

Heat conduction equation

Generated by Doxygen 1.8.11

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	7
4.1	Analytical Class Reference	7
4.1.1	Detailed Description	10
4.1.2	Constructor & Destructor Documentation	10
4.1.2.1	Analytical(Problem problem)	10
4.1.3	Member Function Documentation	10
4.1.3.1	compute_solution()	10
4.1.4	Member Data Documentation	11
4.1.4.1	nr_of_expansions	11
4.2	CrankNicolson Class Reference	11
4.2.1	Detailed Description	14
4.2.2	Constructor & Destructor Documentation	14
4.2.2.1	CrankNicolson(Problem problem)	14
4.2.3	Member Function Documentation	14
4.2.3.1	build_r(Vector previous_step)	14
4.3	DufortFrankel Class Reference	15

4.3.1	Detailed Description	18
4.3.2	Constructor & Destructor Documentation	18
4.3.2.1	DufortFrankel(Problem problem)	18
4.3.3	Member Function Documentation	18
4.3.3.1	build_iteration(Vector current_step, Vector previous_step)	18
4.4	Explicit Class Reference	19
4.4.1	Detailed Description	22
4.4.2	Constructor & Destructor Documentation	22
4.4.2.1	Explicit(Problem problem)	22
4.4.3	Member Function Documentation	22
4.4.3.1	build_iteration(Vector current_step, Vector previous_step)=0	22
4.4.3.2	compute_solution()	23
4.5	FTCS Class Reference	24
4.5.1	Detailed Description	26
4.5.2	Constructor & Destructor Documentation	26
4.5.2.1	FTCS(Problem problem)	26
4.5.3	Member Function Documentation	26
4.5.3.1	build_iteration(Vector current_step, Vector previous_step)	26
4.6	Implicit Class Reference	27
4.6.1	Detailed Description	30
4.6.2	Constructor & Destructor Documentation	30
4.6.2.1	Implicit(Problem problem)	30
4.6.3	Member Function Documentation	30
4.6.3.1	build_r(Vector previous_step)=0	30
4.6.3.2	compute_solution()	31
4.6.3.3	thomas_algorithm(Vector r, double a, double b, double c)	32
4.7	IOManager Class Reference	33
4.7.1	Detailed Description	34
4.7.2	Constructor & Destructor Documentation	34
4.7.2.1	IOManager()	34

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_tables(std::string output_name, std::vector< Method * > method)	35
4.7.3.4	export_outputs(Method *analytical, std::vector< Method * > methods)	35
4.7.3.5	plot_default_deltat_times()	36
4.7.3.6	plot_laasonen_times()	36
4.7.3.7	plot_solutions(std::string output_name, Method *analytical, Method *method)	37
4.7.4	Member Data Documentation	37
4.7.4.1	default_deltat_times	38
4.7.4.2	laasonen_times	38
4.7.4.3	output_path	38
4.8	Laasonen Class Reference	38
4.8.1	Detailed Description	41
4.8.2	Constructor & Destructor Documentation	41
4.8.2.1	Laasonen(Problem problem)	41
4.8.3	Member Function Documentation	41
4.8.3.1	build_r(Vector previous_step)	41
4.9	Matrix Class Reference	42
4.9.1	Detailed Description	45
4.9.2	Member Typedef Documentation	46
4.9.2.1	vec	46
4.9.3	Constructor & Destructor Documentation	46
4.9.3.1	Matrix()	46
4.9.3.2	Matrix(int Nrows, int Ncols)	46
4.9.3.3	Matrix(const Matrix &m)	47
4.9.4	Member Function Documentation	47
4.9.4.1	getNcols() const	47
4.9.4.2	getNrows() const	48
4.9.4.3	mult(const Matrix &a) const	49

4.9.4.4	<code>one_norm() const</code>	49
4.9.4.5	<code>operator*(const Matrix &a) const</code>	50
4.9.4.6	<code>operator*(const Vector &v) const</code>	51
4.9.4.7	<code>operator=(const Matrix &m)</code>	52
4.9.4.8	<code>operator==(const Matrix &m) const</code>	52
4.9.4.9	<code>set_row(int index, Vector v)</code>	53
4.9.4.10	<code>transpose() const</code>	54
4.9.4.11	<code>two_norm() const</code>	55
4.9.4.12	<code>uniform_norm() const</code>	55
4.9.5	Friends And Related Function Documentation	56
4.9.5.1	<code>operator<<</code>	56
4.9.5.2	<code>operator<<</code>	57
4.9.5.3	<code>operator>></code>	57
4.9.5.4	<code>operator>></code>	58
4.10	Method Class Reference	58
4.10.1	Detailed Description	61
4.10.2	Constructor & Destructor Documentation	62
4.10.2.1	<code>Method()</code>	62
4.10.2.2	<code>Method(Problem problem)</code>	62
4.10.3	Member Function Documentation	62
4.10.3.1	<code>compute()</code>	62
4.10.3.2	<code>compute_norms(Matrix analytical_matrix)</code>	63
4.10.3.3	<code>compute_solution()=0</code>	63
4.10.3.4	<code>get_computational_time()</code>	63
4.10.3.5	<code>get_deltat()</code>	64
4.10.3.6	<code>get_name()</code>	64
4.10.3.7	<code>get_solution()</code>	64
4.10.3.8	<code>get_two_norm()</code>	65
4.10.3.9	<code>get_xvalues()</code>	65
4.10.4	Member Data Documentation	66

4.10.4.1	computational_time	66
4.10.4.2	name	66
4.10.4.3	one_norm	66
4.10.4.4	problem	66
4.10.4.5	q	66
4.10.4.6	two_norm	66
4.10.4.7	uniform_norm	66
4.11	Problem Class Reference	67
4.11.1	Detailed Description	69
4.11.2	Constructor & Destructor Documentation	69
4.11.2.1	Problem()	69
4.11.2.2	Problem(double dt, double dx)	69
4.11.3	Member Function Documentation	70
4.11.3.1	get_deltat()	70
4.11.3.2	get_deltax()	71
4.11.3.3	get_first_row()	71
4.11.3.4	get_solution()	72
4.11.3.5	get_tsize()	72
4.11.3.6	get_tvalues()	72
4.11.3.7	get_xsize()	73
4.11.3.8	get_xvalues()	73
4.11.3.9	set_initial_conditions()	74
4.11.3.10	set_t_values()	74
4.11.3.11	set_time_step(Vector step, double time)	75
4.11.3.12	set_x_values()	76
4.11.4	Member Data Documentation	76
4.11.4.1	delta_t	76
4.11.4.2	delta_x	76
4.11.4.3	solution	76
4.11.4.4	t_size	76

4.11.4.5	<code>t_values</code>	77
4.11.4.6	<code>x_size</code>	77
4.11.4.7	<code>x_values</code>	77
4.12	Richardson Class Reference	77
4.12.1	Detailed Description	80
4.12.2	Constructor & Destructor Documentation	80
4.12.2.1	<code>Richardson(Problem problem)</code>	80
4.12.3	Member Function Documentation	80
4.12.3.1	<code>build_iteration(Vector current_step, Vector previous_step)</code>	80
4.13	Vector Class Reference	81
4.13.1	Detailed Description	84
4.13.2	Member Typedef Documentation	84
4.13.2.1	<code>vec</code>	84
4.13.3	Constructor & Destructor Documentation	84
4.13.3.1	<code>Vector()</code>	84
4.13.3.2	<code>Vector(int Num)</code>	85
4.13.3.3	<code>Vector(const Vector &v)</code>	85
4.13.3.4	<code>Vector(std::vector< double > vec)</code>	86
4.13.4	Member Function Documentation	86
4.13.4.1	<code>find(double value)</code>	86
4.13.4.2	<code>getSize() const</code>	86
4.13.4.3	<code>one_norm() const</code>	87
4.13.4.4	<code>operator=(const Vector &v)</code>	87
4.13.4.5	<code>operator==(const Vector &v) const</code>	88
4.13.4.6	<code>push(double value)</code>	88
4.13.4.7	<code>push_front_back(double value)</code>	89
4.13.4.8	<code>two_norm() const</code>	89
4.13.4.9	<code>uniform_norm() const</code>	89
4.13.5	Friends And Related Function Documentation	90
4.13.5.1	<code>operator<<</code>	90
4.13.5.2	<code>operator<<</code>	90
4.13.5.3	<code>operator>></code>	91
4.13.5.4	<code>operator>></code>	91

5 File Documentation	93
5.1 grid/matrix.cpp File Reference	93
5.1.1 Function Documentation	93
5.1.1.1 operator<<(std::ostream &os, const Matrix &m)	93
5.1.1.2 operator<<(std::ofstream &ofs, const Matrix &m)	94
5.1.1.3 operator>>(std::istream &is, Matrix &m)	95
5.1.1.4 operator>>(std::ifstream &ifis, Matrix &m)	96
5.2 grid/matrix.h File Reference	97
5.3 grid/vector.cpp File Reference	97
5.3.1 Function Documentation	98
5.3.1.1 operator<<(std::ostream &os, const Vector &v)	98
5.3.1.2 operator<<(std::ofstream &ofs, const Vector &v)	98
5.3.1.3 operator>>(std::istream &is, Vector &v)	99
5.3.1.4 operator>>(std::ifstream &ifis, Vector &v)	100
5.4 grid/vector.h File Reference	100
5.5 io/iomanager.cpp File Reference	101
5.6 io/iomanager.h File Reference	102
5.7 main.cpp File Reference	103
5.7.1 Function Documentation	104
5.7.1.1 main()	104
5.8 methods/analytical.cpp File Reference	104
5.9 methods/analytical.h File Reference	105
5.10 methods/explicit/dufort_frankel.cpp File Reference	106
5.11 methods/explicit/dufort_frankel.h File Reference	106
5.12 methods/explicit/explicit.cpp File Reference	107
5.13 methods/explicit/explicit.h File Reference	108
5.14 methods/explicit/forward_t_central_s.cpp File Reference	109
5.15 methods/explicit/forward_t_central_s.h File Reference	110
5.16 methods/explicit/richardson.cpp File Reference	111
5.17 methods/explicit/richardson.h File Reference	111

5.18	methods/implicit/crank_nicolson.cpp File Reference	112
5.19	methods/implicit/crank_nicolson.h File Reference	113
5.20	methods/implicit/implicit.cpp File Reference	114
5.21	methods/implicit/implicit.h File Reference	115
5.22	methods/implicit/laasonen.cpp File Reference	115
5.23	methods/implicit/laasonen.h File Reference	116
5.24	methods/method.cpp File Reference	117
5.25	methods/method.h File Reference	118
5.26	variants/problem.cpp File Reference	118
5.27	variants/problem.h File Reference	119
5.28	variants/utls.h File Reference	120
5.28.1	Variable Documentation	121
5.28.1.1	ANALYTICAL	121
5.28.1.2	CRANK_NICHOLSON	121
5.28.1.3	DELTA_T	121
5.28.1.4	DELTA_T_LASSONEN	121
5.28.1.5	DELTA_X	122
5.28.1.6	DIFUSIVITY	122
5.28.1.7	DUFORT_FRANKEL	122
5.28.1.8	FORWARD_TIME_CENTRAL_SPACE	122
5.28.1.9	INITIAL_TEMPERATURE	122
5.28.1.10	LAASONEN	122
5.28.1.11	NUMBER_OF_EXPANSIONS	122
5.28.1.12	NUMBER_TIME_STEPS	122
5.28.1.13	OUTPUT_PATH	123
5.28.1.14	PI	123
5.28.1.15	RICHARDSON	123
5.28.1.16	SURFACE_TEMPERATURE	123
5.28.1.17	THICKNESS	123
5.28.1.18	TIMELIMIT	123

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

IOManager	33
Method	58
Analytical	7
Explicit	19
DufortFrankel	15
FTCS	24
Richardson	77
Implicit	27
CrankNicolson	11
Laasonen	38
Problem	67
vector	
Matrix	42
Vector	81

Chapter 2

Class Index

2.1 Class List

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

Analytical	An Analytical class to compute the solution with standard procedures The implementation is derived from the Method Object	7
CrankNicolson	A CrankNicolson method class that contains a r vector builder	11
DufortFrankel	A DufortFrankel method class that contains an iteration builder	15
Explicit	An explicit method class that contains default methods that only explicit methods use The implementation is derived from the Method class	19
FTCS	A FTCS method class that contains an iteration builder	24
Implicit	An implicit method class that contains default methods that only implicit methods use The implementation is derived from the Method class	27
IOManager	An input/output manager class to handle plot exportations and future implementations of input handling	33
Laasonen	A Laasonen method class that contains a r vector builder	38
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 usto expose only those base class functions we wish to use - in this the array access operator []	42
Method	A Method class to structure information used to solve the problem	58
Problem	A Problem class to structure relevant information related with the problem	67
Richardson	A Richardson method class that contains an iteration builder	77
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 []	81

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

main.cpp	103
grid/matrix.cpp	93
grid/matrix.h	97
grid/vector.cpp	97
grid/vector.h	100
io/iomanager.cpp	101
io/iomanager.h	102
methods/analytical.cpp	104
methods/analytical.h	105
methods/method.cpp	117
methods/method.h	118
methods/explicit/dufort_frankel.cpp	106
methods/explicit/dufort_frankel.h	106
methods/explicit/explicit.cpp	107
methods/explicit/explicit.h	108
methods/explicit/forward_t_central_s.cpp	109
methods/explicit/forward_t_central_s.h	110
methods/explicit/richardson.cpp	111
methods/explicit/richardson.h	111
methods/implicit/crank_nicolson.cpp	112
methods/implicit/crank_nicolson.h	113
methods/implicit/implicit.cpp	114
methods/implicit/implicit.h	115
methods/implicit/laasonen.cpp	115
methods/implicit/laasonen.h	116
variants/problem.cpp	118
variants/problem.h	119
variants/utils.h	120

