color, const double opacity, const double size)>
                        shape = Shapes::Circle,
                const std::string& legend = "");

            DataPlot(const std::vector<double>& Y, const std::vector<double>& X,
                std::function<Color(unsigned int)> color,
                std::function<double(unsigned int)> size,
                std::function<double(unsigned int)> opacity,
                std::function<std::function<Objects2d::Object2d(
                    const AffineSpace::Point2d pos,
                    const Color& color, const double opacity, const double size)>
                    (unsigned int)> shape,
                const std::string& legend = "");

            DataPlot(const std::vector<double>& Y, const std::vector<double>& X,
                std::function<Color(AffineSpace::Point2d&)> color,
                std::function<double(AffineSpace::Point2d&)> size,
                std::function<double(AffineSpace::Point2d&)> opacity,
```

```
                std::function<std::function<Objects2d::Object2d(
                    const AffineSpace::Point2d pos,
                    const Color& color, const double opacity, const double size)>
                    (AffineSpace::Point2d&)> shape,
                const std::string& legend = "");

            DataPlot(const std::vector<AffineSpace::Point2d>& data,
                const Color&color = Color::BLUE, const double size = 1,
                const double opacity = 1, std::function<Objects2d::Object2d(
                    const AffineSpace::Point2d pos,const Color& color,
                    const double opacity, const double size)> shape =
                    Shapes::Circle,
                const std::string& legend = "");

            DataPlot(const std::vector<AffineSpace::Point2d>& data,
                std::function<Color(unsigned int)> color,
                std::function<double(unsigned int)> size,
                std::function<double(unsigned int)> opacity,
                std::function<std::function<Objects2d::Object2d(const AffineSpace::Point2d pos,
                    const Color& color, const double opacity, const double size)>
                    (unsigned int)> shape,
                const std::string& legend = "");

            DataPlot(const std::vector<AffineSpace::Point2d>& data,
                std::function<Color(AffineSpace::Point2d&)> color,
                std::function<double(AffineSpace::Point2d&)> size,
                std::function<double(AffineSpace::Point2d&)> opacity,
                std::function<std::function<Objects2d::Object2d(const AffineSpace::Point2d pos,
                    const Color& color, const double opacity, const double size)>
                    (AffineSpace::Point2d&)> shape,
                const std::string& legend = "");

            double x_min() const override;
            double x_max() const override;
            double y_min() const override;
            double y_max() const override;

            Objects2d::Object2d miniature(const AffineSpace::Point2d& pos) const override;

            Scene2d render_to_scene(
                std::function<AffineSpace::Point2d(const AffineSpace::Point2d& P)> transform,
                const double x_min, const double x_max, const double y_min, const double y_max) const override;
        };
    }
}

#endif // DATA_PLOT_HPP
```

8.13 Graphic.hpp

```
#ifndef GRAPHIC_HPP
#define GRAPHIC_HPP

#include <vector>
#include "GraphicObject.hpp"
#include "../Scene2d.hpp"
#include "Axis.hpp"

namespace CPGF
{
    namespace Plot2d
    {
        enum class LegendPosition
        {
            LEFT,
            ABOVE,
            RIGHT,
            BELOW,
        };

        class Graphic
        {
            public:
            std::vector<std::tuple<GraphicObject*, Axis*, Axis*» graphic_objects;

            bool show_legend;

            LegendPosition legend_position;

            void add(GraphicObject* object, Axis* Y, Axis* X);

            Scene2d render_to_scene() const;
```

```

            Graphic(const bool show_legend = false);

            std::vector<Axis*> axes;

            constexpr static const double LEGEND_MARGIN = 0.25;
            constexpr static const double LEGEND_VERTICAL_DISPLACEMENT_PER_LINE = 0.25;
            constexpr static const double CHARACTER_WIDTH = 0.1;
        };
    }
}

#endif // GRAPHIC_HPP
```

8.14 GraphicObject.hpp

```
#ifndef GRAPHICOBJECT_HPP
#define GRAPHICOBJECT_HPP

#include "../Scene2d.hpp"
#include "../AffineSpace2d/Point2d.hpp"
#include <functional>

namespace CPGF
{
    namespace Plot2d
    {
        class GraphicObject
        {
            public:
            virtual double x_min() const = 0;
            virtual double x_max() const = 0;
            virtual double y_min() const = 0;
            virtual double y_max() const = 0;

            virtual Objects2d::Object2d miniature(const AffineSpace::Point2d& pos) const = 0;

            virtual Scene2d render_to_scene(
                std::function<AffineSpace::Point2d(const AffineSpace::Point2d& P)> transform,
                const double x_min, const double x_max, const double y_min, const double y_max) const =
    0;
            std::string legend;

            constexpr static const double MINIATURE_HALF_WIDTH = 0.5;
        };
    }
}

#endif // GRAPHICOBJECT_HPP
```

8.15 LinePlot.hpp

```
#ifndef LINE_PLOT_HPP
#define LINE_PLOT_HPP

#include "../AffineSpace2d/Point2d.hpp"
#include "../Objects2d/BasicGeometries.hpp"
#include "GraphicObject.hpp"
#include <functional>

namespace CPGF
{
    namespace Plot2d
    {
        class LinePlot: public GraphicObject
        {
            public:
            std::vector<AffineSpace::Point2d> points;
            std::function<Color(unsigned int)> color;
            std::function<double(unsigned int)> line_width;
            std::function<double(unsigned int)> opacity;
            std::function<std::vector<double>(unsigned int)> dash_pattern;

            LinePlot(const std::vector<double>& Y, const std::vector<double>& X,
                const Color& color = Color::BLUE, const double line_width = LineWidth::THIN,
                const double opacity = 1, const std::vector<double>& dash_pattern = DashPatterns::SOLID,
                const std::string& legend = "");

            LinePlot(const std::vector<double>& Y, const std::vector<double>& X,
```

```
            std::function<Color(unsigned int)> color, std::function<double(unsigned int)> line_width,
            std::function<double(unsigned int)> opacity,
            std::function<std::vector<double>(unsigned int)> dash_pattern,
            const std::string& legend = "");

        LinePlot(const std::vector<double>& Y, const std::vector<double>& X,
            std::function<Color(AffineSpace::Point2d&)> color, std::function<double(AffineSpace::Point2d&)> line_width,
            std::function<double(AffineSpace::Point2d&)> opacity,
            std::function<std::vector<double>(AffineSpace::Point2d&)> dash_pattern,
            const std::string& legend = "");

        LinePlot(const std::vector<AffineSpace::Point2d>& data,
            const Color& color = Color::BLUE, const double line_width = LineWidth::THIN,
            const double opacity = 1, const std::vector<double>& dash_pattern = DashPatterns::SOLID,
            const std::string& legend = "");

        LinePlot(const std::vector<AffineSpace::Point2d>& data,
            std::function<Color(unsigned int)> color, std::function<double(unsigned int)> line_width,
            std::function<double(unsigned int)> opacity,
            std::function<std::vector<double>(unsigned int)> dash_pattern,
            const std::string& legend = "");

        LinePlot(const std::vector<AffineSpace::Point2d>& data,
            std::function<Color(AffineSpace::Point2d&)> color, std::function<double(AffineSpace::Point2d&)> line_width,
            std::function<double(AffineSpace::Point2d&)> opacity,
            std::function<std::vector<double>(AffineSpace::Point2d&)> dash_pattern,
            const std::string& legend = "");

        double x_min() const override;
        double x_max() const override;
        double y_min() const override;
        double y_max() const override;

        Objects2d::Object2d miniature(const AffineSpace::Point2d& pos) const override;

        Scene2d render_to_scene(
            std::function<AffineSpace::Point2d(const AffineSpace::Point2d& P)> transform,
            const double x_min, const double x_max, const double y_min, const double y_max) const override;

        protected:
        bool const_parameters;
    };

    class AveragePlot: public LinePlot
    {
        public:
        AveragePlot(const std::vector<double>& Y, const std::vector<double>& partition,
            const Color& color = Color::BLUE, const double line_width = LineWidth::THIN,
            const double opacity = 1, const std::vector<double>& dash_pattern = DashPatterns::SOLID,
            const std::string& legend = "");

        AveragePlot(const std::vector<double>& Y, const std::vector<double>& partition,
            std::function<Color(unsigned int)> color, std::function<double(unsigned int)> line_width,
            std::function<double(unsigned int)> opacity,
            std::function<std::vector<double>(unsigned int)> dash_pattern,
            const std::string& legend = "");

        AveragePlot(const std::vector<double>& Y, const std::vector<double>& partition,
            std::function<Color(AffineSpace::Point2d&)> color, std::function<double(AffineSpace::Point2d&)> line_width,
            std::function<double(AffineSpace::Point2d&)> opacity,
            std::function<std::vector<double>(AffineSpace::Point2d&)> dash_pattern,
            const std::string& legend = "");

        protected:
        static LinePlot builder(const std::vector<double>& Y, const std::vector<double>& partition,
            const Color& color, const double line_width,
            const double opacity, const std::vector<double>& dash_pattern,
            const std::string& legend);

        static LinePlot builder(const std::vector<double>& Y, const std::vector<double>& partition,
            std::function<Color(unsigned int)> color, std::function<double(unsigned int)> line_width,
            std::function<double(unsigned int)> opacity,
            std::function<std::vector<double>(unsigned int)> dash_pattern,
            const std::string& legend);

        static LinePlot builder(const std::vector<double>& Y, const std::vector<double>& partition,
            std::function<Color(AffineSpace::Point2d&)> color, std::function<double(AffineSpace::Point2d&)> line_width,
            std::function<double(AffineSpace::Point2d&)> opacity,
            std::function<std::vector<double>(AffineSpace::Point2d&)> dash_pattern,
            const std::string& legend);
    };
```

```
    }
}

#endif // LINE_PLOT_HPP
```

8.16 Scene2d.hpp

```
#ifndef SCENE2D_HPP
#define SCENE2D_HPP

#include <vector>
#include <string>

#include "Objects2d/Object2d.hpp"
#include "Text/Text.hpp"

namespace CPGF
{
    class Scene2d
    {
        public:
        std::vector<Objects2d::Object2d*> objects;
        std::vector<Text*> texts;

        Scene2d(const std::vector<Objects2d::Object2d*>& objects = std::vector<Objects2d::Object2d*>(),
            const std::vector<Text*>& texts = std::vector<Text*>());

        Scene2d& add(Objects2d::Object2d& object);
        Scene2d& add(Text& texts);

        Scene2d friend operator+(const Scene2d& S1, const Scene2d& S2);
        Scene2d& operator+=(const Scene2d& S2);
        Scene2d& operator+=(Objects2d::Object2d& Obj);
        Scene2d& operator+=(Text& text);

        std::string render_to_string() const;
        void render(const std::string& filename, const unsigned int density = 100) const;
    };
}

#endif // SCENE2D_HPP
```

8.17 Text.hpp

```
#ifndef TEXT_HPP
#define TEXT_HPP

#include "../AffineSpace2d/Point2d.hpp"
#include "../PGFBasics/PGFConf.hpp"

namespace CPGF
{
    enum class TextAlignment
    {
        CENTER,

        LEFT,

        RIGHT,

        TOP,

        BOTTOM,

        BASE,

        TOP_LEFT,

        TOP_RIGHT,

        BOTTOM_LEFT,

        BOTTOM_RIGHT,

        BASE_LEFT,

        BASE_RIGHT,
    };

```

```
    class Text
    {
        public:
         std::string text;

         AffineSpace::Point2d pos;

         double rot;

         TextAlignment text_alignment;

         Color color;

         Text(const AffineSpace::Point2d& pos = AffineSpace::Point2d(0,0),
             const std::string& text = "",
             const Color& color = Color::BLACK,
             const TextAlignment text_alignment = TextAlignment::CENTER,
             const double rot = 0);

         std::string render_to_string() const;
    };
}

#endif // TEXT_HPP
```

8.18 Math/AlgebraicSolvers.hpp File Reference

This file contains functions that can be used to solve non-linear algebraic equations.

