

# COMPUTATIONAL METHODS

---

## Heat Conduction Equation

---

December 10, 2017

**António Pedro Araújo Fraga**

**Student ID: 279654**

**Cranfield University**

**M.Sc. in Software Engineering for Technical Computing**

# Contents

Introduction . . . . .	1
Problem definition . . . . .	2
Numerical analysis . . . . .	2
Procedures . . . . .	3
Explicit Schemes . . . . .	4
Richardson . . . . .	4
DuFort-Frankel . . . . .	5
Implicit Schemes . . . . .	5
Laasonen Simple Implicit . . . . .	6
Crank-Nicholson . . . . .	7
Solution Design . . . . .	8
Results & Discussion . . . . .	9
Laasonen Implicit Scheme: study of time step variation . . . . .	11
Conclusions . . . . .	13
References . . . . .	14
Appendices . . . . .	15
Richardson Method Analysis . . . . .	15
Accuracy . . . . .	15
Stability . . . . .	16
Doxygen Documentation . . . . .	16

### **Abstract**

Four numerical schemes were applied to compute a solution for a parabolic partial differential equation, the heat conduction equation. Two different types of schemes were used, explicit and implicit, and their solutions were evaluated. It could be observed the different behaviours of unstable and stable schemes. Step size variations of a stable method were studied as well. The obtained solutions were compared to the problem analytical solution in order to have a better understanding on these different behaviours.

**Table 1:** Nomenclature

Diffusivity	$D$
First derivative in time	$\frac{\partial f}{\partial t}$
First derivative in space	$\frac{\partial f}{\partial x}$
Time grid position	$n$
Space grid position	$i$
Function at time and space grid position	$f_i^n$
Time step	$\Delta t$
Space step	$\Delta x$
Time value	$t$
Space value	$x$
Analytical function at specific space and time values	$f(x, t)$
Initial Temperature	$T_{in}$
Surface Temperature	$T_{sur}$

## Introduction

Numerical methods are used to obtain an approximated solution to problems with no given analytical solution. These methods can be used in order to save computational time, therefore they can obtain results which are similar to the real solution more efficiently. Four different schemes were applied to compute an approximated solution to a **Parabolic Partial Differential Equation**, in this case the heat conduction equation.

$$\frac{\partial f}{\partial t} = D \frac{\partial^2 f}{\partial x^2}$$

This condition had to be satisfied on a grid in space and time, which means the problem has a structured mesh type, and therefore can be represented as a grid of two dimensions. The previous equation could be written in its discretized form for each method.

## Problem definition

A few initial or boundary conditions were set, including the heat conduction equation. An existing wall with **1 ft** thick had an initial temperature of **100°F** and the surface temperatures at both sides were suddenly increased and maintained to **300°F**. It is also known that the wall is composed of nickel steel (40% Ni) with a Diffusivity of **0.01 ft<sup>2</sup>/h**.

Since the wall has a 1 ft thickness, the problem space domain could be restricted between **0** and **1**, and the diffusivity value, which is considered constant, could be set to **0.01**. The time domain was restricted between **0** to **0.5**:

$$x \in [0, 1], t \in [0, 0.5]$$

$$T_{in} = 100, T_{sur} = 300$$

$$D = 0.01$$

The initial boundaries can be formalized in mathematical expressions:

$$f(x, 0) = T_{in}$$

$$f(0, t) = T_{sur}$$

$$f(1, t) = T_{sur}$$

The analytical solution of this problem was given by the following expression:

$$f(x, t) = T_{sur} + 2(T_{in} - T_{sur}) \sum_{m=1}^{m=\infty} e^{-Dt(m\pi/L)^2} \frac{1 - (-1)^m}{m\pi} \sin\left(\frac{m\pi x}{L}\right)$$

## Numerical analysis

Numerical analysis is the study of the obtained solution. Criticism is very important on this phase, since the solutions are evaluated. Digital computers have problems with round-off errors, and since values were truncated, problems with discretization errors may appear. There are some definitions related with this study: stability, convergence and approximation.