Chapter 4

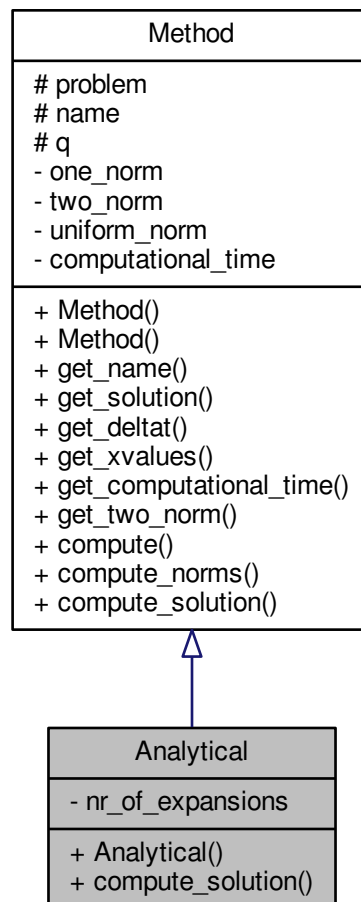
Class Documentation

4.1 Analytical Class Reference

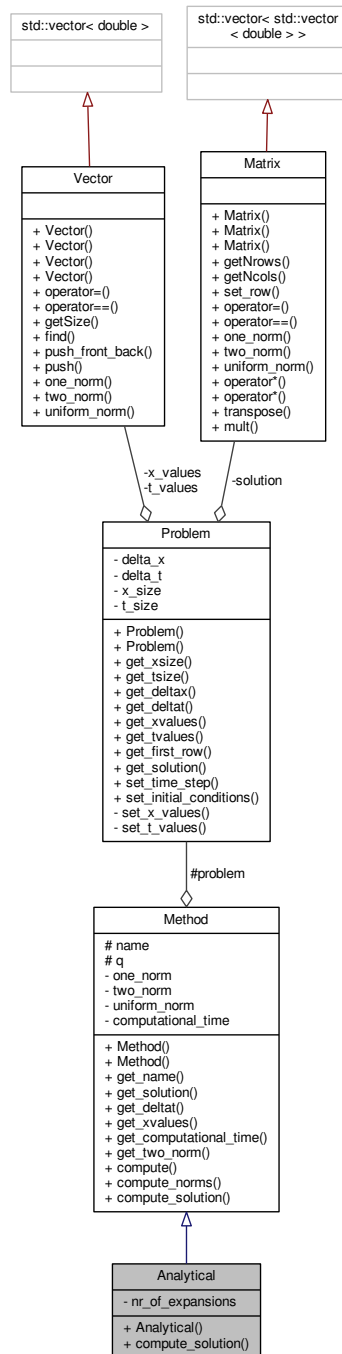
An [Analytical](#) class to compute the solution with standard procedures
The implementation is derived from the [Method](#) Object.

```
#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

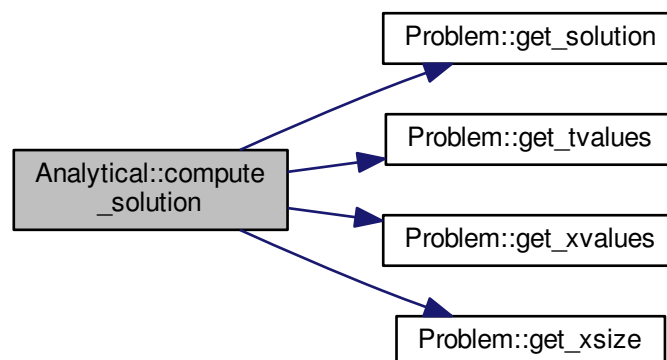
4.1.3.1 [void Analytical::compute_solution](#) () [\[virtual\]](#)

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:

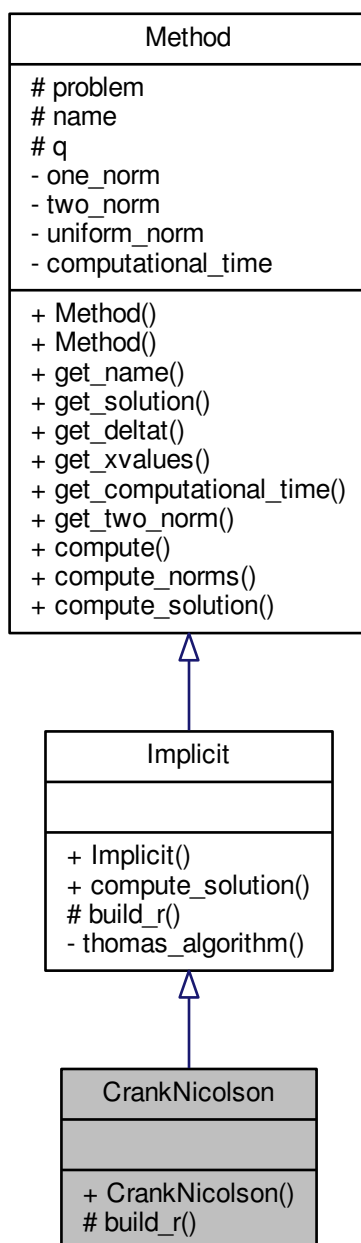
- methods/[analytical.h](#)
- methods/[analytical.cpp](#)

4.2 CrankNicolson Class Reference

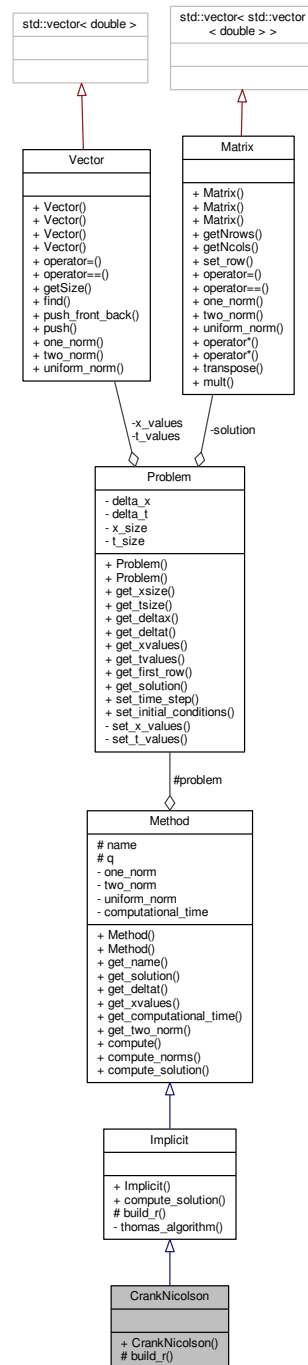
A [CrankNicolson](#) method class that contains a r vector builder.

```
#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

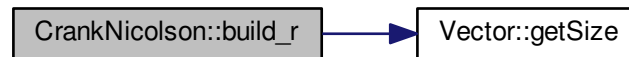
<i>previous_step</i>	Vector representing the solution of the previous time step.
----------------------	---

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:

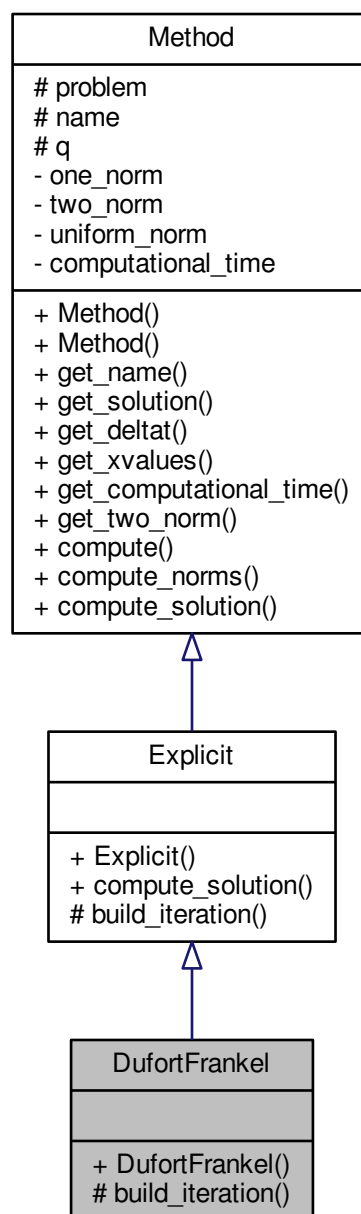
- [methods/implicit/crank_nicolson.h](#)
- [methods/implicit/crank_nicolson.cpp](#)

4.3 DufortFrankel Class Reference

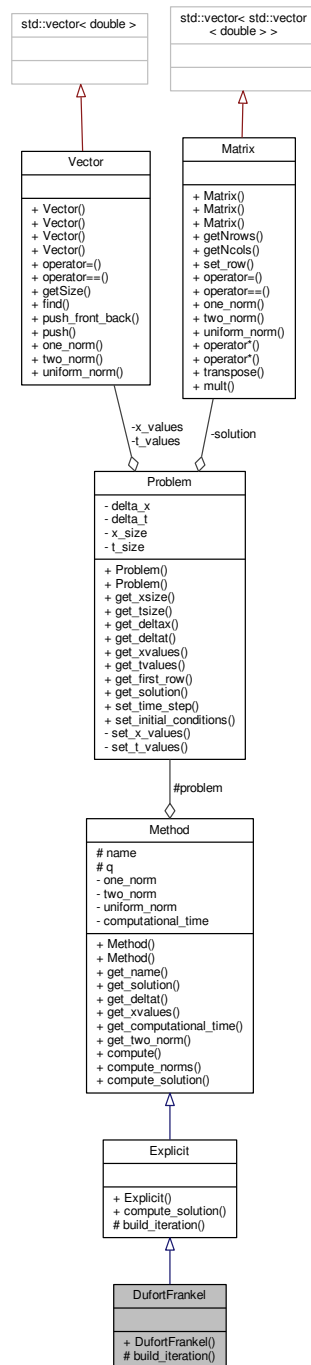
A [DufortFrankel](#) method class that contains an iteration builder.

```
#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

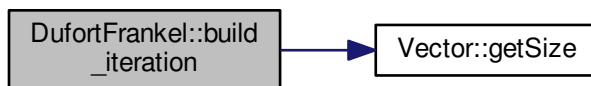
<i>current_step</i>	A vector representing the current time step solution.
<i>previous_step</i>	A vector representing the previous time step solution.

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:

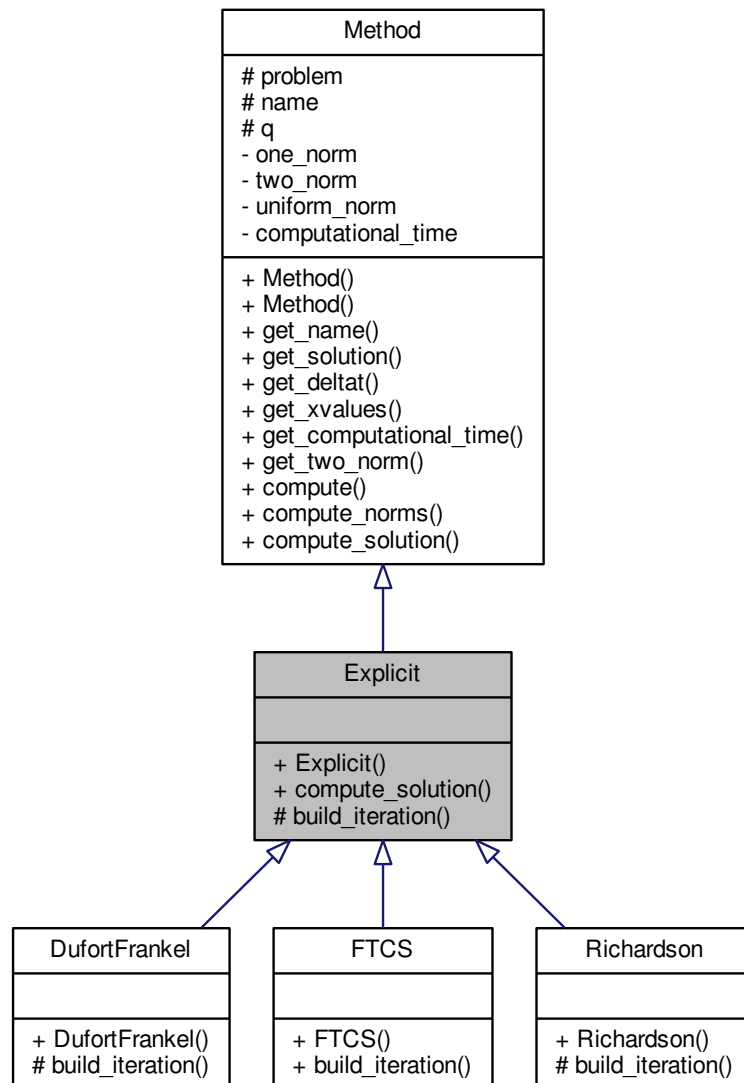
- [methods/explicit/dufort_frankel.h](#)
- [methods/explicit/dufort_frankel.cpp](#)

4.4 Explicit Class Reference

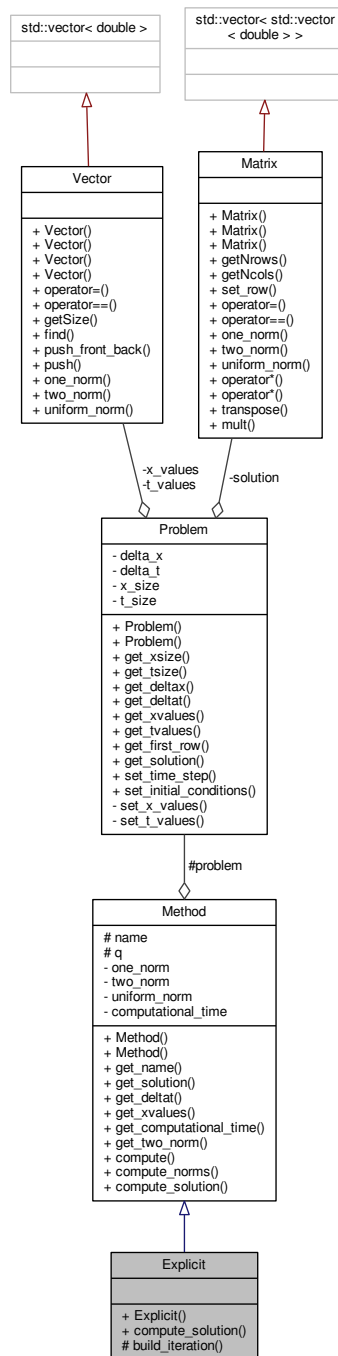
An explicit method class that contains default methods that only explicit methods use
The implementation is derived from the [Method](#) class.

```
#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:

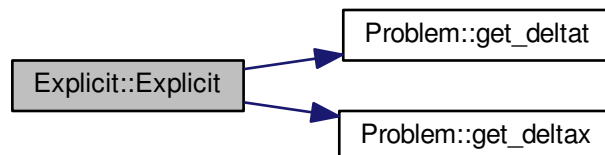
- a basic constructor for creating an explicit method object.
- a method to compute a solution following explicit methods rules

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:



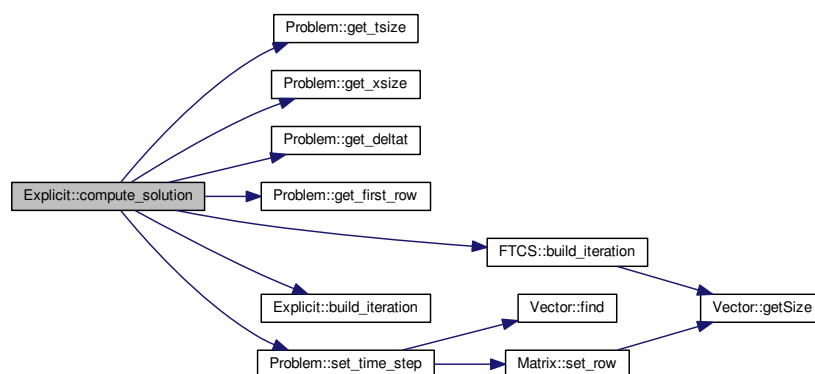
4.4.3.2 void Explicit::compute_solution () [virtual]

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:

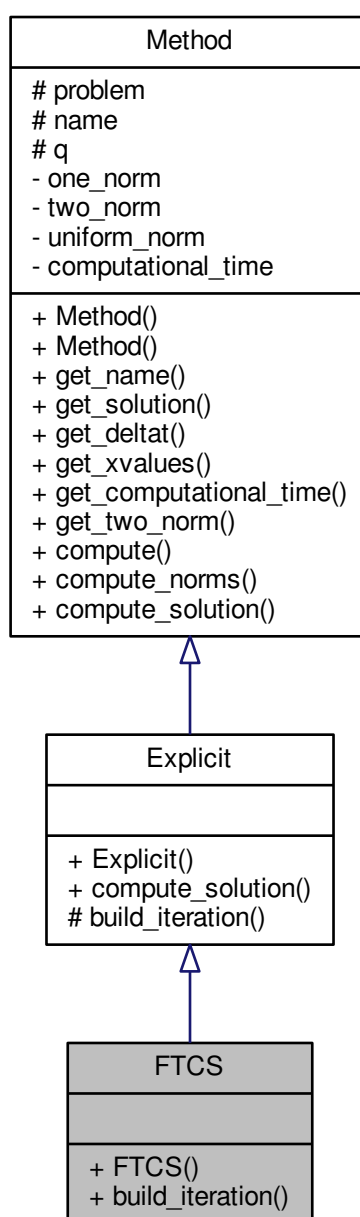
- [methods/explicit/explicit.h](#)
- [methods/explicit/explicit.cpp](#)