```
#include <functional>
#include "DualNumbers.hpp"
#include "Vector.hpp"
#include "Matrix.hpp"
```

Classes

- class Math::AlgebraicSolvers::LinearSystemSolver

  Solves linear systems of the form Ax=b through LU decomposition with partial pivoting.

Namespaces

- namespace Math

  Namespace that includes mathematical objects such as vectors, matrixes, dual numbers and useful numerical methods such as numerical integrators, ode solvers and solvers for algebraic equations.
- namespace Math::AlgebraicSolvers

  Contains solvers for algebraic equations.

Functions

- double Math::AlgebraicSolvers::bisection (std::function< double(const double x)> f, double a, double b, const unsigned int iter_max, const double abs_tol, const double rel_tol)

  Applies the bisection method to the function f in the interval (a,b).
- double Math::AlgebraicSolvers::secant (std::function< double(const double x)> f, double x1, double x2, const unsigned int iter_max, const double abs_tol, const double rel_tol)

  Applies the secant method to the function f with initial guesses x1 and x2.
- double Math::AlgebraicSolvers::newton_raphson (std::function< double(const double x)>f, std::function< double(const double x)>dfdx, const double x0, const unsigned int iter_max, const double abs_tol, const double rel_tol)

  Applies the Newton-Raphson method to the function f with initial guess x0.
- double Math::AlgebraicSolvers::newton_raphson (std::function< DualNumber< double >(const DualNumber< double > x)>f, const double x0, const unsigned int iter_max, const double abs_tol, const double rel_tol)

  Applies the Newton-Raphson method to the function f with initial guess x0.

8.18.1 Detailed Description

This file contains functions that can be used to solve non-linear algebraic equations.

Author

Andrés Laín Sanclemente

Version

0.2.0

Date

9th September 2021

8.19 AlgebraicSolvers.hpp

Go to the documentation of this file.

```
#ifndef ALGEBRAIC_SOLVERS_HPP
#define ALGEBRAIC_SOLVERS_HPP

#include <functional>
#include "DualNumbers.hpp"
#include "Vector.hpp"
#include "Matrix.hpp"

namespace Math
{
    namespace AlgebraicSolvers
    {
        double bisection(std::function<double(const double x)> f,
            double a, double b,
            const unsigned int iter_max, const double abs_tol, const double rel_tol);

        double secant(std::function<double(const double x)> f,
            double x1, double x2,
            const unsigned int iter_max, const double abs_tol, const double rel_tol);

        double newton_raphson(std::function<double(const double x)>f,
            std::function<double(const double x)>dfdx, const double x0,
            const unsigned int iter_max, const double abs_tol, const double rel_tol);

         double newton_raphson(std::function<DualNumber<double>(const DualNumber<double> x)>f,
             const double x0,
             const unsigned int iter_max, const double abs_tol, const double rel_tol);


         class LinearSystemSolver
         {
             public:
             Vector<double> solve(const Vector<double>& b) const;

             explicit LinearSystemSolver(const Matrix<double>& A);

             protected:

             Matrix<double> L;

             Matrix<double> U;

             std::vector<unsigned int> P;
         };
    }
}

#endif // ALGEBRAIC_SOLVERS_HPP
```

8.20 Math/DualNumbers.hpp File Reference

This file implements dual numbers in one variable to allow for automatic differentiation.

```
#include <string>
```

Classes

- class Math::DualNumber< K >

    Represents a dual number. A mathematical object written like x = a + b, where satisfies ^{2}*=0.*

Namespaces

- namespace Math

    Namespace that includes mathematical objects such as vectors, matrixes, dual numbers and useful numerical methods such as numerical integrators, ode solvers and solvers for algebraic equations.

Functions

- template<typename K >
  DualNumber< K > Math::operator+ (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator+ (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > Math::operator+ (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator- (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator- (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > Math::operator- (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator* (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator* (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > Math::operator* (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator/ (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::operator/ (const DualNumber< K > &x, const K y)
- template<typename K >
  DualNumber< K > Math::operator/ (const K x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > Math::cos (const DualNumber< K > &x)

    *Returns cos(x.a) - x.b*sin(x.a)*
- template<typename K >
  DualNumber< K > Math::sin (const DualNumber< K > &x)

    *Returns sin(x.a) + x.b*cos(x.a)*
- template<typename K >
  DualNumber< K > Math::tan (const DualNumber< K > &x)

Returns tan(x.a) + x.b/cos(x.a)/cos(x.a)∗

- template<typename K >
  DualNumber< K > Math::acos (const DualNumber< K > &x)

  Returns acos(x.a) - 1/sqrt(1-x.a∗x.a)∗x.b∗
- template<typename K >
  DualNumber< K > Math::asin (const DualNumber< K > &x)

  Returns asin(x.a) + 1/sqrt(1-x.a∗x.a)∗x.b∗
- template<typename K >
  DualNumber< K > Math::atan (const DualNumber< K > &x)

  Returns atan(x.a) + 1/(1+x.a∗x.a)∗x.b∗
- template<typename K >
  DualNumber< K > Math::cosh (const DualNumber< K > &x)

  Returns cosh(x.a) + sinh(x.a)∗x.b∗
- template<typename K >
  DualNumber< K > Math::sinh (const DualNumber< K > &x)

  Returns sinh(x.a) + cosh(x.a)∗x.b∗
- template<typename K >
  DualNumber< K > Math::tanh (const DualNumber< K > &x)

  Returns tanh(x.a) + 1/cosh(x.a)/cosh(x.a)∗x.b∗
- template<typename K >
  DualNumber< K > **Math::acosh** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::asinh** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::atanh** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::exp** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::log** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::log10** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::exp2** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::expm1** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::log1p** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::log2** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::logb** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::pow** (const DualNumber< K > &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > **Math::pow** (const DualNumber< K > &x, const K &y)
- template<typename K >
  DualNumber< K > **Math::pow** (const K &x, const DualNumber< K > &y)
- template<typename K >
  DualNumber< K > **Math::sqrt** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::cbrt** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::erf** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::erfc** (const DualNumber< K > &x)

- template<typename K >
  DualNumber< K > **Math::tgamma** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::lgamma** (const DualNumber< K > &x)
- template<typename K >
  DualNumber< K > **Math::fabs** (const DualNumber< K > &x)

8.20.1 Detailed Description

This file implements dual numbers in one variable to allow for automatic differentiation.

Author

Andrés Laín Sanclemente

Version

0.2.0

Date

9th September 2021

8.21 DualNumbers.hpp

Go to the documentation of this file.

```
#ifndef DUALNUMBERS_HPP
#define DUALNUMBERS_HPP

#include <string>

namespace Math
{
    // We make a forward declaration of the class in order to be able to declare
    // friend operators.
    template <typename K>
    class DualNumber;

    // Forward declaration of friend operators.
    template <typename K>
    DualNumber<K> operator+(const DualNumber<K>& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator+(const DualNumber<K>& x, const K y);
    template <typename K>
    DualNumber<K> operator+(const K x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator-(const DualNumber<K>& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator-(const DualNumber<K>& x, const K y);
    template <typename K>
    DualNumber<K> operator-(const K x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator*(const DualNumber<K>& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator*(const DualNumber<K>& x, const K y);
    template <typename K>
    DualNumber<K> operator*(const K x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator/(const DualNumber<K>& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> operator/(const DualNumber<K>& x, const K y);
    template <typename K>
    DualNumber<K> operator/(const K x, const DualNumber<K>& y);

    template <typename K>
```

```
class DualNumber
{
    public:
    K a;

    K b;

    static const DualNumber<K> epsilon;

    DualNumber();

    DualNumber(const K a);

    DualNumber(const K a, const K b);

     friend DualNumber<K> operator+<K>(const DualNumber<K>& x, const DualNumber<K>& y);

     friend DualNumber<K> operator+<K>(const DualNumber<K>& x, const K y);

     friend DualNumber<K> operator+<K>(const K x, const DualNumber<K>& y);

     friend DualNumber<K> operator-<K>(const DualNumber<K>& x, const DualNumber<K>& y);

     friend DualNumber<K> operator-<K>(const DualNumber<K>& x, const K y);

     friend DualNumber<K> operator-<K>(const K x, const DualNumber<K>& y);

     friend DualNumber<K> operator*<K>(const DualNumber<K>& x, const DualNumber<K>& y);

     friend DualNumber<K> operator*<K>(const DualNumber<K>& x, const K y);

     friend DualNumber<K> operator*<K>(const K x, const DualNumber<K>& y);

     friend DualNumber<K> operator/<K>(const DualNumber<K>& x, const DualNumber<K>& y);

     friend DualNumber<K> operator/<K>(const DualNumber<K>& x, const K y);

     friend DualNumber<K> operator/<K>(const K x, const DualNumber<K>& y);

     std::string to_string() const;
 };

 template <typename K>
 DualNumber<K> cos(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> sin(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> tan(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> acos(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> asin(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> atan(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> cosh(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> sinh(const DualNumber<K>& x);

 template <typename K>
 DualNumber<K> tanh(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> acosh(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> asinh(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> atanh(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> exp(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> log(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> log10(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> exp2(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> expm1(const DualNumber<K>& x);
 template <typename K>
 DualNumber<K> log1p(const DualNumber<K>& x);
 template <typename K>
```

```
    DualNumber<K> log2(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> logb(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> pow(const DualNumber<K>& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> pow(const DualNumber<K>& x, const K& y);
    template <typename K>
    DualNumber<K> pow(const K& x, const DualNumber<K>& y);
    template <typename K>
    DualNumber<K> sqrt(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> cbrt(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> erf(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> erfc(const DualNumber<K>& x);
    /* To do. Not implemented. */
    template <typename K>
    DualNumber<K> tgamma(const DualNumber<K>& x);
    /* To do. Not implemented. */
    template <typename K>
    DualNumber<K> lgamma(const DualNumber<K>& x);
    template <typename K>
    DualNumber<K> fabs(const DualNumber<K>& x);
}

#endif // DUALNUMBERS_HPP
```

8.22 Integration.hpp

```
#ifndef INTEGRATION_HPP
#define INTEGRATION_HPP

#include <functional>

namespace Math
{
    namespace Integrators
    {
        double rectangle_rule(std::function<double(double x)> f, const double a, const double b,
            const unsigned int N);

        double trapezoidal_rule(std::function<double(double x)> f, const double a, const double b,
            const unsigned int N);

        double Gauss_Konrad_G7_K15(std::function<double(double x)> f, const double a, const double b,
            const double tol = 1e-6);
    }
}

#endif // INTEGRATION_HPP
```

8.23 Interpolation.hpp

```
#ifndef INTERPOLATION_HPP
#define INTERPOLATION_HPP

#include <vector>

namespace Math
{
    namespace Interpolation
    {
        class AverageLinearInterpolation
        {
            public:

            AverageLinearInterpolation(const std::vector<double>& averages,
                const std::vector<double>& partition);
            AverageLinearInterpolation(double* averages,
                double *partition, unsigned int N_cells);

            AverageLinearInterpolation(const AverageLinearInterpolation& f);
            AverageLinearInterpolation& operator=(const AverageLinearInterpolation& f);
            ~AverageLinearInterpolation();

            double operator()(const double x) const;

```

```
            public:
            unsigned int N_cells;
            double* x_part;
            double* b;
            double* c;
        };

        class AverageQuadraticInterpolation
        {
            public:

            AverageQuadraticInterpolation(const std::vector<double>& averages,
                const std::vector<double>& partition);
            AverageQuadraticInterpolation(double* averages,
                double *partition, unsigned int N_cells);

            AverageQuadraticInterpolation(const AverageQuadraticInterpolation& f);
            AverageQuadraticInterpolation& operator=(const AverageQuadraticInterpolation& f);
            ~AverageQuadraticInterpolation();

            double operator()(const double x) const;

            protected:
            unsigned int N_cells;
            double* x_part;
            double* a;
            double* b;
            double* c;
        };

        class LinearInterpolation
        {
            public:
            LinearInterpolation(const std::vector<double>& Y, const std::vector<double>& X);
            LinearInterpolation(const double* Y, const double* X, const unsigned int N);

