

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>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	Analytical Class Reference . . . . .	7
4.1.1	Detailed Description . . . . .	9
4.1.2	Constructor & Destructor Documentation . . . . .	9
4.1.2.1	Analytical(Problem problem) . . . . .	9
4.1.3	Member Function Documentation . . . . .	9
4.1.3.1	compute_solution() . . . . .	9
4.1.4	Member Data Documentation . . . . .	10
4.1.4.1	nr_of_expansions . . . . .	10
4.2	CrankNicolson Class Reference . . . . .	10
4.2.1	Detailed Description . . . . .	13
4.2.2	Constructor & Destructor Documentation . . . . .	13
4.2.2.1	CrankNicolson(Problem problem) . . . . .	13
4.2.3	Member Function Documentation . . . . .	13
4.2.3.1	build_r(Vector previous_step) . . . . .	13
4.3	DufortFrankel Class Reference . . . . .	14

4.3.1	Detailed Description	17
4.3.2	Constructor & Destructor Documentation	17
4.3.2.1	DufortFrankel(Problem problem)	17
4.3.3	Member Function Documentation	17
4.3.3.1	build_iteration(Vector current_step, Vector previous_step)	17
4.4	Explicit Class Reference	18
4.4.1	Detailed Description	21
4.4.2	Constructor & Destructor Documentation	21
4.4.2.1	Explicit(Problem problem)	21
4.4.3	Member Function Documentation	21
4.4.3.1	build_iteration(Vector current_step, Vector previous_step)=0	21
4.4.3.2	compute_solution()	22
4.5	FTCS Class Reference	23
4.5.1	Detailed Description	25
4.5.2	Constructor & Destructor Documentation	25
4.5.2.1	FTCS(Problem problem)	25
4.5.3	Member Function Documentation	25
4.5.3.1	build_iteration(Vector current_step, Vector previous_step)	25
4.6	Implicit Class Reference	26
4.6.1	Detailed Description	29
4.6.2	Constructor & Destructor Documentation	29
4.6.2.1	Implicit(Problem problem)	29
4.6.3	Member Function Documentation	29
4.6.3.1	build_r(Vector previous_step)=0	29
4.6.3.2	compute_solution()	30
4.6.3.3	thomas_algorithm(Vector r, double a, double b, double c)	31
4.7	IOManager Class Reference	32
4.7.1	Detailed Description	33
4.7.2	Constructor & Destructor Documentation	33
4.7.2.1	IOManager()	33

4.7.3	Member Function Documentation . . . . .	34
4.7.3.1	create_output_dir() . . . . .	34
4.7.3.2	double_to_string(int precision, double value) . . . . .	34
4.7.3.3	error_table(std::string output_name, Method *analytical, Method *method) . . . . .	34
4.7.3.4	export_outputs(Method *analytical, std::vector< Method * > methods) . . . . .	35
4.7.3.5	plot_laasonen_times() . . . . .	36
4.7.3.6	plot_solutions(std::string output_name, Method *analytical, Method *method) . . . . .	36
4.7.4	Member Data Documentation . . . . .	37
4.7.4.1	default_deltat_times . . . . .	37
4.7.4.2	laasonen_times . . . . .	37
4.7.4.3	output_path . . . . .	37
4.8	Laasonen Class Reference . . . . .	38
4.8.1	Detailed Description . . . . .	40
4.8.2	Constructor & Destructor Documentation . . . . .	40
4.8.2.1	Laasonen(Problem problem) . . . . .	40
4.8.3	Member Function Documentation . . . . .	40
4.8.3.1	build_r(Vector previous_step) . . . . .	40
4.9	Matrix Class Reference . . . . .	41
4.9.1	Detailed Description . . . . .	44
4.9.2	Member Typedef Documentation . . . . .	45
4.9.2.1	vec . . . . .	45
4.9.3	Constructor & Destructor Documentation . . . . .	45
4.9.3.1	Matrix() . . . . .	45
4.9.3.2	Matrix(int Nrows, int Ncols) . . . . .	45
4.9.3.3	Matrix(const Matrix &m) . . . . .	46
4.9.4	Member Function Documentation . . . . .	46
4.9.4.1	getNcols() const . . . . .	46
4.9.4.2	getNrows() const . . . . .	47
4.9.4.3	mult(const Matrix &a) const . . . . .	48
4.9.4.4	one_norm() const . . . . .	48

4.9.4.5	<code>operator*(const Matrix &amp;a) const</code>	49
4.9.4.6	<code>operator*(const Vector &amp;v) const</code>	50
4.9.4.7	<code>operator=(const Matrix &amp;m)</code>	51
4.9.4.8	<code>operator==(const Matrix &amp;m) const</code>	51
4.9.4.9	<code>set_row(int index, Vector v)</code>	52
4.9.4.10	<code>transpose() const</code>	53
4.9.4.11	<code>two_norm() const</code>	54
4.9.4.12	<code>uniform_norm() const</code>	54
4.9.5	Friends And Related Function Documentation	55
4.9.5.1	<code>operator&lt;&lt;</code>	55
4.9.5.2	<code>operator&lt;&lt;</code>	56
4.9.5.3	<code>operator&gt;&gt;</code>	56
4.9.5.4	<code>operator&gt;&gt;</code>	57
4.10	Method Class Reference	57
4.10.1	Detailed Description	60
4.10.2	Constructor & Destructor Documentation	60
4.10.2.1	<code>Method()</code>	60
4.10.2.2	<code>Method(Problem problem)</code>	61
4.10.3	Member Function Documentation	61
4.10.3.1	<code>compute()</code>	61
4.10.3.2	<code>compute_solution()=0</code>	61
4.10.3.3	<code>get_computational_time()</code>	62
4.10.3.4	<code>get_deltat()</code>	62
4.10.3.5	<code>get_name()</code>	62
4.10.3.6	<code>get_solution()</code>	63
4.10.3.7	<code>get_xvalues()</code>	63
4.10.4	Member Data Documentation	64
4.10.4.1	<code>computational_time</code>	64
4.10.4.2	<code>name</code>	64
4.10.4.3	<code>problem</code>	64

4.10.4.4	q	64
4.11	Problem Class Reference	65
4.11.1	Detailed Description	67
4.11.2	Constructor & Destructor Documentation	67
4.11.2.1	Problem()	67
4.11.2.2	Problem(double dt, double dx)	67
4.11.3	Member Function Documentation	68
4.11.3.1	get_deltat()	68
4.11.3.2	get_deltax()	69
4.11.3.3	get_first_row()	69
4.11.3.4	get_solution()	70
4.11.3.5	get_tsize()	70
4.11.3.6	get_tvalues()	70
4.11.3.7	get_xsize()	71
4.11.3.8	get_xvalues()	71
4.11.3.9	set_initial_conditions()	72
4.11.3.10	set_t_values()	72
4.11.3.11	set_time_step(Vector step, double time)	73
4.11.3.12	set_x_values()	73
4.11.4	Member Data Documentation	74
4.11.4.1	delta_t	74
4.11.4.2	delta_x	74
4.11.4.3	solution	74
4.11.4.4	t_size	74
4.11.4.5	t_values	74
4.11.4.6	x_size	75
4.11.4.7	x_values	75
4.12	Richardson Class Reference	75
4.12.1	Detailed Description	78
4.12.2	Constructor & Destructor Documentation	78

4.12.2.1 Richardson(Problem problem)	78
4.12.3 Member Function Documentation	78
4.12.3.1 build_iteration(Vector current_step, Vector previous_step)	78
4.13 Vector Class Reference	79
4.13.1 Detailed Description	82
4.13.2 Member Typedef Documentation	82
4.13.2.1 vec	82
4.13.3 Constructor & Destructor Documentation	82
4.13.3.1 Vector()	82
4.13.3.2 Vector(int Num)	83
4.13.3.3 Vector(const Vector &v)	83
4.13.3.4 Vector(std::vector< double > vec)	84
4.13.4 Member Function Documentation	84
4.13.4.1 find(double value)	84
4.13.4.2 getSize() const	84
4.13.4.3 one_norm() const	85
4.13.4.4 operator=(const Vector &v)	86
4.13.4.5 operator==(const Vector &v) const	86
4.13.4.6 push(double value)	86
4.13.4.7 push_front_back(double value)	87
4.13.4.8 two_norm() const	87
4.13.4.9 uniform_norm() const	87
4.13.5 Friends And Related Function Documentation	88
4.13.5.1 operator<<	88
4.13.5.2 operator<<	88
4.13.5.3 operator>>	89
4.13.5.4 operator>>	89



<b>5</b>	<b>File Documentation</b>	<b>91</b>
5.1	grid/matrix.cpp File Reference	91
5.1.1	Function Documentation	91
5.1.1.1	operator<<(std::ostream &os, const Matrix &m)	91
5.1.1.2	operator<<(std::ofstream &ofs, const Matrix &m)	92
5.1.1.3	operator>>(std::istream &is, Matrix &m)	93
5.1.1.4	operator>>(std::ifstream &ifis, Matrix &m)	94
5.2	grid/matrix.h File Reference	95
5.3	grid/vector.cpp File Reference	95
5.3.1	Function Documentation	96
5.3.1.1	operator<<(std::ostream &os, const Vector &v)	96
5.3.1.2	operator<<(std::ofstream &ofs, const Vector &v)	96
5.3.1.3	operator>>(std::istream &is, Vector &v)	97
5.3.1.4	operator>>(std::ifstream &ifis, Vector &v)	98
5.4	grid/vector.h File Reference	98
5.5	io/iomanager.cpp File Reference	99
5.6	io/iomanager.h File Reference	100
5.7	main.cpp File Reference	101
5.7.1	Function Documentation	102
5.7.1.1	main()	102
5.8	methods/analytical.cpp File Reference	102
5.9	methods/analytical.h File Reference	103
5.10	methods/explicit/dufort_frankel.cpp File Reference	104
5.11	methods/explicit/dufort_frankel.h File Reference	104
5.12	methods/explicit/explicit.cpp File Reference	105
5.13	methods/explicit/explicit.h File Reference	106
5.14	methods/explicit/forward_t_central_s.cpp File Reference	107
5.15	methods/explicit/forward_t_central_s.h File Reference	108
5.16	methods/explicit/richardson.cpp File Reference	109
5.17	methods/explicit/richardson.h File Reference	109

5.18	methods/implicit/crank_nicolson.cpp File Reference	110
5.19	methods/implicit/crank_nicolson.h File Reference	111
5.20	methods/implicit/implicit.cpp File Reference	112
5.21	methods/implicit/implicit.h File Reference	113
5.22	methods/implicit/laasonen.cpp File Reference	113
5.23	methods/implicit/laasonen.h File Reference	114
5.24	methods/method.cpp File Reference	115
5.25	methods/method.h File Reference	116
5.26	variants/problem.cpp File Reference	116
5.27	variants/problem.h File Reference	117
5.28	variants/utls.h File Reference	118
5.28.1	Variable Documentation	119
5.28.1.1	CRANK_NICHOLSON	119
5.28.1.2	DELTA_T	119
5.28.1.3	DELTA_T_LASSONEN	119
5.28.1.4	DELTA_X	119
5.28.1.5	DIFUSIVITY	120
5.28.1.6	DUFORT_FRANKEL	120
5.28.1.7	FORWARD_TIME_CENTRAL_SPACE	120
5.28.1.8	INITIAL_TEMPERATURE	120
5.28.1.9	LAASONEN	120
5.28.1.10	NUMBER_OF_EXPANSIONS	120
5.28.1.11	NUMBER_TIME_STEPS	120
5.28.1.12	OUTPUT_PATH	120
5.28.1.13	PI	121
5.28.1.14	RICHARDSON	121
5.28.1.15	SURFACE_TEMPERATURE	121
5.28.1.16	THICKNESS	121
5.28.1.17	TIMELIMIT	121

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

IOManager . . . . .	32
Method . . . . .	57
Analytical . . . . .	7
Explicit . . . . .	18
DufortFrankel . . . . .	14
FTCS . . . . .	23
Richardson . . . . .	75
Implicit . . . . .	26
CrankNicolson . . . . .	10
Laasonen . . . . .	38
Problem . . . . .	65
vector	
Matrix . . . . .	41
Vector . . . . .	79



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Analytical</a>	An <a href="#">Analytical</a> class to compute the solution with standard procedures The implementation is derived from the <a href="#">Method</a> Object . . . . .	7
<a href="#">CrankNicolson</a>	A <a href="#">CrankNicolson</a> method class that contains a r vector builder . . . . .	10
<a href="#">DufortFrankel</a>	A <a href="#">DufortFrankel</a> method class that contains an iteration builder . . . . .	14
<a href="#">Explicit</a>	An explicit method class that contains default methods that only explicit methods use The implementation is derived from the <a href="#">Method</a> class . . . . .	18
<a href="#">FTCS</a>	A <a href="#">FTCS</a> method class that contains an iteration builder . . . . .	23
<a href="#">Implicit</a>	An implicit method class that contains default methods that only implicit methods use The implementation is derived from the <a href="#">Method</a> class . . . . .	26
<a href="#">IOManager</a>	An input/output manager class to handle plot exportations and future implementations of input handling . . . . .	32
<a href="#">Laasonen</a>	A <a href="#">Laasonen</a> method class that contains a r vector builder . . . . .	38
<a href="#">Matrix</a>	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 usto expose only those base class functions we wish to use - in this the array access operator [] . . . . .	41
<a href="#">Method</a>	A <a href="#">Method</a> class to structure information used to solve the problem . . . . .	57
<a href="#">Problem</a>	A <a href="#">Problem</a> class to structure relevant information related with the problem . . . . .	65
<a href="#">Richardson</a>	A <a href="#">Richardson</a> method class that contains an iteration builder . . . . .	75
<a href="#">Vector</a>	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 usto expose only those base class functions we wish to use - in this the array access operator [] . . . . .	79



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	101
<a href="#">grid/matrix.cpp</a>	91
<a href="#">grid/matrix.h</a>	95
<a href="#">grid/vector.cpp</a>	95
<a href="#">grid/vector.h</a>	98
<a href="#">io/iomanager.cpp</a>	99
<a href="#">io/iomanager.h</a>	100
<a href="#">methods/analytical.cpp</a>	102
<a href="#">methods/analytical.h</a>	103
<a href="#">methods/method.cpp</a>	115
<a href="#">methods/method.h</a>	116
<a href="#">methods/explicit/dufort_frankel.cpp</a>	104
<a href="#">methods/explicit/dufort_frankel.h</a>	104
<a href="#">methods/explicit/explicit.cpp</a>	105
<a href="#">methods/explicit/explicit.h</a>	106
<a href="#">methods/explicit/forward_t_central_s.cpp</a>	107
<a href="#">methods/explicit/forward_t_central_s.h</a>	108
<a href="#">methods/explicit/richardson.cpp</a>	109
<a href="#">methods/explicit/richardson.h</a>	109
<a href="#">methods/implicit/crank_nicolson.cpp</a>	110
<a href="#">methods/implicit/crank_nicolson.h</a>	111
<a href="#">methods/implicit/implicit.cpp</a>	112
<a href="#">methods/implicit/implicit.h</a>	113
<a href="#">methods/implicit/laasonen.cpp</a>	113
<a href="#">methods/implicit/laasonen.h</a>	114
<a href="#">variants/problem.cpp</a>	116
<a href="#">variants/problem.h</a>	117
<a href="#">variants/utils.h</a>	118





## Chapter 4

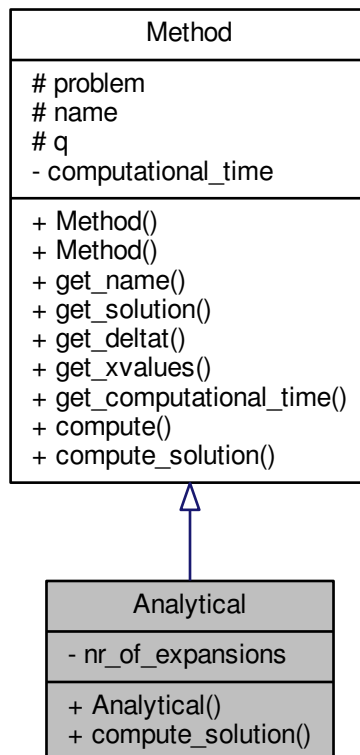
# Class Documentation

### 4.1 Analytical Class Reference

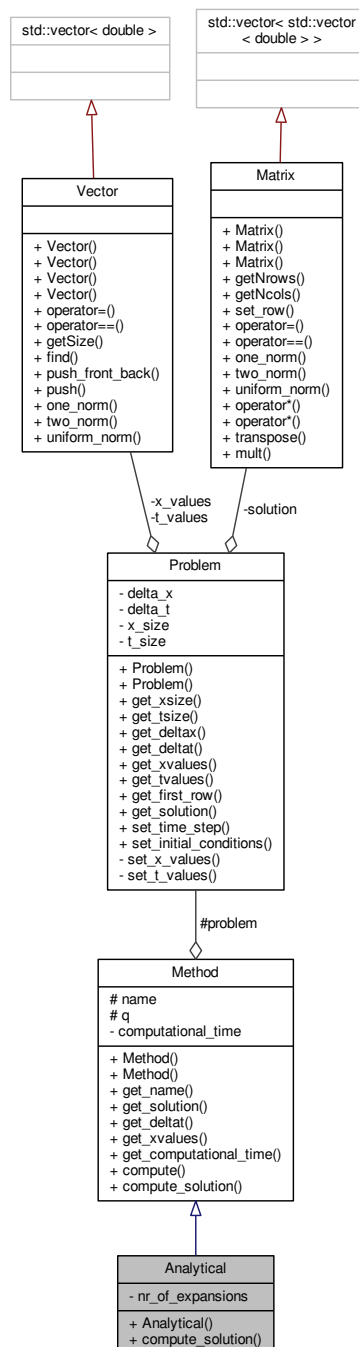
An [Analytical](#) class to compute the solution with standard procedures  
The implementation is derived from the [Method](#) Object.

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

Inheritance diagram for Analytical:



Collaboration diagram for Analytical:



## Public Member Functions

- [Analytical \(Problem problem\)](#)

*Default constructor.*

- void [compute\\_solution \(\)](#)

*Normal public method.*

## Private Attributes

- unsigned int [nr\\_of\\_expansions](#)  
*Private unsigned int nr\_of\_expansions.*

## Additional Inherited Members

### 4.1.1 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

### 4.1.2 Constructor & Destructor Documentation

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

Default constructor.

Intialize a [Analytical](#) object

### 4.1.3 Member Function Documentation

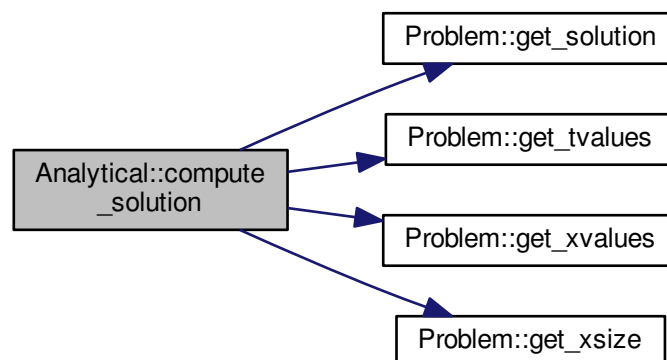
#### 4.1.3.1 [void Analytical::compute\\_solution](#) ( ) [[virtual](#)]

Normal public method.

compute the solution with specific given rules

Implements [Method](#).

Here is the call graph for this function:



#### 4.1.4 Member Data Documentation

##### 4.1.4.1 unsigned int Analytical::nr\_of\_expansions [private]

Private unsigned int nr\_of\_expansions.

Limit of expansions to do in the sum used to compute the solution.

