

## Heat Conduction Equation

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Analytical Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	Analytical(Problem problem) . . . . .	6
3.1.3	Member Function Documentation . . . . .	6
3.1.3.1	compute_solution() . . . . .	6
3.2	CrankNicolson Class Reference . . . . .	7
3.2.1	Detailed Description . . . . .	8
3.2.2	Constructor & Destructor Documentation . . . . .	8
3.2.2.1	CrankNicolson(Problem problem) . . . . .	8
3.2.3	Member Function Documentation . . . . .	8
3.2.3.1	build_r(Vector previous_step) . . . . .	8
3.3	DufortFrankel Class Reference . . . . .	9
3.3.1	Detailed Description . . . . .	10
3.3.2	Constructor & Destructor Documentation . . . . .	10
3.3.2.1	DufortFrankel(Problem problem) . . . . .	10
3.3.3	Member Function Documentation . . . . .	10

3.3.3.1	<a href="#">build_iteration(Vector current_step, Vector previous_step)</a>	10
3.4	<a href="#">Explicit Class Reference</a>	11
3.4.1	<a href="#">Detailed Description</a>	12
3.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	12
3.4.2.1	<a href="#">Explicit(Problem problem)</a>	12
3.4.3	<a href="#">Member Function Documentation</a>	12
3.4.3.1	<a href="#">build_iteration(Vector current_step, Vector previous_step)=0</a>	12
3.4.3.2	<a href="#">compute_solution()</a>	12
3.5	<a href="#">FTCS Class Reference</a>	13
3.5.1	<a href="#">Detailed Description</a>	14
3.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	14
3.5.2.1	<a href="#">FTCS(Problem problem)</a>	14
3.5.3	<a href="#">Member Function Documentation</a>	14
3.5.3.1	<a href="#">build_iteration(Vector current_step, Vector previous_step)</a>	14
3.6	<a href="#">Implicit Class Reference</a>	14
3.6.1	<a href="#">Detailed Description</a>	16
3.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	16
3.6.2.1	<a href="#">Implicit(Problem problem)</a>	16
3.6.3	<a href="#">Member Function Documentation</a>	16
3.6.3.1	<a href="#">build_r(Vector previous_step)=0</a>	16
3.6.3.2	<a href="#">compute_solution()</a>	16
3.7	<a href="#">IOManager Class Reference</a>	17
3.7.1	<a href="#">Detailed Description</a>	17
3.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	17
3.7.2.1	<a href="#">IOManager()</a>	17
3.7.3	<a href="#">Member Function Documentation</a>	17
3.7.3.1	<a href="#">export_outputs(Method *analytical, std::vector&lt; Method * &gt; methods)</a>	17
3.8	<a href="#">Laasonen Class Reference</a>	17
3.8.1	<a href="#">Detailed Description</a>	19
3.8.2	<a href="#">Constructor &amp; Destructor Documentation</a>	19

3.8.2.1	Laasonen(Problem problem)	19
3.8.3	Member Function Documentation	19
3.8.3.1	build_r(Vector previous_step)	19
3.9	Matrix Class Reference	19
3.9.1	Detailed Description	21
3.9.2	Constructor & Destructor Documentation	21
3.9.2.1	Matrix()	21
3.9.2.2	Matrix(int Nrows, int Ncols)	21
3.9.2.3	Matrix(const Matrix &m)	22
3.9.3	Member Function Documentation	22
3.9.3.1	getNcols() const	22
3.9.3.2	getNrows() const	22
3.9.3.3	one_norm() const	22
3.9.3.4	operator*(const Matrix &a) const	23
3.9.3.5	operator*(const Vector &v) const	23
3.9.3.6	operator=(const Matrix &m)	24
3.9.3.7	operator==(const Matrix &m) const	24
3.9.3.8	set_row(int index, Vector v)	24
3.9.3.9	transpose() const	25
3.9.3.10	two_norm() const	25
3.9.3.11	uniform_norm() const	25
3.9.4	Friends And Related Function Documentation	25
3.9.4.1	operator<<	25
3.9.4.2	operator<<	26
3.9.4.3	operator>>	26
3.9.4.4	operator>>	27
3.10	Method Class Reference	27
3.10.1	Detailed Description	28
3.10.2	Constructor & Destructor Documentation	29
3.10.2.1	Method()	29

3.10.2.2	Method(Problem problem)	29
3.10.3	Member Function Documentation	29
3.10.3.1	compute()	29
3.10.3.2	compute_solution() $\Rightarrow$ 0	29
3.10.3.3	get_computational_time()	29
3.10.3.4	get_deltat()	29
3.10.3.5	get_name()	30
3.10.3.6	get_solution()	30
3.10.3.7	get_two_norm()	30
3.10.3.8	get_xvalues()	30
3.10.4	Member Data Documentation	30
3.10.4.1	name	30
3.10.4.2	problem	30
3.10.4.3	q	31
3.11	Problem Class Reference	31
3.11.1	Detailed Description	31
3.11.2	Constructor & Destructor Documentation	31
3.11.2.1	Problem()	31
3.11.2.2	Problem(double dt, double dx)	31
3.11.3	Member Function Documentation	32
3.11.3.1	get_deltat()	32
3.11.3.2	get_deltax()	32
3.11.3.3	get_first_row()	32
3.11.3.4	get_solution()	32
3.11.3.5	get_tsize()	33
3.11.3.6	get_tvalues()	33
3.11.3.7	get_xsize()	33
3.11.3.8	get_xvalues()	33
3.11.3.9	set_initial_conditions()	33
3.11.3.10	set_time_step(Vector step, double time)	33

3.12 Richardson Class Reference . . . . .	34
3.12.1 Detailed Description . . . . .	35
3.12.2 Constructor & Destructor Documentation . . . . .	35
3.12.2.1 Richardson(Problem problem) . . . . .	35
3.12.3 Member Function Documentation . . . . .	35
3.12.3.1 build_iteration(Vector current_step, Vector previous_step) . . . . .	35
3.13 Vector Class Reference . . . . .	36
3.13.1 Detailed Description . . . . .	37
3.13.2 Constructor & Destructor Documentation . . . . .	37
3.13.2.1 Vector() . . . . .	37
3.13.2.2 Vector(int Num) . . . . .	37
3.13.2.3 Vector(const Vector &v) . . . . .	38
3.13.2.4 Vector(std::vector< double > vec) . . . . .	38
3.13.3 Member Function Documentation . . . . .	38
3.13.3.1 find(double value) . . . . .	38
3.13.3.2 getSize() const . . . . .	38
3.13.3.3 one_norm() const . . . . .	38
3.13.3.4 operator=(const Vector &v) . . . . .	39
3.13.3.5 operator==(const Vector &v) const . . . . .	39
3.13.3.6 push(double value) . . . . .	40
3.13.3.7 push_front_back(double value) . . . . .	40
3.13.3.8 two_norm() const . . . . .	40
3.13.3.9 uniform_norm() const . . . . .	40
3.13.4 Friends And Related Function Documentation . . . . .	41
3.13.4.1 operator<< . . . . .	41
3.13.4.2 operator<< . . . . .	41
3.13.4.3 operator>> . . . . .	42
3.13.4.4 operator>> . . . . .	42





# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

IOManager . . . . .	17
Method . . . . .	27
Analytical . . . . .	5
Explicit . . . . .	11
DufortFrankel . . . . .	9
FTCS . . . . .	13
Richardson . . . . .	34
Implicit . . . . .	14
CrankNicolson . . . . .	7
Laasonen . . . . .	17
Problem . . . . .	31
vector	
Matrix . . . . .	19
Vector . . . . .	36



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Analytical</a>	5
<a href="#">CrankNicolson</a>	7
<a href="#">DufortFrankel</a>	9
<a href="#">Explicit</a>	11
<a href="#">FTCS</a>	13
<a href="#">Implicit</a>	14
<a href="#">IOManager</a>	17
<a href="#">Laasonen</a>	17
<a href="#">Matrix</a>	19
<a href="#">Method</a>	27
<a href="#">Problem</a>	31
<a href="#">Richardson</a>	34
<a href="#">Vector</a>	36



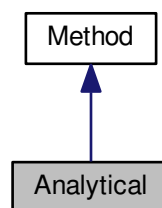
## Chapter 3

# Class Documentation

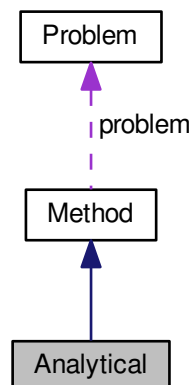
### 3.1 Analytical Class Reference

```
#include <analytical.h>
```

Inheritance diagram for Analytical:



Collaboration diagram for Analytical:



## Public Member Functions

- [Analytical](#) ([Problem](#) *problem*)
- void [compute\\_solution](#) ()

## Additional Inherited Members

### 3.1.1 Detailed Description

An [Analytical](#) class to compute the solution with standard procedures  
The implementation is derived from the [Method](#) Object

The [Analytical](#) class provides:

- a basic constructor for an object,
- a method to compute a solution with the correct procedures

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 [Analytical::Analytical](#) ( [Problem](#) *problem* )

Default constructor. Intialize a [Analytical](#) object

### 3.1.3 Member Function Documentation

#### 3.1.3.1 void [Analytical::compute\\_solution](#) ( ) [[virtual](#)]

Normal public method. compute the solution with specific given rules

Implements [Method](#).

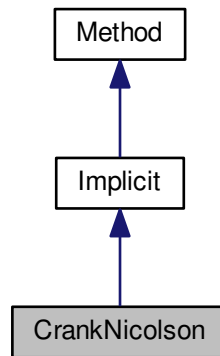
The documentation for this class was generated from the following files:

- methods/analytical.h
- methods/analytical.cpp

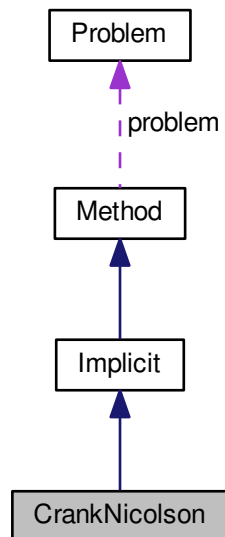
## 3.2 CrankNicolson Class Reference

```
#include <crank_nicolson.h>
```

Inheritance diagram for CrankNicolson:



Collaboration diagram for CrankNicolson:



### Public Member Functions

- [CrankNicolson](#) ([Problem](#) problem)

## Protected Member Functions

- [Vector build\\_r](#) ([Vector](#) previous\_step)

## Additional Inherited Members

### 3.2.1 Detailed Description

A [CrankNicolson](#) method class that contains a r vector builder.  
This builder is used to calculate the r vector in  $A.x = r$  linear equation system.

The [CrankNicolson](#) class provides:

- a basic constructor for creating a [CrankNicolson](#) method object.
- a method to compute the r vector.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 CrankNicolson::CrankNicolson ( [Problem](#) problem )

Default constructor.

### 3.2.3 Member Function Documentation

#### 3.2.3.1 [Vector](#) CrankNicolson::build\_r ( [Vector](#) previous\_step ) [protected], [virtual]

Normal protected method. get the number of rows

##### Parameters

previous_step	<a href="#">Vector</a> representing the solution of the previous time step.
---------------	---

##### Returns

[Vector](#). r vector to be used in  $A.x = r$

Implements [Implicit](#).

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

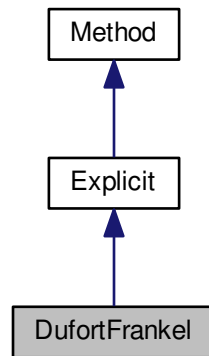
- methods/implicit/krank\_nicolson.h
- methods/implicit/krank\_nicolson.cpp



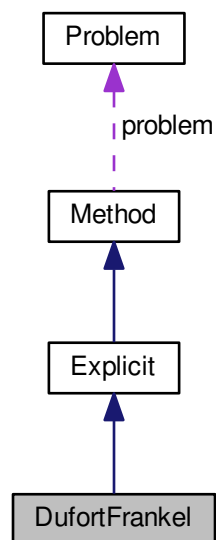
### 3.3 DufortFrankel Class Reference

```
#include <dufort_frankel.h>
```

Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



#### Public Member Functions

- [DufortFrankel](#) ([Problem problem](#))

## Protected Member Functions

- [Vector build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)

## Additional Inherited Members

### 3.3.1 Detailed Description

A [DufortFrankel](#) method class that contains an iteration builder.  
This builder is used to calculate a solution using the Dufort-Frankel method.

The [DufortFrankel](#) class provides:

- a basic constructor for creating a [DufortFrankel](#) method object.
- a method to compute a solution of the current iteration

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 DufortFrankel::DufortFrankel ( *Problem problem* )

Default constructor.

### 3.3.3 Member Function Documentation

#### 3.3.3.1 **Vector** DufortFrankel::build\_iteration ( *Vector current\_step*, *Vector previous\_step* ) [protected], [virtual]

Normal protected method. Calculate a next time step solution requiring a previous time step and a current time step solution.

#### Parameters

<i>current_step</i>	A vector representing the current time step solution.
<i>previous_step</i>	A vector representing the previous time step solution.