            LinearInterpolation(const LinearInterpolation& I);
            LinearInterpolation& operator=(const LinearInterpolation& I);
            ~LinearInterpolation();

            double operator()(const double x) const;

            protected:
            unsigned int N;
            double* x_part;
            double* b;
            double* c;
        };
    }
}

#endif // INTERPOLATION_HPP
```

8.24 Matrix.hpp

```
#ifndef MATRIX_HPP
#define MATRIX_HPP

#include "Vector.hpp"
#include <vector>

double inline fabs(std::complex<double> z)
{
    return std::abs(z);
}

namespace Math
{
    // Forward declaration to make function declaration possible.
    template<typename K>
    class Matrix;

    // We declare all friend operators.
    template <typename K>
    Matrix<K> operator+(const Matrix<K>& A, const Matrix<K>& B);
    template <typename K>
    Matrix<K> operator+(const Matrix<K>& A, const K& alpha);
    template <typename K>
    Matrix<K> operator+( const K& alpha, const Matrix<K>& A);
    template <typename K>
    Matrix<K> operator-(const Matrix<K>& A, const Matrix<K>& B);
    template <typename K>
    Matrix<K> operator-(const Matrix<K>& A, const K& alpha);
```

```
    template <typename K>
    Matrix<K> operator-( const K& alpha, const Matrix<K>& A);
    template <typename K>
    Matrix<K> operator*(const Matrix<K>& A, const K& alpha);
    template <typename K>
    Matrix<K> operator*( const K& alpha, const Matrix<K>& A);
    template <typename K>
    Matrix<K> operator/(const Matrix<K>& A, const K& alpha);
    template <typename K>
    Matrix<K> operator|(const Matrix<K>& A, const Matrix<K>& B);
    template <typename K>
    Vector<K> operator|(const Matrix<K>& A, const Vector<K>& v);
    template <typename K>
    Vector<K> operator|(const Vector<K>& v, const Matrix<K>& A);

    // In the future, define exp, log, sqrt, ...

    template <typename K>
    class Matrix
    {
        public:
        Matrix(unsigned int m = 0, unsigned int n = 0);
        Matrix(unsigned int m, unsigned int n, std::vector<K> elements);

        Matrix(const Matrix<K>& B);
        Matrix& operator=(const Matrix<K>& B);
        ~Matrix();

        operator Matrix<std::complex<K>>() const;

        friend Matrix<K> operator+<K>(const Matrix<K>& A, const Matrix<K>& B);
        friend Matrix<K> operator+<K>(const Matrix<K>& A, const K& alpha);
        friend Matrix<K> operator+<K>(const K& alpha, const Matrix<K>& A);
        friend Matrix<K> operator-<K>(const Matrix<K>& A, const Matrix<K>& B);
        friend Matrix<K> operator-<K>(const Matrix<K>& A, const K& alpha);
        friend Matrix<K> operator-<K>(const K& alpha, const Matrix<K>& A);
        friend Matrix<K> operator*<K>(const Matrix<K>& A, const K& alpha);
        friend Matrix<K> operator*<K>(const K& alpha, const Matrix<K>& A);
        friend Matrix<K> operator/<K>(const Matrix<K>& A, const K& alpha);
        friend Matrix<K> operator|<K>(const Matrix<K>& A, const Matrix<K>& B);
        friend Vector<K> operator|<K>(const Matrix<K>& A, const Vector<K>& v);
        friend Vector<K> operator|<K>(const Vector<K>& v, const Matrix<K>& A);

        Matrix<K>& operator+=(const K& alpha);
        Matrix<K>& operator-=(const K& alpha);
        Matrix<K>& operator*=(const K& alpha);
        Matrix<K>& operator/=(const K& alpha);

        Vector<K>& operator()(const unsigned int i, const std::string& j);

        Vector<K> operator()(const unsigned int i, const std::string& j) const;

        Vector<K> operator()(const std::string& i, const unsigned int j) const;

        K& operator()(const unsigned int i, const unsigned int j);
        K operator()(const unsigned int i, const unsigned int j) const;

        static Matrix<K> zero(const unsigned int m, const unsigned int n);
        static Matrix<K> zero(const unsigned int n);
        static Matrix<K> identity(const unsigned int n);

        unsigned int m() const;

         unsigned int n() const;

         bool is_square() const;

         void LU(Matrix<K>& L, Matrix<K>& U, std::vector<unsigned int>& P) const;

         K det() const;

         K tr() const;

         std::vector<std::complex<double» eigenvalues() const;

         std::vector<Vector<std::complex<double»> eigenvectors(
             std::vector<std::complex<double»& eigenvalues,
             const bool calculate_eigenvalues = true) const;

         std::string to_string() const;

         protected:
         Vector<K>* vec;
         unsigned int _m;
     };
}

```

```
#endif // MATRIX_HPP
```

8.25 ODESolvers.hpp

```
#ifndef ODE_SOLVERS_HPP
#define ODE_SOLVERS_HPP

#include "Vector.hpp"
#include <functional>

namespace Math
{
    namespace ODESolvers
    {
        Vector<double> euler_explicit(const Vector<double>& x, const double t,
            std::function<Vector<double>(const Vector<double>&, const double)> f, const double dt);
        Vector<double> euler_explicit(const Vector<double>& x,
            std::function<Vector<double>(const Vector<double>&)> f, const double dt);

        Vector<double> runge_kutta_order_4(const Vector<double>& x, const double t,
            std::function<Vector<double>(const Vector<double>&, const double)> f, const double dt);
        Vector<double> runge_kutta_order_4(const Vector<double>& x,
            std::function<Vector<double>(const Vector<double>&)> f, const double dt);

        void runge_kutta_fehlberg(Vector<double>& x, double& t, double& dt,
    std::function<Vector<double>(const Vector<double>&, const double)> f,
            const Vector<double>& abs_tol);
        void runge_kutta_fehlberg(Vector<double>& x, double& t, double& dt,
    std::function<Vector<double>(const Vector<double>&)> f,
            const Vector<double>& abs_tol);
    }
}

#endif // ODE_SOLVERS_HPP
```

8.26 Rational.hpp

```
#ifndef RATIONAL_HPP
#define RATIONAL_HPP

#include <string>


namespace Math
{
    // Forward declaration to make function declaration possible.
    template <typename K>
    class Rational;

    // And we now declare all friend operators.
    template <typename K>
    Rational<K>& operator+(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    Rational<K>& operator-(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    Rational<K>& operator*(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    Rational<K>& operator/(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator==(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator!=(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator<(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator<=(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator>(const Rational<K>& p, const Rational<K>& q);
    template <typename K>
    bool operator>=(const Rational<K>& p, const Rational<K>& q);


    template <typename K>
    class Rational
    {
        public:

        K num;
        K den;
```

```
        Rational();
        Rational(const K num);
        Rational(const K num, const K den);

        std::string to_string() const;

        friend Rational<K>& operator+<K>(const Rational<K>& p, const Rational<K>& q);
        friend Rational<K>& operator-<K>(const Rational<K>& p, const Rational<K>& q);
        friend Rational<K>& operator*<K>(const Rational<K>& p, const Rational<K>& q);
        friend Rational<K>& operator/<K>(const Rational<K>& p, const Rational<K>& q);
        friend bool operator==<K>(const Rational<K>& p, const Rational<K>& q);
        friend bool operator!=(const Rational<K>& p, const Rational<K>& q);
        friend bool operator<(const Rational<K>& p, const Rational<K>& q);
        friend bool operator<=(const Rational<K>& p, const Rational<K>& q);
        friend bool operator>(const Rational<K>& p, const Rational<K>& q);
        friend bool operator>=(const Rational<K>& p, const Rational<K>& q);
    };
}


#endif // RATIONAL_HPP
```

8.27 Math/Vector.hpp File Reference

This file contains all prototypes related to class Vector<K>.