A method is declared stable if the error doesn't grow as time advances. Theoretically, conditions that make a scheme becomes stable or unstable can be known.

Approximation can be verified by comparing the computed solution with the analytical solution, and check if there is an approximation at all.

Convergence is defined by how well the computed solution approximates to the analytical solution. This can vary with a change in the number of **time steps** or **space steps**. A smaller number of steps can lead to a bigger error, whereas a bigger number of steps can lead to a considerably more time expensive solution. Every method could be developed using **Taylor Series**. This series were developed for **n** terms. Thus, every method has a given approximation factor, that could be represented in the **Big Oh** annotation. The error related with this approximation is called **Truncation Error**.

A quantitative analysis can be done by comparing solutions of each method. By calculating the **norms** of the error matrix, one can conclude which method is more accurate. The **error matrix** can be calculated by subtracting each cell of the **analytical** solution matrix to the cells of a method matrix. When the result is a matrix of small values, the error is small. Whenever a norm is calculated, one is able to translate an error matrix into a single value:

- **One Norm** - Which is given by adding the absolute values of each cell of the matrix.
- **Two Norm** - Which is obtained by adding the squares of every value in the matrix.
- **Uniform Norm** - Which represents the biggest error, or the maximum value in the matrix.

The second norm "punishes" the biggest values, and "regards" the lowest ones. Notice that a the square of a value between 0 and 1 is lower than the given value. Whereas the square of a value bigger than 1 is higher. Therefore, this norm is a good quality indicator.

A stencil could also be developed for each method, which relates the several grid points, revealing the dependencies for a calculation in a more graphical way.

## Procedures

Four different schemes/methods were used to compute a solution for the given problem, two of them are explicit schemes, **Richardson**, **DuFort-Frankel**, and two of them are implicit schemes, **Laasonen Simple Implicit** and **Crank-Nicholson**. The space step was maintained at **0.05 ft**, and the time step took the value of **0.01 h**, studying every solutions in intervals

of **0.1** hours from **0.0** to **0.5**. The **Laasonen Simple Implicit** solution was also studied with different time steps, always maintaining the same space step,  $\Delta x = 0.05$ :

- $\Delta t = 0.01$
- $\Delta t = 0.025$
- $\Delta t = 0.05$
- $\Delta t = 0.1$

As referred, considering the initial equation, these methods can be written in its discretized form.

## Explicit Schemes

This type of schemes rely only on the previous time steps to calculate the current time step solution. In the case of both used methods, they were relying in known values of the **n - 1** and **n** time steps to calculate a value for the **n + 1** time step. Thereby, the second time step can not be calculated by these methods, because there's no possible value for a negative time step. A different method, for the same equation, with two levels of time steps was used in order to overcome this situation, the **Forward in Time and Central in Space** scheme. It's known that this method is **conditionally stable**, and its stability condition is given by,

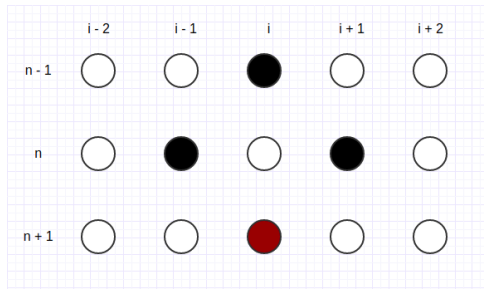
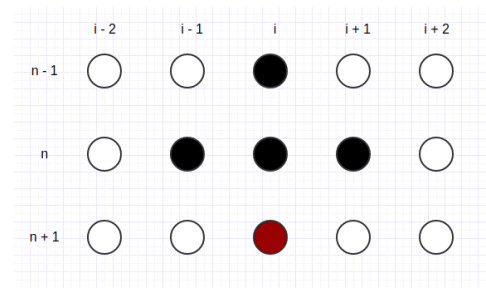
$$\frac{D\Delta t}{(\Delta x)^2} \leq 0.5$$