#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

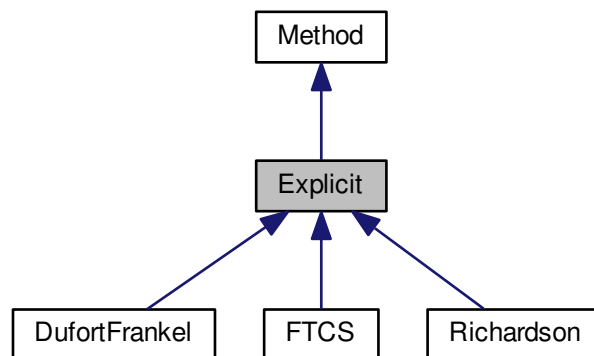
The documentation for this class was generated from the following files:

- methods/explicit/dufort\_frankel.h
- methods/explicit/dufort\_frankel.cpp

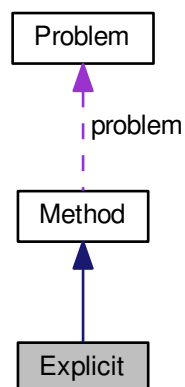
## 3.4 Explicit Class Reference

```
#include <explicit.h>
```

Inheritance diagram for Explicit:



Collaboration diagram for Explicit:



### Public Member Functions

- [Explicit](#) ([Problem](#) problem)
- void [compute\\_solution](#) ()

## Protected Member Functions

- virtual [Vector](#) [build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)=0

## Additional Inherited Members

### 3.4.1 Detailed Description

An explicit method class that contains default methods that only explicit methods use  
The implementation is derived from the [Method](#) class

The [Explicit](#) class provides:

- a basic constructor for creating an explicit method object.
- a method to compute a solution following explicit methods rules

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 `Explicit::Explicit ( Problem problem )`

Default constructor.

### 3.4.3 Member Function Documentation

#### 3.4.3.1 `virtual Vector Explicit::build_iteration ( Vector current_step, Vector previous_step )` `[protected]`, `[pure virtual]`

A pure virtual member. Build the solution of the next time step, using the previous time step and the next time step solutions

#### Parameters

<i>previous_step</i>	A vector containing the previous time step solution.
<i>current_step</i>	A vector containing the current time step solution.

#### Returns

[Vector](#). A vector representing the next time step solution.

Implemented in [FTCS](#), [DufortFrankel](#), and [Richardson](#).

#### 3.4.3.2 `void Explicit::compute_solution ( )` `[virtual]`

Normal public method. Calculates a solution for the given problem by populating the solution grid with the correct values.

Implements [Method](#).

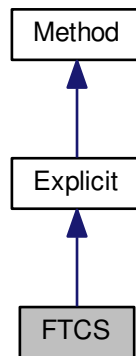
The documentation for this class was generated from the following files:

- methods/explicit/explicit.h
- methods/explicit/explicit.cpp

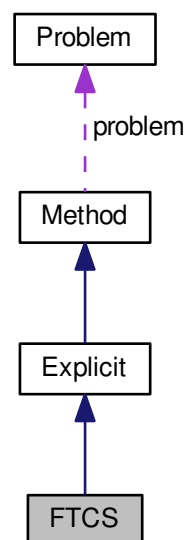
## 3.5 FTCS Class Reference

```
#include <forward_t_central_s.h>
```

Inheritance diagram for FTCS:



Collaboration diagram for FTCS:



### Public Member Functions

- [FTCS](#) ([Problem](#) problem)
- [Vector](#) [build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)

## Additional Inherited Members

### 3.5.1 Detailed Description

A [FTCS](#) method class that contains an iteration builder.

This builder is used to calculate the first iteration of explicit methods, since it only requires the previous step solution to do it.

The [FTCS](#) class provides:

- a basic constructor for creating a [FTCS](#) method object.
- a method to compute the current iteration

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 FTCS::FTCS ( *Problem problem* )

Default constructor.

### 3.5.3 Member Function Documentation

#### 3.5.3.1 **Vector** FTCS::build\_iteration ( *Vector current\_step*, *Vector previous\_step* ) `[virtual]`

Normal public method. Calculate a solution requiring only the previous time step solution.

##### Parameters

<i>current_step</i>	A vector with size 0, it's not required in this method.
<i>previous_step</i>	A vector representing the previous time step solution

##### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

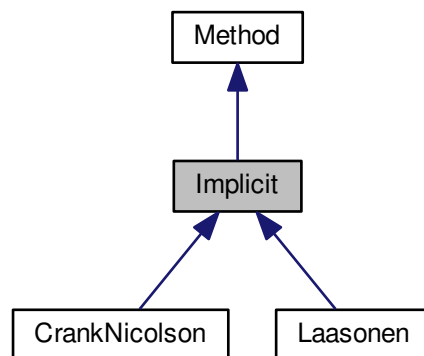
The documentation for this class was generated from the following files:

- methods/explicit/forward\_t\_central\_s.h
- methods/explicit/forward\_t\_central\_s.cpp

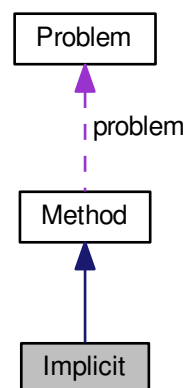
## 3.6 Implicit Class Reference

```
#include <implicit.h>
```

Inheritance diagram for Implicit:



Collaboration diagram for Implicit:



### Public Member Functions

- [Implicit](#) ([Problem](#) problem)
- void [compute\\_solution](#) ()

### Protected Member Functions

- virtual [Vector](#) [build\\_r](#) ([Vector](#) previous\_step)=0

## Additional Inherited Members

### 3.6.1 Detailed Description

An implicit method class that contains default methods that only implicit methods use  
The implementation is derived from the [Method](#) class

The [Implicit](#) class provides:

- a basic constructor for creating an implicit method object.
- a method to compute a solution following implicit methods rules

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Implicit::Implicit ( [Problem](#) *problem* )

Default constructor.

### 3.6.3 Member Function Documentation

#### 3.6.3.1 virtual [Vector](#) Implicit::build\_r ( [Vector](#) *previous\_step* ) [protected],[pure virtual]

A pure virtual member. Build the r vector in a linear system of  $A \cdot x = r$  in which A is a matrix, whereas b and r are vectors.

This method is used to compute a solution using the thomas algorithm, which can be used in a triadiagonal matrix.

##### Parameters

<i>previous_step</i>	A vector containing the previous time step solution.
----------------------	--

##### Returns

[Vector](#). The r vector, which can be used in to calculate the current time step solution with Tomas Algorithm.

Implemented in [CrankNicolson](#), and [Laasonen](#).

#### 3.6.3.2 void Implicit::compute\_solution ( ) [virtual]

Normal public method. Calculates a solution for the given problem by populating the solution grid with the correct values.

Implements [Method](#).

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

- methods/implicit/implicit.h
- methods/implicit/implicit.cpp



## 3.7 IOManager Class Reference

```
#include <iomanager.h>
```

### Public Member Functions

- [IOManager](#) ()
- void [export\\_outputs](#) ([Method](#) \*analytical, std::vector< [Method](#) \* > methods)

### 3.7.1 Detailed Description

An input/output manager class to handle plot exportations and future implementations of input handling

The [IOManager](#) class provides:

-plot method which compares the analytical solution with a set of given methods, plotting them with a custom configuration using gnuplot

### 3.7.2 Constructor & Destructor Documentation

#### 3.7.2.1 IOManager::IOManager ( )

Default constructor. Initialize an [IOManager](#) object.