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

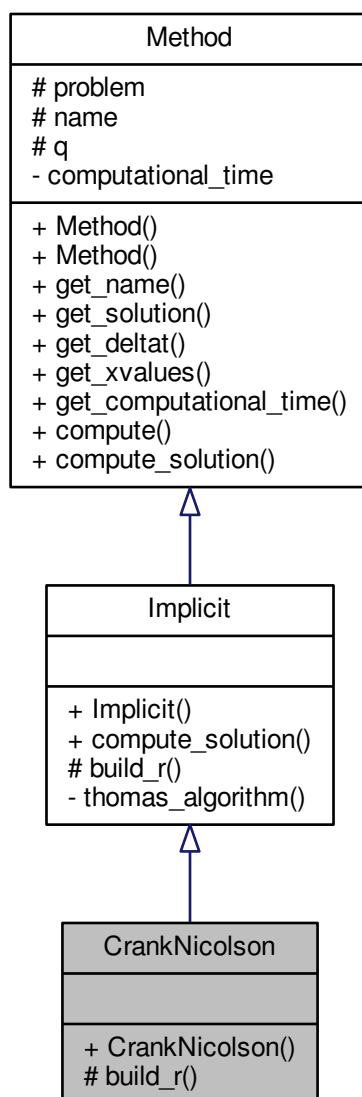
- [methods/analytical.h](#)
- [methods/analytical.cpp](#)

## 4.2 CrankNicolson Class Reference

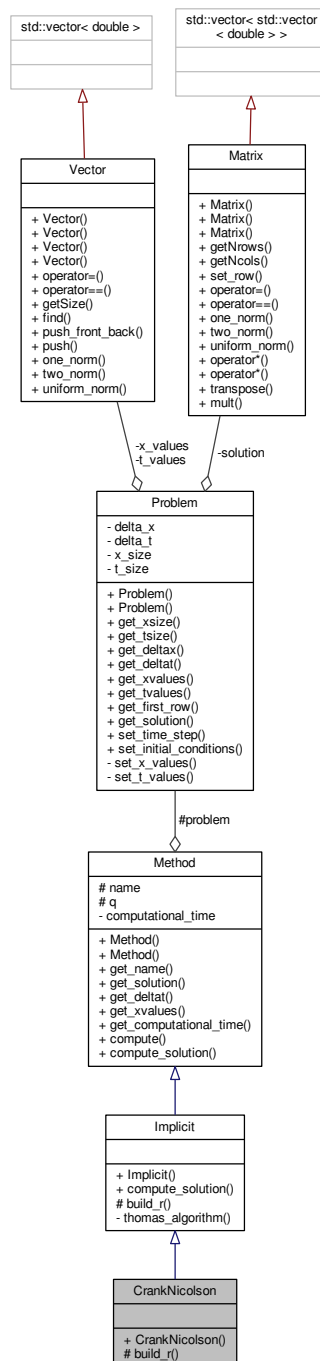
A [CrankNicolson](#) method class that contains a r vector builder.

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

Inheritance diagram for CrankNicolson:



Collaboration diagram for CrankNicolson:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

- [Vector build\\_r](#) ([Vector](#) previous\_step)  
*Normal protected method.*

## Additional Inherited Members

### 4.2.1 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.

### 4.2.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.2.3 Member Function Documentation

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

Normal protected method.

get the number of rows

#### Parameters

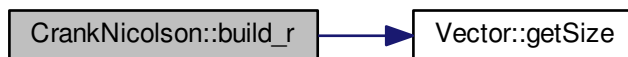
<i>previous_step</i>	<a href="#">Vector</a> representing the solution of the previous time step.
----------------------	---

#### Returns

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

Implements [Implicit](#).

Here is the call graph for this function:



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

- [methods/implicit/crank\\_nicolson.h](#)
- [methods/implicit/crank\\_nicolson.cpp](#)

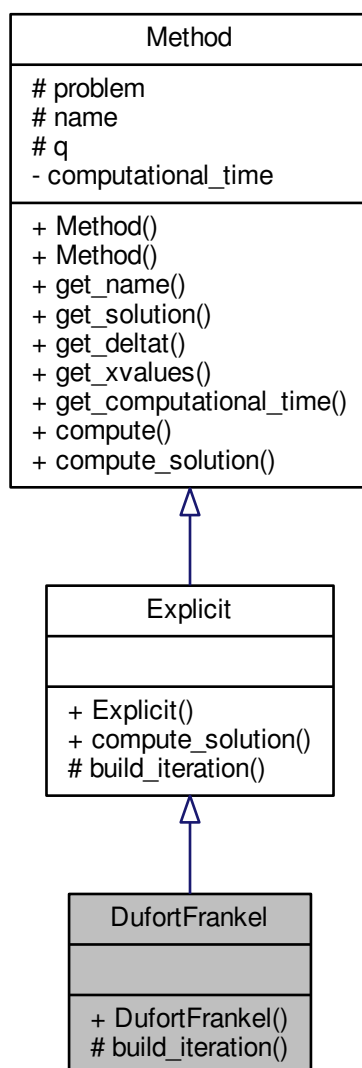
## 4.3 DufortFrankel Class Reference

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

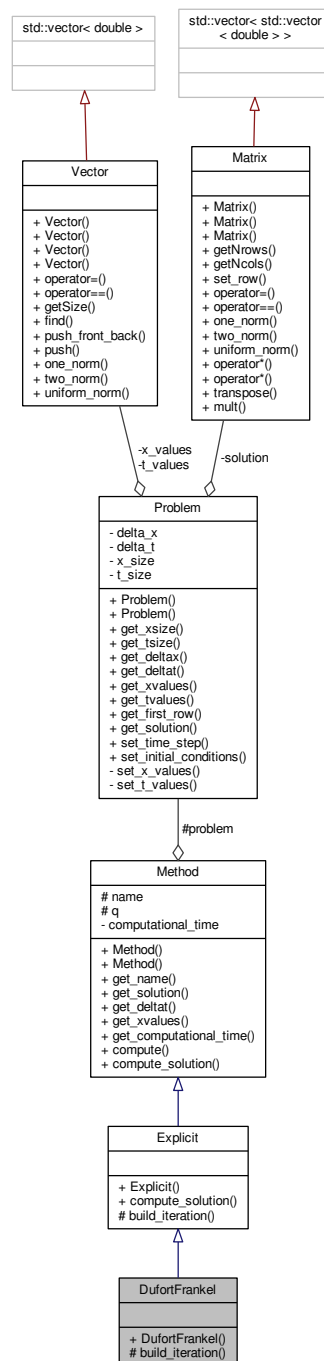
```
#include <dufort_frankel.h>
```



Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

- [Vector build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)  
*Normal protected method.*

## Additional Inherited Members

### 4.3.1 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

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 DufortFrankel::DufortFrankel ( [Problem](#) problem )

Default constructor.

### 4.3.3 Member Function Documentation

#### 4.3.3.1 [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

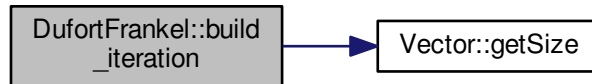
<i>current_step</i>	A vector representing the current time step solution.
<i>previous_step</i>	A vector representing the previous time step solution.

#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

Here is the call graph for this function:



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

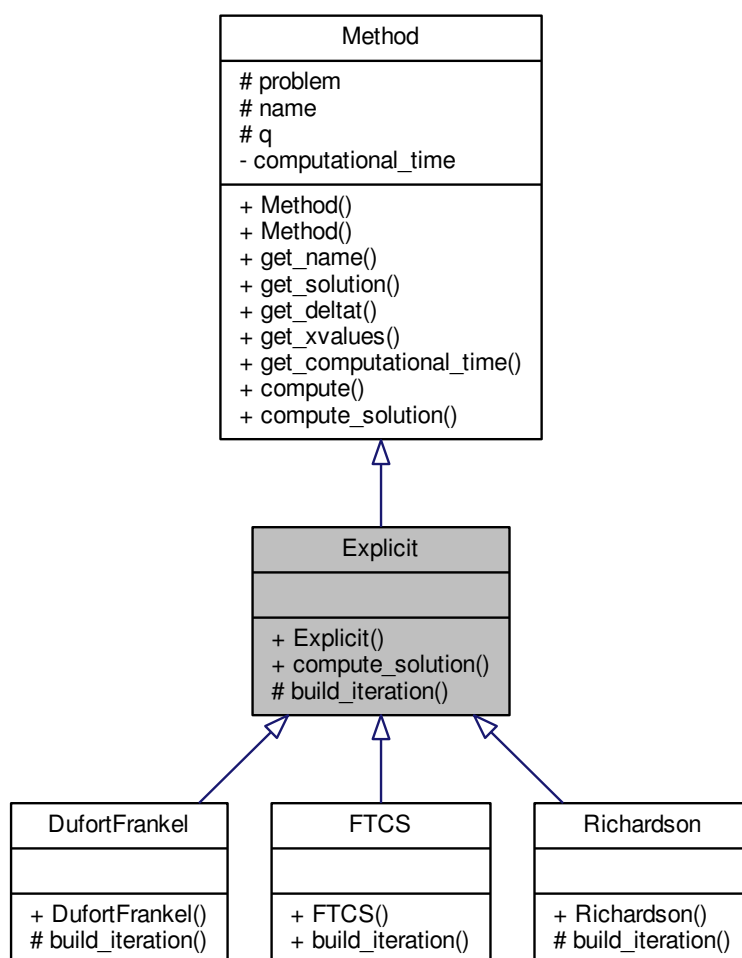
- [methods/explicit/dufort\\_frankel.h](#)
- [methods/explicit/dufort\\_frankel.cpp](#)

## 4.4 Explicit Class Reference

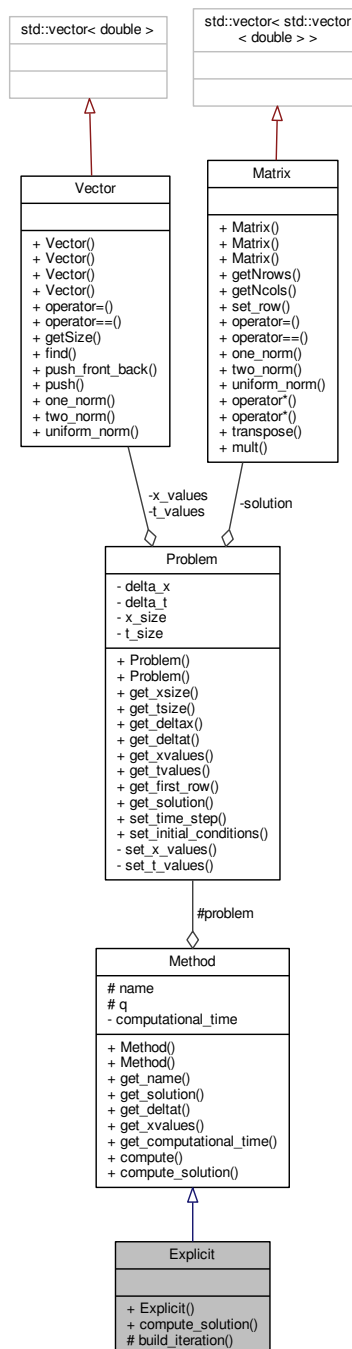
An explicit method class that contains default methods that only explicit methods use  
The implementation is derived from the [Method](#) class.

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

Inheritance diagram for Explicit:



Collaboration diagram for Explicit:



## Public Member Functions

- [Explicit \(Problem problem\)](#)  
*Default constructor.*
- void [compute\\_solution \(\)](#)  
*Normal public method.*

## Protected Member Functions

- virtual [Vector](#) [build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)=0  
*A pure virtual member.*

## Additional Inherited Members

### 4.4.1 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:

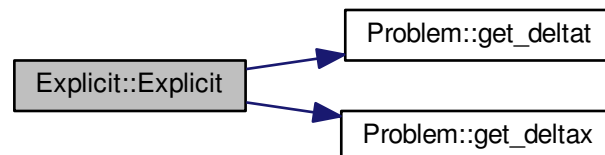
- a basic constructor for creating an explicit method object.
- a method to compute a solution following explicit methods rules

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 Explicit::Explicit ( [Problem](#) *problem* )

Default constructor.

Here is the call graph for this function:



### 4.4.3 Member Function Documentation

#### 4.4.3.1 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](#).

Here is the caller graph for this function:



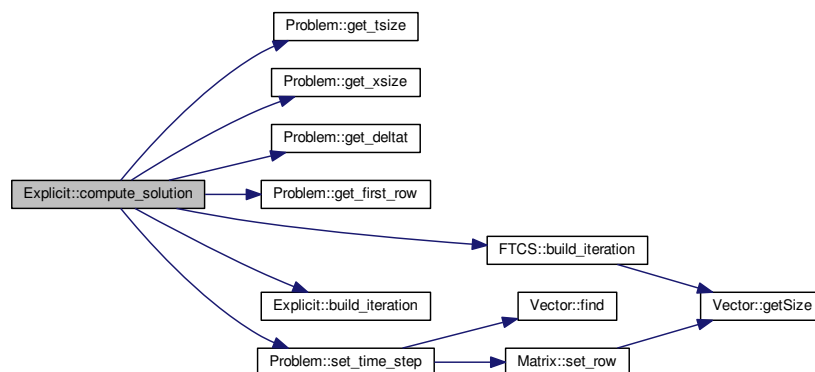
#### 4.4.3.2 void Explicit::compute\_solution ( ) [virtual]

Normal public method.

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

Implements [Method](#).

Here is the call graph for this function:



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

- [methods/explicit/explicit.h](#)
- [methods/explicit/explicit.cpp](#)

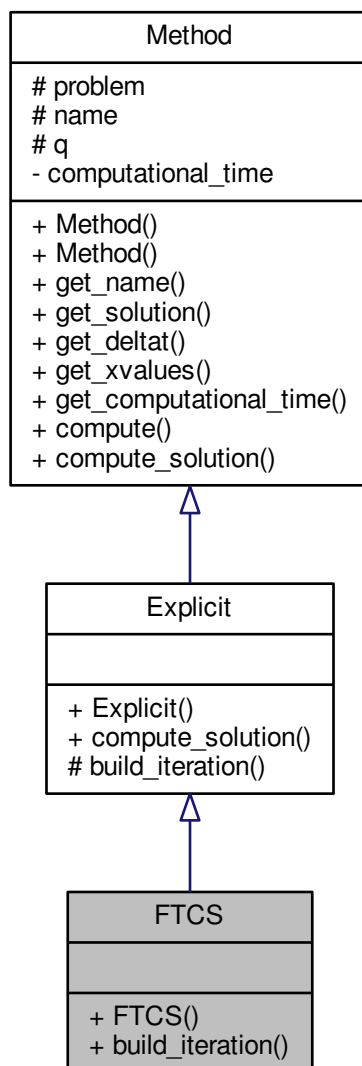


## 4.5 FTCS Class Reference

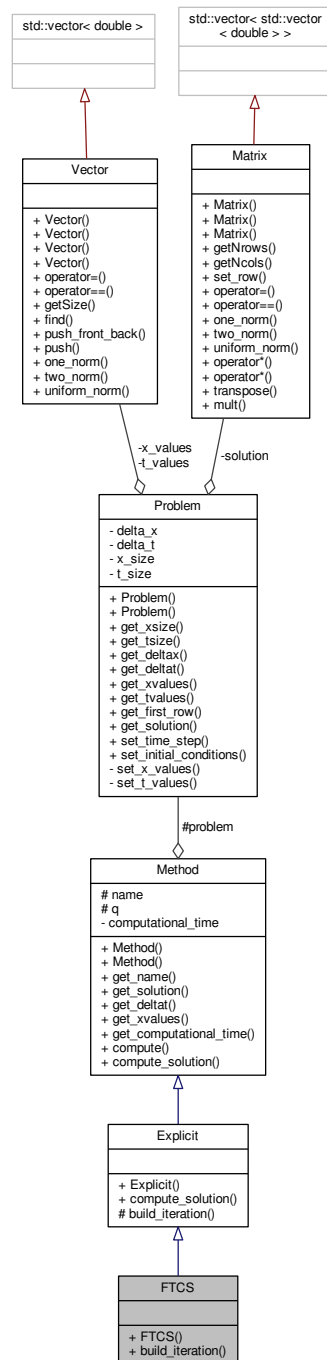
A [FTCS](#) method class that contains an iteration builder.

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

Inheritance diagram for FTCS:



Collaboration diagram for FTCS:



## Public Member Functions

- [FTCS](#) ([Problem problem](#))  
*Default constructor.*
- [Vector build\\_iteration](#) ([Vector current\\_step](#), [Vector previous\\_step](#))  
*Normal public method.*

## Additional Inherited Members

### 4.5.1 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

### 4.5.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.5.3 Member Function Documentation

#### 4.5.3.1 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](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

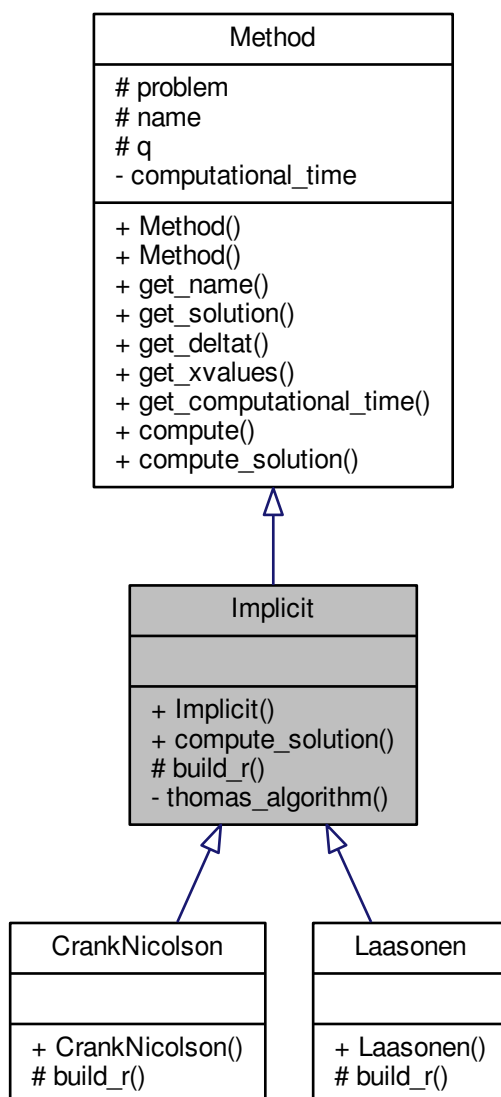
- [methods/explicit/forward\\_t\\_central\\_s.h](#)
- [methods/explicit/forward\\_t\\_central\\_s.cpp](#)

## 4.6 Implicit Class Reference

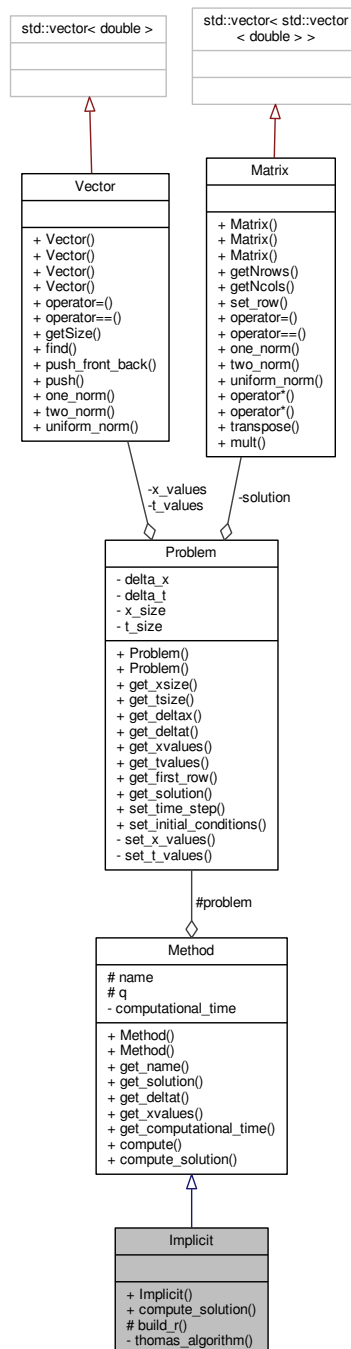
An implicit method class that contains default methods that only implicit methods use  
The implementation is derived from the [Method](#) class.