4.5 FTCS Class Reference

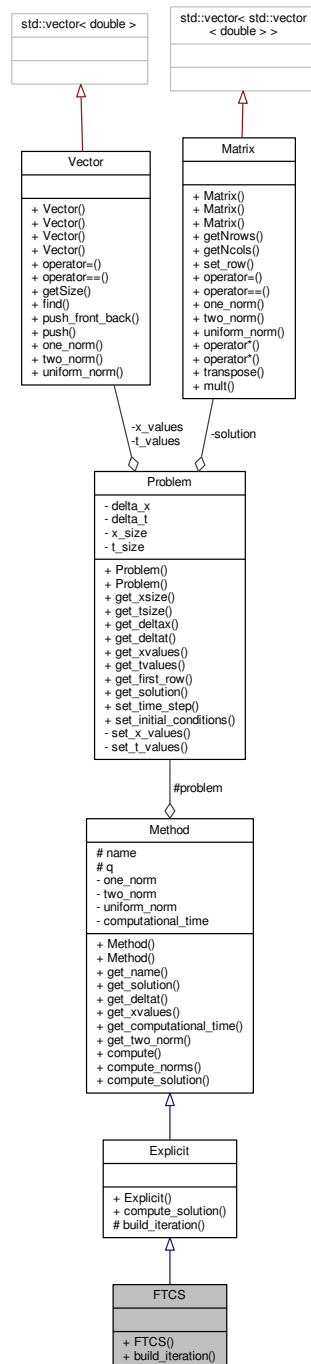
A [FTCS](#) method class that contains an iteration builder.

```
#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:

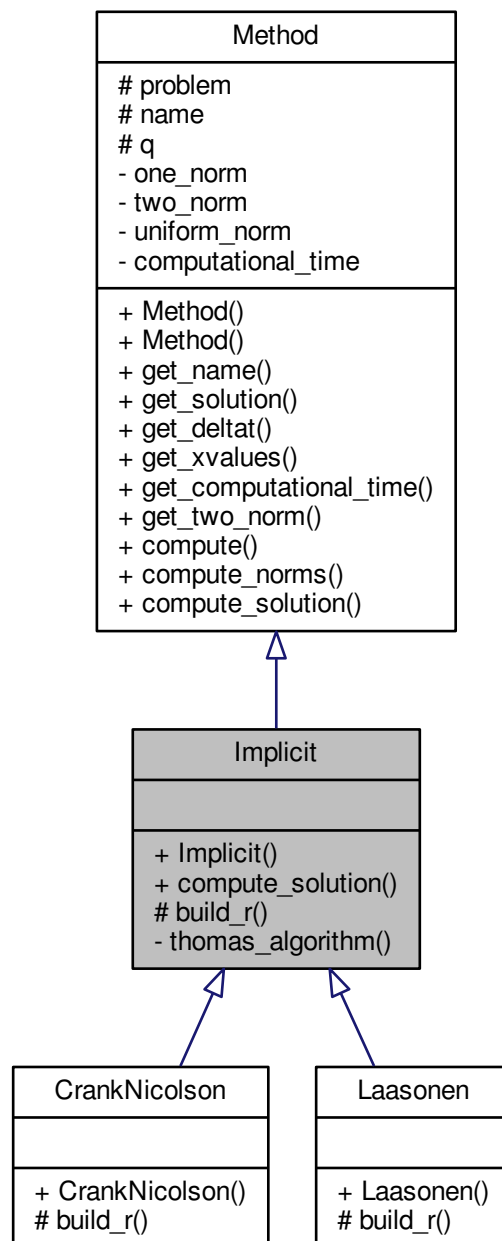
- [methods/explicit/forward_t_central_s.h](#)
- [methods/explicit/forward_t_central_s.cpp](#)

4.6 Implicit Class Reference

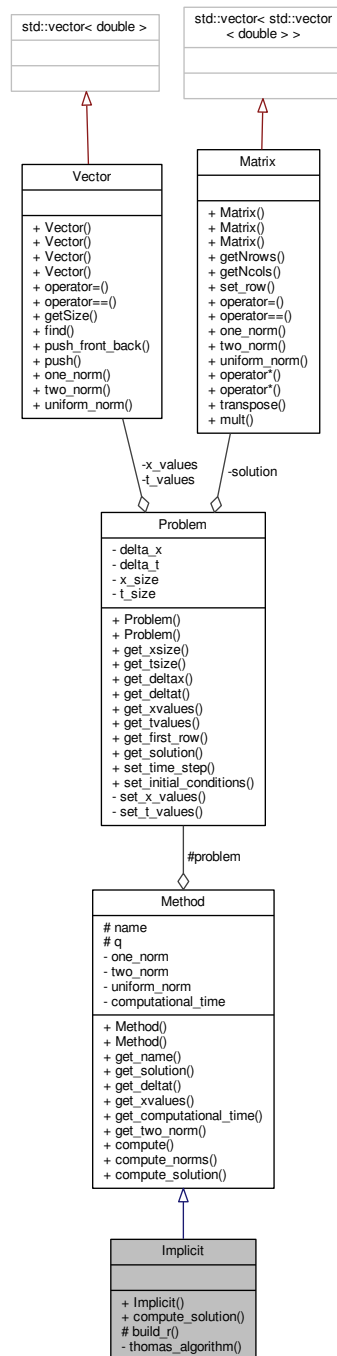
An implicit method class that contains default methods that only implicit methods use
The implementation is derived from the [Method](#) class.

```
#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:

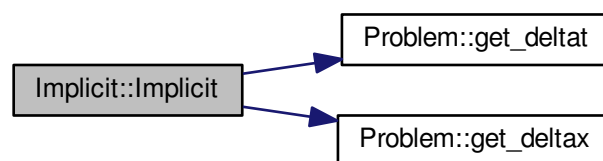
- a basic constructor for creating an implicit method object.
- a method to compute a solution following implicit methods rules

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

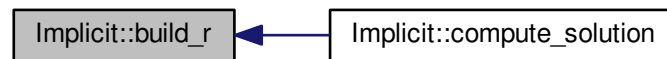
<i>previous_step</i>	A vector containing the previous time step solution.
----------------------	--

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:



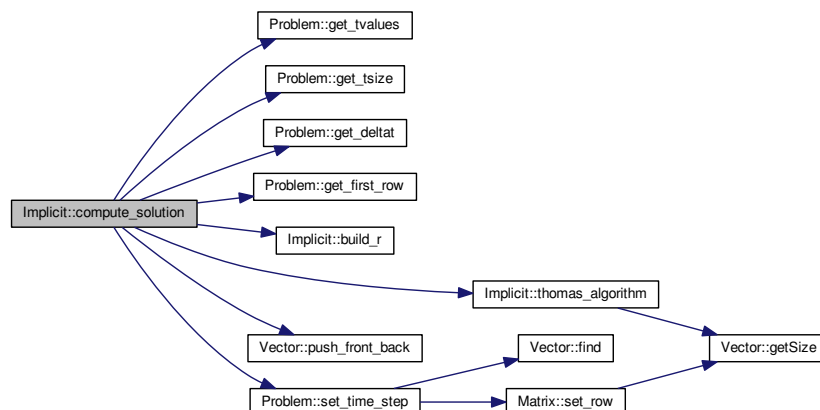
4.6.3.2 void Implicit::compute_solution () [virtual]

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\(Vector previous_step\)](#)

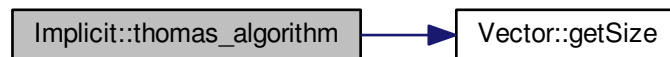
Parameters

<i>r</i>	Vector 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:

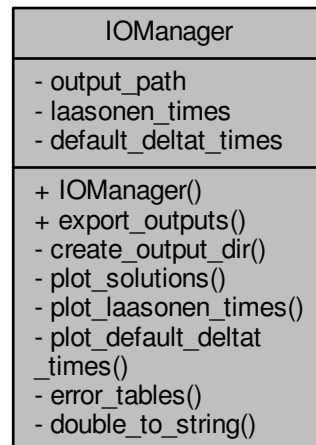
- [methods/implicit/implicit.h](#)
- [methods/implicit/implicit.cpp](#)

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 ouput 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](#) delta t variation computational times.
- void [plot_default_deltat_times](#) ()
Exports a plot with four methods computational times.
- void [error_tables](#) (std::string output_name, std::vector< [Method](#) * > method)
Exports a plot that compares the norms of each solution.
- std::string [double_to_string](#) (int precision, double value)
Converts a double to a string with a precison of 2 decimal places.

Private Attributes

- `std::string output_path`
Private string output_path.
- `std::vector< double > laasonen_times`
Private [Vector](#) laasonen_times.
- `std::vector< double > 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

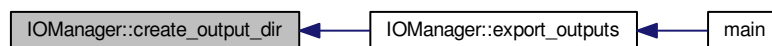
4.7.3.1 bool IOManager::create_output_dir () [private]

[Method](#) to create output folder if the folder does not exist.

Returns

bool. true if successfull, 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.

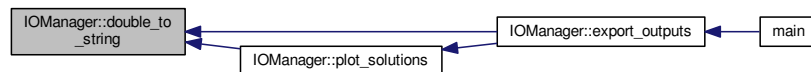
Parameters

<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:



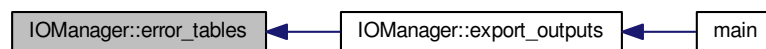
4.7.3.3 void IOManager::error_tables (std::string output_name, std::vector< Method * > method) [private]

Exports a plot that compares the norms of each solution.

Parameters

<i>string</i>	output_name File name to be exported
<i>vector< Method*></i>	vector of methods to plot the second norm

Here is the caller graph for this function:



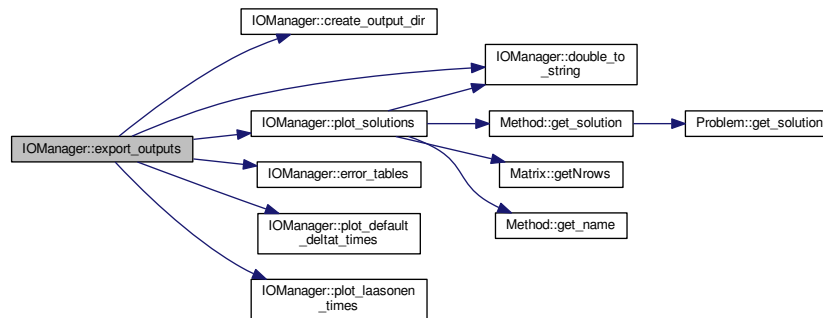
4.7.3.4 void IOManager::export_outputs (Method * analytical, std::vector< Method * > methods)

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

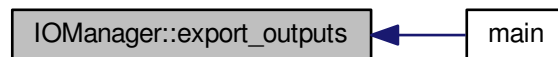
Parameters

<i>Method*</i>	analytical The analytical solution
<i>vector< Method*></i>	methods Vector containing the solutions

Here is the call graph for this function:



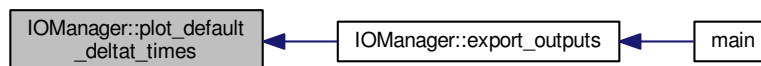
Here is the caller graph for this function:



4.7.3.5 void IOManager::plot_default_deltat_times () [private]

Exports a plot with four methods computational times.

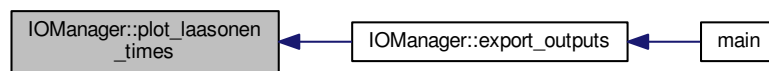
Here is the caller graph for this function:



4.7.3.6 void IOManager::plot_laasonen_times () [private]

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

Here is the caller graph for this function:



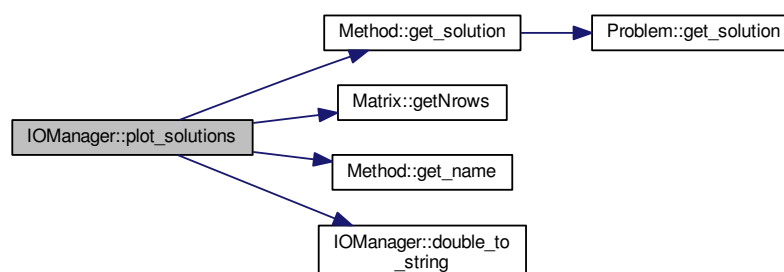
4.7.3.7 `void IOManager::plot_solutions (std::string output_name, Method * analytical, Method * method)`
`[private]`

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 `std::vector<double> 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:

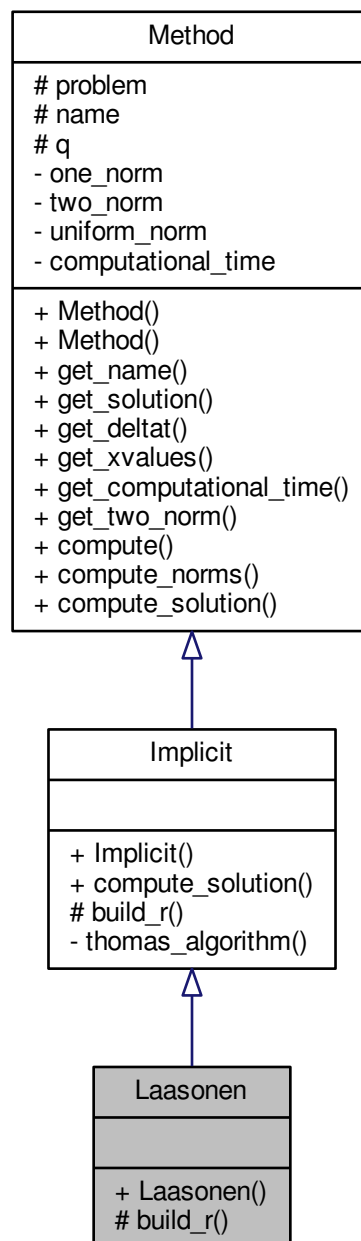
- [io/iomanager.h](#)
- [io/iomanager.cpp](#)

4.8 Laasonen Class Reference

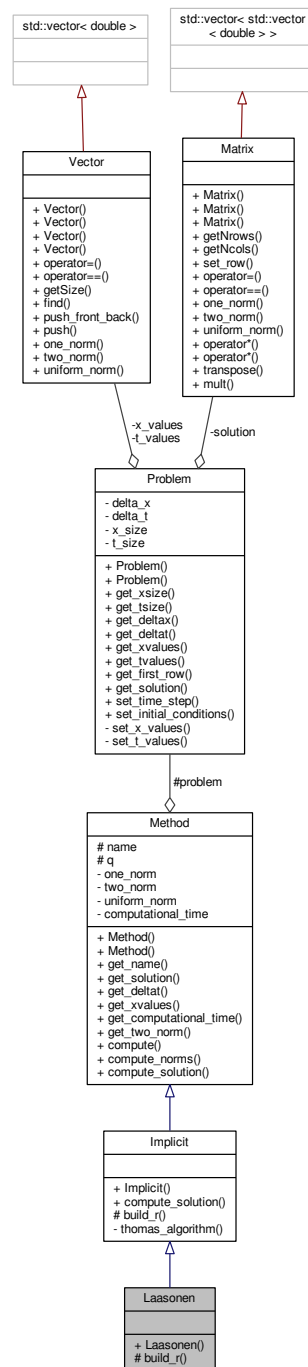
A [Laasonen](#) method class that contains a r vector builder.

```
#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>	Vector 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

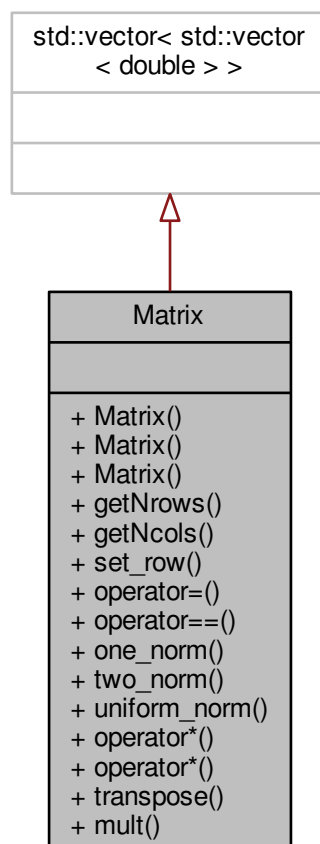
A matrix class for data storage of a 2D array of doubles

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

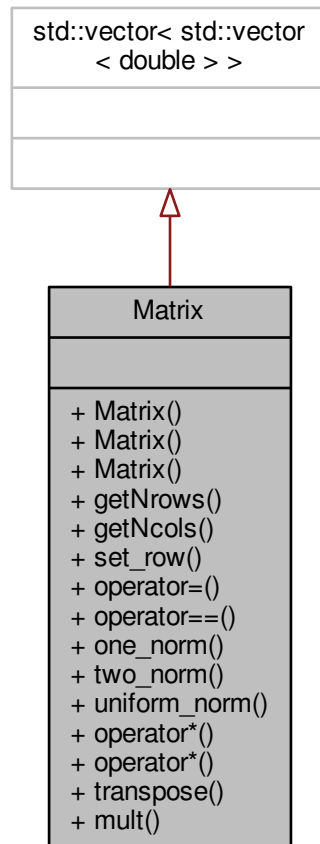
We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`.

```
#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.