### 3.7.3 Member Function Documentation

#### 3.7.3.1 void IOManager::export\_outputs ( [Method](#) \* *analytical*, std::vector< [Method](#) \* > *methods* )

Exports outputs regarding plots images and error tables for each computed solution, comparing them to the analytical solution

#### Parameters

<i>Method*</i>	analytical The analytical solution
<i>vector&lt;Method*&gt;</i>	methods <a href="#">Vector</a> containing the solutions

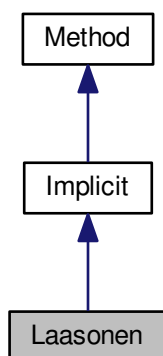
The documentation for this class was generated from the following files:

- io/iomanager.h
- io/iomanager.cpp

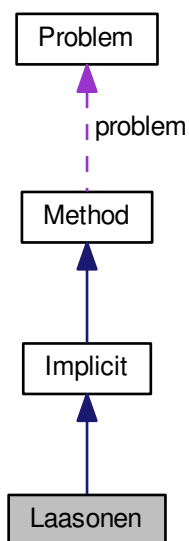
## 3.8 Laasonen Class Reference

```
#include <laasonen.h>
```

Inheritance diagram for Laasonen:



Collaboration diagram for Laasonen:



### Public Member Functions

- [Laasonen](#) ([Problem](#) problem)

### Protected Member Functions

- [Vector build\\_r](#) ([Vector](#) previous\_step)

## Additional Inherited Members

### 3.8.1 Detailed Description

A [Laasonen](#) method class that contains a r vector builder.

This builder is used is used to calculate the r vector in  $A.x = r$  linear equation system.

The [Laasonen](#) class provides:

- a basic constructor for creating a [Laasonen](#) method object.
- a method to compute the r vector.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 [Laasonen::Laasonen](#) ( *Problem problem* )

Default constructor.

### 3.8.3 Member Function Documentation

#### 3.8.3.1 [Vector](#) [Laasonen::build\\_r](#) ( [Vector](#) *previous\_step* ) [protected], [virtual]

Normal protected method. get the number of rows

##### Parameters

<i>previous_step</i>	<a href="#">Vector</a> representing the solution of the previous time step.
----------------------	---

##### Returns

[Vector](#). r vector to be used in  $A.x = r$

Implements [Implicit](#).

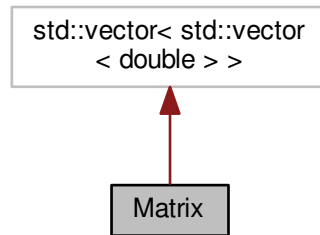
The documentation for this class was generated from the following files:

- methods/implicit/laasonen.h
- methods/implicit/laasonen.cpp

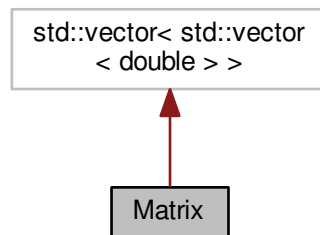
## 3.9 Matrix Class Reference

```
#include <matrix.h>
```

Inheritance diagram for Matrix:



Collaboration diagram for Matrix:



## Public Member Functions

- [Matrix](#) ()
- [Matrix](#) (int Nrows, int Ncols)
- [Matrix](#) (const [Matrix](#) &m)
- int [getNrows](#) () const
- int [getNcols](#) () const
- void [set\\_row](#) (int index, [Vector](#) v)
- [Matrix](#) & [operator=](#) (const [Matrix](#) &m)
- bool [operator==](#) (const [Matrix](#) &m) const
- double [one\\_norm](#) () const
- double [two\\_norm](#) () const
- double [uniform\\_norm](#) () const
- [Matrix](#) [operator\\*](#) (const [Matrix](#) &a) const
- [Vector](#) [operator\\*](#) (const [Vector](#) &v) const
- [Matrix](#) [transpose](#) () const
- [Matrix](#) [mult](#) (const [Matrix](#) &a) const

## Friends

- `std::istream & operator>> (std::istream &is, Matrix &m)`
- `std::ostream & operator<< (std::ostream &os, const Matrix &m)`
- `std::ifstream & operator>> (std::ifstream &ifs, Matrix &m)`
- `std::ofstream & operator<< (std::ofstream &ofs, const Matrix &m)`

### 3.9.1 Detailed Description

A matrix class for data storage of a 2D array of doubles

The implementation is derived from the standard container vector `std::vector`

We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`

The [Matrix](#) class provides:

- basic constructors for creating a matrix object from other matrix object, by creating empty matrix of a given size,
- input and output operation via `>>` and `<<` operators using keyboard or file
- basic operations like access via `[]` operator, assignment and comparison

### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 `Matrix::Matrix ( )`

Default constructor. Initialize an empty [Matrix](#) object

See also

[Matrix\(int Nrows, int Ncols\)](#)  
[Matrix\(const Matrix& m\)](#)

#### 3.9.2.2 `Matrix::Matrix ( int Nrows, int Ncols )`

Alternate constructor. build a matrix Nrows by Ncols

See also

[Matrix\(\)](#)  
[Matrix\(const Matrix& m\)](#)

#### Exceptions

<i>invalid_argument</i>	("matrix size negative or zero")
-------------------------	----------------------------------

#### Parameters

<i>Nrows</i>	int. number of rows in matrix
<i>Ncols</i>	int. number of columns in matrix

### 3.9.2.3 Matrix::Matrix ( const Matrix & *m* )

Copy constructor. build a matrix from another matrix

See also

[Matrix\(\)](#)  
[Matrix\(int Nrows, int Ncols\)](#)

Parameters

<i>m</i>	<a href="#">Matrix</a> &. matrix to copy from
----------	---

## 3.9.3 Member Function Documentation

### 3.9.3.1 int Matrix::getNcols ( ) const

Normal public get method. get the number of columns

See also

int [getNrows\(\)](#)const

Returns

int. number of columns in matrix

### 3.9.3.2 int Matrix::getNrows ( ) const

Normal public get method. get the number of rows

See also

int [getNcols\(\)](#)const

Returns

int. number of rows in matrix

### 3.9.3.3 double Matrix::one\_norm ( ) const

Normal public method that returns a double. It returns L1 norm of matrix

See also

[two\\_norm\(\)](#)const  
[uniform\\_norm\(\)](#)const

Returns

double. matrix L1 norm

### 3.9.3.4 Matrix Matrix::operator\* ( const Matrix & a ) const

Overloaded \*operator that returns a [Matrix](#). It Performs matrix by matrix multiplication.

See also

[operator\\*\(const Matrix & a\) const](#)

#### Exceptions

<i>out_of_range</i>	("Matrix access error") One or more of the matrix have a zero size
<i>std::out_of_range</i>	("uncompatible matrix sizes") Number of columns in first matrix do not match number of columns in second matrix

#### Returns

[Matrix](#). matrix-matrix product

#### Parameters

<i>a</i>	<a href="#">Matrix</a> . matrix to multiply by
----------	--

### 3.9.3.5 Vector Matrix::operator\* ( const Vector & v ) const

Overloaded \*operator that returns a [Vector](#). It Performs matrix by vector multiplication.