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

Inheritance diagram for Implicit:



Collaboration diagram for Implicit:



## Public Member Functions

- [Implicit \(Problem problem\)](#)

*Default constructor.*

- [void compute\\_solution \(\)](#)

*Normal public method.*

## Protected Member Functions

- virtual `Vector build_r (Vector previous_step)=0`  
*A pure virtual member.*

## Private Member Functions

- `Vector thomas_algorithm (Vector r, double a, double b, double c)`  
*Normal private method.*

## Additional Inherited Members

### 4.6.1 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:

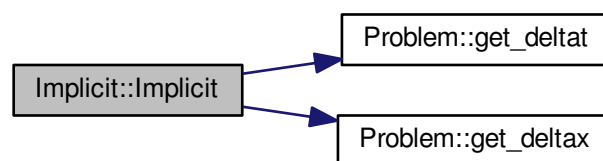
- a basic constructor for creating an implicit method object.
- a method to compute a solution following implicit methods rules

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 `Implicit::Implicit ( Problem problem )`

Default constructor.

Here is the call graph for this function:



### 4.6.3 Member Function Documentation

#### 4.6.3.1 `virtual Vector Implicit::build_r ( Vector previous_step ) [protected],[pure virtual]`

A pure virtual member.

Build the `r` vector in a linear system of  $A.x = r$  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

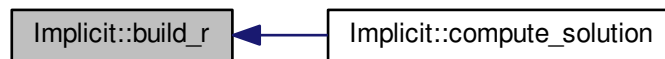
<i>previous_step</i>	A vector containing the previous time step solution.
----------------------	--

## 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](#).

Here is the caller graph for this function:



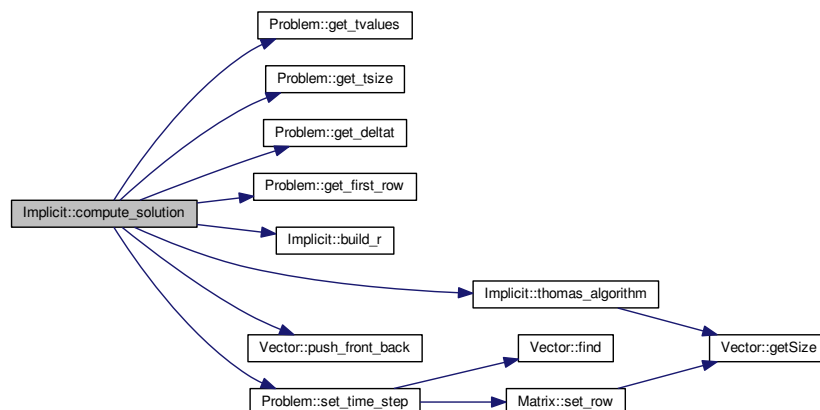
#### 4.6.3.2 void Implicit::compute\_solution ( ) [virtual]

Normal public method.

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

Implements [Method](#).

Here is the call graph for this function:





#### 4.6.3.3 Vector Implicit::thomas\_algorithm ( Vector *r*, double *a*, double *b*, double *c* ) [private]

Normal private method.

Calculates the current time step with Tomas Algorithm. Giving the  $A.x = r$ , in which  $A$  is a matrix, whereas  $b$  and  $r$  are vectors, it calculates the  $b$  vector, since  $A$  and  $b$  are known variables.

See also

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

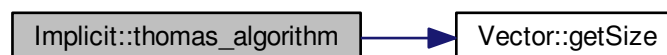
##### Parameters

<i>r</i>	<a href="#">Vector</a> calculated by the build_r method.
<i>a</i>	Lower diagonal value of the tridiagonal matrix
<i>b</i>	Center diagonal value of the tridiagonal matrix
<i>c</i>	Upper diagonal value of the tridiagonal matrix

##### Returns

[Vector](#). [Vector](#) that represents the current time step solution.

Here is the call graph for this function:



Here is the caller graph for this function:



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

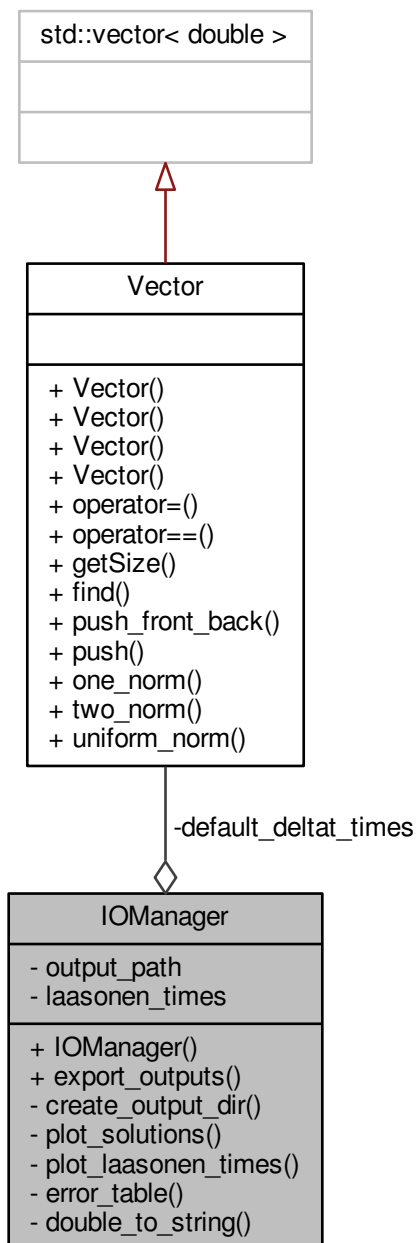
- [methods/implicit/implicit.h](#)
- [methods/implicit/implicit.cpp](#)

## 4.7 IOManager Class Reference

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

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

Collaboration diagram for IOManager:



## Public Member Functions

- [IOManager](#) ()  
*Default constructor.*
- void [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.*

## Private Member Functions

- bool [create\\_output\\_dir](#) ()  
*Method to create output folder if the folder does not exist.*
- void [plot\\_solutions](#) (std::string output\_name, [Method](#) \*analytical, [Method](#) \*method)  
*Exports a plot chart that compares the analytical solution to any other solution using gnuplot.*
- void [plot\\_laasonen\\_times](#) ()  
*Exports a plot with [Laasonen](#) times computational times.*
- void [error\\_table](#) (std::string output\_name, [Method](#) \*analytical, [Method](#) \*method)  
*Exports an error table to a .lsx file that compares the analytical solution to any other solution.*
- std::string [double\\_to\\_string](#) (int precision, double value)  
*Converts a double to a string with a precision of 2 decimal places.*

## Private Attributes

- std::string [output\\_path](#)  
*Private string output\_path.*
- std::vector< double > [laasonen\\_times](#)  
*Private [Vector](#) laasonen\_times.*
- [Vector](#) [default\\_deltat\\_times](#)  
*Private [Vector](#) default\_deltat\_times.*

### 4.7.1 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

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 IOManager::IOManager ( )

Default constructor.

Initialize an [IOManager](#) object.

### 4.7.3 Member Function Documentation

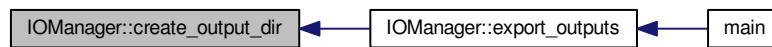
#### 4.7.3.1 `bool IOManager::create_output_dir ( )` `[private]`

**Method** to create output folder if the folder does not exist.

##### Returns

bool. true if successful, false if not

Here is the caller graph for this function:



#### 4.7.3.2 `std::string IOManager::double_to_string ( int precision, double value )` `[private]`

Converts a double to a string with a precision of 2 decimal places.

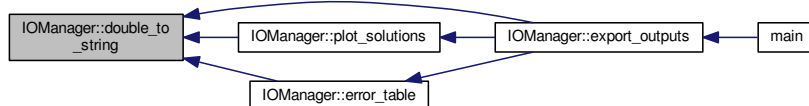
##### Parameters

<i>double</i>	value Number to be converted
<i>int</i>	precision Precision to have

##### Returns

string. String containing the converted number

Here is the caller graph for this function:



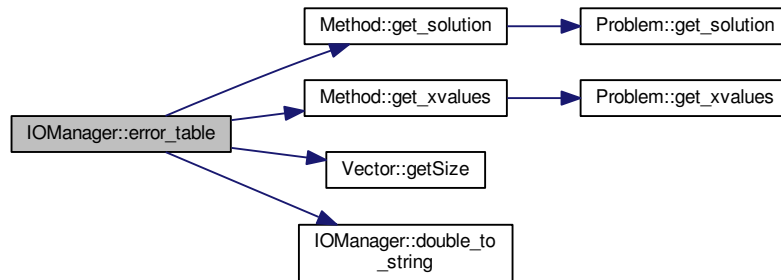
#### 4.7.3.3 `void IOManager::error_table ( std::string output_name, Method * analytical, Method * method )` `[private]`

Exports an error table to a .lsx file that compares the analytical solution to any other solution.

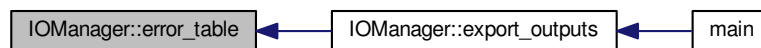
## Parameters

<i>string</i>	output_name File name to be exported
<i>Method*</i>	analytical The analytical solution
<i>Method*</i>	method Any method solution

Here is the call graph for this function:



Here is the caller graph for this function:



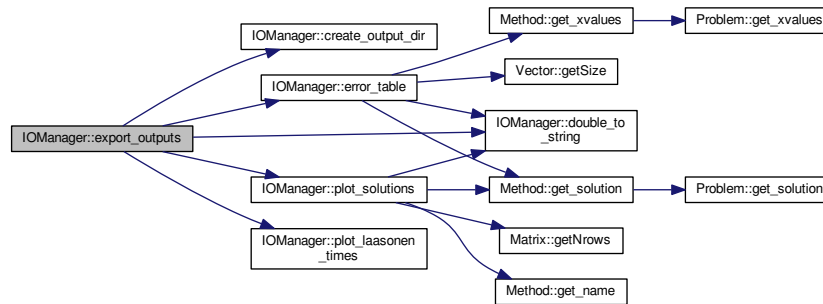
#### 4.7.3.4 void IOManager::export\_outputs ( Method \* analytical, std::vector< Method \* > methods )

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

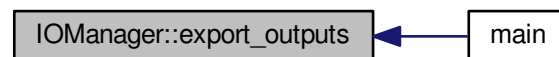
## Parameters

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

Here is the call graph for this function:



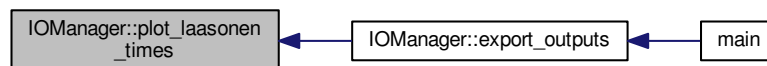
Here is the caller graph for this function:



#### 4.7.3.5 void IOManager::plot\_laasonen\_times ( ) [private]

Exports a plot with [Laasonen](#) times computational times.

Here is the caller graph for this function:



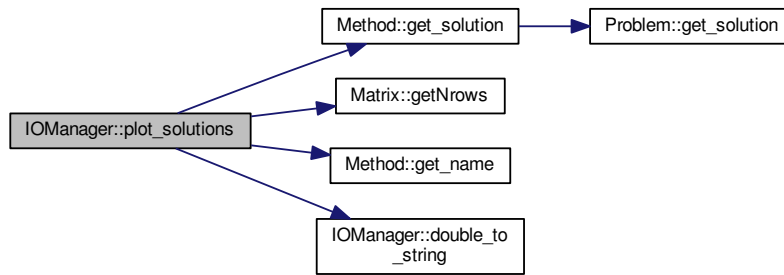
#### 4.7.3.6 void IOManager::plot\_solutions ( std::string output\_name, Method \* analytical, Method \* method ) [private]

Exports a plot chart that compares the analytical solution to any other solution using gnuplot.

##### Parameters

<i>string</i>	output_name File name to be exported
<i>Method*</i>	analytical The analytical solution
<i>Method*</i>	method Any method solution

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.4 Member Data Documentation

##### 4.7.4.1 Vector IOManager::default\_deltat\_times [private]

Private [Vector](#) `default_deltat_times`.

Contains the computation time of each method solution, with a time step of 0.01.

##### 4.7.4.2 std::vector<double> IOManager::laasonen\_times [private]

Private [Vector](#) `laasonen_times`.

Contains the computation time of each laasonen solution, with a different time step.

##### 4.7.4.3 std::string IOManager::output\_path [private]

Private string `output_path`.

Contains the output directory path name.

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

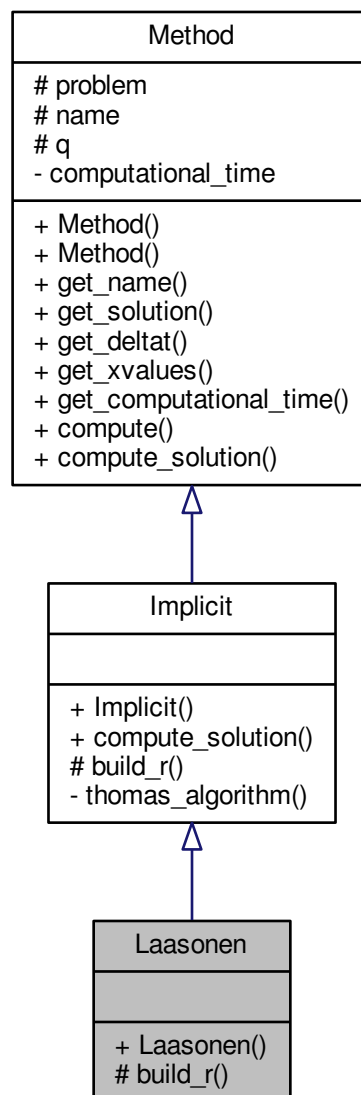
- [io/iomanager.h](#)
- [io/iomanager.cpp](#)

## 4.8 Laasonen Class Reference

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

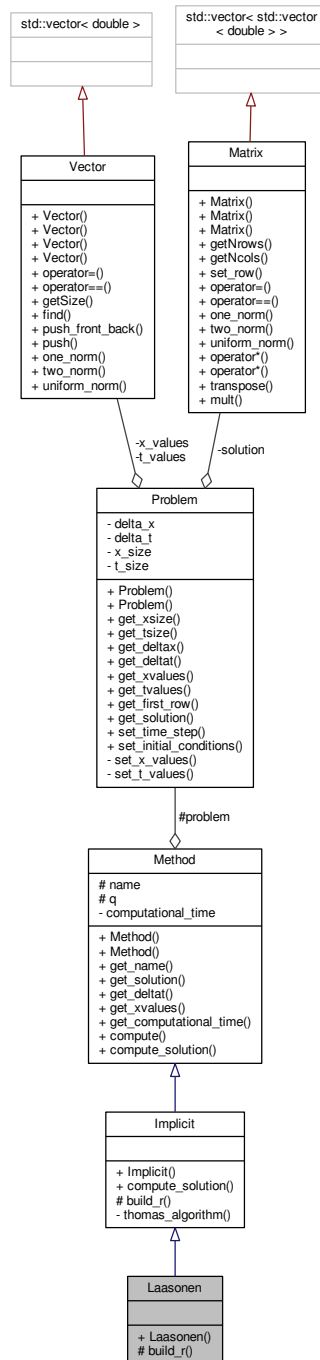
```
#include <laasonen.h>
```

Inheritance diagram for Laasonen:





Collaboration diagram for Laasonen:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

- [Vector build\\_r](#) ([Vector](#) previous\_step)  
*Normal protected method.*

## Additional Inherited Members

### 4.8.1 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.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 [Laasonen::Laasonen](#) ( [Problem](#) problem )

Default constructor.

### 4.8.3 Member Function Documentation

#### 4.8.3.1 [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](#).

Here is the call graph for this function:



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

- [methods/implicit/laasonen.h](#)
- [methods/implicit/laasonen.cpp](#)

## 4.9 Matrix Class Reference

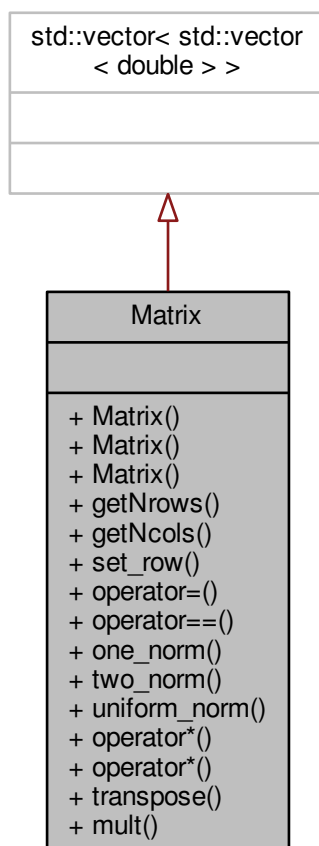
A matrix class for data storage of a 2D array of doubles

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

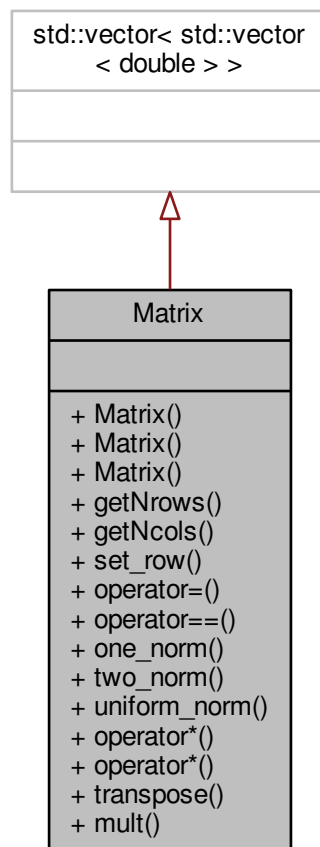
We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`.

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

Inheritance diagram for Matrix:



Collaboration diagram for Matrix:



## Public Member Functions

- [Matrix](#) ()  
*Default constructor.*
- [Matrix](#) (int Nrows, int Ncols)  
*Alternate constructor.*
- [Matrix](#) (const [Matrix](#) &m)  
*Copy constructor.*
- int [getNrows](#) () const  
*Normal public get method.*
- int [getNcols](#) () const  
*Normal public get method.*
- void [set\\_row](#) (int index, [Vector](#) v)  
*Normal public set method.*
- [Matrix](#) & [operator=](#) (const [Matrix](#) &m)  
*Overloaded assignment operator.*
- bool [operator==](#) (const [Matrix](#) &m) const

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

- double `one_norm` () const  
Normal public method that returns a double.
- double `two_norm` () const  
Normal public method that returns a double.
- double `uniform_norm` () const  
Normal public method that returns a double.
- `Matrix operator*` (const `Matrix` &a) const  
Overloaded \*operator that returns a `Matrix`.
- `Vector operator*` (const `Vector` &v) const  
Overloaded \*operator that returns a `Vector`.
- `Matrix transpose` () const  
public method that returns the transpose of the matrix.
- `Matrix mult` (const `Matrix` &a) const

## Private Types

- typedef std::vector< std::vector< double > > `vec`

## Friends

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

### 4.9.1 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

## 4.9.2 Member Typedef Documentation

4.9.2.1 `typedef std::vector<std::vector<double> > Matrix::vec` `[private]`

## 4.9.3 Constructor & Destructor Documentation

4.9.3.1 `Matrix::Matrix ( )`

Default constructor.