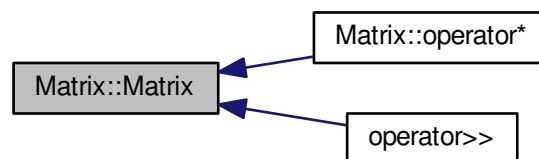
Initialize an empty [Matrix](#) object

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

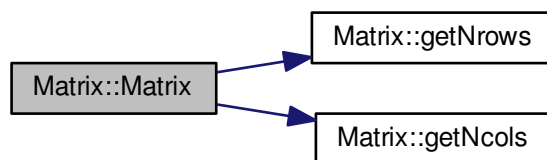
[Matrix\(\)](#)

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

Parameters

<i>m</i>	Matrix& . 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

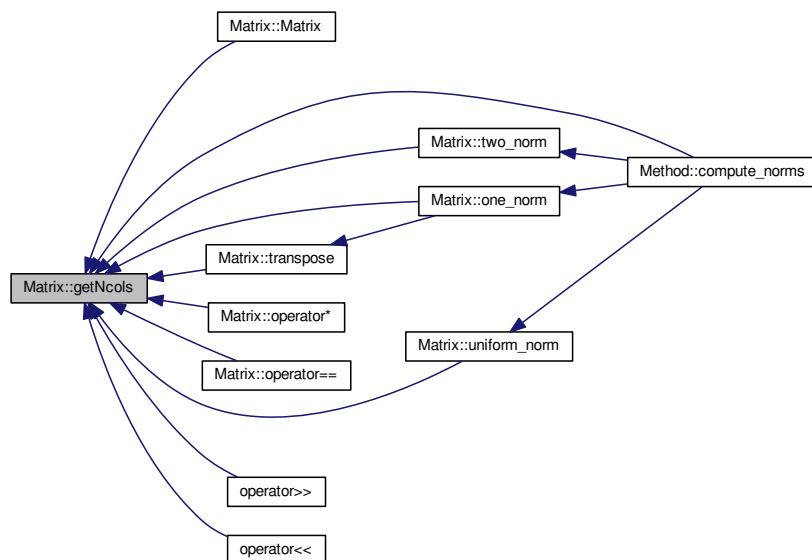
See also

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

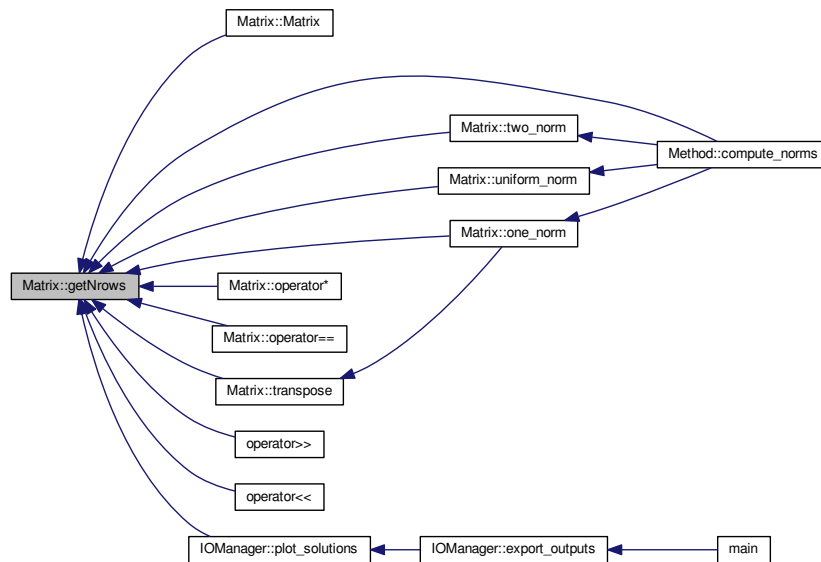
See also

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

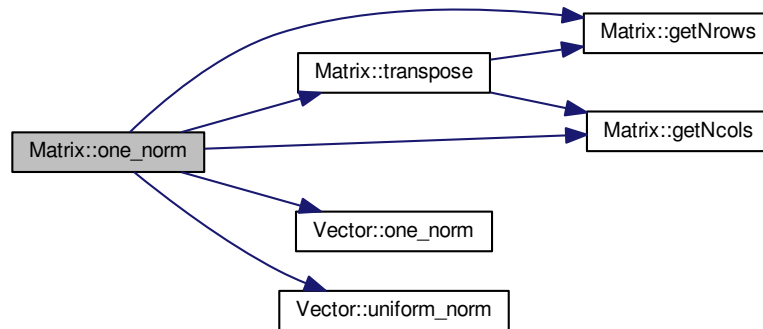
See also

[two_norm\(\)const](#)
[uniform_norm\(\)const](#)

Returns

double. matrix L1 norm

Here is the call graph for this function:



Here is the caller 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

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

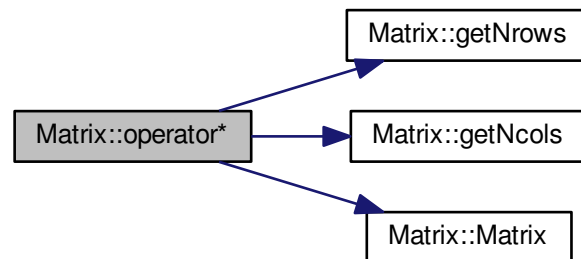
Returns

[Matrix](#). matrix-matrix product

Parameters

<i>a</i>	Matrix . 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

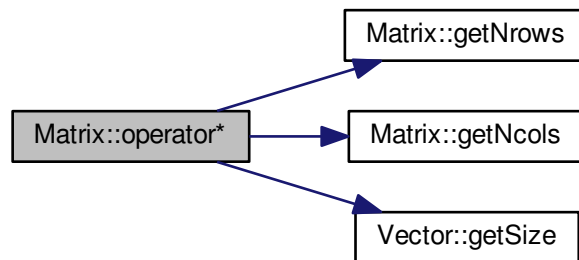
Returns

[Vector](#). matrix-vector product

Parameters

<i>v</i>	Vector . Vector 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>	Matrix& . Matrix 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=\(const Matrix& m\)](#)

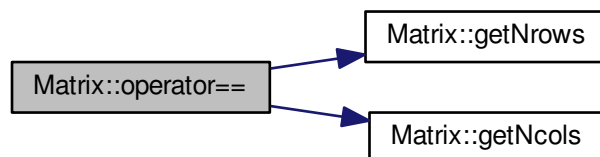
Returns

bool. true or false

Parameters

<i>m</i>	Matrix& . Matrix 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

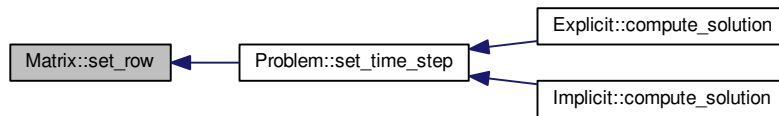
Exceptions

<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`

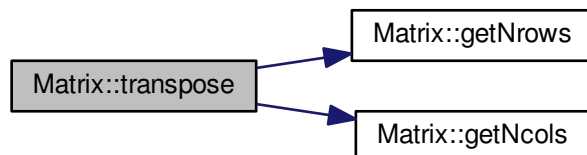
public method that returns the transpose of the matrix.

It returns the transpose of matrix

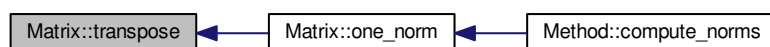
Returns

[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

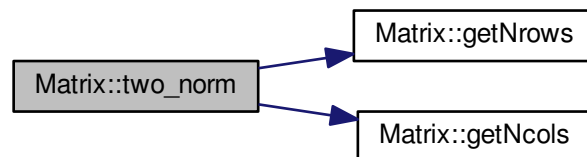
See also

[one_norm\(\)const](#)
[uniform_norm\(\)const](#)

Returns

double. matrix L2 norm

Here is the call graph for this function:



Here is the caller 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

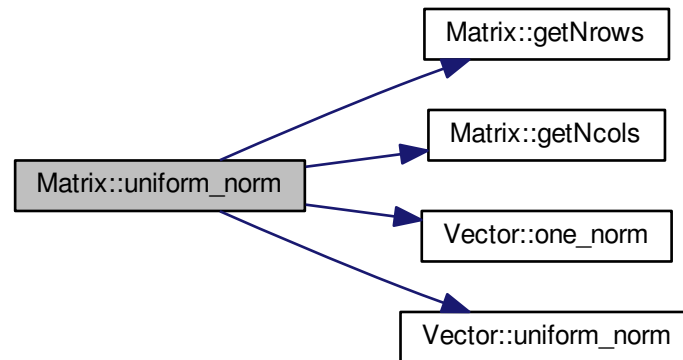
See also

[one_norm\(\)const](#)
[two_norm\(\)const](#)

Returns

double. matrix L_max norm

Here is the call graph for this function:



Here is the caller 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>	Matrix 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>	Matrix 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>	Matrix to write into

4.9.5.4 std::istream& operator>> (std::istream & *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>>\(std::istream& ifs, Matrix& m\)](#)
[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>	Matrix to write into

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

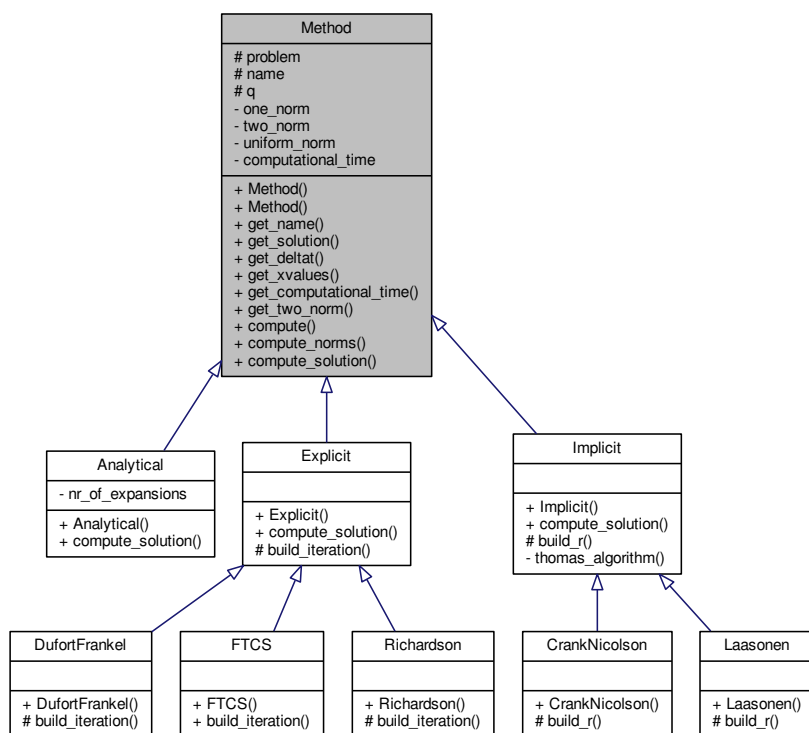
- [grid/matrix.h](#)
- [grid/matrix.cpp](#)