See also

[operator\\*\(const Matrix & a\) const](#)

#### Exceptions

<i>std::out_of_range</i>	("Matrix access error") matrix has a zero size
<i>std::out_of_range</i>	("Vector access error") vector has a zero size
<i>std::out_of_range</i>	("uncompatible matrix-vector sizes") Number of columns in matrix do not match the vector size

#### Returns

[Vector](#). matrix-vector product

#### Parameters

<i>v</i>	<a href="#">Vector</a> . <a href="#">Vector</a> to multiply by
----------	--

### 3.9.3.6 `Matrix & Matrix::operator= ( const Matrix & m )`

Overloaded assignment operator

See also

[operator==\(const Matrix& m\)const](#)

Returns

[Matrix&](#). the matrix on the left of the assignment

Parameters

<i>m</i>	<a href="#">Matrix&amp;</a> . <a href="#">Matrix</a> to assign from
----------	---

### 3.9.3.7 `bool Matrix::operator== ( const Matrix & m ) const`

Overloaded comparison operator returns true or false depending on whether the matrices are the same or not

See also

[operator=\(const Matrix& m\)](#)

Returns

bool. true or false

Parameters

<i>m</i>	<a href="#">Matrix&amp;</a> . <a href="#">Matrix</a> to compare to
----------	--

### 3.9.3.8 `void Matrix::set_row ( int index, Vector v )`

Normal public set method. replace a row with a given vector

Parameters

<i>index</i>	Index of row to mutate
<i>v</i>	New vector

Exceptions

<i>out_of_range</i>	("index out of range.\n")
<i>out_of_range</i>	("vector size is different from matrix columns number.\n")



#### 3.9.3.9 Matrix Matrix::transpose ( ) const

public method that returns the transpose of the matrix. It returns the transpose of matrix

Returns

[Matrix](#). matrix transpose

#### 3.9.3.10 double Matrix::two\_norm ( ) const

Normal public method that returns a double. It returns L2 norm of matrix

See also

[one\\_norm\(\)const](#)  
[uniform\\_norm\(\)const](#)

Returns

double. matrix L2 norm

#### 3.9.3.11 double Matrix::uniform\_norm ( ) const

Normal public method that returns a double. It returns L\_max norm of matrix

See also

[one\\_norm\(\)const](#)  
[two\\_norm\(\)const](#)

Returns

double. matrix L\_max norm

### 3.9.4 Friends And Related Function Documentation

#### 3.9.4.1 std::ostream& operator<< ( std::ostream & os, const Matrix & m ) [friend]

Overloaded ostream << operator. Display output if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

See also

[operator>>\(std::ifstream& ifs, Matrix& m\)](#)  
[operator>>\(std::istream& is, Matrix& m\)](#)  
[operator<<\(std::ostream& os, const Matrix& m\)](#)

Returns

std::ostream&. The ostream object

## Parameters

<i>os</i>	Display output stream
<i>m</i>	<a href="#">Matrix</a> to read from

3.9.4.2 `std::ofstream& operator<< ( std::ofstream & ofs, const Matrix & m )` `[friend]`

Overloaded ofstream << operator. File output the file output operator is compatible with file input operator, ie. everything written can be read later.

## See also

[operator>>\(std::ifstream& ifs, Matrix& m\)](#)  
[operator<<\(std::ofstream& ofs, const Matrix& m\)](#)  
[operator>>\(std::istream& is, Matrix& m\)](#)

## Exceptions

<code>std::invalid_argument</code>	("file read error - negative matrix size");
------------------------------------	---

## Returns

`std::ofstream&`. The ofstream object

## Parameters

<i>m</i>	<a href="#">Matrix</a> to read from
----------	-------------------------------------

3.9.4.3 `std::istream& operator>> ( std::istream & is, Matrix & m )` `[friend]`

Overloaded istream >> operator. Keyboard input if matrix has size user will be asked to input only matrix values if matrix was not initialized user can choose matrix size and input it values

## See also

[operator<<\(std::ofstream& ofs, const Matrix& m\)](#)  
[operator>>\(std::istream& is, Matrix& m\)](#)  
[operator<<\(std::ostream& os, const Matrix& m\)](#)

## Exceptions

<code>std::invalid_argument</code>	("read error - negative matrix size");
------------------------------------	--

## Returns

`std::istream&`. The istream object

## Parameters

<i>is</i>	Keyboard input stream
<i>m</i>	<a href="#">Matrix</a> to write into

3.9.4.4 `std::ifstream& operator>> ( std::ifstream & ifs, Matrix & m )` `[friend]`

Overloaded ifstream >> operator. File input the file output operator is compatible with file input operator, ie. everything written can be read later.

## See also

[operator>>\(std::ifstream& ifs, Matrix& m\)](#)  
[operator<<\(std::ofstream& ofs, const Matrix& m\)](#)  
[operator<<\(std::ostream& os, const Matrix& m\)](#)

## Returns

`std::ifstream&`. The ifstream object

## Parameters

<i>ifs</i>	Input file stream with opened matrix file
<i>m</i>	<a href="#">Matrix</a> to write into

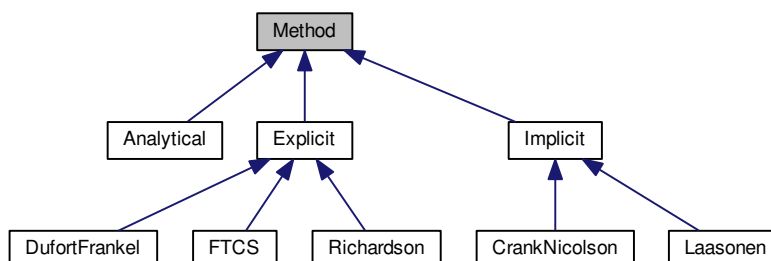
The documentation for this class was generated from the following files:

- `grid/matrix.h`
- `grid/matrix.cpp`

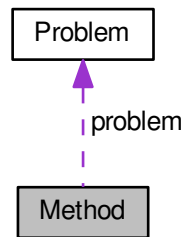
## 3.10 Method Class Reference

```
#include <method.h>
```

Inheritance diagram for Method:



Collaboration diagram for Method:



### Public Member Functions

- [Method](#) ()
- [Method](#) ([Problem](#) problem)
- `std::string` [get\\_name](#) ()
- `Matrix` [get\\_solution](#) ()
- `double` [get\\_deltat](#) ()
- `Vector` [get\\_xvalues](#) ()
- `double` [get\\_computational\\_time](#) ()
- `double` [get\\_two\\_norm](#) ()
- `void` [compute](#) ()
- `void` **compute\_norms** (`Matrix` analytical\_matrix)
- `virtual void` [compute\\_solution](#) ()=0

### Protected Attributes

- [Problem](#) problem
- `std::string` name
- `double` q

#### 3.10.1 Detailed Description

A [Method](#) class to structure information used to solve the problem

The [Method](#) class provides:

- basic constructors for creating a [Method](#) object.
- accessor methods to retrieve valuable information
- mutator methods to change the problem grid system

### 3.10.2 Constructor & Destructor Documentation

#### 3.10.2.1 `Method::Method ( )`

Default constructor. Initialize a [Method](#) object

See also

[Method\(Problem problem\)](#)

#### 3.10.2.2 `Method::Method ( Problem problem )`

Alternate constructor. Initializes a [Method](#) with a given parabolic problem.