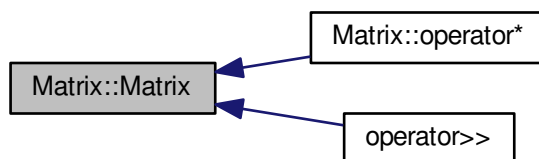
Initialize an empty [Matrix](#) object

See also

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

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

Here is the caller graph for this function:



4.9.3.2 `Matrix::Matrix ( int Nrows, int Ncols )`

Alternate constructor.

build a matrix *Nrows* by *Ncols*

See also

[Matrix\(\)](#)

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

Exceptions

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

## Parameters

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

4.9.3.3 `Matrix::Matrix ( const Matrix & m )`

Copy constructor.

build a matrix from another matrix

## See also

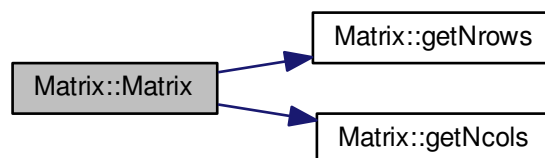
[Matrix\(\)](#)

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

## Parameters

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

Here is the call graph for this function:



## 4.9.4 Member Function Documentation

4.9.4.1 `int Matrix::getNcols ( ) const`

Normal public get method.

get the number of columns

## See also

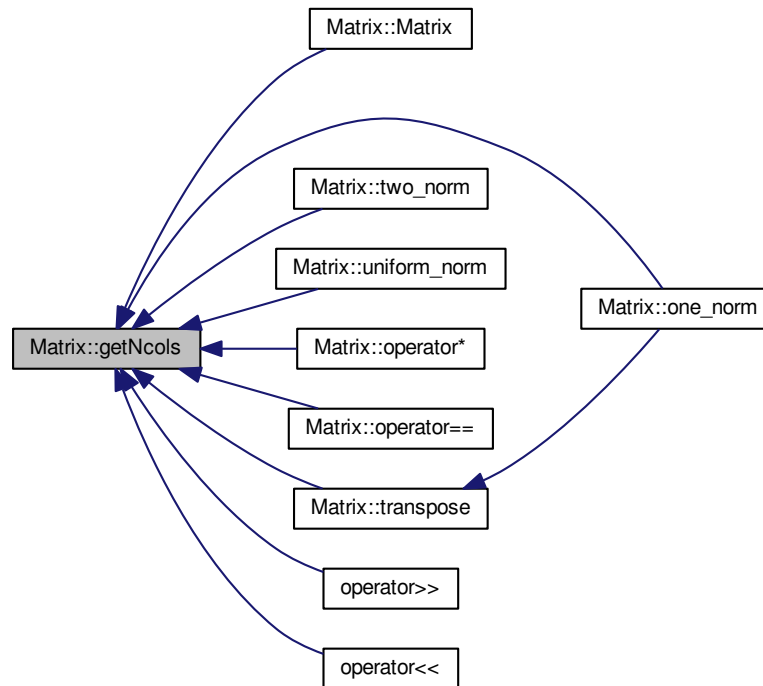
int [getNrows\(\)](#)const



## Returns

int. number of columns in matrix

Here is the caller graph for this function:



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

Normal public get method.

get the number of rows

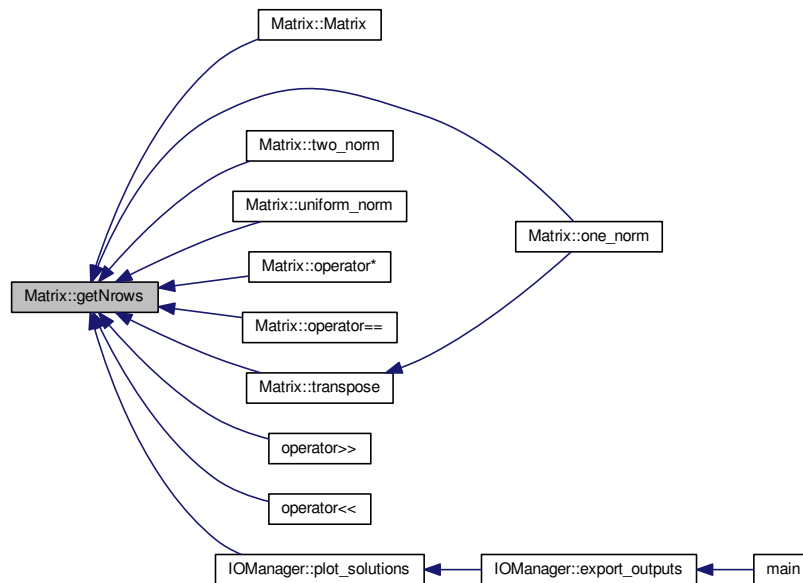
## See also

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

**Returns**

int. number of rows in matrix

Here is the caller graph for this function:



#### 4.9.4.3 `Matrix Matrix::mult ( const Matrix & a ) const`

#### 4.9.4.4 `double Matrix::one_norm ( ) const`

Normal public method that returns a double.

It returns L1 norm of matrix

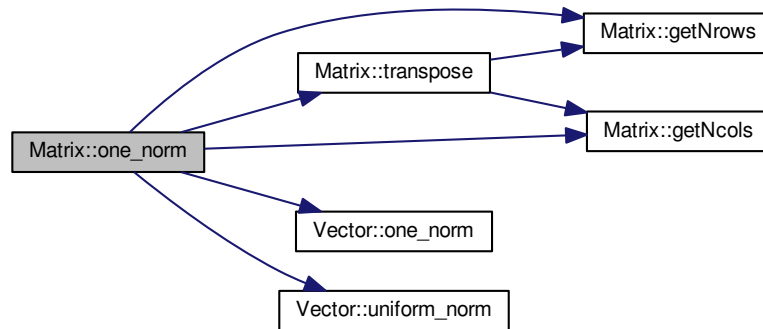
**See also**

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

**Returns**

double. matrix L1 norm

Here is the call graph for this function:

**4.9.4.5 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

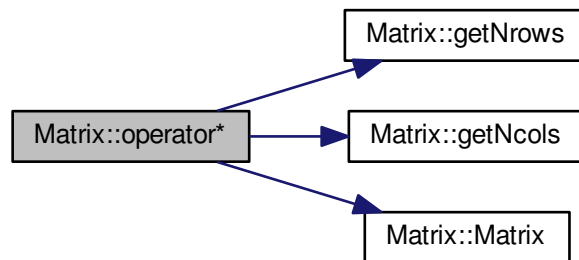
**Returns**

[Matrix](#). matrix-matrix product

**Parameters**

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

Here is the call graph for this function:



#### 4.9.4.6 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

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

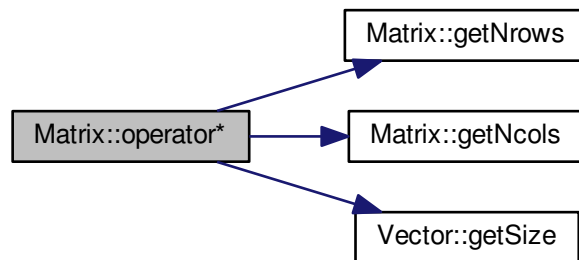
#### Returns

[Vector](#). matrix-vector product

#### Parameters

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

Here is the call graph for this function:



#### 4.9.4.7 `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
----------	---

#### 4.9.4.8 `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=\(const Matrix& m\)](#)

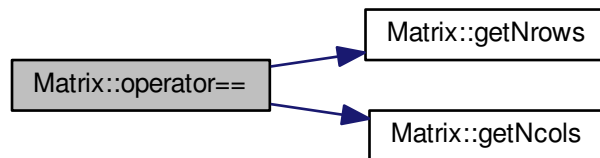
Returns

bool. true or false

## Parameters

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

Here is the call graph for this function:



#### 4.9.4.9 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

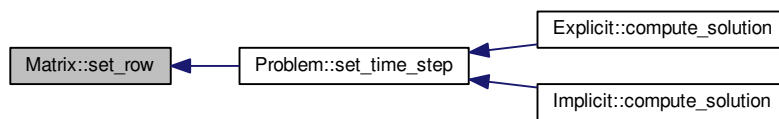
## Exceptions

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.9.4.10 Matrix `Matrix::transpose ( ) const`

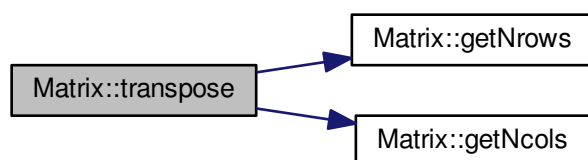
public method that returns the transpose of the matrix.

It returns the transpose of matrix

##### Returns

[Matrix](#). matrix transpose

Here is the call graph for this function:



Here is the caller graph for this function:



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

Normal public method that returns a double.

It returns L2 norm of matrix

See also

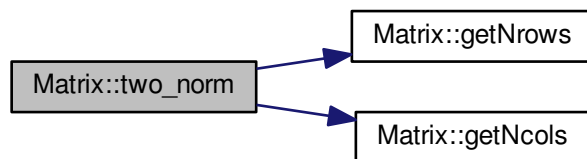
[one\\_norm\(\)const](#)

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

Returns

double. matrix L2 norm

Here is the call graph for this function:



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

Normal public method that returns a double.

It returns L\_max norm of matrix

See also

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

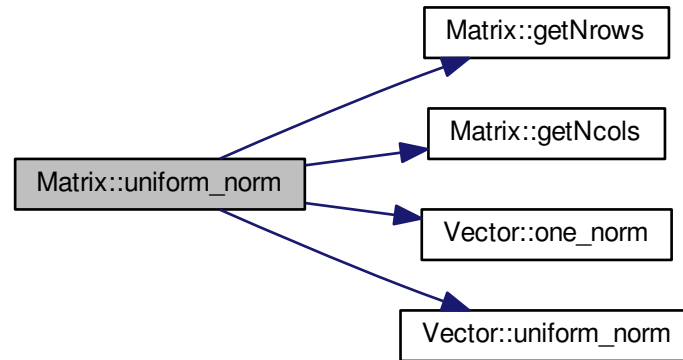
[two\\_norm\(\)const](#)



**Returns**

double. matrix L\_max norm

Here is the call graph for this function:



### 4.9.5 Friends And Related Function Documentation

#### 4.9.5.1 `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

#### 4.9.5.2 `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
----------	-------------------------------------

#### 4.9.5.3 `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

4.9.5.4 `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:

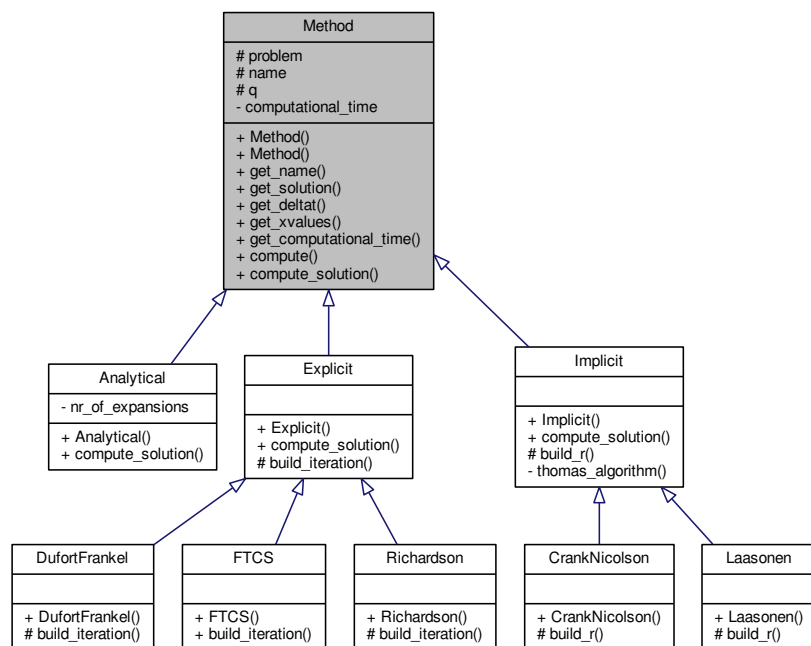
- [grid/matrix.h](#)
- [grid/matrix.cpp](#)

## 4.10 Method Class Reference

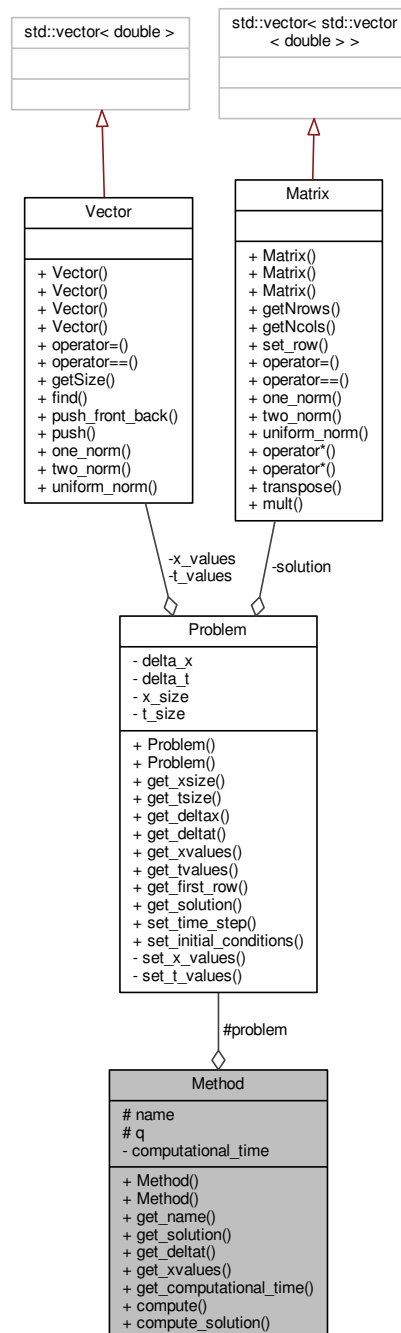
A [Method](#) class to structure information used to solve the problem.

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

Inheritance diagram for Method:



Collaboration diagram for Method:



## Public Member Functions

- [Method \(\)](#)  
*Default constructor.*
- [Method \(Problem problem\)](#)  
*Alternate constructor.*
- `std::string` [get\\_name \(\)](#)

- Normal public get method.*
- [Matrix get\\_solution \(\)](#)  
*Normal public get method.*
- double [get\\_deltat \(\)](#)  
*Normal public get method.*
- [Vector get\\_xvalues \(\)](#)  
*Normal public get method.*
- double [get\\_computational\\_time \(\)](#)  
*Normal public get method.*
- void [compute \(\)](#)  
*Normal public method.*
- virtual void [compute\\_solution \(\)=0](#)  
*A pure virtual member.*

## Protected Attributes

- [Problem problem](#)  
*Protected [Problem](#) problem.*
- std::string [name](#)  
*Protected string name.*
- double [q](#)  
*Protected double q.*

## Private Attributes

- double [computational\\_time](#)  
*Private double computational\_time.*

### 4.10.1 Detailed Description

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

The [Method](#) class provides:

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

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 [Method::Method \( \)](#)

Default constructor.

Intialize a [Method](#) object

See also

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

#### 4.10.2.2 Method::Method ( Problem *problem* )

Alternate constructor.

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

See also

[Method\(\)](#)

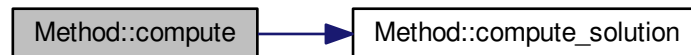
### 4.10.3 Member Function Documentation

#### 4.10.3.1 void Method::compute ( )

Normal public method.

Keeps track of the time to compute a solution

Here is the call graph for this function:



#### 4.10.3.2 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](#).

Here is the caller graph for this function:



#### 4.10.3.3 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.

#### 4.10.3.4 double Method::get\_deltat ( )

Normal public get method.

get the time step of the solution

##### Returns

double. Solution time step.

Here is the call graph for this function:



#### 4.10.3.5 std::string Method::get\_name ( )

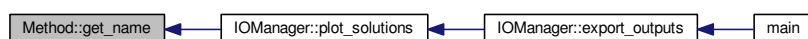
Normal public get method.

get the method name

##### Returns

string. [Method](#) name.

Here is the caller graph for this function:





#### 4.10.3.6 Matrix Method::get\_solution ( )

Normal public get method.

get the solution grid

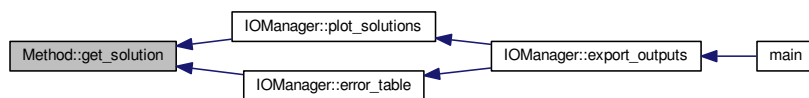
Returns

[Matrix](#). Computed solution grid.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.3.7 Vector Method::get\_xvalues ( )

Normal public get method.

get x values vector

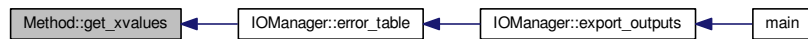
Returns

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.4 Member Data Documentation

##### 4.10.4.1 `double Method::computational_time` `[private]`

Private double computational\_time.

Elapsed time throughout the solution computation.

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

Protected string name.

Name of the method.

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

Protected [Problem](#) problem.

Space step of the solution.

##### 4.10.4.4 `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:

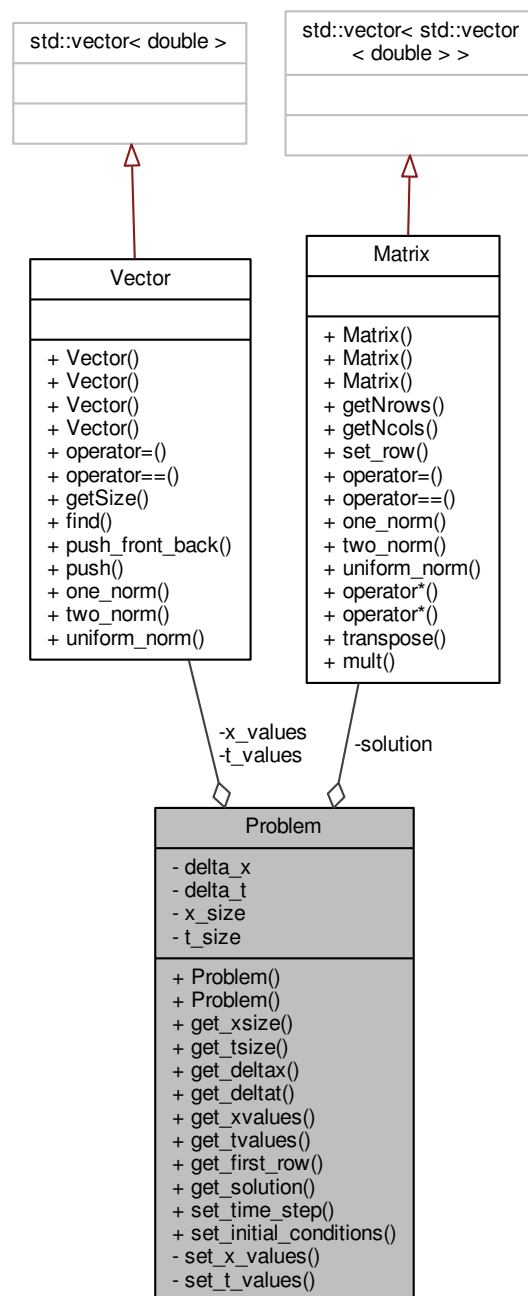
- [methods/method.h](#)
- [methods/method.cpp](#)

## 4.11 Problem Class Reference

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

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

Collaboration diagram for Problem:



## Public Member Functions

- [Problem](#) ()  
*Default constructor.*
- [Problem](#) (double dt, double dx)  
*Initialize [Problem](#) object with specific time and space steps.*
- unsigned int [get\\_xsize](#) ()  
*Normal public get method that returns an unsigned int, the number of columns of the solution.*
- unsigned int [get\\_tsize](#) ()  
*Normal public get method that returns an unsigned int, the number of rows of the solution.*
- double [get\\_deltax](#) ()  
*Normal public get method that returns a double, the space step value of the solution.*
- double [get\\_deltat](#) ()  
*Normal public get method that returns a double, the time step value of the solution.*
- [Vector](#) [get\\_xvalues](#) ()  
*Normal public get method that returns a [Vector](#), containing the space values in each column.*
- [Vector](#) [get\\_tvalues](#) ()  
*Normal public get method that returns a [Vector](#), containing the time values in each row.*
- [Vector](#) [get\\_first\\_row](#) ()  
*Normal public get method that returns a [Vector](#), containing the initial boundaries in the first row of the solution.*
- [Matrix](#) \* [get\\_solution](#) ()  
*Normal public get method that returns a [Matrix](#), containing the solution solution.*
- void [set\\_time\\_step](#) ([Vector](#) step, double time)  
*Normal public set method.*
- void [set\\_initial\\_conditions](#) ()  
*Normal public set method.*

## Private Member Functions

- void [set\\_x\\_values](#) ()  
*Normal private set method.*
- void [set\\_t\\_values](#) ()  
*Normal private set method.*

## Private Attributes

- double [delta\\_x](#)  
*Private double delta\_x.*
- double [delta\\_t](#)  
*Private double delta\_t.*
- unsigned int [x\\_size](#)  
*Private unsigned int x\_size.*
- unsigned int [t\\_size](#)  
*Private unsigned int t\_size.*
- [Vector](#) [x\\_values](#)  
*Private [Vector](#) x\_values.*
- [Vector](#) [t\\_values](#)  
*Private [Vector](#) t\_values.*
- [Matrix](#) [solution](#)  
*Private [Matrix](#) solution.*

### 4.11.1 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

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 [Problem::Problem \( \)](#)

Default constructor.