```
#include <string>
#include <complex>
```

Classes

- class Math::Vector< K >

    Generic Vector over the field K with any number of components.

Namespaces

- namespace Math

    Namespace that includes mathematical objects such as vectors, matrixes, dual numbers and useful numerical methods such as numerical integrators, ode solvers and solvers for algebraic equations.

Functions

- char Math::conj (const char &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned char Math::conj (const unsigned char &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- short int Math::conj (const short int &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned short int Math::conj (const unsigned short int &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- int Math::conj (const int &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned int Math::conj (const unsigned int &val)

    This function exists to make the definition of the scalar product more general. It just returns val .
- long int Math::conj (const long int &val)

This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned long int Math::conj (const unsigned long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- long long int Math::conj (const long long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- unsigned long long int Math::conj (const unsigned long long int &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- float Math::conj (const float &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- double Math::conj (const double &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- long double Math::conj (const long double &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- wchar_t Math::conj (const wchar_t &val)

  This function exists to make the definition of the scalar product more general. It just returns val .
- template<typename K >
  std::complex< K > Math::conj (const std::complex< K > &val)

  This function exists to make the definition of the scalar product more general. It just returns the complex conjugate of val .
- template<typename K >
  std::string **Math::to_string** (const std::complex< K > &z)
- template<typename K >
  Vector< K > Math::operator+ (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator+ (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > Math::operator+ (const K &alpha, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator- (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator- (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > Math::operator- (const K &alpha, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator* (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator* (const K &alpha, const Vector< K > &v)
- template<typename K >
  Vector< K > Math::operator* (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > Math::operator/ (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator/ (const Vector< K > &v, const K &alpha)
- template<typename K >
  Vector< K > Math::operator/ (const K &alpha, const Vector< K > &w)
- template<typename K >
  bool **Math::operator<** (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool **Math::operator<** (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool **Math::operator<** (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool **Math::operator<=** (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool **Math::operator<=** (const Vector< K > &v, const K &alpha)

- template<typename K >
  bool **Math::operator<=** (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool Math::operator> (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool Math::operator> (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool Math::operator> (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool Math::operator>= (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool Math::operator>= (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool Math::operator>= (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool Math::operator== (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool Math::operator== (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool Math::operator== (const K &alpha, const Vector< K > &v)
- template<typename K >
  bool Math::operator!= (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  bool Math::operator!= (const Vector< K > &v, const K &alpha)
- template<typename K >
  bool Math::operator!= (const K &alpha, const Vector< K > &v)
- template<typename K >
  Vector< K > Math::operator& (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::operator& (const Vector< K > &v, const K &w)
- template<typename K >
  Vector< K > Math::operator& (const K &v, const Vector< K > &w)
- template<typename K >
  K Math::operator| (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  K Math::vector_product_2d (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  Vector< K > Math::vector_product_3d (const Vector< K > &v, const Vector< K > &w)
- template<typename K >
  std::ostream & **Math::operator<<** (std::ostream &os, const Vector< K > &v)
- template<typename K >
  K Math::min (const Vector< K > &v)
- template<typename K >
  K Math::max (const Vector< K > &v)
- template<typename K >
  K Math::sum (const Vector< K > &v)
- template<typename K >
  K Math::multiply (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::cos (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::sin (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::tan (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::acos (const Vector< K > &v)

- template< typename K >
  Vector< K > Math::asin (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::atan (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::atan2 (const K v, const Vector< K > &w)
- template< typename K >
  Vector< K > Math::atan2 (const Vector< K > &v, const K w)
- template< typename K >
  Vector< K > Math::atan2 (const Vector< K > &v, const Vector< K > &w)
- template< typename K >
  Vector< K > Math::cosh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::sinh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::tanh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::acosh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::asinh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::atanh (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::exp (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::frexp (const Vector< K > &v, Vector< int > *exp)
- template< typename K >
  Vector< K > Math::ldexp (const Vector< K > &v, const int exp)
- template< typename K >
  Vector< K > Math::ldexp (const Vector< K > &v, const Vector< int > &exp)
- template< typename K >
  Vector< K > Math::log (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::log10 (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::modf (const Vector< K > &v, Vector< K > *intpart)
- template< typename K >
  Vector< K > Math::exp2 (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::expm1 (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::ilogb (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::log1p (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::log2 (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::logb (const Vector< K > &v)
- template< typename K >
  Vector< K > Math::scalbn (const Vector< K > &v, const int n)
- template< typename K >
  Vector< K > Math::scalbn (const Vector< K > &v, const Vector< int > &n)
- template< typename K >
  Vector< K > Math::scalbln (const Vector< K > &v, const long int n)
- template< typename K >
  Vector< K > Math::scalbln (const Vector< K > &v, const Vector< long int > &n)

- template<typename K >
  Vector< K > Math::pow (const K v, const Vector< K > &exponent)
- template<typename K >
  Vector< K > Math::pow (const Vector< K > &v, const K exponent)
- template<typename K >
  Vector< K > Math::pow (const Vector< K > &v, const Vector< K > &exponent)
- template<typename K >
  Vector< K > Math::sqrt (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::cbrt (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::hypot (const K x, const Vector< K > &y)
- template<typename K >
  Vector< K > Math::hypot (const Vector< K > &x, const K y)
- template<typename K >
  Vector< K > Math::hypot (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< K > Math::erf (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::erfc (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::tgamma (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::lgamma (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::ceil (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::floor (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::fmod (const K numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > Math::fmod (const Vector< K > &numer, const K denom)
- template<typename K >
  Vector< K > Math::fmod (const Vector< K > &numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > Math::trunc (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::round (const Vector< K > &v)
- template<typename K >
  Vector< long int > Math::lround (const Vector< K > &v)
- template<typename K >
  Vector< long long int > Math::llround (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::rint (const Vector< K > &v)
- template<typename K >
  Vector< long int > Math::lrint (const Vector< K > &v)
- template<typename K >
  Vector< long long int > Math::llrint (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::nearbyint (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::remainder (const K numer, const Vector< K > &denom)
- template<typename K >
  Vector< K > Math::remainder (const Vector< K > &numer, const K denom)
- template<typename K >
  Vector< K > Math::remainder (const Vector< K > &numer, const Vector< K > &denom)

- template<typename K >
  Vector< K > Math::remquo (const K numer, const Vector< K > &denom, Vector< int > *quot)
- template<typename K >
  Vector< K > Math::remquo (const Vector< K > &numer, const K denom, Vector< int > *quot)
- template<typename K >
  Vector< K > Math::remquo (const Vector< K > &numer, const Vector< K > &denom, Vector< int > *quot)
- template<typename K >
  Vector< K > Math::copysign (const Vector< K > &x, const K y)
- template<typename K >
  Vector< K > Math::copysign (const Vector< K > &x, const Vector< K > &y)
- template Vector< std::complex< double > > Math::nan (const unsigned int N, const char *tagp)
- template<typename K >
  Vector< K > Math::nextafter (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< K > Math::fdim (const K x, const Vector< K > &y)
- template<typename K >
  Vector< K > Math::fdim (const Vector< K > &x, K y)
- template<typename K >
  Vector< K > Math::fdim (const Vector< K > &x, const Vector< K > &y)
- template<typename K >
  Vector< double > Math::fabs (const Vector< K > &v)
- template<typename K >
  Vector< double > Math::abs (const Vector< K > &v)
- template<typename K >
  Vector< K > Math::fma (const K x, const Vector< K > &y, const Vector< K > &z)
- template<typename K >
  Vector< K > Math::fma (const Vector< K > &x, const K y, const Vector< K > &z)
- template<typename K >
  Vector< K > Math::fma (const Vector< K > &x, const Vector< K > &y, const K z)
- template<typename K >
  Vector< K > Math::fma (const K x, const K y, const Vector< K > &z)
- template<typename K >
  Vector< K > Math::fma (const K x, const Vector< K > &y, const K z)
- template<typename K >
  Vector< K > Math::fma (const Vector< K > &x, const K y, const K z)
- template<typename K >
  Vector< K > Math::fma (const Vector< K > &x, const Vector< K > &y, const Vector< K > &z)
- template<typename K >
  Vector< int > Math::fpclassify (const Vector< K > &v)
- template<typename K >
  bool Math::isfinite (const Vector< K > &v)
- template<typename K >
  bool Math::isinf (const Vector< K > &v)
- template<typename K >
  bool Math::isnan (const Vector< K > &v)
- template<typename K >
  bool Math::isnormal (const Vector< K > &v)
- template<typename K >
  Vector< K > **Math::real** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > **Math::imag** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > **Math::arg** (const Vector< std::complex< K > > &v)
- template<typename K >
  Vector< K > Math::conj (const Vector< K > &v)

8.27.1 Detailed Description

This file contains all prototypes related to class Vector<K>.

Author

Andrés Laín Sanclemente

Version

0.9.0

Date

9th September 2021

8.28 Vector.hpp

Go to the documentation of this file.

```
#ifndef VECTOR_HPP
#define VECTOR_HPP

#include <string>
#include <complex>

namespace Math
{
    char inline conj(const char& val)
    {
        return val;
    }

    unsigned char inline conj(const unsigned char& val)
    {
        return val;
    }

    short int inline conj(const short int& val)
    {
        return val;
    }

    unsigned short int inline conj(const unsigned short int& val)
    {
        return val;
    }

    int inline conj(const int& val)
    {
        return val;
    }

    unsigned int inline conj(const unsigned int& val)
    {
        return val;
    }

    long int inline conj(const long int& val)
    {
        return val;
    }

    unsigned long int inline conj(const unsigned long int& val)
    {
        return val;
    }

    long long int inline conj(const long long int& val)
    {
        return val;
```

```
    }

    unsigned long long int inline conj(const unsigned long long int& val)
    {
        return val;
    }

    float inline conj(const float& val)
    {
        return val;
    }

    double inline conj(const double& val)
    {
        return val;
    }

    long double inline conj(const long double& val)
    {
        return val;
    }

    wchar_t inline conj(const wchar_t& val)
    {
        return val;
    }

    template<typename K>
    std::complex<K> inline conj(const std::complex<K>& val)
    {
        return std::conj(val);
    }

    template<typename K>
    std::string inline to_string(const std::complex<K>& z)
    {
        using std::to_string;
        return to_string(z.real()) + " + " + to_string(z.imag()) + "i";
    }

    // Forward declaration to make function declaration possible.
    template<typename K>
    class Vector;

    // And we now declare all friend operators.
    template<typename K>
    Vector<K> operator+(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    Vector<K> operator+(const Vector<K>& v, const K& alpha);
    template<typename K>
    Vector<K> operator+(const K& alpha, const Vector<K>& w);
    template<typename K>
    Vector<K> operator-(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    Vector<K> operator-(const Vector<K>& v, const K& alpha);
    template<typename K>
    Vector<K> operator-(const K& alpha, const Vector<K>& w);
    template<typename K>
    Vector<K> operator*(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    Vector<K> operator*(const K& alpha, const Vector<K>& v);
    template<typename K>
    Vector<K> operator*(const Vector<K>& v, const K& alpha);
    template<typename K>
    Vector<K> operator/(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    Vector<K> operator/(const Vector<K>& v, const K& alpha);
    template<typename K>
    Vector<K> operator/(const K& alpha, const Vector<K>& w);

    template<typename K>
    bool operator<(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    bool operator<(const Vector<K>& v, const K& alpha);
    template<typename K>
    bool operator<(const K& alpha, const Vector<K>& v);
    template<typename K>
    bool operator<=(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    bool operator<=(const Vector<K>& v, const K& alpha);
    template<typename K>
    bool operator<=(const K& alpha, const Vector<K>& v);
    template<typename K>
    bool operator>(const Vector<K>& v, const Vector<K>& w);
    template<typename K>
    bool operator>(const Vector<K>& v, const K& alpha);
    template<typename K>
```

```
bool operator>(const K& alpha, const Vector<K>& v);
template<typename K>
bool operator>=(const Vector<K>& v, const Vector<K>& w);
template<typename K>
bool operator>=(const Vector<K>& v, const K& alpha);
template<typename K>
bool operator>=(const K& alpha, const Vector<K>& v);
template<typename K>
bool operator==(const Vector<K>& v, const Vector<K>& w);
template<typename K>
bool operator==(const Vector<K>& v, const K& alpha);
template<typename K>
bool operator==(const K& alpha, const Vector<K>& v);
template<typename K>
bool operator!=(const Vector<K>& v, const Vector<K>& w);
template<typename K>
bool operator!=(const Vector<K>& v, const K& alpha);
template<typename K>
bool operator!=(const K& alpha, const Vector<K>& v);

template<typename K>
Vector<K> operator&(const Vector<K>& v, const Vector<K>& w);
template<typename K>
Vector<K> operator&(const Vector<K>& v, const K& w);
template<typename K>
Vector<K> operator&(const K& v, const Vector<K>& w);

template<typename K>
K operator|(const Vector<K>& v, const Vector<K>& w);
template<typename K>
K vector_product_2d(const Vector<K>& v, const Vector<K>& w);
template<typename K>
Vector<K> vector_product_3d(const Vector<K>& v, const Vector<K>& w);

template<typename K>
std::ostream& operator« (std::ostream& os, const Vector<K>& v);

template<typename K>
K min(const Vector<K>& v);
template<typename K>
K max(const Vector<K>& v);
template<typename K>
K sum(const Vector<K>& v);
template<typename K>
K multiply(const Vector<K>& v);
template<typename K>
Vector<K> cos(const Vector<K>& v);
template<typename K>
Vector<K> sin(const Vector<K>& v);
template<typename K>
Vector<K> tan(const Vector<K>& v);
template<typename K>
Vector<K> acos(const Vector<K>& v);
template<typename K>
Vector<K> asin(const Vector<K>& v);
template<typename K>
Vector<K> atan(const Vector<K>& v);
template<typename K>
Vector<K> atan2(const K v, const Vector<K>& w);
template<typename K>
Vector<K> atan2(const Vector<K>& v, const K w);
template<typename K>
Vector<K> atan2(const Vector<K>& v, const Vector<K>& w);
template<typename K>
Vector<K> cosh(const Vector<K>& v);
template<typename K>
Vector<K> sinh(const Vector<K>& v);
template<typename K>
Vector<K> tanh(const Vector<K>& v);
template<typename K>
Vector<K> acosh(const Vector<K>& v);
template<typename K>
Vector<K> asinh(const Vector<K>& v);
template<typename K>
Vector<K> atanh(const Vector<K>& v);
template<typename K>
Vector<K> exp(const Vector<K>& v);
template<typename K>
Vector<K> frexp(const Vector<K>& v, Vector<int>* exp);
template<typename K>
Vector<K> ldexp(const Vector<K>& v, const int exp);
template<typename K>
Vector<K> ldexp(const Vector<K>& v, const Vector<int>& exp);
template<typename K>
Vector<K> log(const Vector<K>& v);
template<typename K>
Vector<K> log10(const Vector<K>& v);
```