See also

[Method\(\)](#)

### 3.10.3 Member Function Documentation

#### 3.10.3.1 `void Method::compute ( )`

Normal public method. Keeps track of the time to compute a solution

#### 3.10.3.2 `virtual void Method::compute_solution ( )` [pure virtual]

A pure virtual member. compute the solution following the rules of a given method.

Implemented in [Implicit](#), [Explicit](#), and [Analytical](#).

#### 3.10.3.3 `double Method::get_computational_time ( )`

Normal public get method. get the elapsed time value to compute a solution

Returns

double. Elapsed time throughout the computation.

#### 3.10.3.4 `double Method::get_deltat ( )`

Normal public get method. get the time step of the solution

Returns

double. Solution time step.

#### 3.10.3.5 `std::string Method::get_name ( )`

Normal public get method. get the method name

##### Returns

string. [Method](#) name.

#### 3.10.3.6 `Matrix Method::get_solution ( )`

Normal public get method. get the solution grid

##### Returns

[Matrix](#). Computed solution grid.

#### 3.10.3.7 `double Method::get_two_norm ( )`

Normal public get method. get the second norm

##### Returns

double. Second norm value.

#### 3.10.3.8 `Vector Method::get_xvalues ( )`

Normal public get method. get x values vector

##### Returns

[Vector](#). x values [Vector](#).

### 3.10.4 Member Data Documentation

#### 3.10.4.1 `std::string Method::name` [protected]

Protected string name. Name of the method.

#### 3.10.4.2 `Problem Method::problem` [protected]

Protected [Problem](#) problem. Space step of the solution.

## 3.10.4.3 double Method::q [protected]

Protected double q. A coefficient which value depends of way the equation is written, it may vary from method to method.

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

- methods/method.h
- methods/method.cpp

## 3.11 Problem Class Reference

```
#include <problem.h>
```

### Public Member Functions

- [Problem](#) ()
- [Problem](#) (double dt, double dx)
- unsigned int [get\\_xsize](#) ()
- unsigned int [get\\_tsize](#) ()
- double [get\\_deltax](#) ()
- double [get\\_deltat](#) ()
- [Vector](#) [get\\_xvalues](#) ()
- [Vector](#) [get\\_tvalues](#) ()
- [Vector](#) [get\\_first\\_row](#) ()
- [Matrix](#) \* [get\\_solution](#) ()
- void [set\\_time\\_step](#) ([Vector](#) step, double time)
- void [set\\_initial\\_conditions](#) ()

### 3.11.1 Detailed Description

A [Problem](#) class to structure relevant information related with the problem

The [Problem](#) class provides:

- basic constructors for creating a [Problem](#) object.
- acessor methods to retrieve valuable information
- mutator methods to change the solution system

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 Problem::Problem ( )

Default constructor. Initialize an empty [Problem](#) object

See also

[Problem\(double dt, double dx\)](#)

#### 3.11.2.2 Problem::Problem ( double dt, double dx )

Initialize [Problem](#) object with specific time and space steps

See also

[Problem\(\)](#)

## Parameters

$dt$	Time step to assign
$dx$	Space step to assign

## Exceptions

<i>out_of_range</i>	("space step can't be negative or zero")
<i>out_of_range</i>	("time step can't be negative or zero")

## 3.11.3 Member Function Documentation

3.11.3.1 `double Problem::get_deltat ( )`

Normal public get method that returns a double, the time step value of the solution

## Returns

double. The time step value of the solution.

3.11.3.2 `double Problem::get_deltax ( )`

Normal public get method that returns a double, the space step value of the solution

## Returns

double. The space step value of the solution.

3.11.3.3 `Vector Problem::get_first_row ( )`

Normal public get method that returns a [Vector](#), containing the initial boundaries in the first row of the solution

## Returns

[Vector](#). The initial boundaries in the first row of the solution.

3.11.3.4 `Matrix * Problem::get_solution ( )`

Normal public get method that returns a [Matrix](#), containing the solution solution.

## Returns

Matrix\*. The solution solution.



#### 3.11.3.5 unsigned int Problem::get\_tsize ( )

Normal public get method that returns an unsigned int, the number of rows of the solution

##### Returns

unsigned int. The number of rows of the solution.

#### 3.11.3.6 Vector Problem::get\_tvalues ( )

Normal public get method that returns a [Vector](#), containing the time values in each row

##### Returns

[Vector](#). The time values in each row.

#### 3.11.3.7 unsigned int Problem::get\_xsize ( )

Normal public get method that returns an unsigned int, the number of columns of the solution

##### Returns

unsigned int. The number of columns of the solution.

#### 3.11.3.8 Vector Problem::get\_xvalues ( )

Normal public get method that returns a [Vector](#), containing the space values in each column

##### Returns

[Vector](#). The space values in each column.

#### 3.11.3.9 void Problem::set\_initial\_conditions ( )

Normal public set method. set the problem initial boundaries.

#### 3.11.3.10 void Problem::set\_time\_step ( Vector *step*, double *time* )

Normal public set method. replace a row of the solution for a given [Vector](#).

##### Parameters

<i>step</i>	<a href="#">Vector</a> conatining the new values.
<i>time</i>	Corresponding row to be replaced

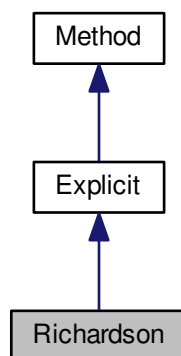
The documentation for this class was generated from the following files:

- variants/problem.h
- variants/problem.cpp

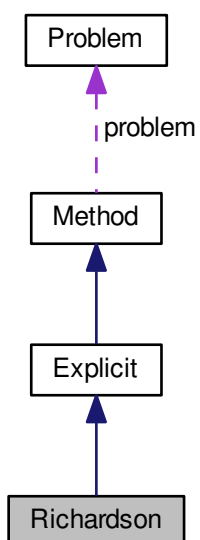
### 3.12 Richardson Class Reference

```
#include <richardson.h>
```

Inheritance diagram for Richardson:



Collaboration diagram for Richardson:



## Public Member Functions

- [Richardson](#) ([Problem problem](#))

## Protected Member Functions

- [Vector build\\_iteration](#) ([Vector current\\_step](#), [Vector previous\\_step](#))

## Additional Inherited Members

### 3.12.1 Detailed Description

A [Richardson](#) method class that contains an iteration builder.  
This builder is used to calculate a solution using the [Richardson](#) method.