Intialize an empty [Problem](#) object

See also

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

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

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

See also

[Problem\(\)](#)

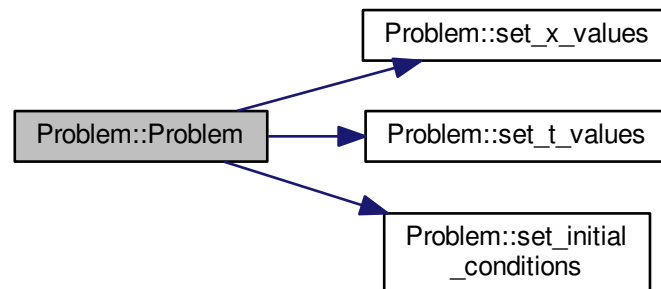
#### Parameters

<i>dt</i>	Time step to assign
<i>dx</i>	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")

Here is the call graph for this function:



### 4.11.3 Member Function Documentation

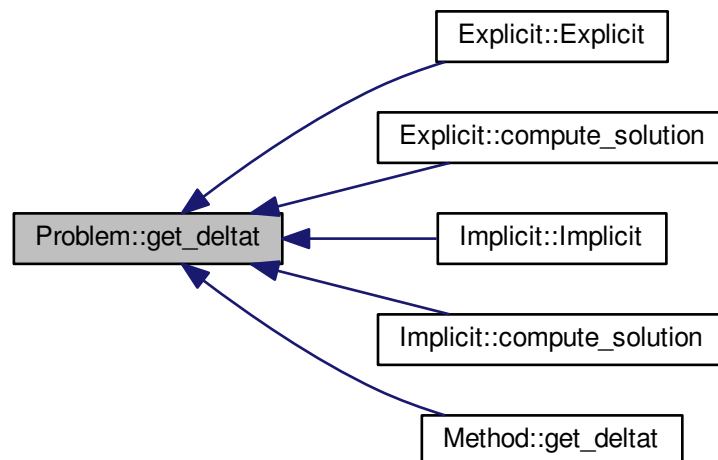
#### 4.11.3.1 `double Problem::get_deltat ( )`

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

##### Returns

double. The time step value of the solution.

Here is the caller graph for this function:



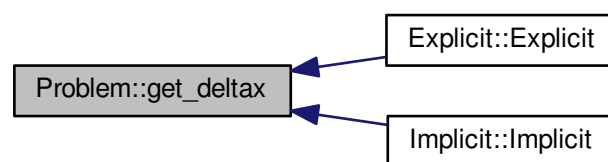
#### 4.11.3.2 double Problem::get\_deltax ( )

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

##### Returns

double. The space step value of the solution.

Here is the caller graph for this function:



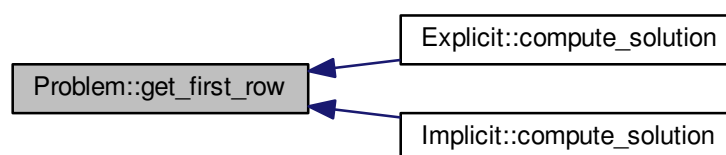
#### 4.11.3.3 Vector Problem::get\_first\_row ( )

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.

Here is the caller graph for this function:



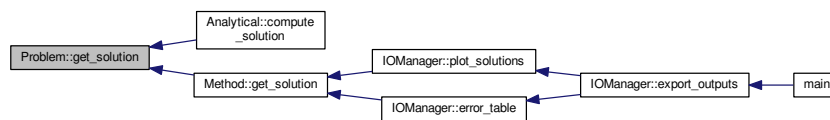
#### 4.11.3.4 **Matrix \*** Problem::get\_solution ( )

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

##### Returns

Matrix\*. The solution solution.

Here is the caller graph for this function:



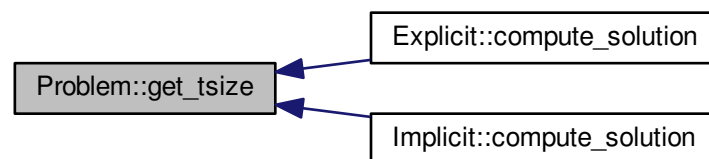
#### 4.11.3.5 **unsigned int** Problem::get\_tsize ( )

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.

Here is the caller graph for this function:



#### 4.11.3.6 **Vector** Problem::get\_tvalues ( )

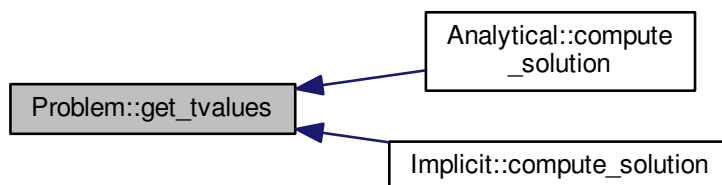
Normal public get method that returns a [Vector](#), containing the time values in each row.



## Returns

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

Here is the caller graph for this function:



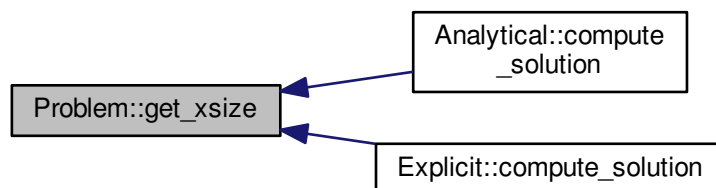
## 4.11.3.7 unsigned int Problem::get\_xsize ( )

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.

Here is the caller graph for this function:



## 4.11.3.8 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.

Here is the caller graph for this function:

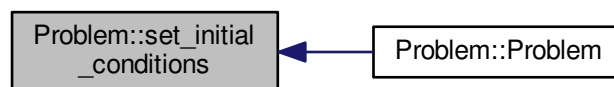


#### 4.11.3.9 void Problem::set\_initial\_conditions ( )

Normal public set method.

set the problem initial boundaries.

Here is the caller graph for this function:



#### 4.11.3.10 void Problem::set\_t\_values ( ) [private]

Normal private set method.

Initialize [Vector](#) t\_values with the correct values.

See also

[t\\_values](#)

Here is the caller graph for this function:



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

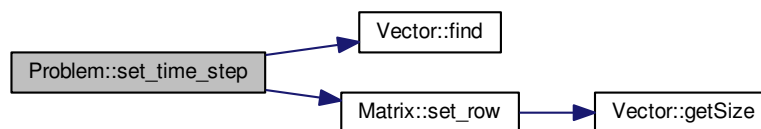
Normal public set method.

replace a row of the solution for a given [Vector](#).

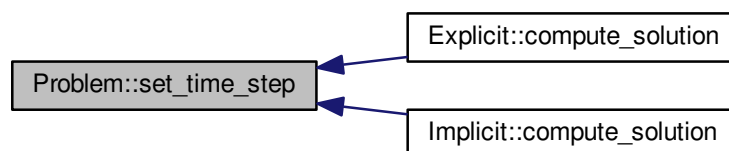
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.11.3.12 void Problem::set\_x\_values ( ) [private]

Normal private set method.

Initialize [Vector](#) `x_values` with the correct values.

See also

[x\\_values](#)

Here is the caller graph for this function:



#### 4.11.4 Member Data Documentation

##### 4.11.4.1 `double Problem::delta_t` [private]

Private double `delta_t`.

Time step of the solution.

##### 4.11.4.2 `double Problem::delta_x` [private]

Private double `delta_x`.

Space step of the solution.

##### 4.11.4.3 `Matrix Problem::solution` [private]

Private [Matrix](#) solution.

[Matrix](#) containing the computed solution.

##### 4.11.4.4 `unsigned int Problem::t_size` [private]

Private unsigned int `t_size`.

Time size of the solution.

##### 4.11.4.5 `Vector Problem::t_values` [private]

Private [Vector](#) `t_values`.

Time correspondent value for each row index.

#### 4.11.4.6 unsigned int Problem::x\_size [private]

Private unsigned int x\_size.

Space size of the solution.

#### 4.11.4.7 Vector Problem::x\_values [private]

Private [Vector](#) x\_values.

Space correspondent value for each column index.

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

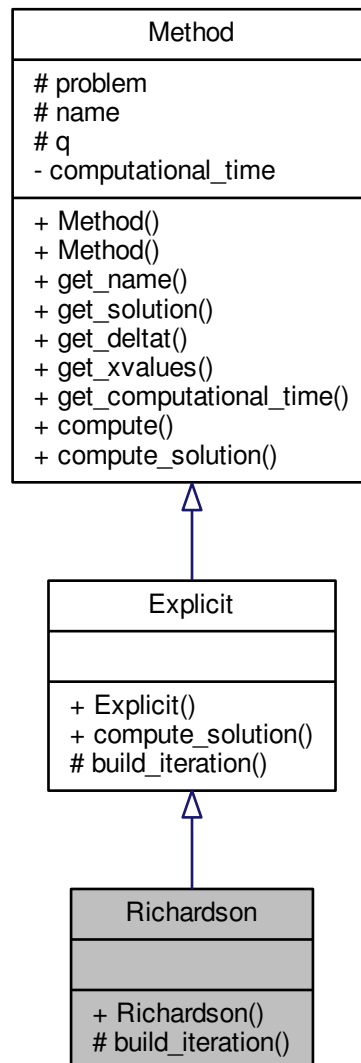
- [variants/problem.h](#)
- [variants/problem.cpp](#)

## 4.12 Richardson Class Reference

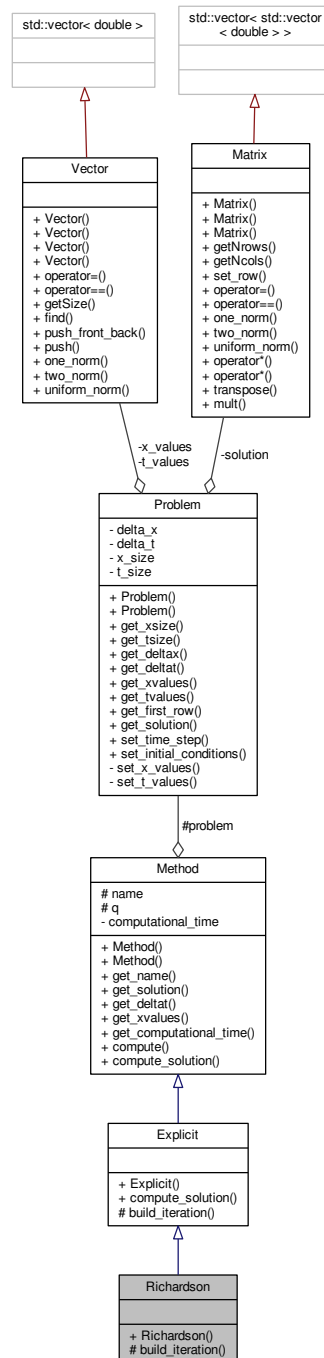
A [Richardson](#) method class that contains an iteration builder.

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

Inheritance diagram for Richardson:



Collaboration diagram for Richardson:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

- [Vector build\\_iteration](#) ([Vector](#) current\_step, [Vector](#) previous\_step)  
*Normal protected method.*

## Additional Inherited Members

### 4.12.1 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

### 4.12.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.12.3 Member Function Documentation

#### 4.12.3.1 [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.

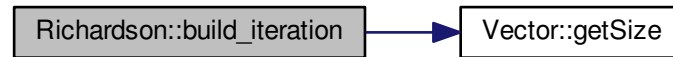
#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).



Here is the call graph for this function:



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

- [methods/explicit/richardson.h](#)
- [methods/explicit/richardson.cpp](#)

## 4.13 Vector Class Reference

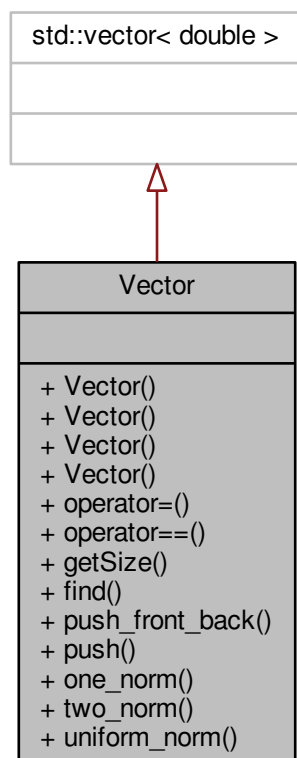
A vector class for data storage of a 1D array of doubles

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

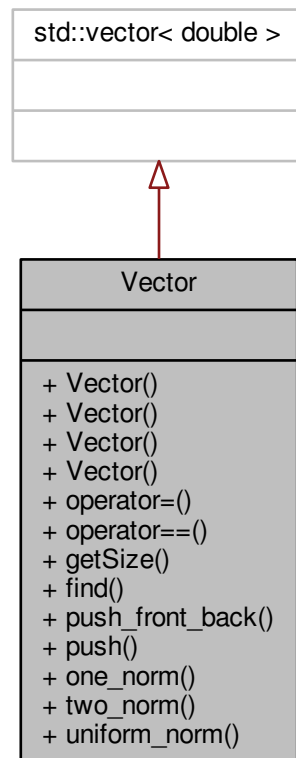
We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`.

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

Inheritance diagram for Vector:



Collaboration diagram for Vector:



## Public Member Functions

- `Vector ()`  
*Default constructor.*
- `Vector (int Num)`  
*Explicit alternative constructor takes an integer.*
- `Vector (const Vector &v)`  
*Copy constructor takes an Vector object reference.*
- `Vector (std::vector< double > vec)`  
*Copy constructor takes an vector<double> object reference.*
- `Vector & operator= (const Vector &v)`  
*Overloaded assignment operator.*
- `bool operator== (const Vector &v) const`  
*Overloaded comparison operator returns true if vectors are the same within a tolerance (1.e-07)*
- `int getSize () const`  
*Normal get method that returns integer, the size of the vector.*
- `int find (double value)`  
*Method to find the value index in a vector.*
- `void push_front_back (double value)`  
*Method to push a value to the first and last position of a Vector.*

- void [push](#) (double value)  
*Method to push a value to the last position of a [Vector](#).*
- double [one\\_norm](#) () const  
*Normal public method that returns a double.*
- double [two\\_norm](#) () const  
*Normal public method that returns a double.*
- double [uniform\\_norm](#) () const  
*Normal public method that returns a double.*

## Private Types

- typedef std::vector< double > [vec](#)

## Friends

- std::istream & [operator>>](#) (std::istream &is, [Vector](#) &v)  
*Overloaded istream >> operator.*
- std::ostream & [operator<<](#) (std::ostream &os, const [Vector](#) &v)  
*Overloaded ostream << operator.*
- std::ifstream & [operator>>](#) (std::ifstream &ifs, [Vector](#) &v)  
*Overloaded ifstream >> operator.*
- std::ofstream & [operator<<](#) (std::ofstream &ofs, const [Vector](#) &v)  
*Overloaded ofstream << operator.*

### 4.13.1 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

### 4.13.2 Member Typedef Documentation

4.13.2.1 typedef std::vector<double> [Vector::vec](#) `[private]`

### 4.13.3 Constructor & Destructor Documentation

4.13.3.1 [Vector::Vector](#) ( )

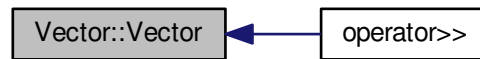
Default constructor.

Initialize an empty [Vector](#) object

See also

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

Here is the caller graph for this function:



#### 4.13.3.2 `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

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

#### Parameters

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

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

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

Initialize [Vector](#) object with another [Vector](#) object

See also

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

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

### 4.13.4 Member Function Documentation

#### 4.13.4.1 int Vector::find ( double value )

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

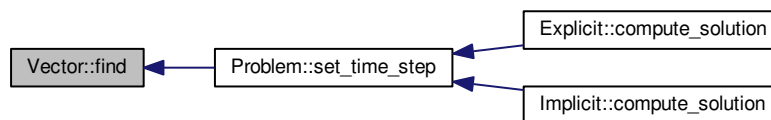
Parameters

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

Returns

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

Here is the caller graph for this function:



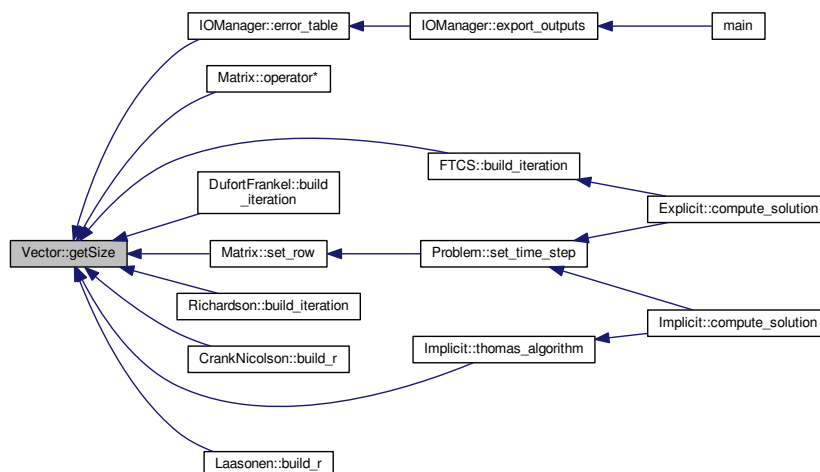
#### 4.13.4.2 int Vector::getSize ( ) const

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

**Returns**

int. the size of the vector

Here is the caller graph for this function:

**4.13.4.3 double Vector::one\_norm ( ) const**

Normal public method that returns a double.

It returns L1 norm of vector

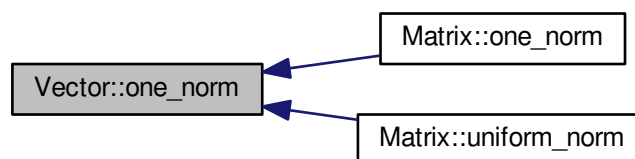
**See also**

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

**Returns**

double. vectors L1 norm

Here is the caller graph for this function:



#### 4.13.4.4 `Vector & Vector::operator= ( const Vector & v )`

Overloaded assignment operator.