4.10 Method Class Reference

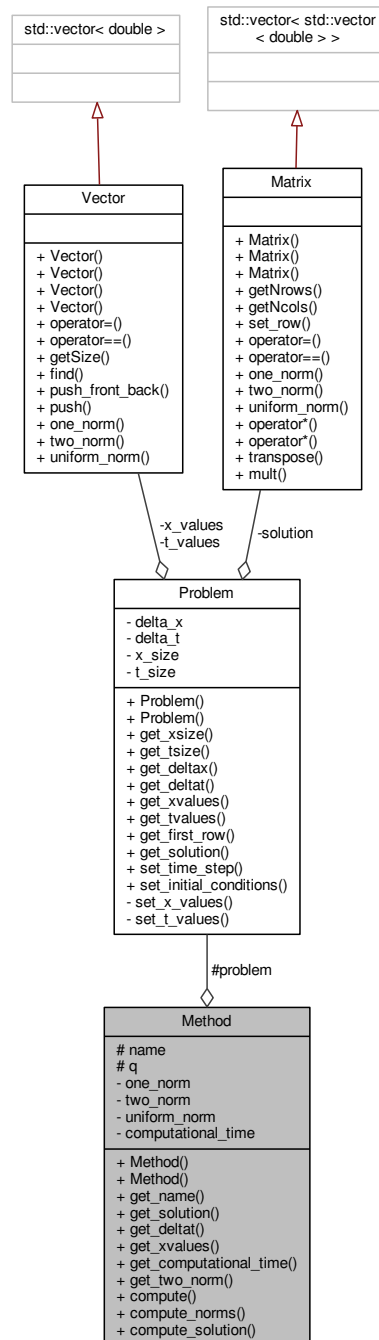
A [Method](#) class to structure information used to solve the problem.

```
#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.*
- double [get_two_norm](#) ()
- Normal public get method.*
- void [compute](#) ()
- Normal public method.*
- void [compute_norms](#) ([Matrix](#) analytical_matrix)
- 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 [one_norm](#)
- double [two_norm](#)
- double [uniform_norm](#)
- 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.
- accessor 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.

Initialize 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.

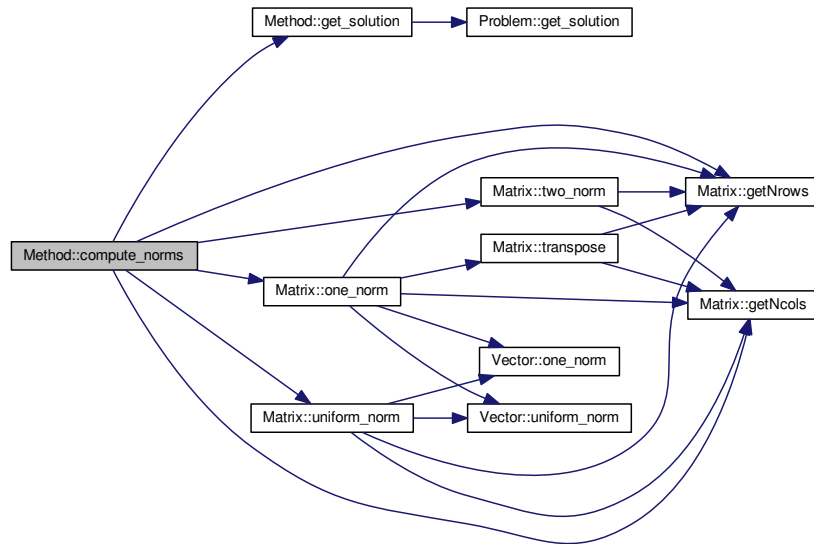
Keeps track of the time to compute a solution

Here is the call graph for this function:



4.10.3.2 void Method::compute_norms (Matrix analytical_matrix)

Here is the call graph for this function:



4.10.3.3 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:



4.10.3.4 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.

4.10.3.5 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:



4.10.3.6 std::string Method::get_name ()

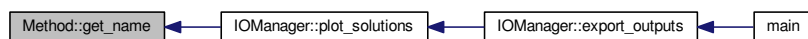
Normal public get method.

get the method name

Returns

string. [Method](#) name.

Here is the caller graph for this function:



4.10.3.7 Matrix Method::get_solution ()

Normal public get method.

get the solution grid

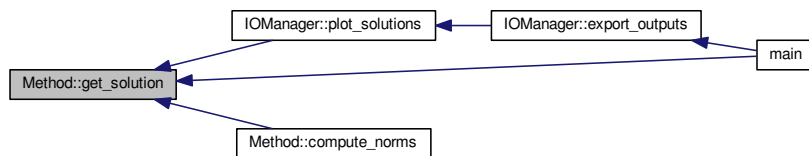
Returns

[Matrix](#). Computed solution grid.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.10.3.8 double Method::get_two_norm ()**

Normal public get method.

get the second norm

Returns

double. Second norm value.

4.10.3.9 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:



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 `double Method::one_norm` `[private]`

4.10.4.4 **Problem** `Method::problem` `[protected]`

Protected [Problem](#) `problem`.

Space step of the solution.

4.10.4.5 `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.

4.10.4.6 `double Method::two_norm` `[private]`

4.10.4.7 `double Method::uniform_norm` `[private]`

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

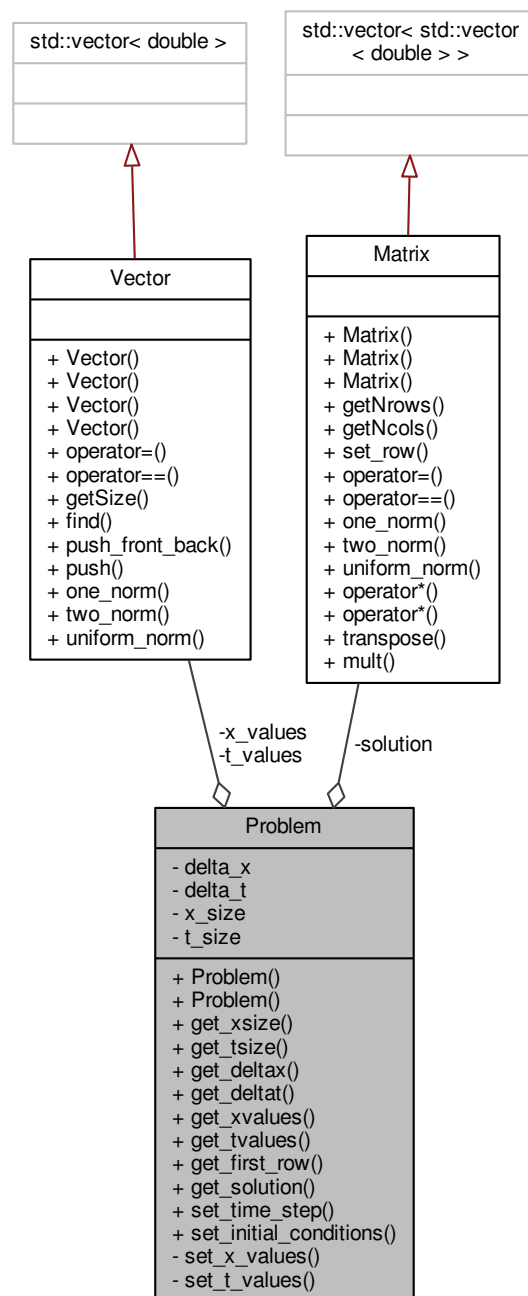
- [methods/method.h](#)
- [methods/method.cpp](#)

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\(\)](#)

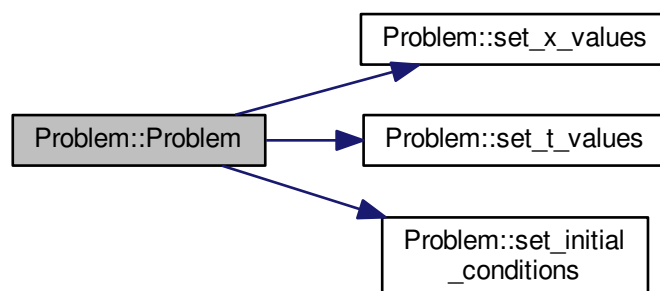
Parameters

<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

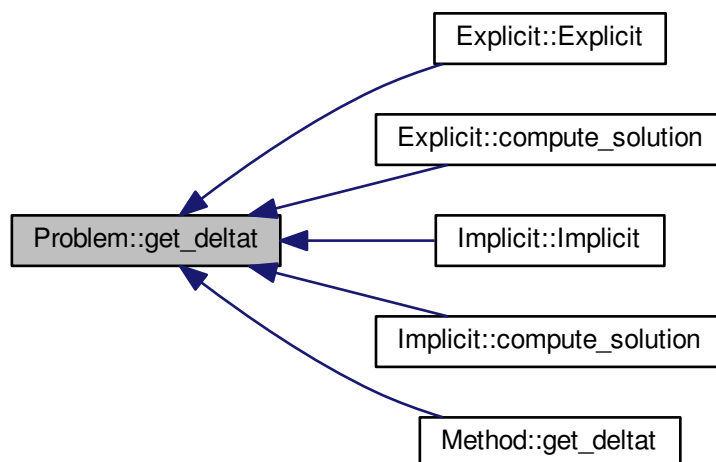
4.11.3.1 `double Problem::get_deltat ()`

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:



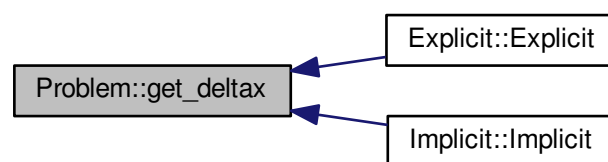
4.11.3.2 `double Problem::get_deltax ()`

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:



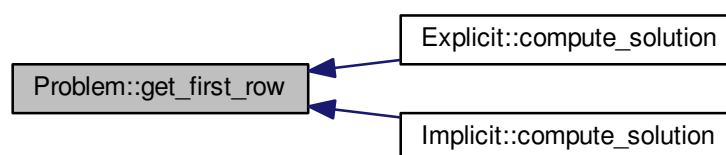
4.11.3.3 `Vector Problem::get_first_row ()`

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:



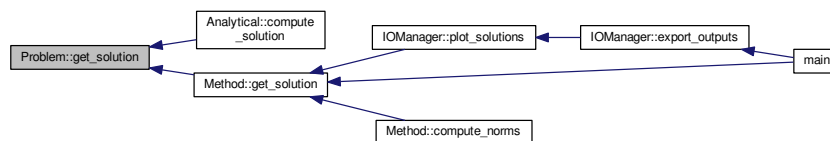
4.11.3.4 **Matrix *** Problem::get_solution ()

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:



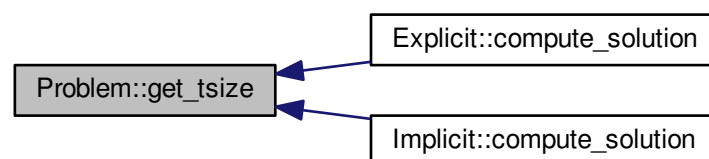
4.11.3.5 **unsigned int** Problem::get_tsize ()

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:



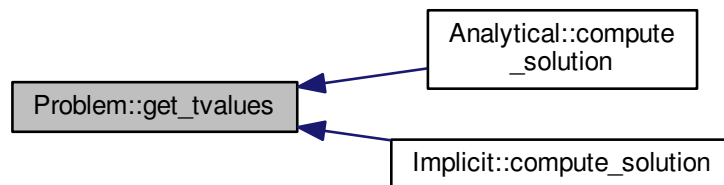
4.11.3.6 **Vector** Problem::get_tvalues ()

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:

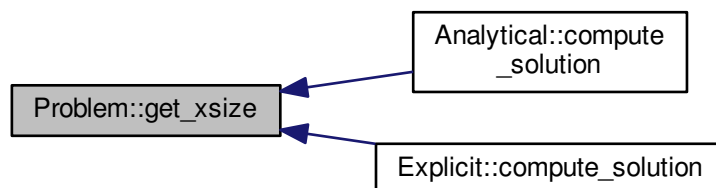
4.11.3.7 `unsigned int Problem::get_xsize ()`

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:

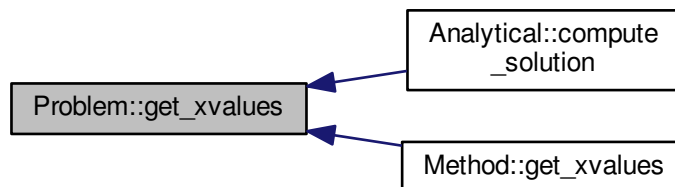
4.11.3.8 `Vector Problem::get_xvalues ()`

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*)

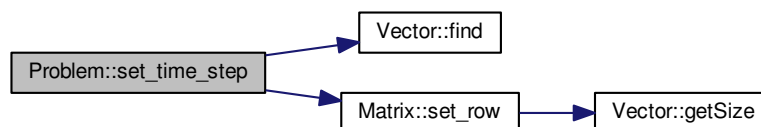
Normal public set method.

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

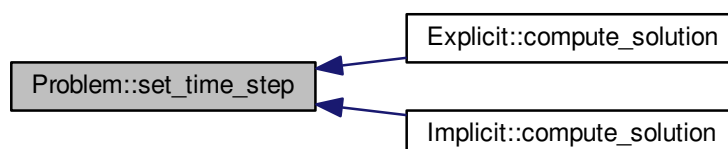
Parameters

<i>step</i>	Vector 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:

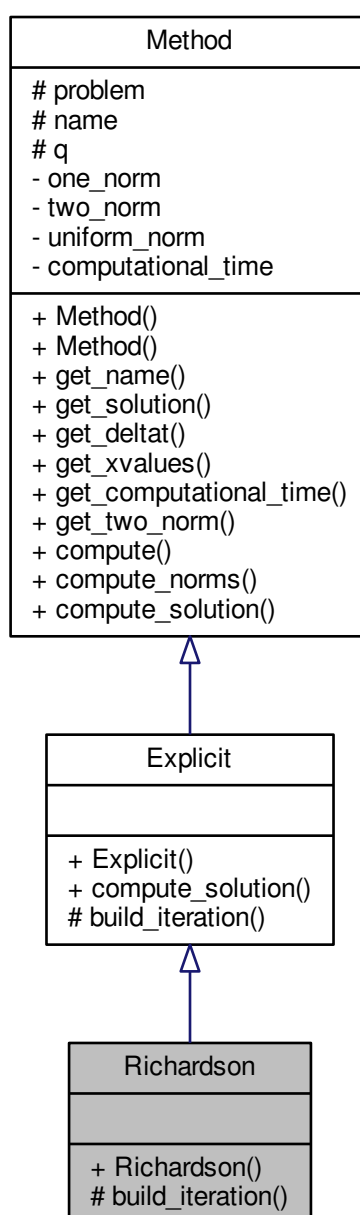
- [variants/problem.h](#)
- [variants/problem.cpp](#)

4.12 Richardson Class Reference

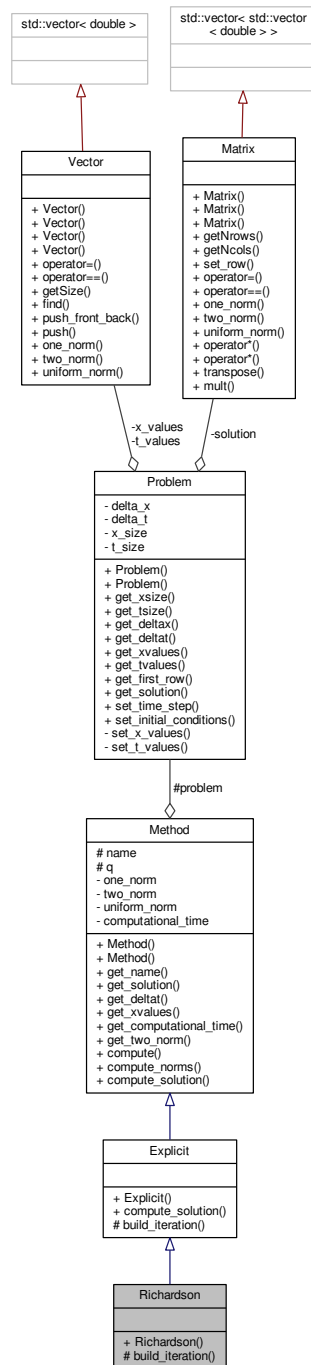
A [Richardson](#) method class that contains an iteration builder.

```
#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

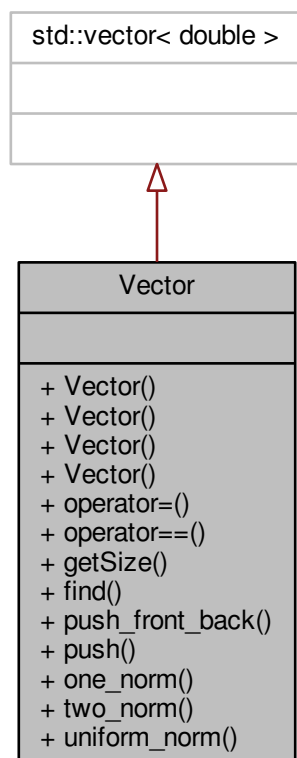
A vector class for data storage of a 1D array of doubles

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

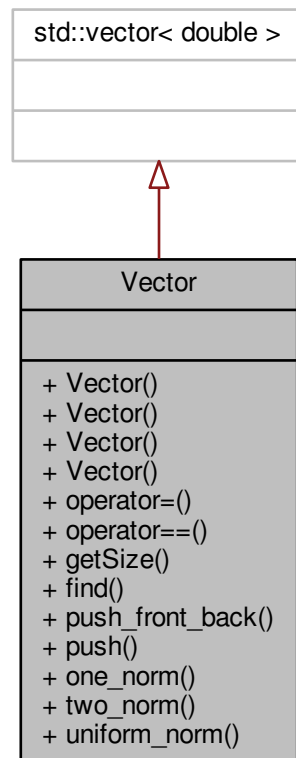
We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`.

```
#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](#) ()

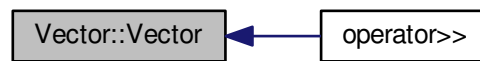
Default constructor.

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

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

Parameters

<i>Num</i>	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.

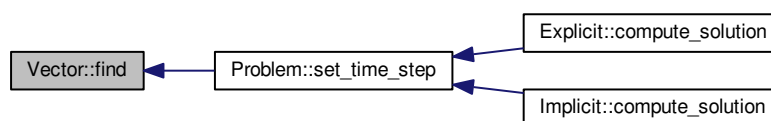
Parameters

<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:



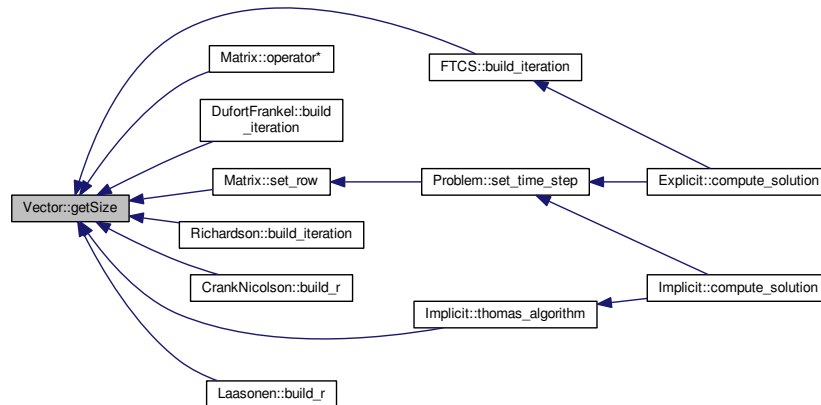
4.13.4.2 int Vector::getSize () const

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

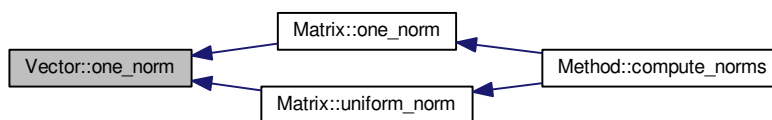
See also

[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

<i>v</i>	Vector to assign from
----------	---------------------------------------

Returns

the object on the left of the assignment

Parameters

<i>v</i>	Vector&. Vector 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

<i>v</i>	Vector &. 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](#)

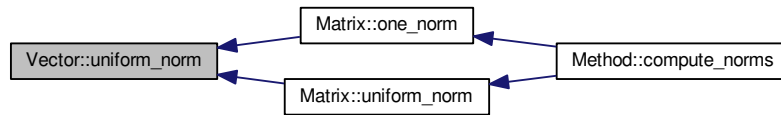
Exceptions

<i>out_of_range</i>	("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>	Vector &. 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>	Vector &. 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>	Vector &. vector to write to

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

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

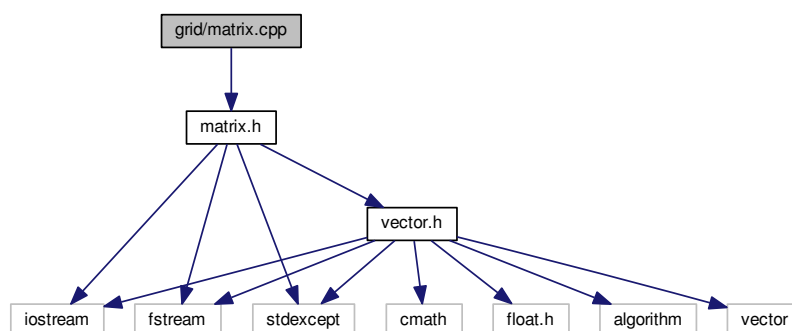
Chapter 5

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<<\(std::ostream& os, const Matrix& m\)](#)

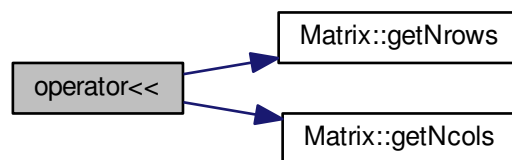
Returns

std::ostream&. The ostream object

Parameters

<i>os</i>	Display output stream
<i>m</i>	Matrix 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");
------------------------------------	---

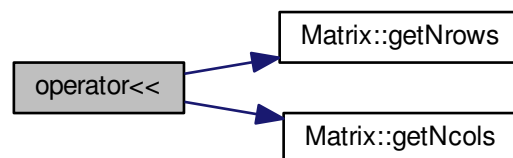
Returns

std::ofstream&. The ofstream object

Parameters

<i>m</i>	Matrix 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");
------------------------------------	--

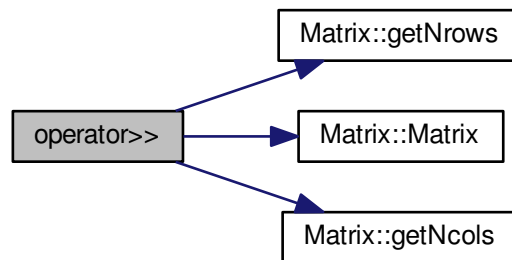
Returns

`std::istream&`. The istream object

Parameters

<i>is</i>	Keyboard input stream
<i>m</i>	Matrix 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

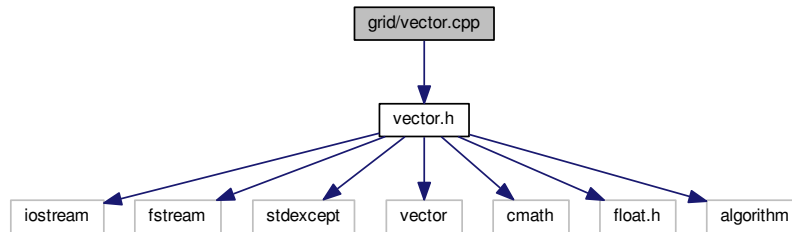
Parameters

<i>ifs</i>	Input file stream with opened matrix file
<i>m</i>	Matrix to write into

Here is the call graph for this function:



Include dependency graph for vector.cpp:



Functions

- `std::istream & operator>> (std::istream &is, Vector &v)`
- `std::ifstream & operator>> (std::ifstream &if, 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>	Vector &. 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>	Vector &. 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>	Vector& . vector to write to

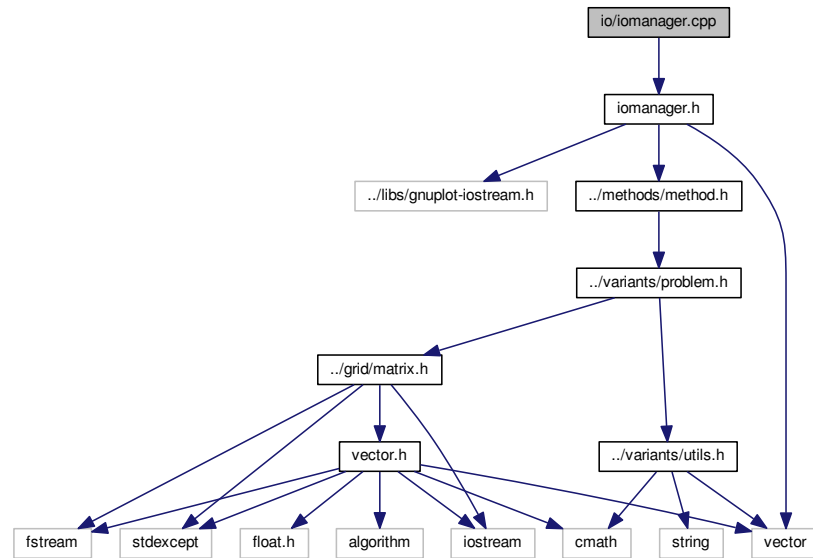
Here is the call graph for this function:



5.4 grid/vector.h File Reference

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

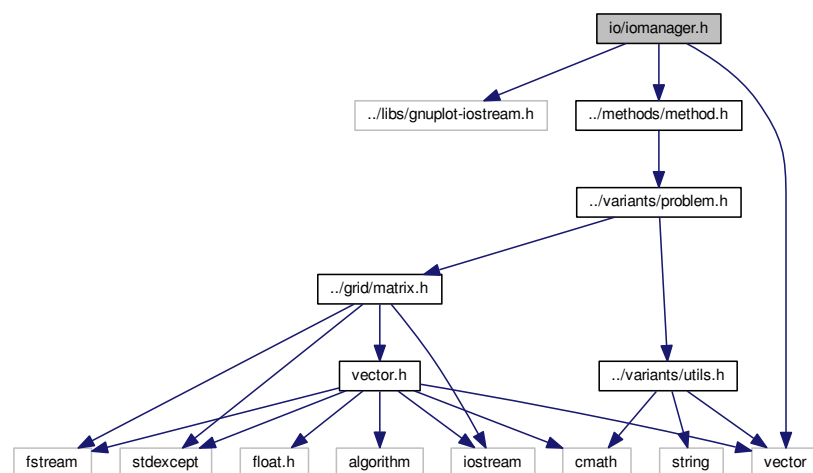

Include dependency graph for iomanager.cpp:



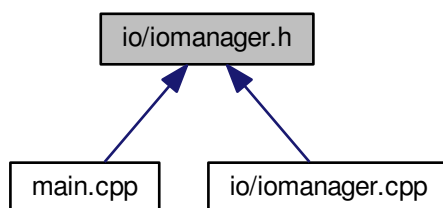
5.6 io/iomanager.h File Reference