The [Richardson](#) class provides:

- a basic constructor for creating a [Richardson](#) method object.
- a method to compute a solution of the current iteration

### 3.12.2 Constructor & Destructor Documentation

#### 3.12.2.1 [Richardson::Richardson](#) ( [Problem problem](#) )

Default constructor.

### 3.12.3 Member Function Documentation

#### 3.12.3.1 [Vector Richardson::build\\_iteration](#) ( [Vector current\\_step](#), [Vector previous\\_step](#) ) [protected], [virtual]

Normal protected method. Calculate a next time step solution requiring a previous time step and a current time step solution.

#### Parameters

<i>current_step</i>	A vector representing the current time step solution.
<i>previous_step</i>	A vector representing the previous time step solution.

#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

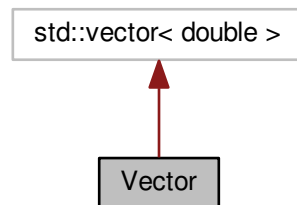
The documentation for this class was generated from the following files:

- methods/explicit/richardson.h
- methods/explicit/richardson.cpp

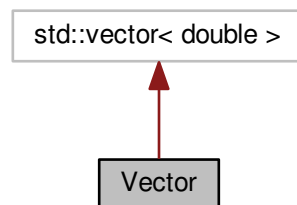
### 3.13 Vector Class Reference

```
#include <vector.h>
```

Inheritance diagram for Vector:



Collaboration diagram for Vector:



#### Public Member Functions

- [Vector](#) ()
- [Vector](#) (int Num)
- [Vector](#) (const [Vector](#) &v)
- [Vector](#) (std::vector< double > vec)
- [Vector](#) & [operator=](#) (const [Vector](#) &v)
- bool [operator==](#) (const [Vector](#) &v) const
- int [getSize](#) () const
- int [find](#) (double value)
- void [push\\_front\\_back](#) (double value)
- void [push](#) (double value)
- double [one\\_norm](#) () const
- double [two\\_norm](#) () const
- double [uniform\\_norm](#) () const

## Friends

- `std::istream & operator>> (std::istream &is, Vector &v)`
- `std::ostream & operator<< (std::ostream &os, const Vector &v)`
- `std::ifstream & operator>> (std::ifstream &ifs, Vector &v)`
- `std::ofstream & operator<< (std::ofstream &ofs, const Vector &v)`

### 3.13.1 Detailed Description

A vector class for data storage of a 1D array of doubles

The implementation is derived from the standard container vector `std::vector`

We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`

The [Vector](#) class provides:

- basic constructors for creating vector object from other vector object, or by creating empty vector of a given size,
- input and output operation via `>>` and `<<` operators using keyboard or file
- basic operations like access via `[]` operator, assignment and comparison

### 3.13.2 Constructor & Destructor Documentation

#### 3.13.2.1 `Vector::Vector ( )`

Default constructor. Initialize an empty [Vector](#) object

See also

[Vector\(int Num\)](#)  
[Vector\(const Vector& v\)](#)

#### 3.13.2.2 `Vector::Vector ( int Num ) [explicit]`

[Explicit](#) alternative constructor takes an integer. it is explicit since implicit type conversion `int -> vector` doesn't make sense Initialize [Vector](#) object of size Num

See also

[Vector\(\)](#)  
[Vector\(const Vector& v\)](#)

#### Exceptions

<i>invalid_argument</i>	("vector size negative")
-------------------------	--------------------------

#### Parameters

<i>Num</i>	int. Size of a vector
------------	-----------------------

### 3.13.2.3 `Vector::Vector ( const Vector & v )`

Copy constructor takes an [Vector](#) object reference. Initialize [Vector](#) object with another [Vector](#) object

See also

[Vector\(\)](#)  
[Vector\(int Num\)](#)

### 3.13.2.4 `Vector::Vector ( std::vector< double > vec )`

Copy constructor takes an `vector<double>` object reference. Initialize [Vector](#) object with an `vector<double>` object

See also

[Vector\(\)](#)  
[Vector\(int Num\)](#)  
[Vector\(const Vector& v\)](#)

## 3.13.3 Member Function Documentation

### 3.13.3.1 `int Vector::find ( double value )`

[Method](#) to find the value index in a vector

Parameters

<i>value</i>	Value to find
--------------	---------------

Returns

int. -1 if value was not found or the value index otherwise

### 3.13.3.2 `int Vector::getSize ( ) const`

Normal get method that returns integer, the size of the vector

Returns

int. the size of the vector

### 3.13.3.3 `double Vector::one_norm ( ) const`

Normal public method that returns a double. It returns L1 norm of vector

See also

[two\\_norm\(\)const](#)  
[uniform\\_norm\(\)const](#)

Returns

double. vectors L1 norm

#### 3.13.3.4 Vector & Vector::operator= ( const Vector & v )

Overloaded assignment operator

See also

[operator==\(const Vector& v\)const](#)

Parameters

<i>v</i>	<a href="#">Vector</a> to assign from
----------	---------------------------------------

Returns

the object on the left of the assignment

Parameters

<i>v</i>	<a href="#">Vector&amp;</a> . <a href="#">Vector</a> to assign from
----------	---

#### 3.13.3.5 bool Vector::operator== ( const Vector & v ) const

Overloaded comparison operator returns true if vectors are the same within a tolerance (1.e-07)

See also

[operator=\(const Vector& v\)](#)  
[operator\[\]\(int i\)](#)  
[operator\[\]\(int i\)const](#)

Returns

bool. true or false

Exceptions

<i>invalid_argument</i>	("incompatible vector sizes\n")
-------------------------	---------------------------------

## Parameters

<i>v</i>	<a href="#">Vector</a> &. vector to compare
----------	---

3.13.3.6 void `Vector::push ( double value )`

[Method](#) to push a value to the last position of a [Vector](#)

## Parameters

<i>value</i>	Value to be pushed
--------------	--------------------

3.13.3.7 void `Vector::push_front_back ( double value )`

[Method](#) to push a value to the first and last position of a [Vector](#)

## Parameters

<i>value</i>	Value to insert
--------------	-----------------

3.13.3.8 double `Vector::two_norm ( ) const`

Normal public method that returns a double. It returns L2 norm of vector

## See also

[one\\_norm\(\)const](#)  
[uniform\\_norm\(\)const](#)

## Returns

double. vectors L2 norm

3.13.3.9 double `Vector::uniform_norm ( ) const`

Normal public method that returns a double. It returns L\_max norm of vector

## See also

[one\\_norm\(\)const](#)  
[two\\_norm\(\)const](#)



## Exceptions

<code>out_of_range</code>	("vector access error") vector has zero size
---------------------------	--

## Returns

double. vectors Lmax norm

## 3.13.4 Friends And Related Function Documentation

3.13.4.1 `std::ostream& operator<< ( std::ostream & os, const Vector & v )` [*friend*]

Overloaded ifstream << operator. Display output.