See also

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

##### Parameters

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

##### Returns

the object on the left of the assignment

##### Parameters

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

#### 4.13.4.5 `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

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

#### 4.13.4.6 `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
--------------	--------------------

4.13.4.7 `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
--------------	-----------------

Here is the caller graph for this function:



4.13.4.8 `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

4.13.4.9 `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](#)

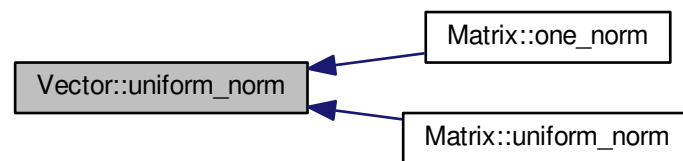
**Exceptions**

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

**Returns**

double. vectors Lmax norm

Here is the caller graph for this function:

**4.13.5 Friends And Related Function Documentation****4.13.5.1 `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

**4.13.5.2 `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

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

#### 4.13.5.3 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

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

Parameters

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

#### 4.13.5.4 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](#)

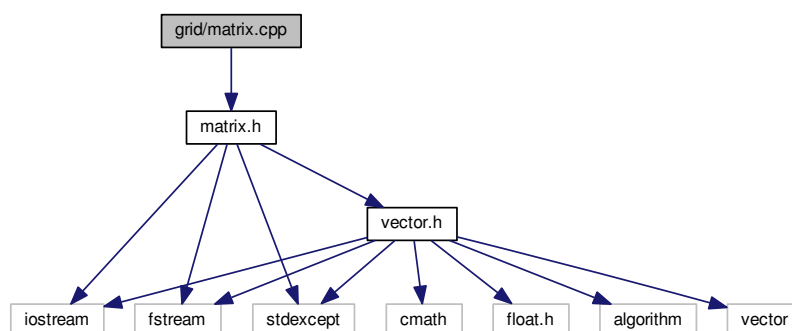
## Chapter 5

# File Documentation

### 5.1 grid/matrix.cpp File Reference

```
#include "matrix.h"
```

Include dependency graph for matrix.cpp:



### Functions

- `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)`

#### 5.1.1 Function Documentation

##### 5.1.1.1 `std::ostream& operator<< ( std::ostream & os, const Matrix & m )`

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)
```

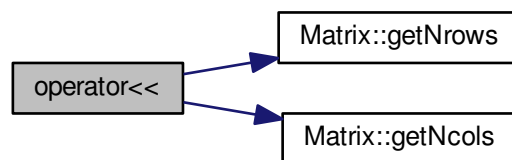
## Returns

std::ostream&. The ostream object

## Parameters

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

Here is the call graph for this function:



#### 5.1.1.2 std::ofstream& operator<< ( std::ofstream & ofs, const Matrix & m )

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");
------------------------------------	---

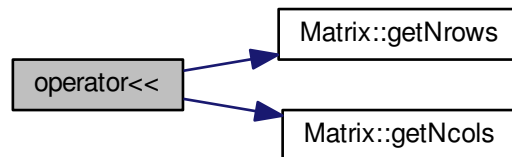
## Returns

std::ofstream&. The ofstream object

## Parameters

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

Here is the call graph for this function:

5.1.1.3 `std::istream& operator>> ( std::istream & is, Matrix & m )`

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");
------------------------------------	--

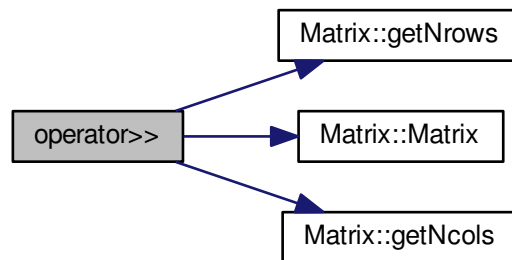
## Returns

`std::istream&`. The istream object

## Parameters

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

Here is the call graph for this function:



#### 5.1.1.4 std::ifstream& operator>> ( std::ifstream & ifs, Matrix & m )

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

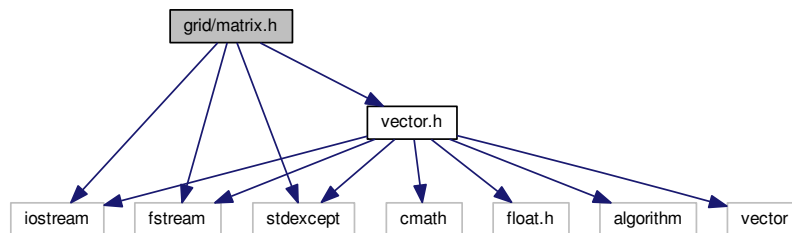
Here is the call graph for this function:



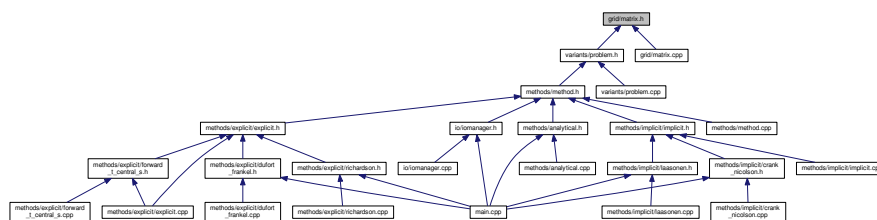


## 5.2 grid/matrix.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdexcept>
#include "vector.h"
Include dependency graph for matrix.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [Matrix](#)

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

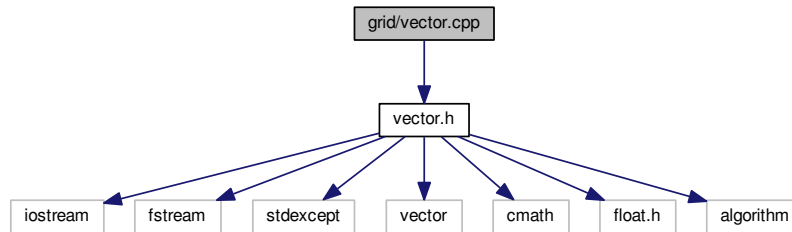
*The implementation is derived from the standard container vector `std::vector`*

*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 `[]`.*

## 5.3 grid/vector.cpp File Reference

```
#include "vector.h"
```

Include dependency graph for vector.cpp:



## Functions

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

### 5.3.1 Function Documentation

#### 5.3.1.1 `std::ostream& operator<< ( std::ostream & os, const Vector & v )`

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

#### 5.3.1.2 `std::ofstream& operator<< ( std::ofstream & ofs, const Vector & v )`

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

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

5.3.1.3 std::istream& operator>> ( std::istream & *is*, Vector & *v* )

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

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

## Parameters

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

Here is the call graph for this function:



#### 5.3.1.4 std::ifstream& operator>> ( std::ifstream & ifs, Vector & v )

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

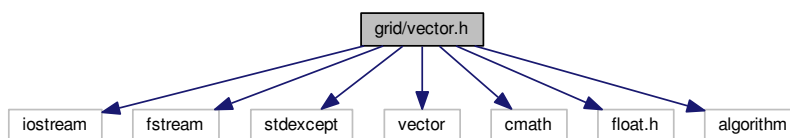
Here is the call graph for this function:



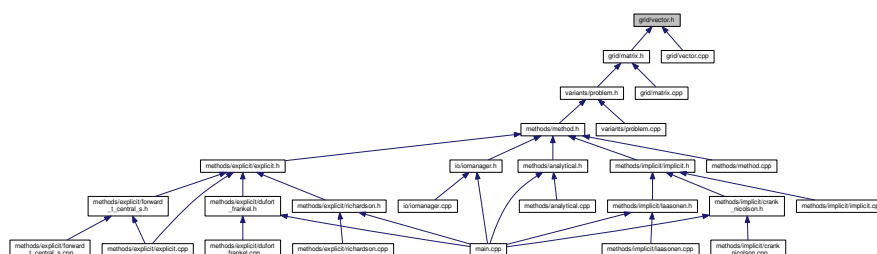
## 5.4 grid/vector.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdexcept>
#include <vector>
#include <cmath>
#include <float.h>
#include <algorithm>
```

This graph shows which files directly or indirectly include this file:



This graph shows which files directly or indirectly include this file:



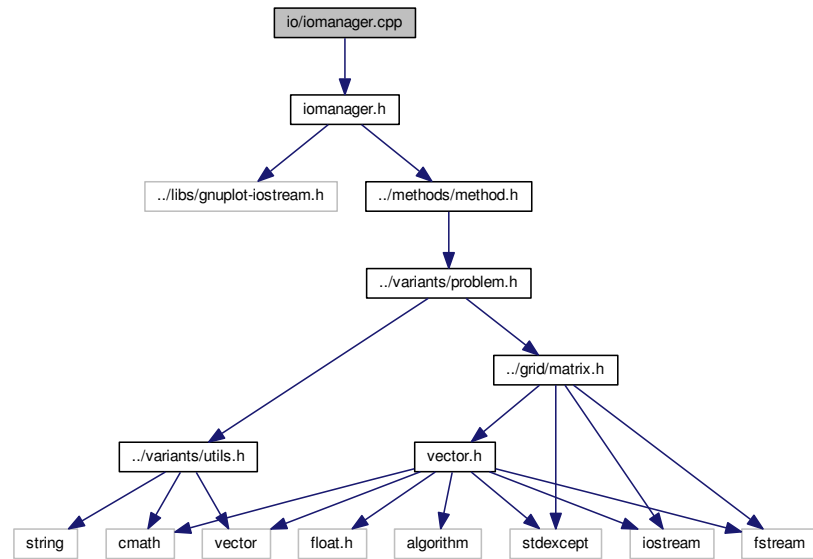
- class **Vector**

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 `usto` expose only those base class functions we wish to use - in this the array access operator `[]`.

## 5.5 io/iomanager.cpp File Reference

```
#include "iomanager.h"
```

Include dependency graph for iomanager.cpp:

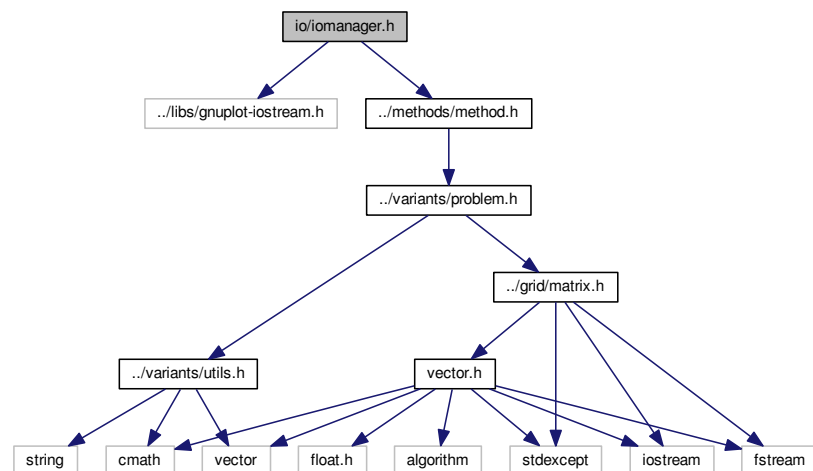


## 5.6 io/iomanager.h File Reference

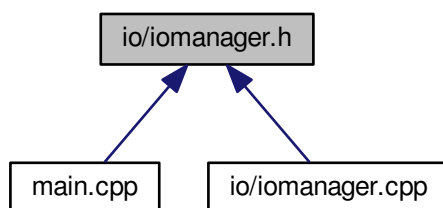
```
#include "../libs/gnuplot-iostream.h"
```

```
#include "../methods/method.h"
```

Include dependency graph for iomanager.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [IOManager](#)

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

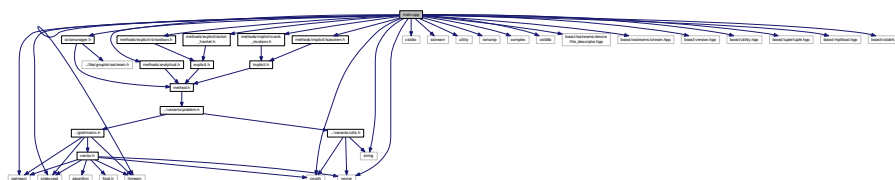
## 5.7 main.cpp File Reference

```

#include <iostream>
#include "methods/analytical.h"
#include "methods/explicit/dufort_frankel.h"
#include "methods/explicit/richardson.h"
#include "methods/implicit/laasonen.h"
#include "methods/implicit/crank_nicolson.h"
#include "io/iomanager.h"

```

Include dependency graph for main.cpp:



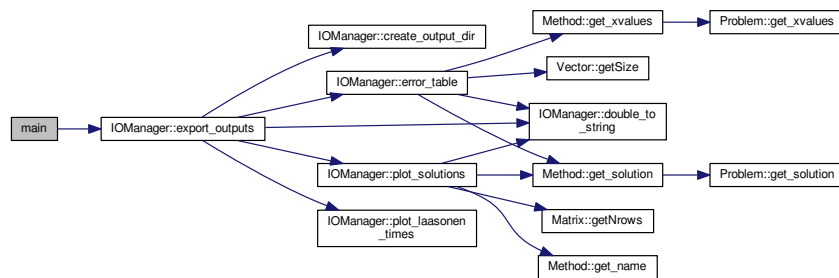
## Functions

- int [main](#) ()

## 5.7.1 Function Documentation

### 5.7.1.1 `int main ( )`

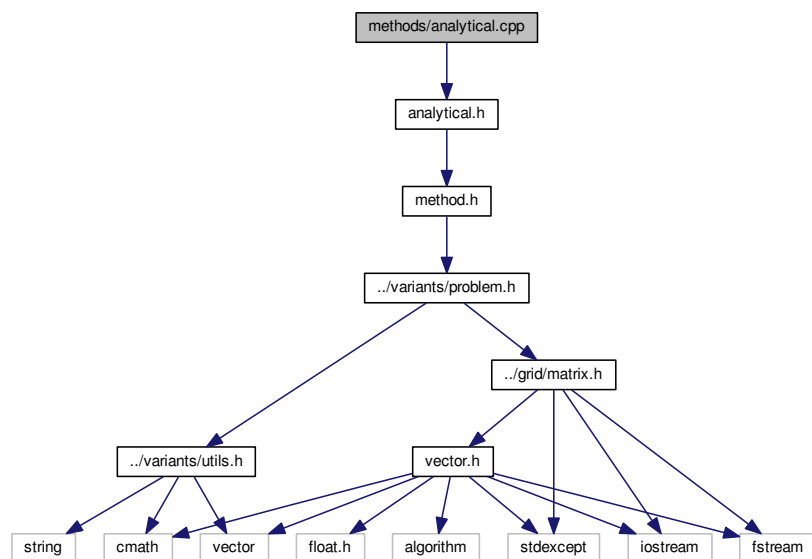
Here is the call graph for this function:



## 5.8 methods/analytical.cpp File Reference

```
#include "analytical.h"
```

Include dependency graph for analytical.cpp:

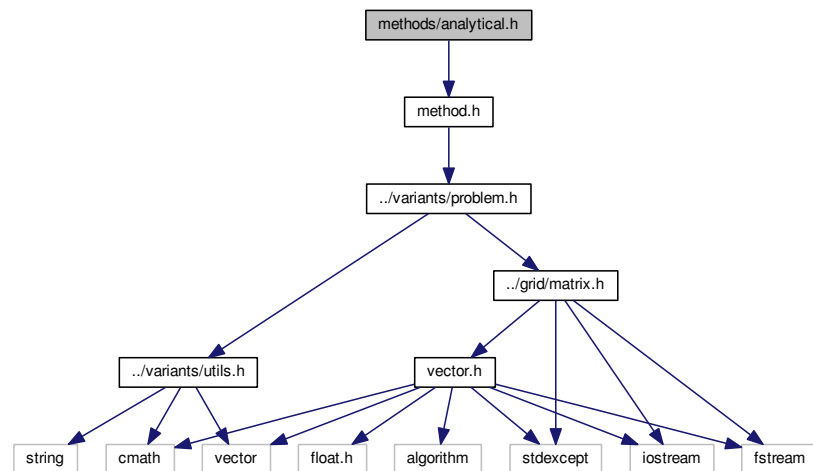




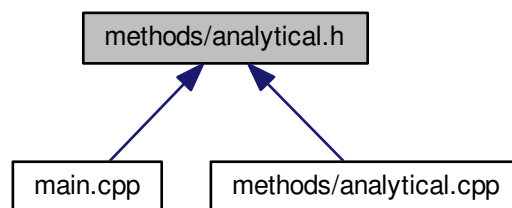
## 5.9 methods/analytical.h File Reference

```
#include "method.h"
```

Include dependency graph for analytical.h:



This graph shows which files directly or indirectly include this file:



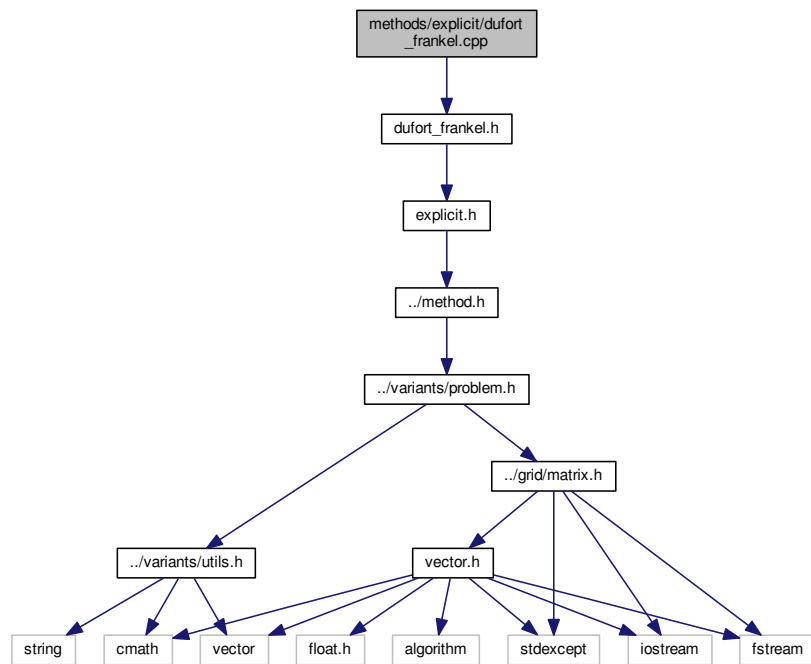
## Classes

- class [Analytical](#)

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

## 5.10 methods/explicit/dufort\_frankel.cpp File Reference

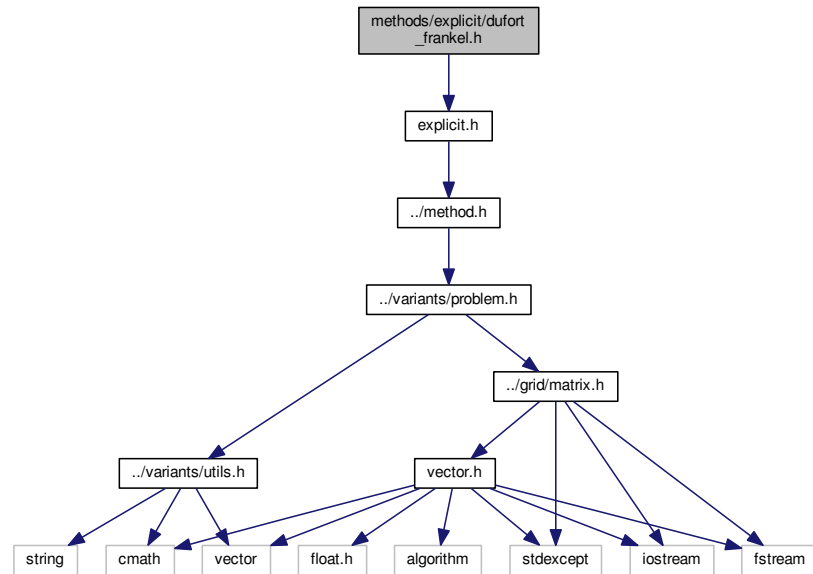
```
#include "dufort_frankel.h"
Include dependency graph for dufort_frankel.cpp:
```



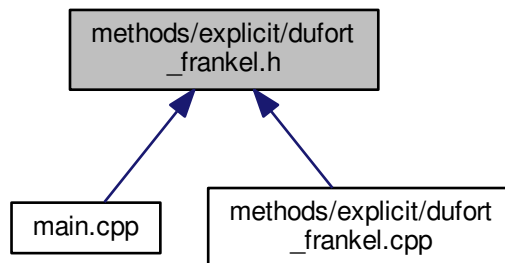
## 5.11 methods/explicit/dufort\_frankel.h File Reference

```
#include "explicit.h"
```

Include dependency graph for dufort\_frankel.h:



This graph shows which files directly or indirectly include this file:



## Classes

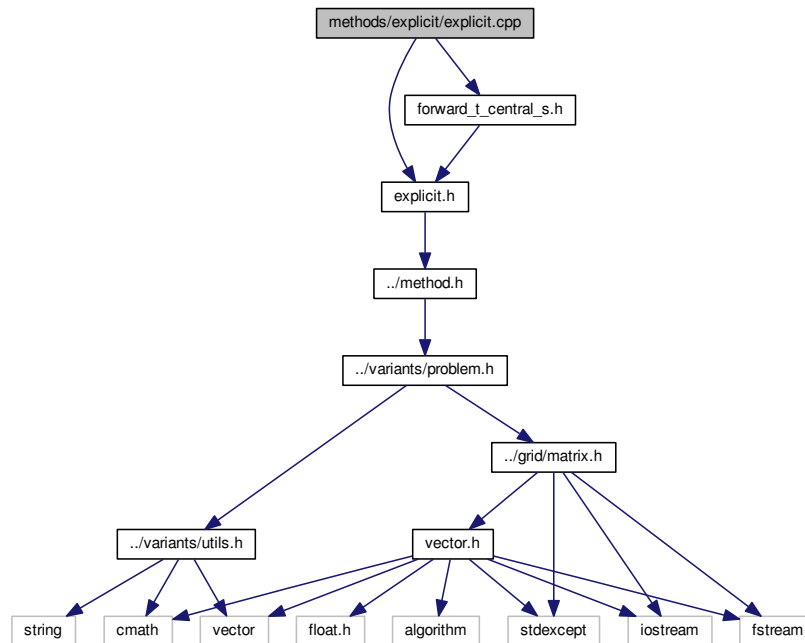
- class [DufortFrankel](#)  
A [DufortFrankel](#) method class that contains an iteration builder.