```
template<typename K>
Vector<K> modf(const Vector<K>& v, Vector<K>* intpart);
template<typename K>
Vector<K> exp2(const Vector<K>& v);
template<typename K>
Vector<K> expm1(const Vector<K>& v);
template<typename K>
Vector<K> ilogb(const Vector<K>& v);
template<typename K>
Vector<K> log1p(const Vector<K>& v);
template<typename K>
Vector<K> log2(const Vector<K>& v);
template<typename K>
Vector<K> logb(const Vector<K>& v);
template<typename K>
Vector<K> scalbn(const Vector<K>& v, const int n);
template<typename K>
Vector<K> scalbn(const Vector<K>& v, const Vector<int>& n);
template<typename K>
Vector<K> scalbln(const Vector<K>& v, const long int n);
template<typename K>
Vector<K> scalbln(const Vector<K>& v, const Vector<long int>& n);
template<typename K>
Vector<K> pow(const K v, const Vector<K>& exponent);
template<typename K>
Vector<K> pow(const Vector<K>& v, const K exponent);
template<typename K>
Vector<K> pow(const Vector<K>& v, const Vector<K>& exponent);
template<typename K>
Vector<K> sqrt(const Vector<K>& v);
template<typename K>
Vector<K> cbrt(const Vector<K>& v);
template<typename K>
Vector<K> hypot(const K x, const Vector<K>& y);
template<typename K>
Vector<K> hypot(const Vector<K>& x, const K y);
template<typename K>
Vector<K> hypot(const Vector<K>& x, const Vector<K>& y);
template<typename K>
Vector<K> erf(const Vector<K>& v);
template<typename K>
Vector<K> erfc(const Vector<K>& v);
template<typename K>
Vector<K> tgamma(const Vector<K>& v);
template<typename K>
Vector<K> lgamma(const Vector<K>& v);
template<typename K>
Vector<K> ceil(const Vector<K>& v);
template<typename K>
Vector<K> floor(const Vector<K>& v);
template<typename K>
Vector<K> fmod(const K numer, const Vector<K>& denom);
template<typename K>
Vector<K> fmod(const Vector<K>& numer, const K denom);
template<typename K>
Vector<K> fmod(const Vector<K>& numer, const Vector<K>& denom);
template<typename K>
Vector<K> trunc(const Vector<K>& v);
template<typename K>
Vector<K> round(const Vector<K>& v);
template<typename K>
Vector<long int> lround(const Vector<K>& v);
template<typename K>
Vector<long long int> llround(const Vector<K>& v);
template<typename K>
Vector<K> rint(const Vector<K>& v);
template<typename K>
Vector<long int> lrint(const Vector<K>& v);
template<typename K>
Vector<long long int> llrint(const Vector<K>& v);
template<typename K>
Vector<K> nearbyint(const Vector<K>& v);
template<typename K>
Vector<K> remainder(const K numer, const Vector<K>& denom);
template<typename K>
Vector<K> remainder(const Vector<K>& numer, const K denom);
template<typename K>
Vector<K> remainder(const Vector<K>& numer, const Vector<K>& denom);
template<typename K>
Vector<K> remquo(const K numer, const Vector<K>& denom, Vector<int>* quot);
template<typename K>
Vector<K> remquo(const Vector<K>& numer, const K denom, Vector<int>* quot);
template<typename K>
Vector<K> remquo(const Vector<K>& numer, const Vector<K>& denom, Vector<int>* quot);
template<typename K>
Vector<K> copysign(const Vector<K>& x, const K y);
template<typename K>
```

```
Vector<K> copysign(const Vector<K>& x, const Vector<K>& y);
template<typename K>
Vector<K> nan(const unsigned int N, const char* tagp);
template<typename K>
Vector<K> nextafter(const Vector<K>& x, const Vector<K>& y);
template<typename K>
Vector<K> fdim(const K x, const Vector<K>& y);
template<typename K>
Vector<K> fdim(const Vector<K>& x, K y);
template<typename K>
Vector<K> fdim(const Vector<K>& x, const Vector<K>& y);
template<typename K>
Vector<double> fabs(const Vector<K>& v);
template<typename K>
Vector<double> abs(const Vector<K>& v);
template<typename K>
Vector<K> fma(const K x, const Vector<K>& y, const Vector<K>& z);
template<typename K>
Vector<K> fma(const Vector<K>& x, const K y, const Vector<K>& z);
template<typename K>
Vector<K> fma(const Vector<K>& x, const Vector<K>& y, const K z);
template<typename K>
Vector<K> fma(const K x, const K y, const Vector<K>& z);
template<typename K>
Vector<K> fma(const K x, const Vector<K>& y, const K z);
template<typename K>
Vector<K> fma(const Vector<K>& x, const K y, const K z);
template<typename K>
Vector<K> fma(const Vector<K>& x, const Vector<K>& y, const Vector<K>& z);
template<typename K>
Vector<int> fpclassify(const Vector<K>& v);
template<typename K>
bool isfinite(const Vector<K>& v);
template<typename K>
bool isinf(const Vector<K>& v);
template<typename K>
bool isnan(const Vector<K>& v);
template<typename K>
bool isnormal(const Vector<K>& v);

template<typename K>
Vector<K> real(const Vector<std::complex<K»& v);
template<typename K>
Vector<K> imag(const Vector<std::complex<K»& v);
template<typename K>
Vector<K> arg(const Vector<std::complex<K»& v);
template<typename K>
Vector<K> conj(const Vector<K>& v);


template<typename K>
class Vector
{
    public:
    K* components;

    unsigned int N;

    public:
    // Constructors and destructor.

    Vector<K>();

    explicit Vector<K>(const unsigned int N);

    Vector<K>(const Vector<K>& w);

    Vector<K>(Vector<K>&& w);

    Vector<K>(const K* components, const unsigned int N);

    Vector<K>(std::initializer_list<K> list);

    ~Vector<K>();

    // Assignment operators

    Vector<K>& operator=(const K& alpha);

    Vector<K>& operator=(const Vector<K>& w);

    Vector<K>& operator=(Vector<K>&& w);

    // Operators defined componentwise

    friend Vector<K> operator+<K>(const Vector<K>& v, const Vector<K>& w);

```

```
        friend Vector<K> operator+<K>(const Vector<K>& v, const K& alpha);
        friend Vector<K> operator+<K>(const K& alpha, const Vector<K>& w);
        friend Vector<K> operator-<K>(const Vector<K>& v, const Vector<K>& w);
        friend Vector<K> operator-<K>(const Vector<K>& v, const K& alpha);
        friend Vector<K> operator-<K>(const K& alpha, const Vector<K>& w);
        friend Vector<K> operator*<K>(const Vector<K>& v, const Vector<K>& w);
        friend Vector<K> operator*<K>(const K& alpha, const Vector<K>& v);
        friend Vector<K> operator*<K>(const Vector<K>& v, const K& alpha);
        friend Vector<K> operator/<K>(const Vector<K>& v, const Vector<K>& w);
        friend Vector<K> operator/<K>(const Vector<K>& v, const K& alpha);
        friend Vector<K> operator/<K>(const K& alpha, const Vector<K>& w);
        // The following operators are done component by component and && together.
        friend bool operator< <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator< <K>(const Vector<K>& v, const K& alpha);
        friend bool operator< <K>(const K& alpha, const Vector<K>& v);
        friend bool operator<= <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator<= <K>(const Vector<K>& v, const K& alpha);
        friend bool operator<= <K>(const K& alpha, const Vector<K>& v);
        friend bool operator> <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator> <K>(const Vector<K>& v, const K& alpha);
        friend bool operator> <K>(const K& alpha, const Vector<K>& v);
        friend bool operator>= <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator>= <K>(const Vector<K>& v, const K& alpha);
        friend bool operator>= <K>(const K& alpha, const Vector<K>& v);
        friend bool operator== <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator== <K>(const Vector<K>& v, const K& alpha);
        friend bool operator== <K>(const K& alpha, const Vector<K>& v);
        friend bool operator!= <K>(const Vector<K>& v, const Vector<K>& w);
        friend bool operator!= <K>(const Vector<K>& v, const K& alpha);
        friend bool operator!= <K>(const K& alpha, const Vector<K>& v);
        friend Vector<K> operator&<K>(const Vector<K>& v, const Vector<K>& w);
        friend Vector<K> operator&<K>(const Vector<K>& v, const K& w);
        friend Vector<K> operator&<K>(const K& v, const Vector<K>& w);
        friend K operator|<K>(const Vector<K>& v, const Vector<K>& w);
        friend K vector_product_2d<K>(const Vector<K>& v, const Vector<K>& w);
        friend Vector<K> vector_product_3d<K>(const Vector<K>& v, const Vector<K>& w);
        // Unary operators: +, -
        Vector<K> operator+();
        Vector<K> operator-();
        // Basic math-assignment operators: +=, -=, *=, /=
        Vector<K>& operator+=(const K& alpha);
        Vector<K>& operator+=(const Vector<K>& w);
        Vector<K>& operator-=(const K& alpha);
        Vector<K>& operator-=(const Vector<K>& w);
        Vector<K>& operator*=(const K& alpha);
```

```
        Vector<K>& operator*=(const Vector<K>& w);

        Vector<K>& operator/=(const K& alpha);

        Vector<K>& operator/=(const Vector<K>& w);

        // Basic unary operators: +, -
        Vector<K> operator+() const;

        Vector<K> operator-() const;

        // Other operators and functions
        K& operator[] (const int i);

        K operator[] (const int i) const;

        Vector<K> slice(int first, int last) const;

        Vector<K> reverse() const;

        unsigned int size() const;

        double norm_2() const;

        double norm_1() const;

        double norm_inf() const;

        double norm_p(const double p) const;

        std::string to_string() const;

        friend std::ostream& operator« <K>(std::ostream& os, const Vector<K>& v);

        void read_from_file(FILE* file);

        void write_to_file(FILE* file) const;

        friend K min<K>(const Vector<K>& v);

        friend K max<K>(const Vector<K>& v);

        friend K sum<K>(const Vector<K>& v);

        friend K multiply<K>(const Vector<K>& v);

        // All math functions component by component
        friend Vector<K> cos<K>(const Vector<K>& v);

        friend Vector<K> sin<K>(const Vector<K>& v);

        friend Vector<K> tan<K>(const Vector<K>& v);

        friend Vector<K> acos<K>(const Vector<K>& v);

        friend Vector<K> asin<K>(const Vector<K>& v);

        friend Vector<K> atan<K>(const Vector<K>& v);

        friend Vector<K> atan2<K>(const K v, const Vector<K>& w);

        friend Vector<K> atan2<K>(const Vector<K>& v, const K w);

        friend Vector<K> atan2<K>(const Vector<K>& v, const Vector<K>& w);

        friend Vector<K> cosh<K>(const Vector<K>& v);

        friend Vector<K> sinh<K>(const Vector<K>& v);

        friend Vector<K> tanh<K>(const Vector<K>& v);

        friend Vector<K> acosh<K>(const Vector<K>& v);

        friend Vector<K> asinh<K>(const Vector<K>& v);

        friend Vector<K> atanh<K>(const Vector<K>& v);

        friend Vector<K> exp<K>(const Vector<K>& v);

        friend Vector<K> frexp<K>(const Vector<K>& v, Vector<int>* exp);

        friend Vector<K> ldexp<K>(const Vector<K>& v, const int exp);

        friend Vector<K> ldexp<K>(const Vector<K>& v, const Vector<int>& exp);