```
#include "../libs/gnuplot-iostream.h"
#include "../methods/method.h"
#include <vector>
```

Include dependency graph for iomanager.h:



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



Classes

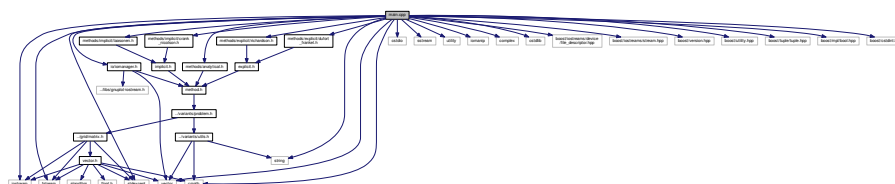
- class `IOManager`

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:



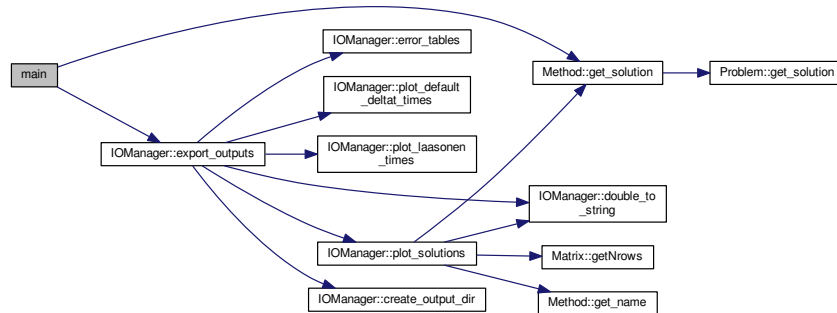
Functions

- int `main` ()

5.7.1 Function Documentation

5.7.1.1 `int main ()`

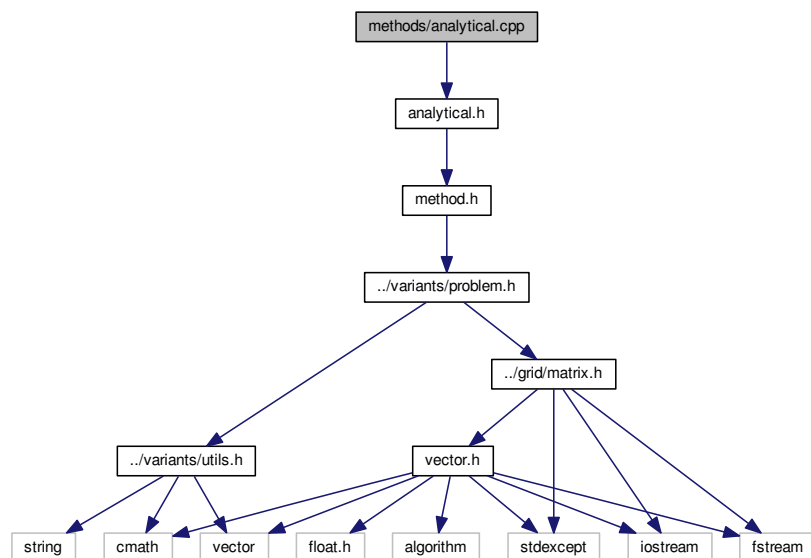
Here is the call graph for this function:



5.8 methods/analytical.cpp File Reference

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

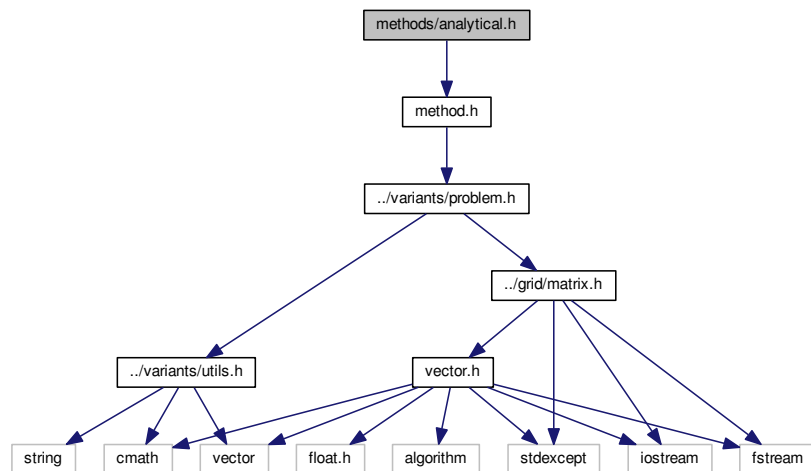
Include dependency graph for analytical.cpp:



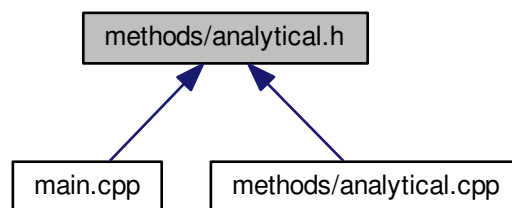
5.9 methods/analytical.h File Reference

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

Include dependency graph for analytical.h:



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



Classes

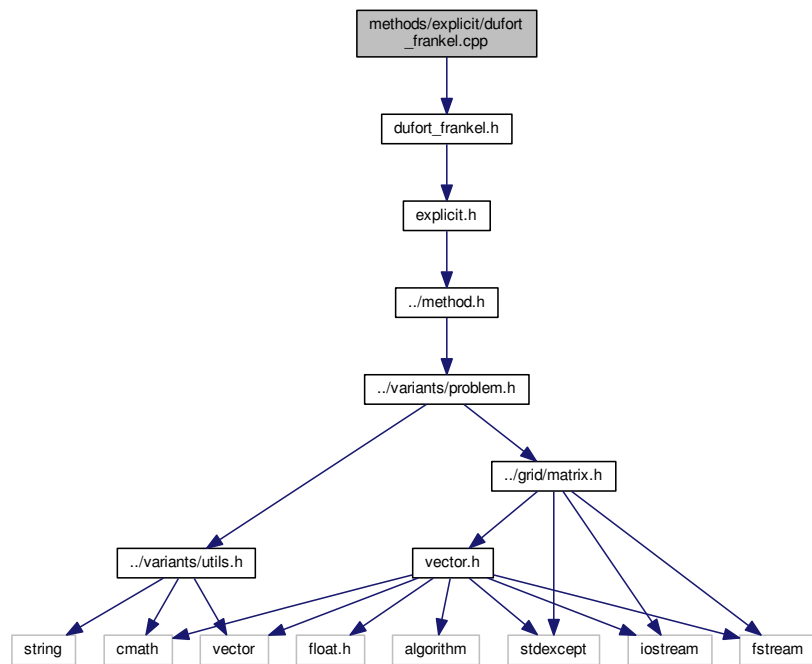
- class [Analytical](#)

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"
```

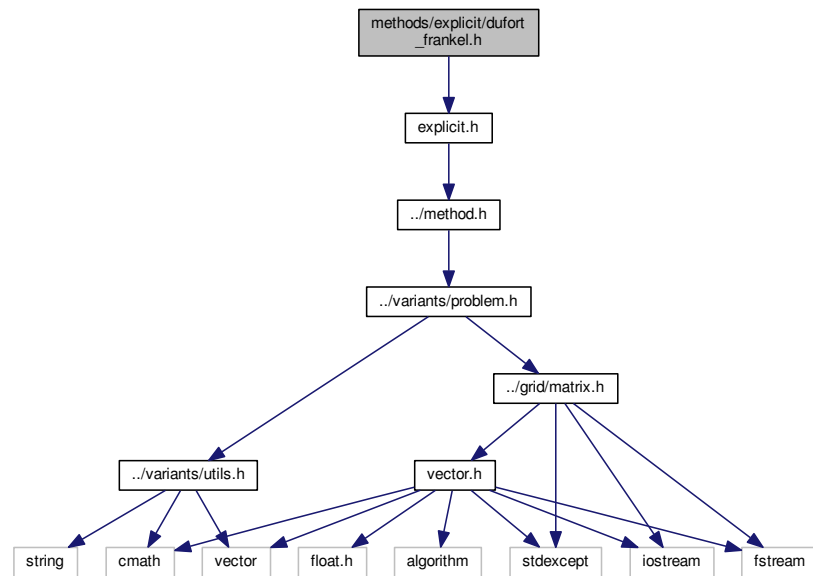
Include dependency graph for dufort_frankel.cpp:



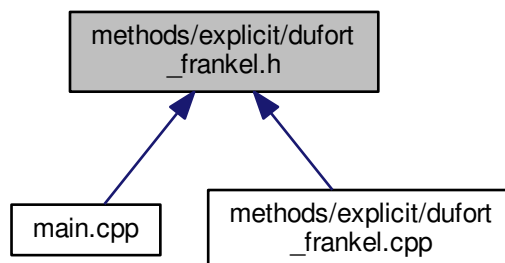
5.11 methods/explicit/dufort_frankel.h File Reference

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

Include dependency graph for dufort_frinkel.h:



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



Classes

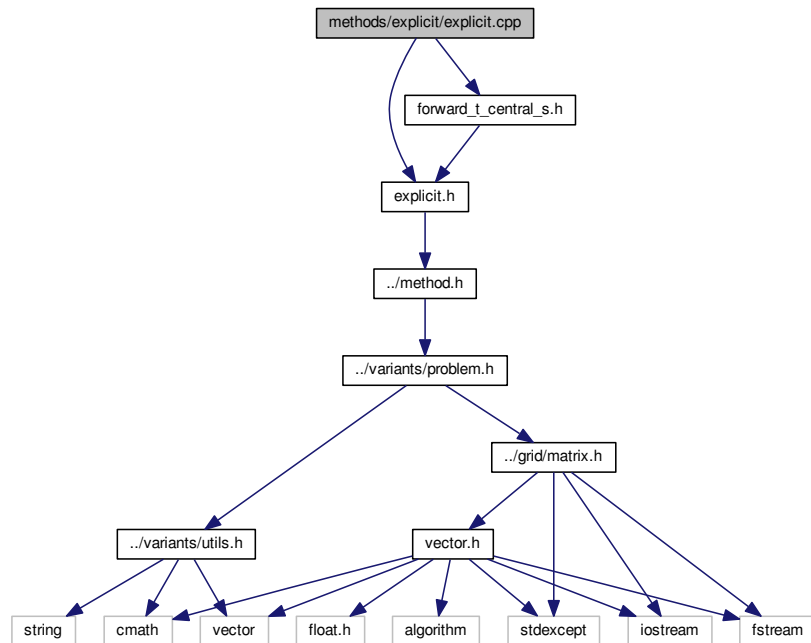
- class [DufortFrankel](#)
A [DufortFrankel](#) method class that contains an iteration builder.

5.12 methods/explicit/explicit.cpp File Reference

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

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

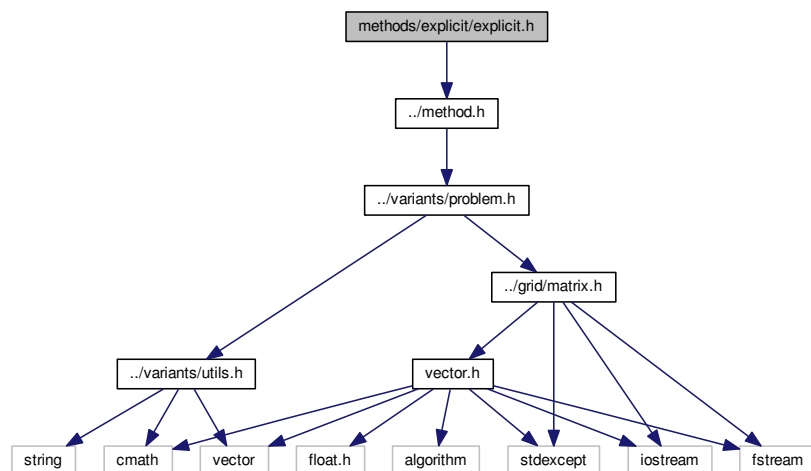
Include dependency graph for explicit.cpp:



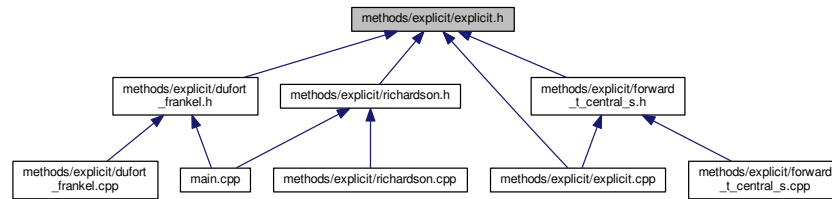
5.13 methods/explicit/explicit.h File Reference

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

Include dependency graph for explicit.h:



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



Classes

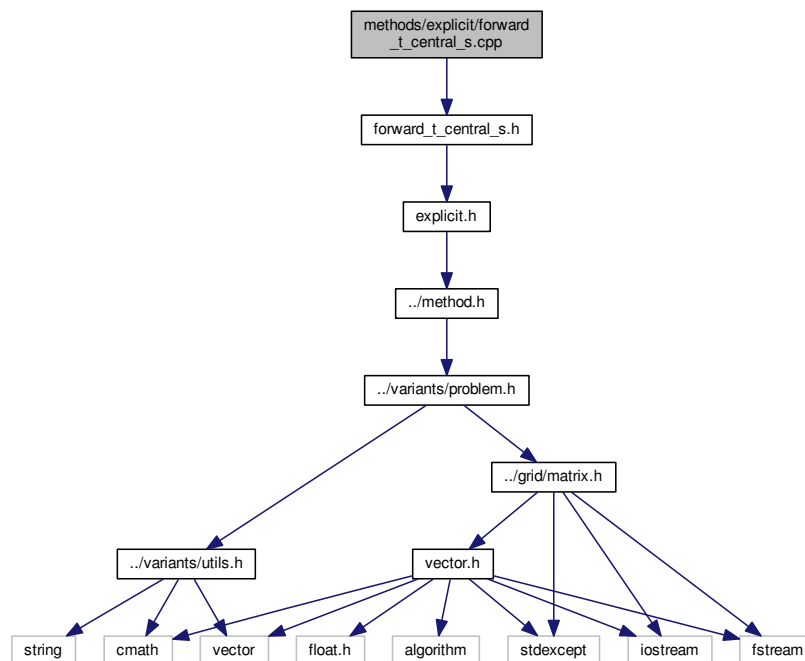
- class [Explicit](#)

*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"
```

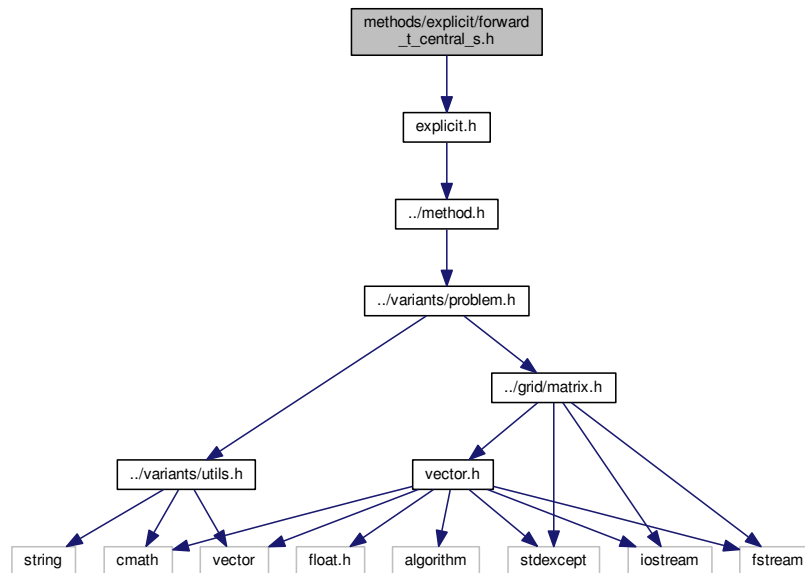
Include dependency graph for forward_t_central_s.cpp:



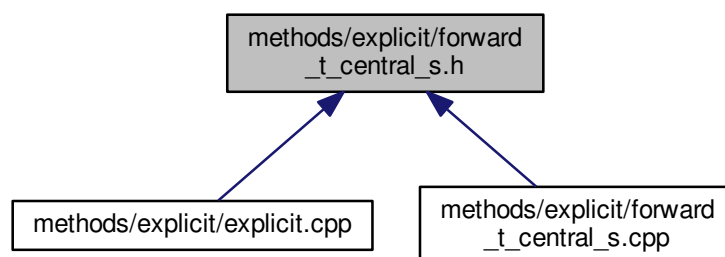
5.15 methods/explicit/forward_t_central_s.h File Reference

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

Include dependency graph for forward_t_central_s.h:



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



Classes

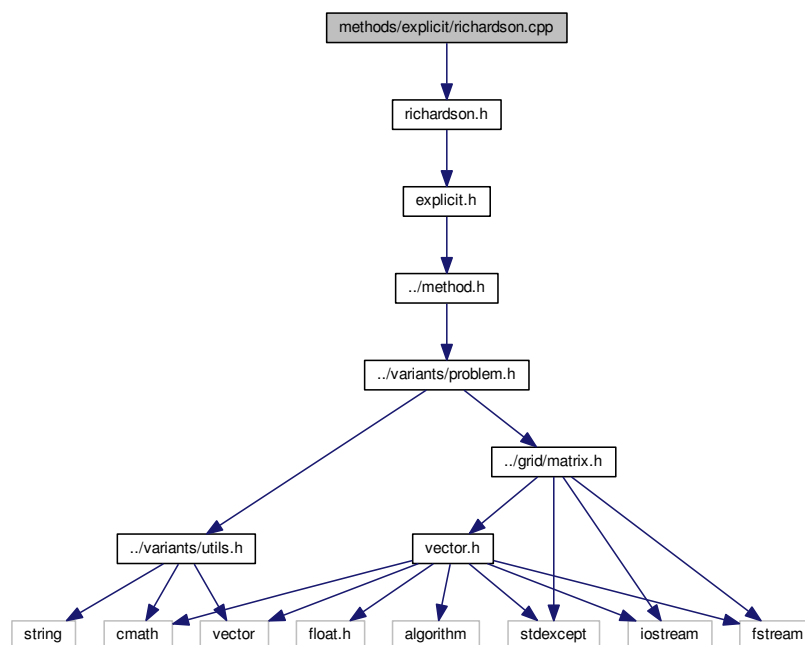
- class [FTCS](#)

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

5.16 methods/explicit/richardson.cpp File Reference

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

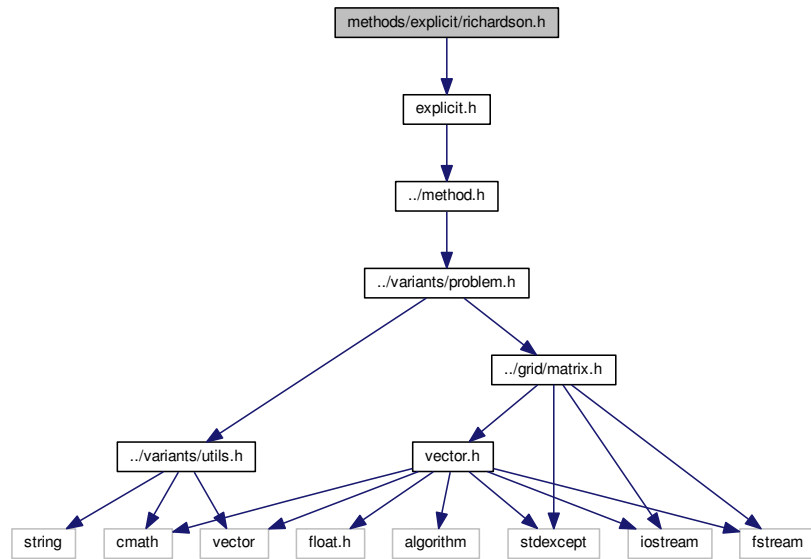
Include dependency graph for richardson.cpp:



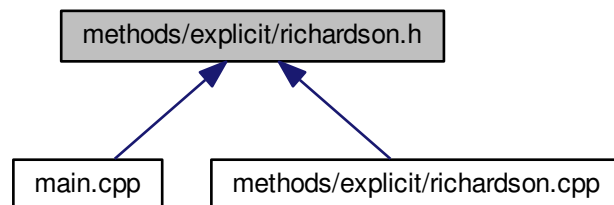
5.17 methods/explicit/richardson.h File Reference

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

Include dependency graph for richardson.h:



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



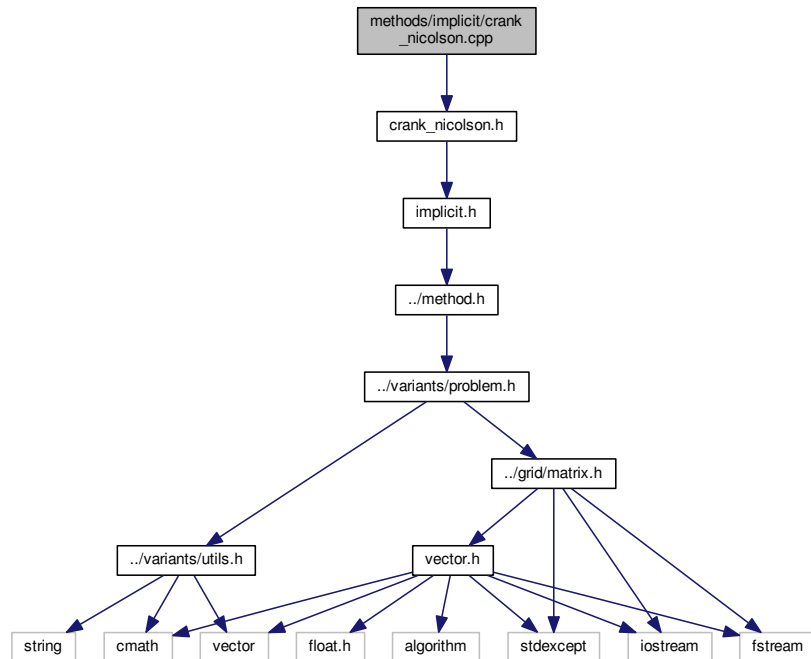
Classes

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

5.18 methods/implicit/crank_nicolson.cpp File Reference

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

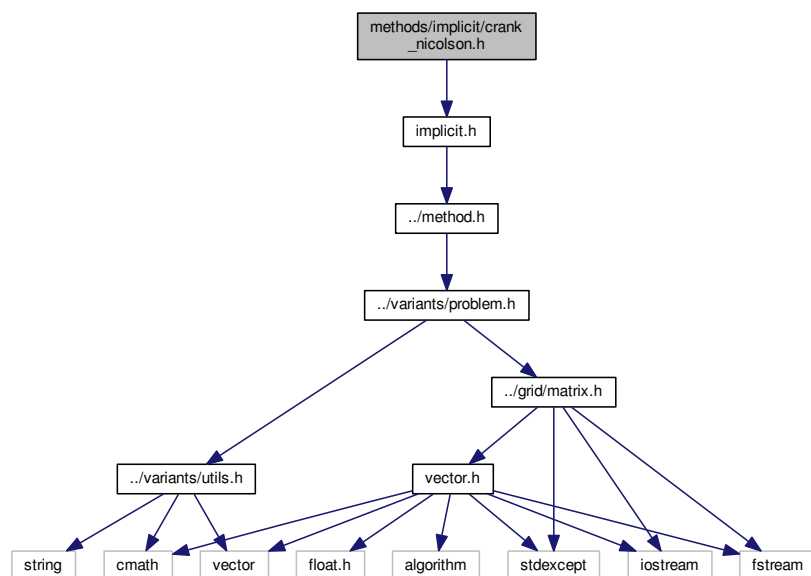
Include dependency graph for krank_nicolson.cpp:



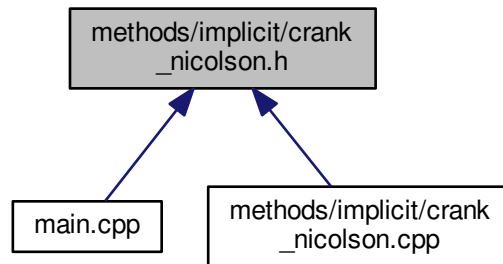
5.19 methods/implicit/krank_nicolson.h File Reference

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

Include dependency graph for krank_nicolson.h:



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



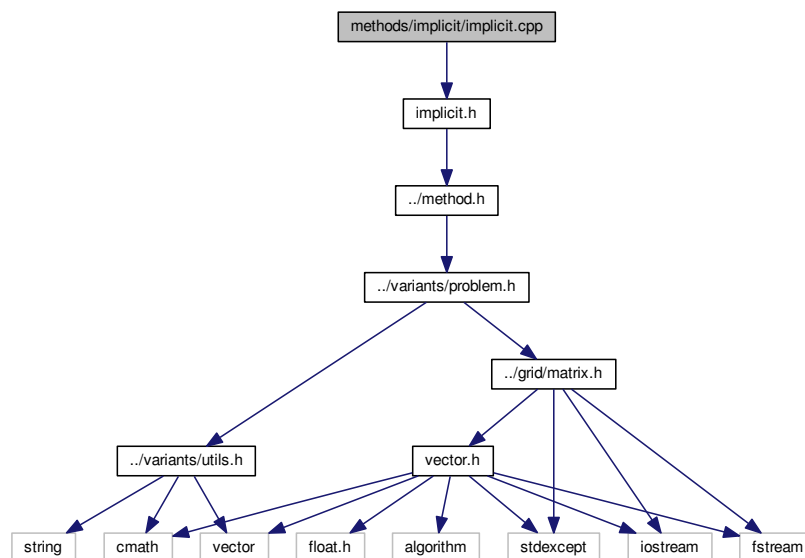
Classes

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

5.20 methods/implicit/implicit.cpp File Reference

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

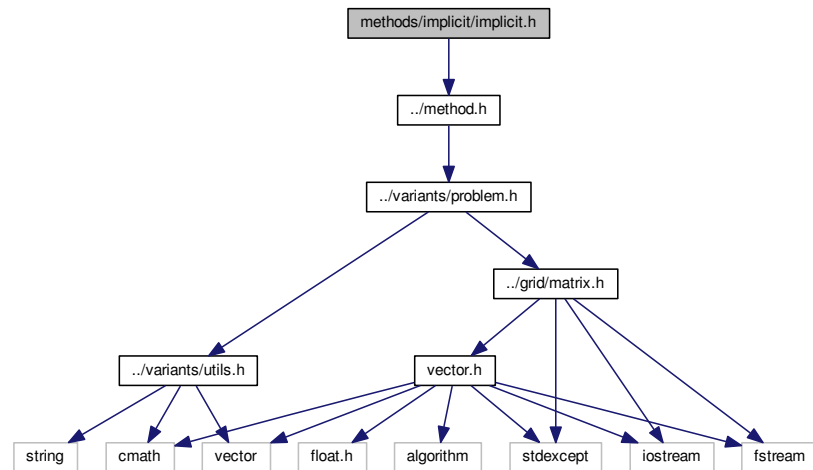
Include dependency graph for implicit.cpp:



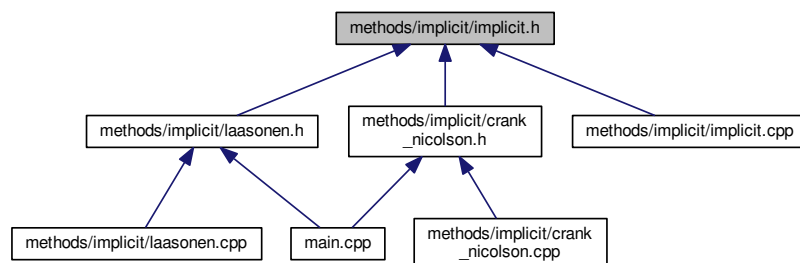
5.21 methods/implicit/implicit.h File Reference

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

Include dependency graph for implicit.h:



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



Classes

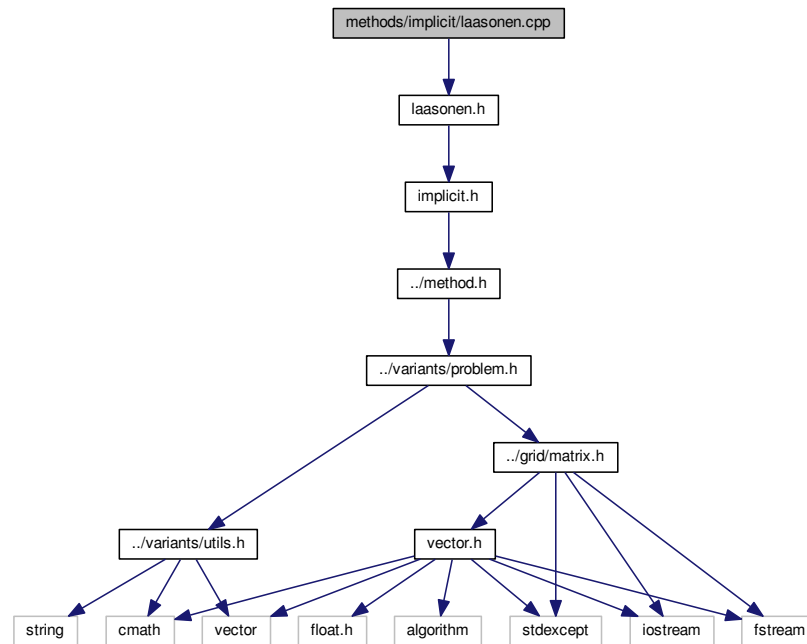
- class [Implicit](#)

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"
```