## 5.12 methods/explicit/explicit.cpp File Reference

```
#include "explicit.h"
```

```
#include "forward_t_central_s.h"
```

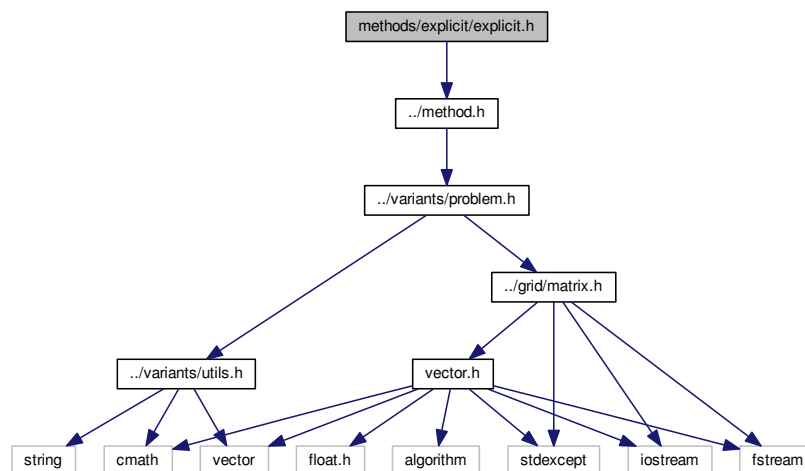
Include dependency graph for explicit.cpp:



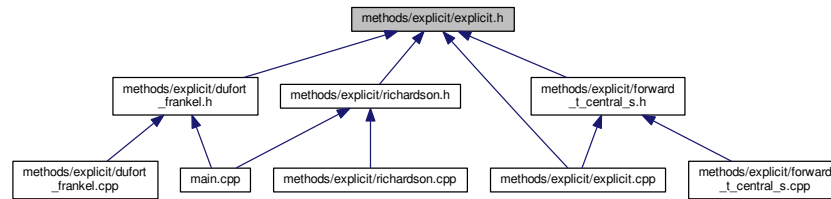
### 5.13 methods/explicit/explicit.h File Reference

```
#include "../method.h"
```

Include dependency graph for explicit.h:



This graph shows which files directly or indirectly include this file:



## Classes

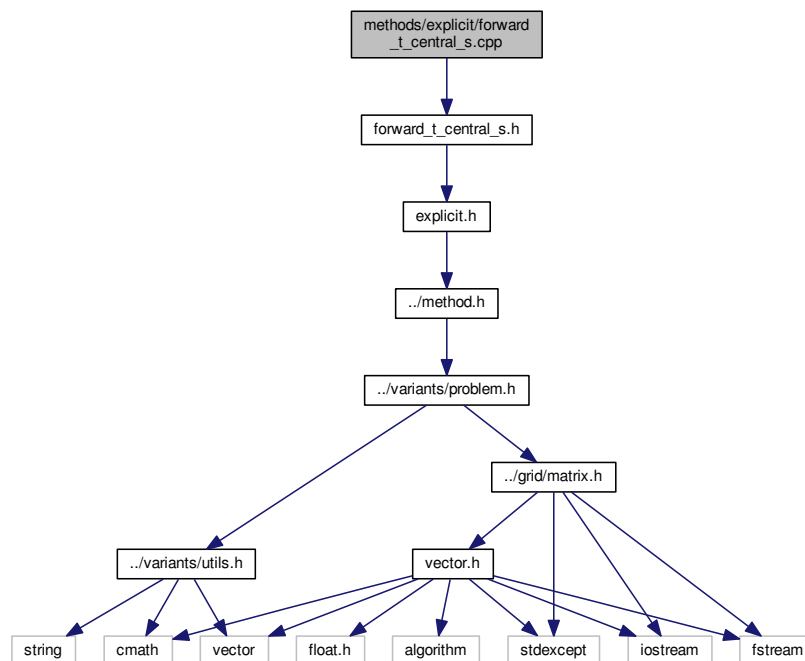
- class [Explicit](#)

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

## 5.14 methods/explicit/forward\_t\_central\_s.cpp File Reference

```
#include "forward_t_central_s.h"
```

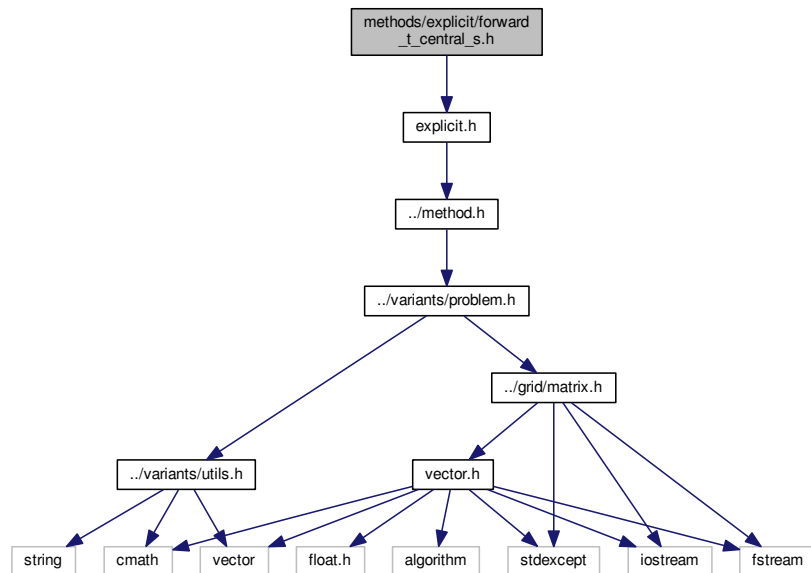
Include dependency graph for forward\_t\_central\_s.cpp:



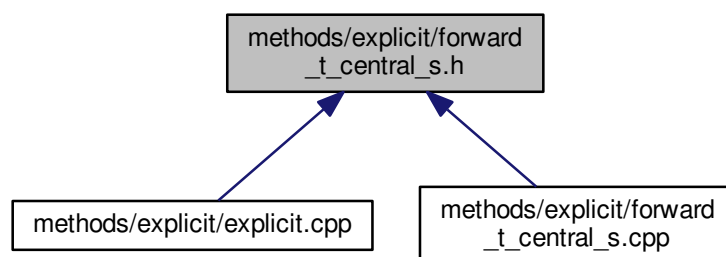
## 5.15 methods/explicit/forward\_t\_central\_s.h File Reference

```
#include "explicit.h"
```

Include dependency graph for forward\_t\_central\_s.h:



This graph shows which files directly or indirectly include this file:



## Classes

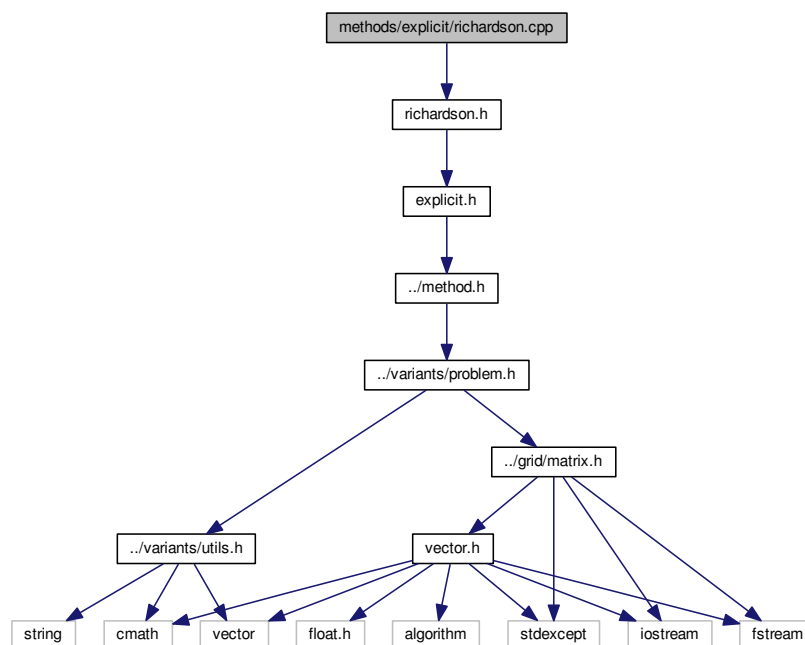
- class [FTCS](#)

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

## 5.16 methods/explicit/richardson.cpp File Reference

```
#include "richardson.h"
```

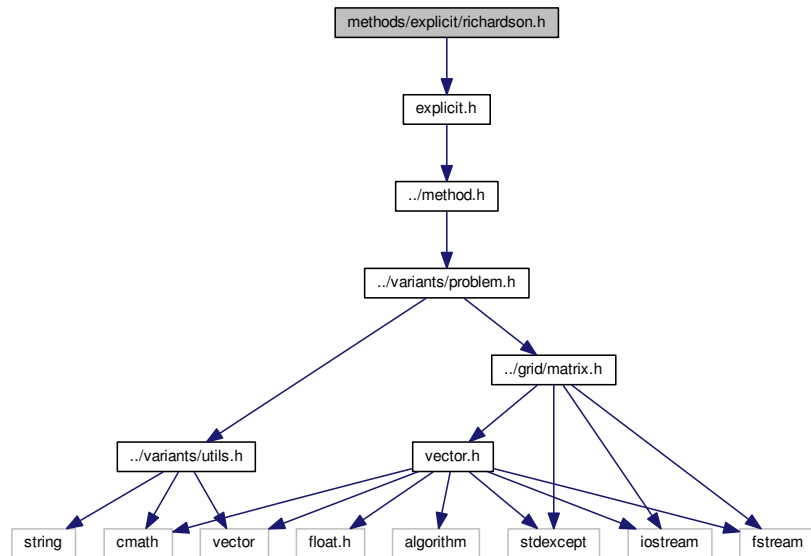
Include dependency graph for richardson.cpp:



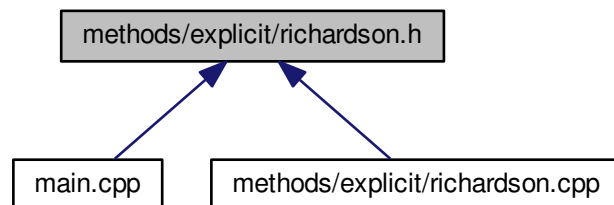
## 5.17 methods/explicit/richardson.h File Reference

```
#include "explicit.h"
```

Include dependency graph for richardson.h:



This graph shows which files directly or indirectly include this file:



## Classes

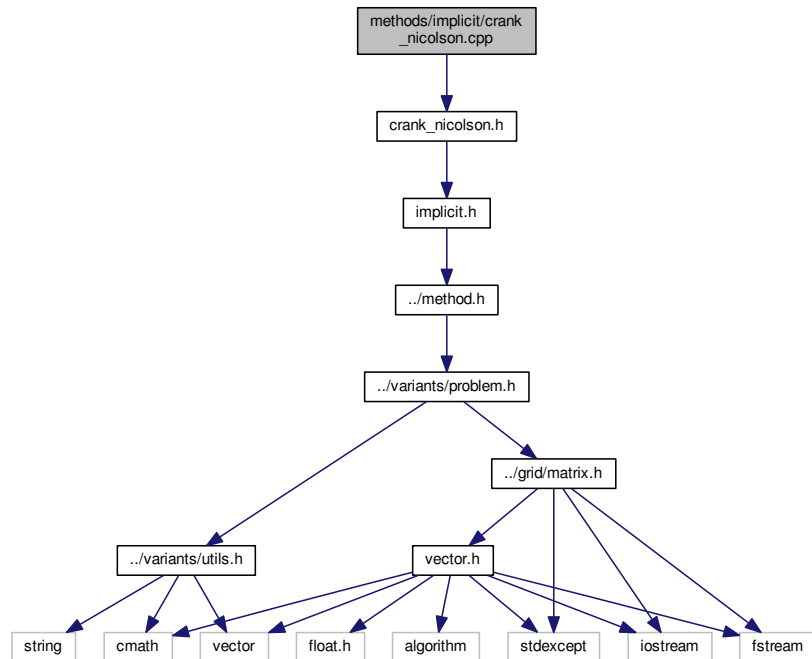
- class [Richardson](#)  
A [Richardson](#) method class that contains an iteration builder.

## 5.18 methods/implicit/crank\_nicolson.cpp File Reference

```
#include "crank_nicolson.h"
```



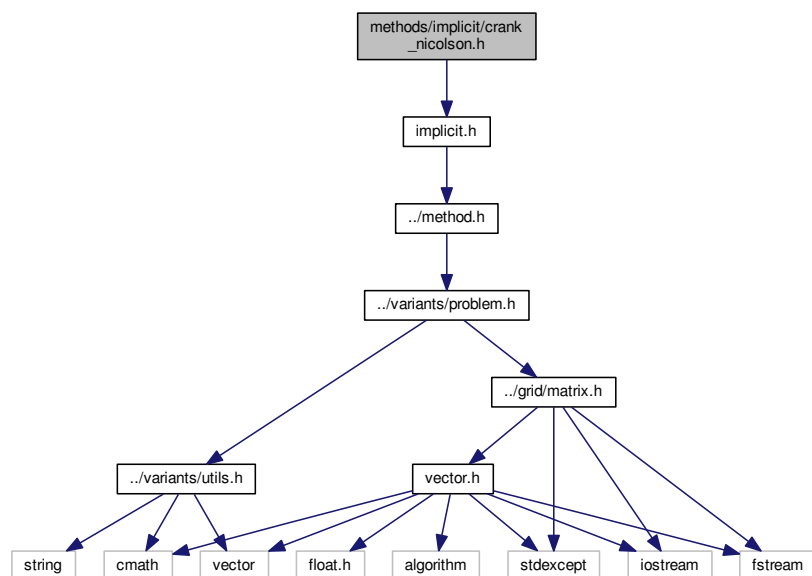
Include dependency graph for krank\_nicolson.cpp:



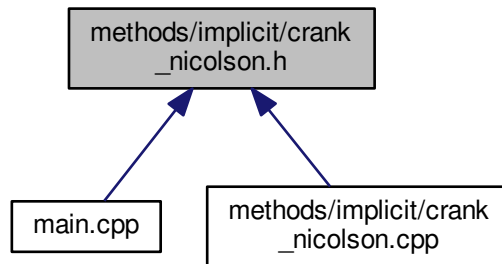
## 5.19 methods/implicit/krank\_nicolson.h File Reference

```
#include "implicit.h"
```

Include dependency graph for krank\_nicolson.h:



This graph shows which files directly or indirectly include this file:



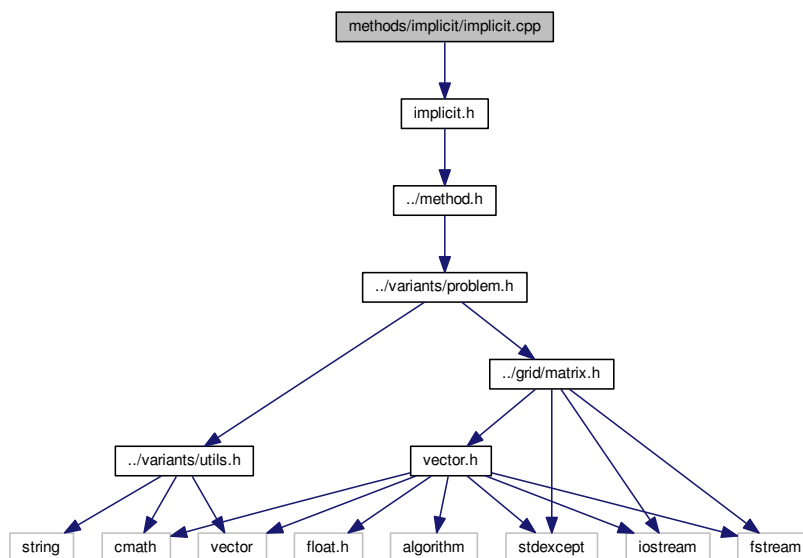
## Classes

- class [CrankNicolson](#)  
A [CrankNicolson](#) method class that contains a *r* vector builder.