Therefore, considering  $\Delta t = 0.01$ ,  $\Delta x = 0.05$ , and  $D = 0.1$ , this method is declared stable. It's important to have a stable solution for the first iteration, since it is a major influence on the overall solution. Therefore, this iteration could be calculated with the following expression,

$$f_i^{n+1} = f_i^n + \frac{D\Delta t}{(\Delta x)^2} (f_{i+1}^n - 2f_i^n + f_{i-1}^n)$$

## Richardson

The Richardson method can be applied by having a central in time and central in space scheme. Regarding to stability issues, this method is unconditionally unstable. This method is of order  $\mathbf{O}(\Delta x^2, \Delta t^2)$ . Following the heat conduction equation, the expression can be written as following:

**Figure 1:** DuFort-Frankel's method stencil.**Figure 2:** Richardson's method stencil.

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t} = D \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2}$$

Which corresponds to,

$$f_i^{n+1} = f_i^{n-1} - \frac{2\Delta t D}{(\Delta x)^2} (f_{i+1}^n - 2f_i^n + f_{i-1}^n)$$

### DuFort-Frankel

The DuFort-Frankel scheme can be applied by having central differences in both derivatives, but to prevent stability issues, the space derivative term  $f_i^n$  can be written as the average value of  $f_i^{n+1}$  and  $f_i^{n-1}$ . Therefore this method is of order  $\mathcal{O}(\Delta x^2, \Delta t^2, (\frac{\Delta t}{\Delta x})^2)$  and is declared as unconditionally stable and it may be formulated as follows:

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t} = D \frac{f_{i+1}^n - f_i^{n+1} - f_i^{n-1} + f_{i-1}^n}{(\Delta x)^2}$$

Which is equivalent to,

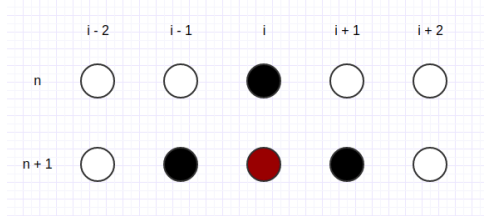
$$f_i^{n+1} = f_i^{n-1} - \frac{2\Delta t D}{(\Delta x)^2} (f_{i+1}^n - f_i^{n+1} - f_i^{n-1} + f_{i-1}^n)$$

### Implicit Schemes

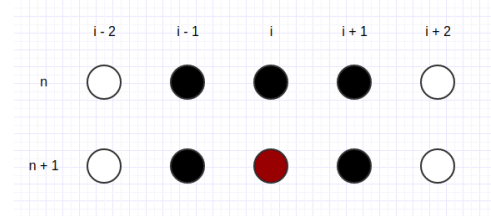
In other hand, implicit schemes rely not only on lower time steps to calculate a solution, but also on the current time step known values. Each time step solution can often be solved by applying the Thomas Algorithm, which is an algorithm that can solve tridiagonal matrix systems,  $Ax = r$ . This algorithm is a special case of the LU decomposition, with a better performance. The matrix  $A$  can be decomposed in a lower triangular matrix  $L$  and an upper triangular matrix  $U$ , therefore  $A = LU$ . This algorithm consists of two steps, the downwards



phase where the equation  $Lp = r$  is solved and the upwards phase, solving  $Ux = p$ , obtaining a solution for  $x$ .