## See also

[operator>>\(std::istream& is, Vector& v\)](#)  
[operator>>\(std::ifstream& ifs, Vector& v\)](#)  
[operator<<\(std::ofstream& ofs, const Vector& v\)](#)

## Returns

`std::ostream&`. the output stream object `os`

## Parameters

<code>os</code>	output file stream
<code>v</code>	vector to read from

3.13.4.2 `std::ofstream& operator<< ( std::ofstream & ofs, const Vector & v )` [*friend*]

Overloaded ofstream << operator. File output. the file output operator is compatible with file input operator, ie. everything written can be read later.

## See also

[operator>>\(std::istream& is, Vector& v\)](#)  
[operator>>\(std::ifstream& ifs, Vector& v\)](#)  
[operator<<\(std::ostream& os, const Vector& v\)](#)

## Returns

`std::ofstream&`. the output ofstream object `ofs`

## Parameters

<code>ofs</code>	outputfile stream. With opened file
<code>v</code>	<a href="#">Vector&amp;</a> . vector to read from

### 3.13.4.3 `std::istream& operator>> ( std::istream & is, Vector & v )` [*friend*]

Overloaded istream >> operator. Keyboard input if vector has size user will be asked to input only vector values if vector was not initialized user can choose vector size and input it values

See also

[operator>>\(std::ifstream& ifs, Vector& v\)](#)  
[operator<<\(std::ostream& os, const Vector& v\)](#)  
[operator<<\(std::ofstream& ofs, const Vector& v\)](#)

Returns

`std::istream&`. the input stream object is

Exceptions

<code>std::invalid_argument</code>	("read error - negative vector size");
------------------------------------	--

Parameters

<i>is</i>	keyboard input stream. For user input
<i>v</i>	<a href="#">Vector&amp;</a> . vector to write to

### 3.13.4.4 `std::ifstream& operator>> ( std::ifstream & ifs, Vector & v )` [*friend*]

Overloaded ifstream >> operator. File input the file output operator is compatible with file input operator, ie. everything written can be read later.

See also

[operator>>\(std::istream& is, Vector& v\)](#)  
[operator<<\(std::ostream& os, const Vector& v\)](#)  
[operator<<\(std::ofstream& ofs, const Vector& v\)](#)

Returns

`ifstream&`. the input ifstream object ifs

Exceptions

<code>std::invalid_argument</code>	("file read error - negative vector size");
------------------------------------	---

Parameters

<i>ifs</i>	input file stream. With opened matrix file
<i>v</i>	<a href="#">Vector&amp;</a> . vector to write to

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

- `grid/vector.h`
- `grid/vector.cpp`



# Index

- Analytical, [5](#)
  - Analytical, [6](#)
  - compute\_solution, [6](#)
- build\_iteration
  - DufortFrankel, [10](#)
  - Explicit, [12](#)
  - FTCS, [14](#)
  - Richardson, [35](#)
- build\_r
  - CrankNicolson, [8](#)
  - Implicit, [16](#)
  - Laasonen, [19](#)
- compute
  - Method, [29](#)
- compute\_solution
  - Analytical, [6](#)
  - Explicit, [12](#)
  - Implicit, [16](#)
  - Method, [29](#)
- CrankNicolson, [7](#)
  - build\_r, [8](#)
  - CrankNicolson, [8](#)
- DufortFrankel, [9](#)
  - build\_iteration, [10](#)
  - DufortFrankel, [10](#)
- Explicit, [11](#)
  - build\_iteration, [12](#)
  - compute\_solution, [12](#)
  - Explicit, [12](#)
- export\_outputs
  - IOManager, [17](#)
- FTCS, [13](#)
  - build\_iteration, [14](#)
  - FTCS, [14](#)
- find
  - Vector, [38](#)
- get\_computational\_time
  - Method, [29](#)
- get\_deltat
  - Method, [29](#)
  - Problem, [32](#)
- get\_deltax
  - Problem, [32](#)
- get\_first\_row
  - Problem, [32](#)
- get\_name
  - Method, [29](#)
- get\_solution
  - Method, [30](#)
  - Problem, [32](#)
- get\_tsize
  - Problem, [32](#)
- get\_tvalues
  - Problem, [33](#)
- get\_two\_norm
  - Method, [30](#)
- get\_xsize
  - Problem, [33](#)
- get\_xvalues
  - Method, [30](#)
  - Problem, [33](#)
- getNcols
  - Matrix, [22](#)
- getNrows
  - Matrix, [22](#)
- getSize
  - Vector, [38](#)
- IOManager, [17](#)
  - export\_outputs, [17](#)
  - IOManager, [17](#)
- Implicit, [14](#)
  - build\_r, [16](#)
  - compute\_solution, [16](#)
  - Implicit, [16](#)
- Laasonen, [17](#)
  - build\_r, [19](#)
  - Laasonen, [19](#)
- Matrix, [19](#)
  - getNcols, [22](#)
  - getNrows, [22](#)
  - Matrix, [21](#), [22](#)
  - one\_norm, [22](#)
  - operator<<, [25](#), [26](#)
  - operator>>, [26](#), [27](#)
  - operator\*, [22](#), [23](#)
  - operator=, [23](#)
  - operator==, [24](#)
  - set\_row, [24](#)
  - transpose, [25](#)
  - two\_norm, [25](#)
  - uniform\_norm, [25](#)
- Method, [27](#)

- compute, 29
- compute\_solution, 29
- get\_computational\_time, 29
- get\_deltat, 29
- get\_name, 29
- get\_solution, 30
- get\_two\_norm, 30
- get\_xvalues, 30
- Method, 29
- name, 30
- problem, 30
- q, 30

name

- Method, 30

one\_norm

- Matrix, 22
- Vector, 38

operator<<

- Matrix, 25, 26
- Vector, 41

operator>>

- Matrix, 26, 27
- Vector, 42

operator\*

- Matrix, 22, 23

operator=

- Matrix, 23
- Vector, 39

operator==

- Matrix, 24
- Vector, 39

Problem, 31

- get\_deltat, 32
- get\_deltax, 32
- get\_first\_row, 32
- get\_solution, 32
- get\_tsize, 32
- get\_tvalues, 33
- get\_xsize, 33
- get\_xvalues, 33
- Problem, 31
- set\_initial\_conditions, 33
- set\_time\_step, 33

problem

- Method, 30

push

- Vector, 40

push\_front\_back

- Vector, 40

q

- Method, 30

Richardson, 34

- build\_iteration, 35
- Richardson, 35

- set\_initial\_conditions
  - Problem, 33
- set\_row
  - Matrix, 24
- set\_time\_step
  - Problem, 33
- transpose
  - Matrix, 25
- two\_norm
  - Matrix, 25
  - Vector, 40
- uniform\_norm
  - Matrix, 25
  - Vector, 40
- Vector, 36
  - find, 38
  - getSize, 38
  - one\_norm, 38
  - operator<<, 41
  - operator>>, 42
  - operator=, 39
  - operator==, 39
  - push, 40
  - push\_front\_back, 40
  - two\_norm, 40
  - uniform\_norm, 40
  - Vector, 37, 38