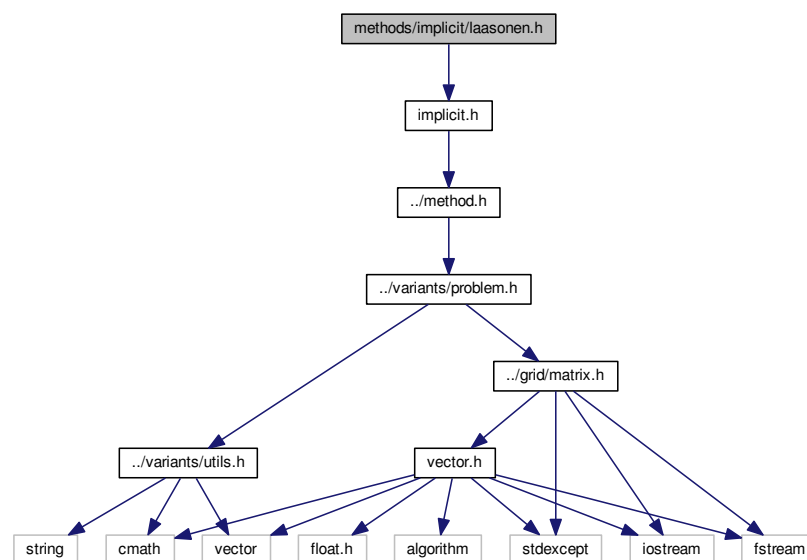
Include dependency graph for laasonen.cpp:



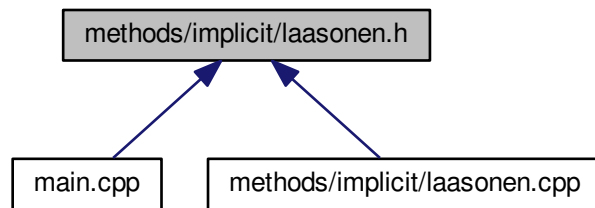
5.23 methods/implicit/laasonen.h File Reference

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

Include dependency graph for laasonen.h:



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



Classes

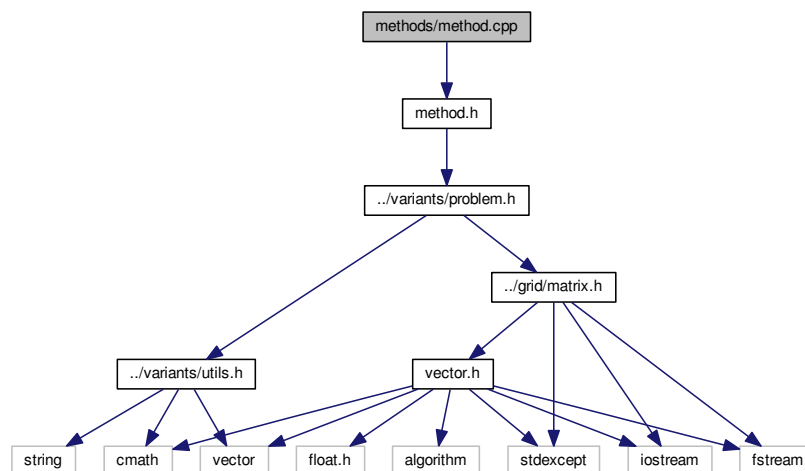
- class [Laasonen](#)

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

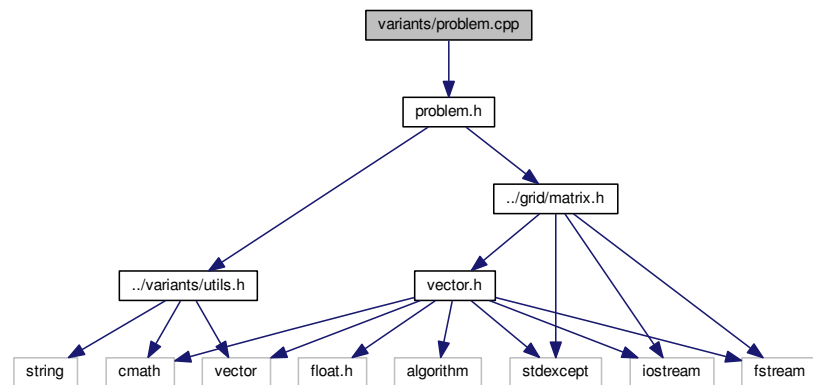
5.24 methods/method.cpp File Reference

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

Include dependency graph for `method.cpp`:



Include dependency graph for problem.cpp:

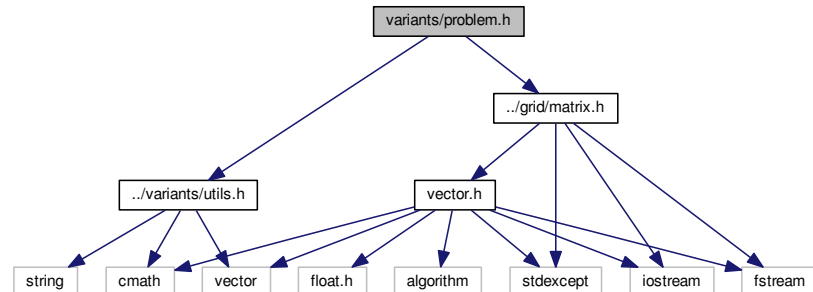


5.27 variants/problem.h File Reference

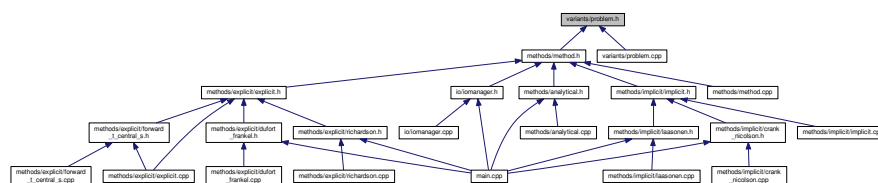
```
#include "../variants/utils.h"
```

```
#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 `ANALYTICAL` = `"Analytical"`
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 ANALYTICAL = "Analytical"

Macro string.

Forward in Time and Central in Space method name.

5.28.1.2 const std::string CRANK_NICHOLSON = "Crank-Nicholson"

Macro string.

Crank-Nicholson method name.

5.28.1.3 const double DELTA_T = 0.01

Macro double.

The default time step.

5.28.1.4 const std::vector<double> DELTA_T_LAASONEN = {0.01, 0.025, 0.05, 0.1}

Macro double.

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

5.28.1.5 `const double DELTA_X = 0.05`

Macro double.

The default space step.

5.28.1.6 `const double DIFUSIVITY = 0.1`

Macro double.

The default value of difusivity.

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

Macro string.

DuFort-Frankel method name.

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

Macro string.

Forward in Time and Central in Space method name.

5.28.1.9 `const double INITIAL_TEMPERATURE = 100.0`

Macro double.

The default initial temperature.

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

Macro string.

[Laasonen](#) method name.

5.28.1.11 `const unsigned int NUMBER_OF_EXPANSIONS = 20`

Macro unsigned int.

Number of expansions to calculate the analytical solution sum expansion.

5.28.1.12 `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

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

Macro string.

Default outputs path.

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

Macro double.

Approximated value of PI.

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

Macro string.

[Richardson](#) method name.

5.28.1.16 `const double SURFACE_TEMPERATURE = 300.0`

Macro double.

The default surface temperature.

5.28.1.17 `const double THICKNESS = 1.0`

Macro double.

The default value of thickness.

5.28.1.18 `const double TIMELIMIT = 0.5`

Macro double.

The default value of time limit.

Index

ANALYTICAL

utils.h, [121](#)

Analytical, [7](#)

Analytical, [10](#)

compute_solution, [10](#)

nr_of_expansions, [11](#)

build_iteration

DufortFrankel, [18](#)

Explicit, [22](#)

FTCS, [26](#)

Richardson, [80](#)

build_r

CrankNicolson, [14](#)

Implicit, [30](#)

Laasonen, [41](#)

CRANK_NICHOLSON

utils.h, [121](#)

computational_time

Method, [66](#)

compute

Method, [62](#)

compute_norms

Method, [62](#)

compute_solution

Analytical, [10](#)

Explicit, [23](#)

Implicit, [31](#)

Method, [63](#)

CrankNicolson, [11](#)

build_r, [14](#)

CrankNicolson, [14](#)

create_output_dir

IOManager, [34](#)

DELTA_T_LASSONEN

utils.h, [121](#)

DELTA_T

utils.h, [121](#)

DELTA_X

utils.h, [121](#)

DIFUSIVITY

utils.h, [122](#)

DUFORT_FRANKEL

utils.h, [122](#)

default_deltat_times

IOManager, [37](#)

delta_t

Problem, [76](#)

delta_x

Problem, [76](#)

double_to_string

IOManager, [34](#)

DufortFrankel, [15](#)

build_iteration, [18](#)

DufortFrankel, [18](#)

error_tables

IOManager, [35](#)

Explicit, [19](#)

build_iteration, [22](#)

compute_solution, [23](#)

Explicit, [22](#)

export_outputs

IOManager, [35](#)

FORWARD_TIME_CENTRAL_SPACE

utils.h, [122](#)

FTCS, [24](#)

build_iteration, [26](#)

FTCS, [26](#)

find

Vector, [86](#)

get_computational_time

Method, [63](#)

get_deltat

Method, [63](#)

Problem, [70](#)

get_deltax

Problem, [70](#)

get_first_row

Problem, [71](#)

get_name

Method, [64](#)

get_solution

Method, [64](#)

Problem, [71](#)

get_tsize

Problem, [72](#)

get_tvalues

Problem, [72](#)

get_two_norm

Method, [65](#)

get_xsize

Problem, [73](#)

get_xvalues

Method, [65](#)

Problem, [73](#)

- getNcols
 - Matrix, 47
- getNrows
 - Matrix, 48
- getSize
 - Vector, 86
- grid/matrix.cpp, 93
- grid/matrix.h, 97
- grid/vector.cpp, 97
- grid/vector.h, 100
- INITIAL_TEMPERATURE
 - utils.h, 122
- IOManager, 33
 - create_output_dir, 34
 - default_deltat_times, 37
 - double_to_string, 34
 - error_tables, 35
 - export_outputs, 35
 - IOManager, 34
 - laasonen_times, 38
 - output_path, 38
 - plot_default_deltat_times, 36
 - plot_laasonen_times, 36
 - plot_solutions, 37
- Implicit, 27
 - build_r, 30
 - compute_solution, 31
 - Implicit, 30
 - thomas_algorithm, 31
- io/iomanager.cpp, 101
- io/iomanager.h, 102
- LAASONEN
 - utils.h, 122
- Laasonen, 38
 - build_r, 41
 - Laasonen, 41
- laasonen_times
 - IOManager, 38
- main
 - main.cpp, 104
- main.cpp, 103
 - main, 104
- Matrix, 42
 - getNcols, 47
 - getNrows, 48
 - Matrix, 46, 47
 - mult, 49
 - one_norm, 49
 - operator<<, 56, 57
 - operator>>, 57, 58
 - operator*, 50, 51
 - operator=, 52
 - operator==, 52
 - set_row, 53
 - transpose, 54
 - two_norm, 54
 - uniform_norm, 55
 - vec, 46
- matrix.cpp
 - operator<<, 93, 94
 - operator>>, 95, 96
- Method, 58
 - computational_time, 66
 - compute, 62
 - compute_norms, 62
 - compute_solution, 63
 - get_computational_time, 63
 - get_deltat, 63
 - get_name, 64
 - get_solution, 64
 - get_two_norm, 65
 - get_xvalues, 65
 - Method, 62
 - name, 66
 - one_norm, 66
 - problem, 66
 - q, 66
 - two_norm, 66
 - uniform_norm, 66
- methods/analytical.cpp, 104
- methods/analytical.h, 105
- methods/explicit/dufort_frankel.cpp, 106
- methods/explicit/dufort_frankel.h, 106
- methods/explicit/explicit.cpp, 107
- methods/explicit/explicit.h, 108
- methods/explicit/forward_t_central_s.cpp, 109
- methods/explicit/forward_t_central_s.h, 110
- methods/explicit/richardson.cpp, 111
- methods/explicit/richardson.h, 111
- methods/implicit/crank_nicolson.cpp, 112
- methods/implicit/crank_nicolson.h, 113
- methods/implicit/implicit.cpp, 114
- methods/implicit/implicit.h, 115
- methods/implicit/laasonen.cpp, 115
- methods/implicit/laasonen.h, 116
- methods/method.cpp, 117
- methods/method.h, 118
- mult
 - Matrix, 49
- NUMBER_OF_EXPANSIONS
 - utils.h, 122
- NUMBER_TIME_STEPS
 - utils.h, 122
- name
 - Method, 66
- nr_of_expansions
 - Analytical, 11
- OUTPUT_PATH
 - utils.h, 122
- one_norm
 - Matrix, 49
 - Method, 66
 - Vector, 87

- operator<<
 - Matrix, [56](#), [57](#)
 - matrix.cpp, [93](#), [94](#)
 - Vector, [90](#)
 - vector.cpp, [98](#)
- operator>>
 - Matrix, [57](#), [58](#)
 - matrix.cpp, [95](#), [96](#)
 - Vector, [91](#)
 - vector.cpp, [99](#), [100](#)
- operator*
 - Matrix, [50](#), [51](#)
- operator=
 - Matrix, [52](#)
 - Vector, [87](#)
- operator==
 - Matrix, [52](#)
 - Vector, [88](#)
- output_path
 - IOManager, [38](#)
- PI
 - utils.h, [123](#)
- plot_default_deltat_times
 - IOManager, [36](#)
- plot_laasonen_times
 - IOManager, [36](#)
- plot_solutions
 - IOManager, [37](#)
- Problem, [67](#)
 - delta_t, [76](#)
 - delta_x, [76](#)
 - get_deltat, [70](#)
 - get_deltax, [70](#)
 - get_first_row, [71](#)
 - get_solution, [71](#)
 - get_tsize, [72](#)
 - get_tvalues, [72](#)
 - get_xsize, [73](#)
 - get_xvalues, [73](#)
 - Problem, [69](#)
 - set_initial_conditions, [74](#)
 - set_t_values, [74](#)
 - set_time_step, [75](#)
 - set_x_values, [76](#)
 - solution, [76](#)
 - t_size, [76](#)
 - t_values, [76](#)
 - x_size, [77](#)
 - x_values, [77](#)
- problem
 - Method, [66](#)
- push
 - Vector, [88](#)
- push_front_back
 - Vector, [88](#)
- q
 - Method, [66](#)
- RICHARDSON
 - utils.h, [123](#)
- Richardson, [77](#)
 - build_iteration, [80](#)
 - Richardson, [80](#)
- SURFACE_TEMPERATURE
 - utils.h, [123](#)
- set_initial_conditions
 - Problem, [74](#)
- set_row
 - Matrix, [53](#)
- set_t_values
 - Problem, [74](#)
- set_time_step
 - Problem, [75](#)
- set_x_values
 - Problem, [76](#)
- solution
 - Problem, [76](#)
- t_size
 - Problem, [76](#)
- t_values
 - Problem, [76](#)
- THICKNESS
 - utils.h, [123](#)
- TIMELIMIT
 - utils.h, [123](#)
- thomas_algorithm
 - Implicit, [31](#)
- transpose
 - Matrix, [54](#)
- two_norm
 - Matrix, [54](#)
 - Method, [66](#)
 - Vector, [89](#)
- uniform_norm
 - Matrix, [55](#)
 - Method, [66](#)
 - Vector, [89](#)
- utils.h
 - ANALYTICAL, [121](#)
 - CRANK_NICHOLSON, [121](#)
 - DELTA_T_LASSONEN, [121](#)
 - DELTA_T, [121](#)
 - DELTA_X, [121](#)
 - DIFUSIVITY, [122](#)
 - DUFORT_FRANKEL, [122](#)
 - FORWARD_TIME_CENTRAL_SPACE, [122](#)
 - INITIAL_TEMPERATURE, [122](#)
 - LAASONEN, [122](#)
 - NUMBER_OF_EXPANSIONS, [122](#)
 - NUMBER_TIME_STEPS, [122](#)
 - OUTPUT_PATH, [122](#)
 - PI, [123](#)
 - RICHARDSON, [123](#)
 - SURFACE_TEMPERATURE, [123](#)

- THICKNESS, [123](#)
- TIMELIMIT, [123](#)
- variants/problem.cpp, [118](#)
- variants/problem.h, [119](#)
- variants/utls.h, [120](#)
- vec
 - Matrix, [46](#)
 - Vector, [84](#)
- Vector, [81](#)
 - find, [86](#)
 - getSize, [86](#)
 - one_norm, [87](#)
 - operator<<, [90](#)
 - operator>>, [91](#)
 - operator=, [87](#)
 - operator==, [88](#)
 - push, [88](#)
 - push_front_back, [88](#)
 - two_norm, [89](#)
 - uniform_norm, [89](#)
 - vec, [84](#)
 - Vector, [84](#), [85](#)
- vector.cpp
 - operator<<, [98](#)
 - operator>>, [99](#), [100](#)
- x_size
 - Problem, [77](#)
- x_values
 - Problem, [77](#)