**Figure 3:** Laasonen's method stencil.



**Figure 4:** Crank-Nicholson's method stencil.

### Laasonen Simple Implicit

The time derivative is considered forward in time. Central difference is used in space derivative, and the scheme is of order  $\mathcal{O}(\Delta x, \Delta t^2)$ , and unconditionally stable. Concluding, the below equation could be established:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = D \frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{(\Delta x)^2}$$

Assuming that  $c = \frac{\Delta t D}{(\Delta x)^2}$ , the equation could be represented as:

$$(1 - 2c)f_i^{n+1} = f_i^n + c[f_{i+1}^{n+1} + f_{i-1}^{n+1}]$$

The values of the first and last space position of each time step are known, they are represent by the  $T_{sur}$  value. Therefore, in every second and penultimate space step, two terms of the previous equation could be successfully inquired. For the second space step, the equation could be divided by having the unknown terms in the left side and the known terms in the right side:

$$(1 - 2c)f_i^{n+1} - cf_{i+1}^{n+1} = f_i^n + cf_{i-1}^{n+1}$$

And the same could be done for the penultimate space step:

$$(1 - 2c)f_i^{n+1} - cf_{i-1}^{n+1} = f_i^n + cf_{i+1}^{n+1}$$

For every other space steps with unknown values, the expression could be generalized as:

$$(1 - 2c)f_i^{n+1} - c[f_{i+1}^{n+1} + f_{i-1}^{n+1}] = f_i^n$$

Considering that the maximum number of space steps is  $\mathbf{m}$ , the previous expressions could form a system of linear equations,  $A.x = r$ :

$$\begin{bmatrix} (1-2c) & -c & 0 & 0 & \dots & 0 & 0 \\ -c & (1-2c) & -c & 0 & \dots & 0 & 0 \\ 0 & -c & (1-2c) & -c & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -c & (1-2c) \end{bmatrix} \begin{bmatrix} f_1^{n+1} \\ f_2^{n+1} \\ f_3^{n+1} \\ \dots \\ f_{\mathbf{m}-1}^{n+1} \end{bmatrix} = \begin{bmatrix} f_1^n + cf_0^{n+1} \\ f_2^n \\ f_3^n \\ \dots \\ f_{\mathbf{m}-1}^n + cf_{\mathbf{m}}^{n+1} \end{bmatrix}$$

### Crank-Nicholson

The time derivative is considered forward in time, and the space derivative can be replaced by the average of central differences in time steps  $\mathbf{n}$  and  $\mathbf{n} + 1$ . The method is of order  $\mathbf{O}(\Delta x^2, \Delta t^2)$  and declared unconditionally stable. Thus:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = \frac{1}{2} D \left[ \frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{(\Delta x)^2} + \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2} \right]$$

In this method, the coefficient had a new value,  $c = \frac{1}{2} \frac{\Delta t D}{(\Delta x)^2}$ , and assuming that  $p = f_{i+1}^n + f_{i-1}^n$ , the equation could be written as follows,

$$(1 - 2c)f_i^{n+1} = (1 - 2c)f_i^n + c[f_{i+1}^{n+1} + f_{i-1}^{n+1} + p]$$

Following the same logical principles of the previous scheme, some expressions could be generalized for the second,

$$(1 - 2c)f_i^{n+1} - cf_{i+1}^{n+1} = (1 - 2c)f_i^n + c[f_{i-1}^{n+1} + p]$$

, penultimate,

$$(1 - 2c)f_i^{n+1} - cf_{i+1}^{n+1} = (1 - 2c)f_i^n + c[f_{i+1}^{n+1} + p]$$

, and every other space steps with unknown values.