## 5.20 methods/implicit/implicit.cpp File Reference

```
#include "implicit.h"
```

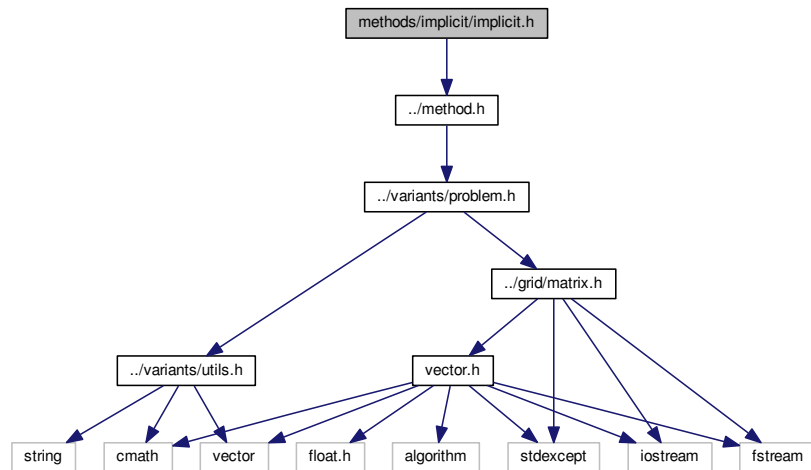
Include dependency graph for `implicit.cpp`:



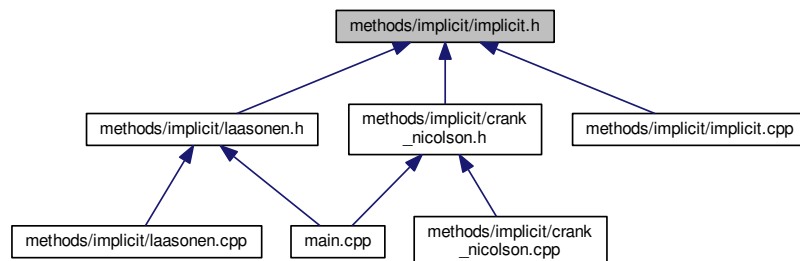
## 5.21 methods/implicit/implicit.h File Reference

```
#include "../method.h"
```

Include dependency graph for implicit.h:



This graph shows which files directly or indirectly include this file:



### Classes

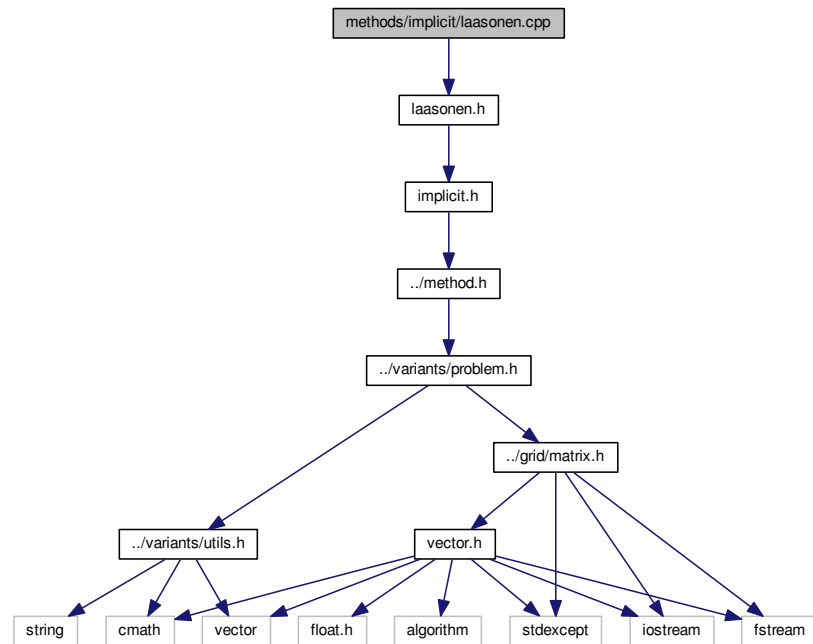
- class [Implicit](#)

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

## 5.22 methods/implicit/laasonen.cpp File Reference

```
#include "laasonen.h"
```

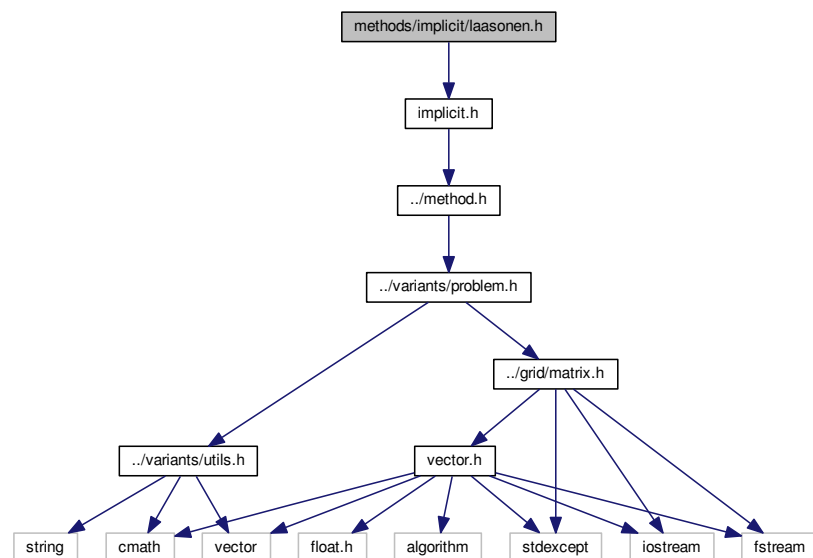
Include dependency graph for laasonen.cpp:



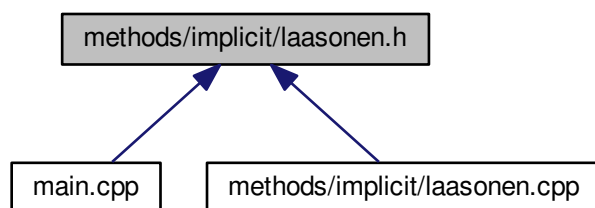
## 5.23 methods/implicit/laasonen.h File Reference

```
#include "implicit.h"
```

Include dependency graph for laasonen.h:



This graph shows which files directly or indirectly include this file:



## Classes

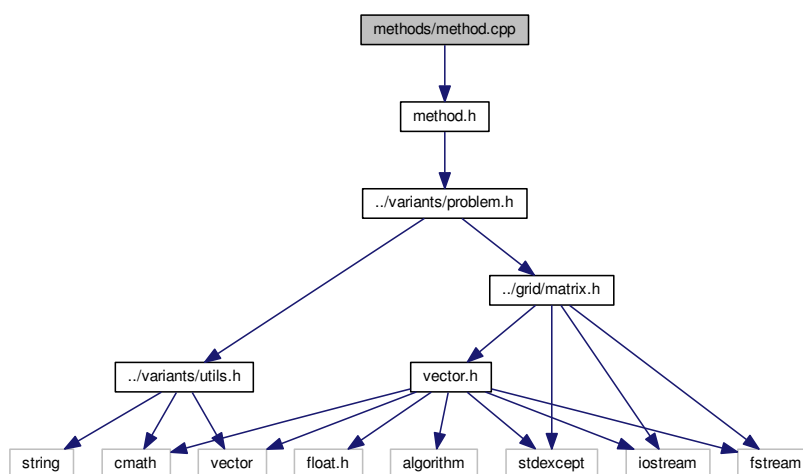
- class [Laasonen](#)

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

## 5.24 methods/method.cpp File Reference

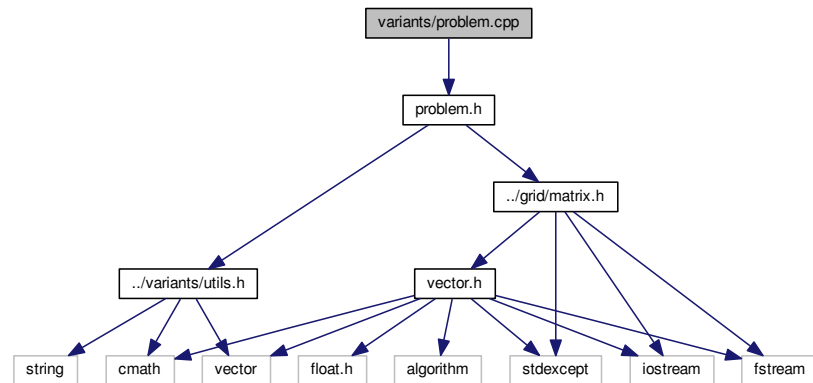
```
#include "method.h"
```

Include dependency graph for `method.cpp`:





Include dependency graph for problem.cpp:

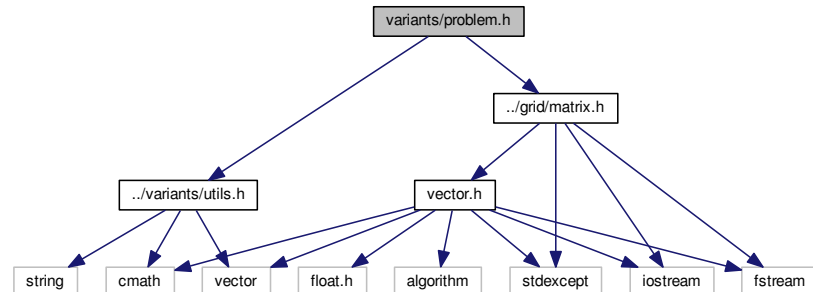


## 5.27 variants/problem.h File Reference

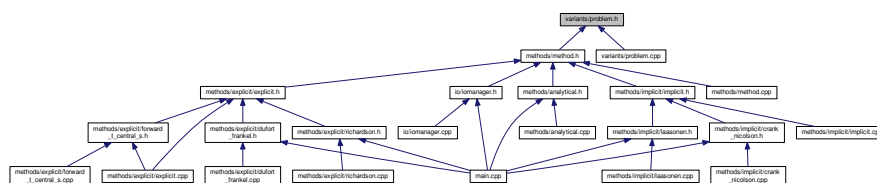
```
#include "../variants/utils.h"
```

```
#include "../grid/matrix.h"
```

Include dependency graph for problem.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class Problem

A **Problem** class to structure relevant information related with the problem.





- const double `NUMBER_TIME_STEPS` = 6.0  
*Macro double.*
- const unsigned int `NUMBER_OF_EXPANSIONS` = 20  
*Macro unsigned int.*
- const double `PI` = `std::atan(1) * 4`  
*Macro double.*
- const std::string `OUTPUT_PATH` = `"../outputs"`  
*Macro string.*
- const std::string `FORWARD_TIME_CENTRAL_SPACE` = `"Forward Time Central Space"`  
*Macro string.*
- const std::string `RICHARDSON` = `"Richardson"`  
*Macro string.*
- const std::string `DUFORT_FRANKEL` = `"DuFort-Frankel"`  
*Macro string.*
- const std::string `LAASONEN` = `"Laasonen"`  
*Macro string.*
- const std::string `CRANK_NICHOLSON` = `"Crank-Nicholson"`  
*Macro string.*

### 5.28.1 Variable Documentation

5.28.1.1 const std::string `CRANK_NICHOLSON` = `"Crank-Nicholson"`

Macro string.

Crank-Nicholson method name.

5.28.1.2 const double `DELTA_T` = 0.01

Macro double.

The default time step.

5.28.1.3 const std::vector<double> `DELTA_T_LASSONEN` = {0.01, 0.025, 0.05, 0.1}

Macro double.

Time steps to study in [Laasonen Implicit](#) Scheme.

5.28.1.4 const double `DELTA_X` = 0.05

Macro double.

The default space step.

5.28.1.5 `const double DIFUSIVITY = 0.1`

Macro double.

The default value of difusivity.

5.28.1.6 `const std::string DUFORT_FRANKEL = "DuFort-Frankel"`

Macro string.

DuFort-Frankel method name.

5.28.1.7 `const std::string FORWARD_TIME_CENTRAL_SPACE = "Forward Time Central Space"`

Macro string.

Forward in Time and Central in Space method name.

5.28.1.8 `const double INITIAL_TEMPERATURE = 100.0`

Macro double.

The default initial temperature.

5.28.1.9 `const std::string LAASONEN = "Laasonen"`

Macro string.

[Laasonen](#) method name.

5.28.1.10 `const unsigned int NUMBER_OF_EXPANSIONS = 20`

Macro unsigned int.

Number of expansions to calculate the analytical solution sum expansion.

5.28.1.11 `const double NUMBER_TIME_STEPS = 6.0`

Macro double.

The default limit of time steps. 0, 0.1, 0.2, 0.3, 0.4, 0.5

5.28.1.12 `const std::string OUTPUT_PATH = "../outputs"`

Macro string.

Default outputs path.

5.28.1.13 `const double PI = std::atan(1) * 4`

Macro double.

Approximated value of PI.

5.28.1.14 `const std::string RICHARDSON = "Richardson"`

Macro string.

[Richardson](#) method name.

5.28.1.15 `const double SURFACE_TEMPERATURE = 300.0`

Macro double.

The default surface temperature.

5.28.1.16 `const double THICKNESS = 1.0`

Macro double.

The default value of thickness.

5.28.1.17 `const double TIMELIMIT = 0.5`

Macro double.

The default value of time limit.



# Index

Analytical, [7](#)  
    Analytical, [9](#)  
    compute\_solution, [9](#)  
    nr\_of\_expansions, [10](#)

build\_iteration  
    DufortFrankel, [17](#)  
    Explicit, [21](#)  
    FTCS, [25](#)  
    Richardson, [78](#)

build\_r  
    CrankNicolson, [13](#)  
    Implicit, [29](#)  
    Laasonen, [40](#)

CRANK\_NICHOLSON  
    utils.h, [119](#)

computational\_time  
    Method, [64](#)

compute  
    Method, [61](#)

compute\_solution  
    Analytical, [9](#)  
    Explicit, [22](#)  
    Implicit, [30](#)  
    Method, [61](#)

CrankNicolson, [10](#)  
    build\_r, [13](#)  
    CrankNicolson, [13](#)

create\_output\_dir  
    IOManager, [34](#)

DELTA\_T\_LASSONEN  
    utils.h, [119](#)

DELTA\_T  
    utils.h, [119](#)

DELTA\_X  
    utils.h, [119](#)

DIFUSIVITY  
    utils.h, [119](#)

DUFORT\_FRANKEL  
    utils.h, [120](#)

default\_deltat\_times  
    IOManager, [37](#)

delta\_t  
    Problem, [74](#)

delta\_x  
    Problem, [74](#)

double\_to\_string  
    IOManager, [34](#)

DufortFrankel, [14](#)  
    build\_iteration, [17](#)  
    DufortFrankel, [17](#)

error\_table  
    IOManager, [34](#)  
Explicit, [18](#)  
    build\_iteration, [21](#)  
    compute\_solution, [22](#)  
    Explicit, [21](#)

export\_outputs  
    IOManager, [35](#)

FORWARD\_TIME\_CENTRAL\_SPACE  
    utils.h, [120](#)

FTCS, [23](#)  
    build\_iteration, [25](#)  
    FTCS, [25](#)

find  
    Vector, [84](#)

get\_computational\_time  
    Method, [61](#)

get\_deltat  
    Method, [62](#)  
    Problem, [68](#)

get\_deltax  
    Problem, [68](#)

get\_first\_row  
    Problem, [69](#)

get\_name  
    Method, [62](#)

get\_solution  
    Method, [62](#)  
    Problem, [69](#)

get\_tsize  
    Problem, [70](#)

get\_tvalues  
    Problem, [70](#)

get\_xsize  
    Problem, [71](#)

get\_xvalues  
    Method, [63](#)  
    Problem, [71](#)

getNcols  
    Matrix, [46](#)

getNrows  
    Matrix, [47](#)

getSize  
    Vector, [84](#)

- grid/matrix.cpp, 91
- grid/matrix.h, 95
- grid/vector.cpp, 95
- grid/vector.h, 98
- INITIAL\_TEMPERATURE
  - utils.h, 120
- IOManager, 32
  - create\_output\_dir, 34
  - default\_deltat\_times, 37
  - double\_to\_string, 34
  - error\_table, 34
  - export\_outputs, 35
  - IOManager, 33
  - laasonen\_times, 37
  - output\_path, 37
  - plot\_laasonen\_times, 36
  - plot\_solutions, 36
- Implicit, 26
  - build\_r, 29
  - compute\_solution, 30
  - Implicit, 29
  - thomas\_algorithm, 30
- io/iomanager.cpp, 99
- io/iomanager.h, 100
- LAASONEN
  - utils.h, 120
- Laasonen, 38
  - build\_r, 40
  - Laasonen, 40
- laasonen\_times
  - IOManager, 37
- main
  - main.cpp, 102
- main.cpp, 101
  - main, 102
- Matrix, 41
  - getNcols, 46
  - getNrows, 47
  - Matrix, 45, 46
  - mult, 48
  - one\_norm, 48
  - operator<<, 55
  - operator>>, 56, 57
  - operator\*, 49, 50
  - operator=, 51
  - operator==, 51
  - set\_row, 52
  - transpose, 53
  - two\_norm, 53
  - uniform\_norm, 54
  - vec, 45
- matrix.cpp
  - operator<<, 91, 92
  - operator>>, 93, 94
- Method, 57
  - computational\_time, 64
  - compute, 61
  - compute\_solution, 61
  - get\_computational\_time, 61
  - get\_deltat, 62
  - get\_name, 62
  - get\_solution, 62
  - get\_xvalues, 63
  - Method, 60
  - name, 64
  - problem, 64
  - q, 64
- methods/analytical.cpp, 102
- methods/analytical.h, 103
- methods/explicit/dufort\_frankel.cpp, 104
- methods/explicit/dufort\_frankel.h, 104
- methods/explicit/explicit.cpp, 105
- methods/explicit/explicit.h, 106
- methods/explicit/forward\_t\_central\_s.cpp, 107
- methods/explicit/forward\_t\_central\_s.h, 108
- methods/explicit/richardson.cpp, 109
- methods/explicit/richardson.h, 109
- methods/implicit/crank\_nicolson.cpp, 110
- methods/implicit/crank\_nicolson.h, 111
- methods/implicit/implicit.cpp, 112
- methods/implicit/implicit.h, 113
- methods/implicit/laasonen.cpp, 113
- methods/implicit/laasonen.h, 114
- methods/method.cpp, 115
- methods/method.h, 116
- mult
  - Matrix, 48
- NUMBER\_OF\_EXPANSIONS
  - utils.h, 120
- NUMBER\_TIME\_STEPS
  - utils.h, 120
- name
  - Method, 64
- nr\_of\_expansions
  - Analytical, 10
- OUTPUT\_PATH
  - utils.h, 120
- one\_norm
  - Matrix, 48
  - Vector, 85
- operator<<
  - Matrix, 55
  - matrix.cpp, 91, 92
  - Vector, 88
  - vector.cpp, 96
- operator>>
  - Matrix, 56, 57
  - matrix.cpp, 93, 94
  - Vector, 89
  - vector.cpp, 97, 98
- operator\*
  - Matrix, 49, 50
- operator=

- Matrix, [51](#)
- Vector, [85](#)
- operator==
  - Matrix, [51](#)
  - Vector, [86](#)
- output\_path
  - IOManager, [37](#)
- PI
  - utils.h, [120](#)
- plot\_laasonen\_times
  - IOManager, [36](#)
- plot\_solutions
  - IOManager, [36](#)
- Problem, [65](#)
  - delta\_t, [74](#)
  - delta\_x, [74](#)
  - get\_deltat, [68](#)
  - get\_deltax, [68](#)
  - get\_first\_row, [69](#)
  - get\_solution, [69](#)
  - get\_tsize, [70](#)
  - get\_tvalues, [70](#)
  - get\_xsize, [71](#)
  - get\_xvalues, [71](#)
  - Problem, [67](#)
  - set\_initial\_conditions, [72](#)
  - set\_t\_values, [72](#)
  - set\_time\_step, [72](#)
  - set\_x\_values, [73](#)
  - solution, [74](#)
  - t\_size, [74](#)
  - t\_values, [74](#)
  - x\_size, [74](#)
  - x\_values, [75](#)
- problem
  - Method, [64](#)
- push
  - Vector, [86](#)
- push\_front\_back
  - Vector, [87](#)
- q
  - Method, [64](#)
- RICHARDSON
  - utils.h, [121](#)
- Richardson, [75](#)
  - build\_iteration, [78](#)
  - Richardson, [78](#)
- SURFACE\_TEMPERATURE
  - utils.h, [121](#)
- set\_initial\_conditions
  - Problem, [72](#)
- set\_row
  - Matrix, [52](#)
- set\_t\_values
  - Problem, [72](#)
- set\_time\_step
  - Problem, [72](#)
- set\_x\_values
  - Problem, [73](#)
- solution
  - Problem, [74](#)
- t\_size
  - Problem, [74](#)
- t\_values
  - Problem, [74](#)
- THICKNESS
  - utils.h, [121](#)
- TIMELIMIT
  - utils.h, [121](#)
- thomas\_algorithm
  - Implicit, [30](#)
- transpose
  - Matrix, [53](#)
- two\_norm
  - Matrix, [53](#)
  - Vector, [87](#)
- uniform\_norm
  - Matrix, [54](#)
  - Vector, [87](#)
- utils.h
  - CRANK\_NICHOLSON, [119](#)
  - DELTA\_T\_LASSONEN, [119](#)
  - DELTA\_T, [119](#)
  - DELTA\_X, [119](#)
  - DIFUSIVITY, [119](#)
  - DUFORT\_FRANKEL, [120](#)
  - FORWARD\_TIME\_CENTRAL\_SPACE, [120](#)
  - INITIAL\_TEMPERATURE, [120](#)
  - LAASONEN, [120](#)
  - NUMBER\_OF\_EXPANSIONS, [120](#)
  - NUMBER\_TIME\_STEPS, [120](#)
  - OUTPUT\_PATH, [120](#)
  - PI, [120](#)
  - RICHARDSON, [121](#)
  - SURFACE\_TEMPERATURE, [121](#)
  - THICKNESS, [121](#)
  - TIMELIMIT, [121](#)
- variants/problem.cpp, [116](#)
- variants/problem.h, [117](#)
- variants/utils.h, [118](#)
- vec
  - Matrix, [45](#)
  - Vector, [82](#)
- Vector, [79](#)
  - find, [84](#)
  - getSize, [84](#)
  - one\_norm, [85](#)
  - operator<<, [88](#)
  - operator>>, [89](#)
  - operator=, [85](#)
  - operator==, [86](#)

- push, [86](#)
- push\_front\_back, [87](#)
- two\_norm, [87](#)
- uniform\_norm, [87](#)
- vec, [82](#)
- Vector, [82](#), [83](#)
- vector.cpp
  - operator<<, [96](#)
  - operator>>, [97](#), [98](#)
- x\_size
  - Problem, [74](#)
- x\_values
  - Problem, [75](#)