        friend Vector<K> log<K>(const Vector<K>& v);
```

```
        friend Vector<K> log10<K>(const Vector<K>& v);
        friend Vector<K> modf<K>(const Vector<K>& v, Vector<K>* intpart);
        friend Vector<K> exp2<K>(const Vector<K>& v);
        friend Vector<K> expm1<K>(const Vector<K>& v);
        friend Vector<K> ilogb<K>(const Vector<K>& v);
        friend Vector<K> log1p<K>(const Vector<K>& v);
        friend Vector<K> log2<K>(const Vector<K>& v);
        friend Vector<K> logb<K>(const Vector<K>& v);
        friend Vector<K> scalbn<K>(const Vector<K>& v, const int n);
        friend Vector<K> scalbn<K>(const Vector<K>& v, const Vector<int>& n);
        friend Vector<K> scalbln<K>(const Vector<K>& v, const long int n);
        friend Vector<K> scalbln<K>(const Vector<K>& v, const Vector<long int>& n);
        friend Vector<K> pow<K>(const K v, const Vector<K>& exponent);
        friend Vector<K> pow<K>(const Vector<K>& v, const K exponent);
        friend Vector<K> pow<K>(const Vector<K>& v, const Vector<K>& exponent);
        friend Vector<K> sqrt<K>(const Vector<K>& v);
        friend Vector<K> cbrt<K>(const Vector<K>& v);
        friend Vector<K> hypot<K>(const K x, const Vector<K>& y);
        friend Vector<K> hypot<K>(const Vector<K>& x, const K y);
        friend Vector<K> hypot<K>(const Vector<K>& x, const Vector<K>& y);
        friend Vector<K> erf<K>(const Vector<K>& v);
        friend Vector<K> erfc<K>(const Vector<K>& v);
        friend Vector<K> tgamma<K>(const Vector<K>& v);
        friend Vector<K> lgamma<K>(const Vector<K>& v);
        friend Vector<K> ceil<K>(const Vector<K>& v);
        friend Vector<K> floor<K>(const Vector<K>& v);
        friend Vector<K> fmod<K>(const K numer, const Vector<K>& denom);
        friend Vector<K> fmod<K>(const Vector<K>& numer, const K denom);
        friend Vector<K> fmod<K>(const Vector<K>& numer, const Vector<K>& denom);
        friend Vector<K> trunc<K>(const Vector<K>& v);
        friend Vector<K> round<K>(const Vector<K>& v);
        friend Vector<long int> lround<K>(const Vector<K>& v);
        friend Vector<long long int> llround<K>(const Vector<K>& v);
        friend Vector<K> rint<K>(const Vector<K>& v);
        friend Vector<long int> lrint<K>(const Vector<K>& v);
        friend Vector<long long int> llrint<K>(const Vector<K>& v);
        friend Vector<K> nearbyint<K>(const Vector<K>& v);
        friend Vector<K> remainder<K>(const K numer, const Vector<K>& denom);
        friend Vector<K> remainder<K>(const Vector<K>& numer, const K denom);
        friend Vector<K> remainder<K>(const Vector<K>& numer, const Vector<K>& denom);
        friend Vector<K> remquo<K>(const K numer, const Vector<K>& denom, Vector<int>* quot);
        friend Vector<K> remquo<K>(const Vector<K>& numer, const K denom, Vector<int>* quot);
        friend Vector<K> remquo<K>(const Vector<K>& numer, const Vector<K>& denom, Vector<int>* quot);
        friend Vector<K> copysign<K>(const Vector<K>& x, const K y);
```

```
        friend Vector<K> copysign<K>(const Vector<K>& x, const Vector<K>& y);

        friend Vector<K> nan<K>(const unsigned int N, const char* tagp);

        friend Vector<K> nextafter<K>(const Vector<K>& x, const Vector<K>& y);

        friend Vector<K> fdim<K>(const K x, const Vector<K>& y);

        friend Vector<K> fdim<K>(const Vector<K>& x, K y);

        friend Vector<K> fdim<K>(const Vector<K>& x, const Vector<K>& y);

        friend Vector<double> fabs<K>(const Vector<K>& v);

        friend Vector<double> abs<K>(const Vector<K>& v);

        friend Vector<K> fma<K>(const K x, const Vector<K>& y, const Vector<K>& z);

        friend Vector<K> fma<K>(const Vector<K>& x, const K y, const Vector<K>& z);

        friend Vector<K> fma<K>(const Vector<K>& x, const Vector<K>& y, const K z);

        friend Vector<K> fma<K>(const K x, const K y, const Vector<K>& z);

        friend Vector<K> fma<K>(const K x, const Vector<K>& y, const K z);

        friend Vector<K> fma<K>(const Vector<K>& x, const K y, const K z);

        friend Vector<K> fma<K>(const Vector<K>& x, const Vector<K>& y, const Vector<K>& z);

        friend Vector<int> fpclassify<K>(const Vector<K>& v);

        friend bool isfinite<K>(const Vector<K>& v);

        friend bool isinf<K>(const Vector<K>& v);

        friend bool isnan<K>(const Vector<K>& v);

        friend bool isnormal<K>(const Vector<K>& v);

        // Complex functions
        friend Vector<K> conj<K>(const Vector<K>& v);
    };
}

#endif // VECTOR_HPP
```

8.29 Mesh/Cell.hpp File Reference

This files contains all declarations related to the basic objects of all meshes: the cells.

```
#include "../Math/Vector.hpp"
#include "../Chemistry/Reaction.hpp"
#include <functional>
#include <string>
```

Classes

- class BaseCell

  The basic structure of the space discretization.
- class GasCell
- class SolidCell

8.29.1 Detailed Description

This files contains all declarations related to the basic objects of all meshes: the cells.

Author

Andrés Laín Sanclemente

Version

0.2.0

Date

9th September 2021

8.30 Cell.hpp

Go to the documentation of this file.

```
#ifndef CELL_HPP
#define CELL_HPP

#include "../Math/Vector.hpp"
#include "../Chemistry/Reaction.hpp"
#include <functional>
#include <string>


class BaseCell
{
    public:
    Math::Vector<double> U;

    double a;

    double b;

    double A;

    Chemistry::SolidGasReaction* QR;

    BaseCell(const Math::Vector<double>& U, const double a, const double b, const double A,
        Chemistry::SolidGasReaction* QR);

    BaseCell(FILE* file, Chemistry::SolidGasReaction* QR);

    double x() const;

    double len() const;

    void read_from_file(FILE* file);

    void write_to_file(FILE* file) const;

    std::string to_string() const;
};


class GasCell: public BaseCell
{
    public:
    // Constructors.
    GasCell(const Math::Vector<double>& U, double a, double b, const double A,
        Chemistry::SolidGasReaction* QR);

    GasCell(FILE* file, Chemistry::SolidGasReaction* QR);

    Math::Vector<double> F() const;

    GasCell* right_neighbour;
```

```
    GasCell* left_neighbour;
    double rho;
    double c;
    double v;
    double T;
    double P;
    double H;
    double E;
    double M;
    void update();
    void read_from_file(FILE* file);
    std::string to_string() const;
};
class SolidCell: public BaseCell
{
    public:
    SolidCell(const Math::Vector<double>& U, double a, double b, const double A,
        Chemistry::SolidGasReaction* QR);
    SolidCell(FILE* file, Chemistry::SolidGasReaction* QR);
    SolidCell* right_neighbour;
    SolidCell* left_neighbour;
    double T;
    void update();
    void read_from_file(FILE* file);
    std::string to_string() const;
};
#endif // CELL_HPP
```

8.31 Mesh/Mesh.hpp File Reference

This files contains all declarations related to the mesh.

```
#include "Cell.hpp"
#include <functional>
#include <vector>
```

Classes

- class Mesh< Cell >

  This object represents a collection of cells. The number of cells used can be changed. Methods for adaptive refinement are included.
- class Mesh< Cell >::Iterator

  Iterator object to loop through all the cells of the mesh.
- class GasMesh
- class SolidMesh

8.31.1 Detailed Description

This files contains all declarations related to the mesh.

Author

Andrés Laín Sanclemente

Version

0.2.0

Date

9th September 2021

8.32 Mesh.hpp

Go to the documentation of this file.

```
#ifndef MESH_HPP
#define MESH_HPP

#include "Cell.hpp"
#include <functional>
#include <vector>

template <class Cell>
class Mesh
{
    public:
    Cell* first_cell;

    Cell* last_cell;

    Chemistry::SolidGasReaction QR;

    unsigned int N_cells() const;

    double detail_subdivide_threshold;

    double detail_merge_threshold;

    double max_length_factor;

    double min_length_factor;

    double boundary_cell_max_length_factor;

    std::function<double(const double x)> A_func;

    class Iterator
    {
        public:
         Cell* cell;

         Iterator();

         Iterator(Cell* cell);

         Iterator(const Iterator& J);

         Iterator& operator=(const Iterator& J);

         Iterator operator+(const int i);

         Iterator operator-(const int i);

         Iterator operator++(int);

         Iterator& operator++();

```

```
        Iterator operator--(int);

        Iterator& operator--();

        Cell& operator*();

        Cell* operator->();

        operator Cell*() const;

        bool operator==(const Iterator& J);

        bool operator!=(const Iterator& J);
    };

    // Constructors.

    Mesh(Cell* first_cell = nullptr, Cell* last_cell = nullptr,
        const Chemistry::SolidGasReaction& QR = Chemistry::SolidGasReaction(),
        std::function<double(const double x)> A_func = [](double x){return 0;},
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.001,
        const double max_length_factor = 1./50,
        const double min_length_factor = 1./2000,
        const double boundary_cell_max_length_factor = 1./1000);

    Mesh(const Mesh& mesh);

    Mesh(Mesh&& mesh);

    Mesh& operator=(const Mesh& mesh);

    ~Mesh();

    void free();

    Mesh::Iterator begin() const;

    Mesh::Iterator rbegin() const;

    Mesh::Iterator end() const;

    Cell* subdivide_at(Cell* C);

    Cell* merge_cells(Cell* L, Cell* R);

    void calculate_variable_ranges();

    double detail(Cell* L, Cell* R) const;

    void optimize_mesh();

    std::vector<double> x() const;

    std::vector<double> x_partition() const;

    std::vector<double> A() const;

    std::vector<Math::Vector<double>> U() const;

    void read_from_file(FILE* file);

    void write_to_file(FILE* file) const;

    protected:
    double ranges;
};

class GasMesh: public Mesh<GasCell>
{
    public:
    GasMesh(GasCell* first_cell = nullptr, GasCell* last_cell = nullptr,
        const Chemistry::SolidGasReaction& QR = Chemistry::SolidGasReaction(),
        std::function<double(const double x)> A_func = [](double x){return 0;},
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.0005,
        const double max_length_factor = 1./20,
        const double min_length_factor = 1./10000,
        const double boundary_cell_max_length_factor = 1./1000);

    std::vector<double> c() const;

    std::vector<double> rho() const;

    std::vector<double> v() const;

    std::vector<double> T() const;
```

```
    std::vector<double> P() const;

    std::vector<double> M() const;
};

class SolidMesh: public Mesh<SolidCell>
{
    public:
    SolidMesh(SolidCell* first_cell = nullptr, SolidCell* last_cell = nullptr,
        const Chemistry::SolidGasReaction& QR = Chemistry::SolidGasReaction(),
        std::function<double(const double x)> A_func = [](double x){return 0;},
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.0005,
        const double max_length_factor = 1./20,
        const double min_length_factor = 1./2000,
        const double boundary_cell_max_length_factor = 1./1000);

    std::vector<double> T() const;
};


#endif // MESH_HPP
```

8.33 Simulation.hpp

```
#ifndef SIMULATION_HPP
#define SIMULATION_HPP

#include "Mesh/Mesh.hpp"
#include "Chemistry/Reaction.hpp"
#include "Utilities/FileArray.hpp"
#include "Math/Integration.hpp"
#include "CPGF/Plot2d/Graphic.hpp"
#include "Utilities/Progress.hpp"
#include "Solvers/Solid/SolidSolvers.hpp"
#include <functional>
#include <limits>
#include <vector>

enum class GasBoundaryConditionsType
{
    WALL,

    FREE,

    PERIODIC,

    FIXED_PRESSURE,

    FIXED_FLOW_ENTHALPY,

    FIXED_DENSITY_SPEED_PRESSURE,

    NONE,
};

class GasBoundaryConditions
{
    public:
    GasBoundaryConditionsType left;

    GasBoundaryConditionsType right;

    std::function<double(double t)> left_condition_1;

    std::function<double(double t)> left_condition_2;

    std::function<double(double t)> left_condition_3;

    std::function<double(double t)> right_condition_1;

    std::function<double(double t)> right_condition_2;

    std::function<double(double t)> right_condition_3;

    GasBoundaryConditions(const GasBoundaryConditionsType& left = GasBoundaryConditionsType::WALL,
        const GasBoundaryConditionsType& right = GasBoundaryConditionsType::WALL,
        std::function<double(double t)> left_condition_1 = [](double t){return 0;},
        std::function<double(double t)> left_condition_2 = [](double t){return 0;},
        std::function<double(double t)> left_condition_3 = [](double t){return 0;},
        std::function<double(double t)> right_condition_1 = [](double t){return 0;},
        std::function<double(double t)> right_condition_2 = [](double t){return 0;},
```

```
        std::function<double(double t)> right_condition_3 = [](double t){return 0;});
};

enum class SimulationType
{
    GAS,

    SOLID,

    BOTH,
};

class Simulation
{
    public:

    GasMesh instant_gas_mesh;

    SolidMesh instant_solid_mesh;

    double instant_v_q;

    double instant_x_q;

    double instant_t;

    unsigned int refine;

    std::string name;

    SimulationType simulation_type;

    Utilities::FileArray<GasMesh>* gas_mesh;

    Utilities::FileArray<SolidMesh>* solid_mesh;

    std::vector<double> x_q_array;

    std::vector<double> v_q_array;

    std::vector<double> t_array;

    GasBoundaryConditions gas_BC;

    SolidBoundaryConditions solid_BC;

    double CFL;

    unsigned int N_saves;

    bool adaptive_refinement_solid;

    bool adaptive_refinement_gas;

    unsigned int adaptive_refinement_period;

    unsigned int N_tasks;

    std::function<double(const GasCell& C, const double t)> external_forces;

    std::function<void(const GasCell& A, const GasCell& B, Math::Vector<double>* F, double* S_max)>
   convection_solver;

    std::function<void(SolidMesh& mesh, double& dt, const double CFL, const SolidBoundaryConditions BC,
   const double t)> diffusion_solver;

    Utilities::Progress<double> progress;

    Simulation
    (
        const std::string& name,
        const Chemistry::SolidGasReaction& QR,
        const double a,
        const double b,
        const unsigned int N_cells_solid,
        const unsigned int N_cells_gas,
        const bool adaptive_refinement_solid,
        const bool adaptive_refinement_gas,
        const unsigned int N_tasks,
        const double CFL,
        const unsigned int N_saves,
        const SolidBoundaryConditions& solid_BC,
        const GasBoundaryConditions& gas_BC,
        const double x_q,
        std::function<double(double x)> v,
        std::function<double(double x)> P,
        std::function<double(double x)> T,
```

```
        std::function<double(double x)> A,
        std::function<void(const GasCell& A, const GasCell& B, Math::Vector<double>* F, double* S_max)> convection_solver,
        std::function<void(SolidMesh& mesh, double& dt, const double CFL,
            const SolidBoundaryConditions BC, const double t)> diffusion_solver,
        std::function<double(std::function<double(double x)> f, const double a, const double b)> integrator =
            [] (std::function<double(double x)> f, const double a, const double b)
                {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
        std::function<double(const GasCell& C, const double t)> external_forces = [] (const GasCell& C, const double t){return 0.;},
        const unsigned int adaptive_refinement_period = 1,
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.001,
        const double max_length_factor = 1./50,
        const double min_length_factor = 1./2000,
        const double boundary_cell_max_length_factor = 1./1000
    );

    Simulation
    (
        const std::string& name,
        const Chemistry::SolidGasReaction& QR,
        const double a,
        const double b,
        const unsigned int N_cells,
        const bool adaptive_refinement,
        const unsigned int N_tasks,
        const double CFL,
        const unsigned int N_saves,
        const GasBoundaryConditions& BC,
        std::function<double(double x)> rho,
        std::function<double(double x)> v,
        std::function<double(double x)> P,
        std::function<double(double x)> A,
        std::function<void(const GasCell& A, const GasCell& B, Math::Vector<double>* F, double* S_max)> convection_solver,
        std::function<double(std::function<double(double x)> f, const double a, const double b)> integrator =
            [] (std::function<double(double x)> f, const double a, const double b)
                {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
        std::function<double(const GasCell& C, const double t)> external_forces = [] (const GasCell& C, const double t){return 0.;},
        const unsigned int adaptive_refinement_period = 1,
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.001,
        const double max_length_factor = 1./50,
        const double min_length_factor = 1./2000,
        const double boundary_cell_max_length_factor = 1./1000
    );

    Simulation
    (
        const std::string& name,
        const Chemistry::SolidGasReaction& QR,
        const double a,
        const double b,
        const unsigned int N_cells,
        const bool adaptive_refinement,
        const unsigned int N_tasks,
        const double CFL,
        const unsigned int N_saves,
        const SolidBoundaryConditions& BC,
        std::function<double(double x)> T,
        std::function<double(double x)> A,
        std::function<void(SolidMesh& mesh, double& dt, const double CFL,
            const SolidBoundaryConditions BC, const double t)> diffusion_solver,
        std::function<double(std::function<double(double x)> f, const double a, const double b)> integrator =
            [] (std::function<double(double x)> f, const double a, const double b)
                {return Math::Integrators::Gauss_Konrad_G7_K15(f, a, b);},
        const unsigned int adaptive_refinement_period = 1,
        const double detail_subdivide_threshold = 0.01,
        const double detail_merge_threshold = 0.001,
        const double max_length_factor = 1./50,
        const double min_length_factor = 1./2000,
        const double boundary_cell_max_length_factor = 1./1000
    );

    explicit Simulation(const std::string& file);

    ~Simulation();

    void update(double dt = std::numeric_limits<double>::max());

    void simulate_until(const double t);

```

```
    void write_to_file(const std::string& file) const;

    double x_q(const double t) const;

    double v_q(const double t) const;

    std::function<double(const double x)> A(const double t) const;

    std::function<double(const double x)> rho(const double t) const;

    std::function<double(const double x)> c(const double t) const;

    std::function<double(const double x)> v(const double t) const;

    std::function<double(const double x)> T(const double t) const;

    std::function<double(const double x)> P(const double t) const;

    std::function<double(const double x)> M(const double t) const;

    CPGF::Plot2d::Graphic* mesh_plot() const;

    protected:
    double a;

    double b;

    double d;

    double rho_g;

    double v_g;
};

#endif // SIMULATION_HPP
```

8.34 ExactRiemannSolver.hpp

```
#ifndef EXACT_RIEMANN_SOLVER_HPP
#define EXACT_RIEMANN_SOLVER_HPP

namespace Solvers
{
    namespace Gas
    {
        enum class VacuumState
        {
            VACUUM_LEFT_STATE,
            VACUUM_RIGHT_STATE,
            GENERATED_VACUUM,
            NO_VACUUM,
        };

        class ExactRiemannSolver
        {
            public:
            double rho(const double x, const double t) const;

            double v(const double x, const double t) const;

            double P(const double x, const double t) const;

            double S_max() const;

            ExactRiemannSolver(const double rho_L, const double v_L, const double P_L,
                const double rho_R, const double v_R, const double P_R, const double gamma,
                const double tol);

            protected:

            double gamma;

            VacuumState vacuum_state;

            double rho_L;

            double v_L;

            double P_L;

            double rho_R;

            double v_R;
```

```
        double P_R;

        // Gamma constants.
        double G1;

        double G2;

        double G3;

        double G4;

        double G5;

        double G6;

        double G7;

        double G8;

        double A_L;

        double A_R;

        double B_L;

        double B_R;

        double c_L;

        double c_R;

        double f_L(const double P) const;

        double f_R(const double P) const;

        double f(const double P) const;

        double df_LdP(const double P) const;

        double df_RdP(const double P) const;

        double dfdP(const double P) const;

        double S_L;

        double S_HL;

        double S_TL;

        double S_starL;

        double rho_starL;

        double v_star;

        double P_star;

        double S_R;

        double S_HR;

        double S_TR;

        double S_starR;

        double rho_starR;

        double rho_Lrf(const double S) const;

        double v_Lrf(const double S) const;

        double P_Lrf(const double S) const;

        double rho_Rrf(const double S) const;

        double v_Rrf(const double S) const;

        double P_Rrf(const double S) const;

        double rho_L0(const double S) const;

        double v_L0(const double S) const;

        double P_L0(const double S) const;

        double rho_R0(const double S) const;
```

```
            double v_R0(const double S) const;

            double P_R0(const double S) const;
        };
    }
}

#endif // EXACT_RIEMANN_SOLVER_HPP
```

8.35 ExactSteadySolver.hpp

```
#ifndef EXACT_STEADY_SOLVER_HPP
#define EXACT_STEADY_SOLVER_HPP

#include "../../Math/Interpolation.hpp"
#include <functional>
#include <vector>

namespace Solvers
{
    namespace Gas
    {
        enum class SolutionType
        {
            SUBSONIC,
            SUPERSONIC,
        };

        class ExactSteadySolver
        {
            public:
            ExactSteadySolver(const double rho_0, const double P_0, const double T_0,
                const double M_x, const double A_x, const double gamma,
                const SolutionType solution_type);

            double M(const double A) const;

            double rho(const double A) const;

            double v(const double A) const;

            double P(const double A) const;

            double T(const double A) const;

            protected:
            SolutionType solution_type;

            double C;

            double rho_0;

            double P_0;

            double T_0;

            double R;

            double gamma;

            double G1;

            double G2;

            double G3;

            double G4;
        };
    }
}


#endif // EXACT_STEADY_SOLVER_HPP
```

8.36 GasSolvers.hpp

```
#ifndef GAS_SOLVERS_HPP
#define GAS_SOLVERS_HPP
```

```

#include "../../Math/Vector.hpp"
#include "../../Mesh/Cell.hpp"

namespace Solvers
{
    namespace Gas
    {
        const double EXACT_RIEMANN_SOLVER_RELATIVE_TOLERANCE = 1e-6;

        void exact(const GasCell& L, const GasCell& R, Math::Vector<double>* F, double* S_max);

        void HLL(const GasCell& L, const GasCell& R, Math::Vector<double>* F, double* S_max);

        void HLLC(const GasCell& L, const GasCell& R, Math::Vector<double>* F, double* S_max);

        void Roe(const GasCell&L, const GasCell& R, Math::Vector<double>* F, double* S_max);
    }
}

#endif // GAS_SOLVERS_HPP
```

8.37 RocketSolver.hpp

```
#ifndef ROCKET_SOLVER_HPP
#define ROCKET_SOLVER_HPP

#include <string>

namespace Solvers
{
    namespace Rocket
    {
        class SteadySolver
        {
            public:
            SteadySolver(const double T_c, const double M_catm, const double n,
                const double gamma, const double R, const double A_c);

            void solve_for_exit_area(const double A_e);

            void calculate_optimum_parameters();

            double A_c;

            double A_e;

            double M_catm;

            double gamma;

            double R;

            double n;

            double T_c;

            double v_c;

            double M_c;

            double P_c;

            double rho_c;

            double v_e;

            double M_e;

            double P_e;

            double T_e;

            double rho_e;

            double m_dot;

            double thrust;

            std::string to_string() const;

            protected:
            constexpr static double P_atm = 101325;
```

```

            double G1;

            double G2;

            double G3;

            double G4;
        };
    }
}

#endif // ROCKET_SOLVER_HPP
```

8.38 SolidSolvers.hpp

```
#ifndef SOLID_SOLVERS_HPP
#define SOLID_SOLVERS_HPP

#include <functional>
#include "../../Mesh/Mesh.hpp"

enum class SolidBoundaryConditionsType
{
    FIXED_TEMPERATURE,

    FIXED_GRADIENT,

    COMBUSTION_FRONT,
};

class SolidBoundaryConditions
{
    public:
    SolidBoundaryConditionsType left;

    SolidBoundaryConditionsType right;

    std::function<double(double T, double t)> left_condition;

    std::function<double(double T, double t)> right_condition;

    SolidBoundaryConditions(const SolidBoundaryConditionsType left = SolidBoundaryConditionsType::FIXED_TEMPERATURE,
        const SolidBoundaryConditionsType right = SolidBoundaryConditionsType::FIXED_TEMPERATURE,
        std::function<double(double T, double t)> left_condition = [](double T, double t){return 0;},
        std::function<double(double T, double t)> right_condition = [](double T, double t){return 0;});
};

namespace Solvers
{
    namespace Solid
    {
        void euler_explicit(SolidMesh& mesh, double& dt, const double CFL, const SolidBoundaryConditions BC, const double t);

         void euler_implicit(SolidMesh& mesh, double& dt, const double CFL, const SolidBoundaryConditions BC, const double t);
     }
}