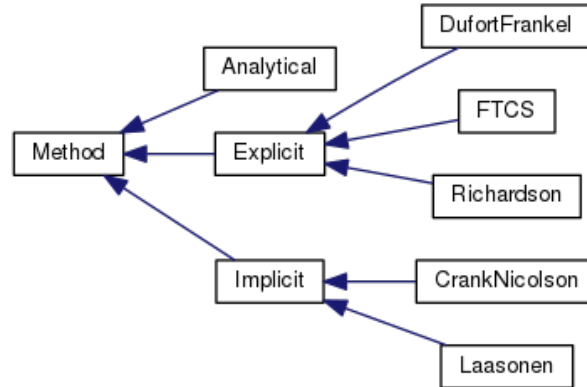
$$(1 - 2c)f_i^{n+1} - c[f_{i+1}^{n+1} + f_{i-1}^{n+1}] = (1 - 2c)f_i^n + cp$$

Thus, a tridiagonal matrix system is obtained,

$$\begin{bmatrix} (1-2c) & -c & 0 & 0 & \dots & 0 & 0 \\ -c & (1-2c) & -c & 0 & \dots & 0 & 0 \\ 0 & -c & (1-2c) & -c & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -c & (1-2c) \end{bmatrix} \begin{bmatrix} f_1^{n+1} \\ f_2^{n+1} \\ f_3^{n+1} \\ \dots \\ f_{m-1}^{n+1} \end{bmatrix} = \begin{bmatrix} (1-2c)f_1^n + c[f_0^{n+1} + p] \\ (1-2c)f_2^n + cp \\ (1-2c)f_3^n + cp \\ \dots \\ (1-2c)f_{m-1}^n + c[f_m^{n+1} + p] \end{bmatrix}$$

## Solution Design

The code was first planned with an initial structure and suffered incremental upgrades. A **method** class was created, being a prototype with multiple inheritance, containing three sub classes: **Analytical**, **Implicit** and **Explicit**. Therefore, the **Implicit** class is an Abstract class as well. This class has three sub classes, representing the three explicit methods used in this problem. Similarly, the **Implicit** class is also an abstract class, having two implicit methods classes as sub classes. The previously described inheritance structure can be more easily visualized on **Figure 5**.



**Figure 5:** Method Class inheritance diagram.

A **Method** class contains a **Problem** object. The **Problem** class represents the Heat Conduction problem, containing informations about the time and space steps, the solution and initial conditions.

An **Input and Output Manager** class was developed so that the code related with plots and tables exportations could be separated from the logical source code. This class was developed with several methods regarding data interpretation and structuration in order to easily export plot charts. A **gnuplot c++ library** was used, therefore the gnuplot syntax could

be directly used from the c++ code, cutting down the need of developing external bash scripts for this specific purpose.

Despite the referred classes, a header file with useful **macros** was declared. This file contains information about which conditions to test, like the initial temperature and the surface temperature. Therefore, if for some reason, one of this values changes, it can be easily corrected.

The **Matrix** and **Vector** classes, which were provided in the c++ module were reused to represent a solution matrix or a solution vector of a certain iteration.

The several objects in this structure could be instantiated in the main file, calling methods to compute the several solutions and to export their plot charts.

## Results & Discussion

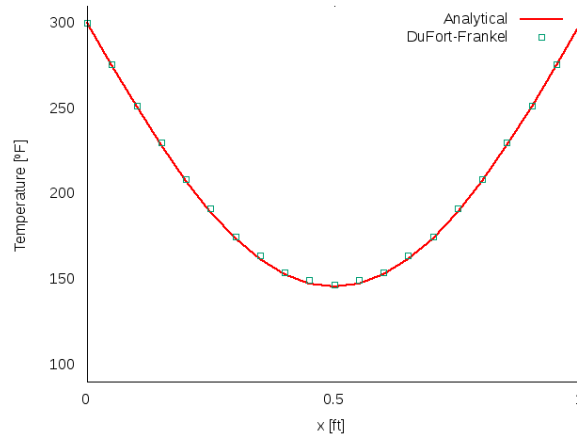
The results of the four methods, **Richardson**, **DuFort-Frankel**, **Laasonen Simple Implicit** and **Crank-Nicholson** can be seen in the following figures/tables. These results were used to analyze each solution quantitatively and qualitatively. In most of the plot charts, the obtained solution was compared to the analytical solution so that it would be possible to realize whether the solution was a good approximation or not. Notice that the next results are regarding to the "default" values of time and space steps,  $\Delta t = 0.01$  and  $\Delta x = 0.05$ .