#endif // SOLID_SOLVERS_HPP
```

8.39 FileArray.hpp

```
#ifndef FILE_ARRAY_HPP
#define FILE_ARRAY_HPP

#include <cstdio>
#include <initializer_list>
#include <string>
#include <vector>

namespace Utilities
{
    template <typename T>
    class FileArray
    {
        public:
```

```
        FILE* file;

        unsigned long size() const;

        class Iterator
        {
            public:

            FileArray* file_array;
            unsigned long i;

            explicit Iterator(FileArray* file_array, const unsigned long i = 0);

            bool operator==(const Iterator& J);
            bool operator!=(const Iterator& J);

            operator T() const;

            Iterator& operator=(const T& val);

            Iterator& operator++();
            Iterator operator++(int);
            Iterator& operator--();
            Iterator operator--(int);
            Iterator& operator+(const unsigned long j);
            Iterator& operator-(const unsigned long j);
        };

        T operator[](const unsigned long i) const;
        Iterator operator[](const unsigned long i);
        T at(const unsigned long i) const;
        Iterator at(const unsigned long i);
        T front() const;
        Iterator front();
        T back() const;
        Iterator back();

        Iterator begin();
        Iterator end();
        Iterator rbegin();
        Iterator rend();

        void push_back(const T& val);
        void pop_back();

        // void insert(const unsigned long i, const T& val);
        // void insert(const unsigned long i, const unsigned long n, const T& val);
        // void insert(const unsigned long i, const std::initializer_list<T>& list);
        // void erase(const unsigned long i);
        // void erase(const unsigned long start, unsigned long stop);
        void clear();

        explicit FileArray();
        explicit FileArray(char* file, const bool temp = false);
        explicit FileArray(const std::string& file, const bool temp = false);

        explicit FileArray(const std::vector<T>& vector);
        FileArray(char* file, const std::vector<T>& vector, const bool temp = false);
        FileArray(const std::string& file, const std::vector<T>& vector, const bool temp = false);

        FileArray(T* array, unsigned int N);
        FileArray(char* file, T* array, unsigned int N, const bool temp = false);
        FileArray(const std::string& file, T* array, unsigned int N, const bool temp = false);

        explicit FileArray(const std::initializer_list<T>& list);
        FileArray(char* file, const std::initializer_list<T>& list, const bool temp = false);
        FileArray(const std::string& file, const std::initializer_list<T>& list, const bool temp =
    false);

        // There is no copy constructor and assigment operator.
        FileArray(const FileArray<T>& f_array) = delete;
        FileArray& operator=(const FileArray<T>& f_array) = delete;
        ~FileArray();

        protected:

        unsigned long N;

         std::string filename;

         bool temp;

         std::vector<unsigned long> indexes;
    };
}

#endif // FILE_ARRAY_HPP
```

8.40 FileOperations.hpp

```
#ifndef FILE_OPERATIONS_HPP
#define FILE_OPERATIONS_HPP

#include <complex>
#include <vector>
#include <cstdio>
#include <string>

template <class T>
void inline read_from_file(T& object, FILE* file)
{
    object.read_from_file(file);
}

void inline read_from_file(char& val, FILE* file)
{
    fread(&val, sizeof(char), 1, file);
}

void inline read_from_file(signed char& val, FILE* file)
{
    fread(&val, sizeof(signed char), 1, file);
}

void inline read_from_file(unsigned char& val, FILE* file)
{
    fread(&val, sizeof(unsigned char), 1, file);
}

void inline read_from_file(short int& val, FILE* file)
{
    fread(&val, sizeof(short int), 1, file);
}

void inline read_from_file(unsigned short int& val, FILE* file)
{
    fread(&val, sizeof(unsigned short int), 1, file);
}

void inline read_from_file(int& val, FILE* file)
{
    fread(&val, sizeof(int), 1, file);
}

void inline read_from_file(unsigned int& val, FILE* file)
{
    fread(&val, sizeof(unsigned int), 1, file);
}

void inline read_from_file(long int& val, FILE* file)
{
    fread(&val, sizeof(long int), 1, file);
}

void inline read_from_file(unsigned long int& val, FILE* file)
{
    fread(&val, sizeof(unsigned long int), 1, file);
}

void inline read_from_file(long long int& val, FILE* file)
{
    fread(&val, sizeof(long long int), 1, file);
}

void inline read_from_file(unsigned long long int& val, FILE* file)
{
    fread(&val, sizeof(unsigned long long int), 1, file);
}

void inline read_from_file(float& val, FILE* file)
{
    fread(&val, sizeof(float), 1, file);
}

void inline read_from_file(double& val, FILE* file)
{
    fread(&val, sizeof(double), 1, file);
}

void inline read_from_file(long double& val, FILE* file)
{
    fread(&val, sizeof(long double), 1, file);
}

void inline read_from_file(wchar_t& val, FILE* file)
```

```
{
    fread(&val, sizeof(wchar_t), 1, file);
}

template <class K>
void inline read_from_file(std::complex<K>& z, FILE* file)
{
    K x, y;
    read_from_file(x, file);
    read_from_file(y, file);
    z = std::complex<K>(x, y);
}

void inline read_from_file(std::string& val, FILE* file)
{
    char buffer[1024];
    unsigned int i = 0;
    fread(buffer, sizeof(char), 1, file);
    while(buffer[i] != '\0')
    {
        i++;
        fread(buffer + i, sizeof(char), 1, file);
    }
    val = std::string(buffer);
}

template <class T>
void inline read_from_file(std::vector<T>& object, FILE* file)
{
    unsigned int size;
    read_from_file(size, file);
    object = std::vector<T>(size);
    for (unsigned int i = 0; i < size; i++)
    {
        read_from_file(object[i], file);
    }
}

void inline read_from_file(bool& val, FILE* file)
{
    char temp;
    read_from_file(temp, file);
    val = temp;
}


template <class T>
void inline write_to_file(const T& object, FILE* file)
{
    object.write_to_file(file);
}

void inline write_to_file(const signed char& val, FILE* file)
{
    fwrite(&val, sizeof(signed char), 1, file);
}

void inline write_to_file(const char& val, FILE* file)
{
    fwrite(&val, sizeof(char), 1, file);
}

void inline write_to_file(const unsigned char& val, FILE* file)
{
    fwrite(&val, sizeof(unsigned char), 1, file);
}

void inline write_to_file(const short int& val, FILE* file)
{
    fwrite(&val, sizeof(short int), 1, file);
}

void inline write_to_file(const unsigned short int& val, FILE* file)
{
    fwrite(&val, sizeof(unsigned short int), 1, file);
}

void inline write_to_file(const int& val, FILE* file)
{
    fwrite(&val, sizeof(int), 1, file);
}

void inline write_to_file(const unsigned int& val, FILE* file)
{
    fwrite(&val, sizeof(unsigned int), 1, file);
}

```

```
void inline write_to_file(const long int& val, FILE* file)
{
    fwrite(&val, sizeof(long int), 1, file);
}

void inline write_to_file(const unsigned long int& val, FILE* file)
{
    fwrite(&val, sizeof(unsigned long int), 1, file);
}

void inline write_to_file(const long long int& val, FILE* file)
{
    fwrite(&val, sizeof(long long int), 1, file);
}

void inline write_to_file(const unsigned long long int& val, FILE* file)
{
    fwrite(&val, sizeof(unsigned long long int), 1, file);
}

void inline write_to_file(const float& val, FILE* file)
{
    fwrite(&val, sizeof(float), 1, file);
}

void inline write_to_file(const double& val, FILE* file)
{
    fwrite(&val, sizeof(double), 1, file);
}

void inline write_to_file(const long double& val, FILE* file)
{
    fwrite(&val, sizeof(long double), 1, file);
}

void inline write_to_file(const wchar_t& val, FILE* file)
{
    fwrite(&val, sizeof(wchar_t), 1, file);
}

template <class K>
void inline write_to_file(const std::complex<K>& z, FILE* file)
{
    K x = z.real();
    K y = z.imag();
    fwrite(&x, sizeof(K), 1, file);
    fwrite(&y, sizeof(K), 1, file);
}

void inline write_to_file(const std::string& val, FILE* file)
{
    fwrite(val.c_str(), sizeof(char), val.size()+1, file);
}

template <class T>
void inline write_to_file(const std::vector<T>& object, FILE* file)
{
    write_to_file((unsigned int)object.size(), file);
    for (unsigned int i = 0; i < object.size(); i++)
    {
        write_to_file(object[i], file);
    }
}

void inline write_to_file(const bool& val, FILE* file)
{
    write_to_file((char) val, file);
}

#endif // FILE_OPERATIONS_HPP
```

8.41 FormatNumber.hpp

```
#ifndef FORMAT_NUMBER_HPP
#define FORMAT_NUMBER_HPP

#include <functional>
#include <string>

namespace Utilities
{
    int position_of_most_significant_digit(const double x);

```

```
    unsigned long long pow10(const int n);

    double round_to_precision(double x, const unsigned int precision);
    double ceil_to_precision(double x, const unsigned int precision);
    double floor_to_precision(double x, const unsigned int precision);

    std::string format_number(double x, const bool latex = true,
        const unsigned int precision = 3, const bool show_sign = false,
        const int lim_inf = -3, const int lim_sup = 3);
}

#endif // FORMAT_NUMBER_HPP
```

8.42 Progress.hpp

```
#ifndef PROGRESS_HPP
#define PROGRESS_HPP

#include <string>
#include <vector>
#include <atomic>
#include <chrono>
#include <map>
#include <mutex>
#include <future>

namespace Utilities
{
    std::string time_to_string(const std::chrono::system_clock::time_point date);
    std::string duration_to_string(unsigned int seconds);
    std::string duration_to_string(const double seconds);
    std::string duration_to_string(const std::chrono::system_clock::duration time);

    enum class ProgressStatus
    {
        HOLDING,
        RUNNING,
        PAUSED,
        FINISHED,
    };

    enum class ProgressEstimation
    {
        LINEAR,
        LOGARITHMIC,
    };

    class ProgressBase
    {
        protected:
        ProgressEstimation estimation;
        std::string name;
        std::chrono::system_clock::time_point t_start;
        std::chrono::system_clock::time_point t_paused;
        std::chrono::system_clock::time_point t_finish;
        std::chrono::system_clock::duration inactive_time;
        std::map<std::string, ProgressBase*> children;
        std::mutex M;
        std::future<void> task;

        public:
        ProgressStatus status;
        void start();
        void pause();
        void resume();
        void finish();
        virtual std::string report(const unsigned int level) const;

        ProgressBase(const std::string& name = "", const ProgressEstimation estimation =
    ProgressEstimation::LINEAR);
        ProgressBase(const ProgressBase& progress);
        ProgressBase& operator=(const ProgressBase& progress);

        ProgressBase& operator[] (std::string name);

        void add_child(ProgressBase& progress);
        void eliminate_child(ProgressBase& progress);

        void update_to_terminal(unsigned int period = 250);
    };


    template<class T>
```

```
    class Progress: public ProgressBase
    {
        protected:
        T initial;
        T objective;
        std::atomic<T> current;

        public:
        Progress(const std::string& name = "", ProgressEstimation estimation =
    ProgressEstimation::LINEAR,
            const T initial = T(), const T objective = T());
        Progress(Progress<T>& progress);
        Progress<T>& operator=(const Progress<T>& progress);
        std::string report(const unsigned int level) const override;

        Progress<T>& operator=(const T& val);
        Progress<T>& operator+=(const T& val);
        Progress<T>& operator++();
        Progress<T>& operator-=(const T& val);
        Progress<T>& operator--();
        Progress<T>& operator*=(const T& val);
        Progress<T>& operator/=(const T& val);
        Progress<T>& operator%=(const T& val);

        operator std::atomic<T>&();
        operator T() const;

    };
}

#endif // PROGRESS_HPP
```

8.43 ToString.hpp

```
#ifndef TO_STRING_HPP
#define TO_STRING_HPP

#include <string>

template<class T>
std::string inline read_from_file(const T& object)
{
    return object.to_string();
}

#endif // TO_STRING_HPP
```