**Table 2:** Richardson method error table.

$\begin{matrix} x \\ t \end{matrix}$	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
0.1	1.05136e+06	631856	123707	8417.73	300.81	8417.73	123707	631856	1.05136e+06
0.2	1.33245e+11	1.39e+11	7.02854e+10	2.06123e+10	6.93136e+09	2.06123e+10	7.02854e+10	1.39e+11	1.33245e+11
0.3	2.14659e+16	2.74969e+16	1.97012e+16	9.88337e+15	5.98161e+15	9.88337e+15	1.97012e+16	2.74969e+16	2.14659e+16
0.4	3.91917e+21	5.60267e+21	4.87086e+21	3.31281e+21	2.58429e+21	3.31281e+21	4.87086e+21	5.60267e+21	3.91917e+21
0.5	7.74272e+26	1.19047e+27	1.18021e+27	9.72231e+26	8.60626e+26	9.72231e+26	1.18021e+27	1.19047e+27	7.74272e+26

By examining **Table 2**, it could be concluded that the solution given by the Richardson method was considerably different from the analytical solution. This was due to the fact that this method is declared as **unconditionally unstable**. As referred before, when a method is declared unstable, the error grows as the time advances. The error growth was responsible

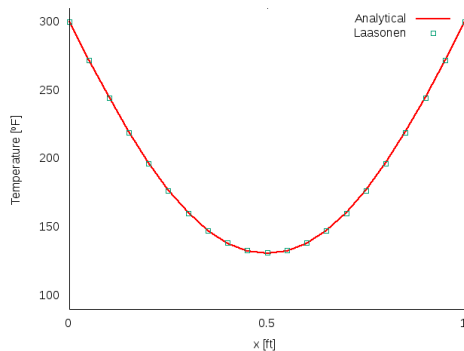
for obtaining a different solution, or a solution to a different problem. The mathematical calculations regarding the stability and accuracy properties of this method can be found under the appendix section.



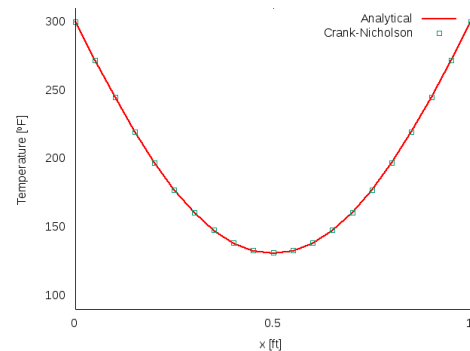
**Figure 6:** DuFort-Frankel's solution at  $t = 0.5$ .

When looking at **Figure 6**, it can be observed that the DuFort-Frankel solution is quite approximated to the real solution. This scheme, as it could be observed at **Figure 10**, is more time efficient comparing to the implicit unconditionally stable methods, the only disadvantage is the fact that it requires a different method for the first iteration.

Similarly of what could be concluded on DuFort-Frankel results, by observing **Figure 7** and **Figure 8**, it can also be deducted that these are good solutions. These schemes, Crank-Nicholson and Laasonen, are unconditionally stable as well. Therefore good results were expected.



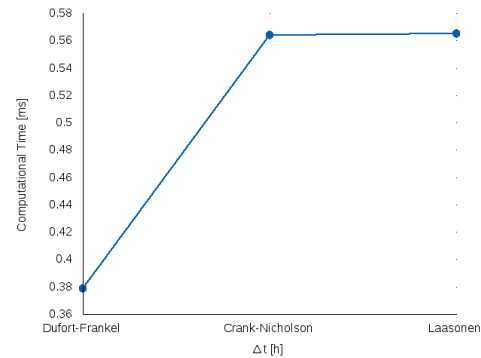
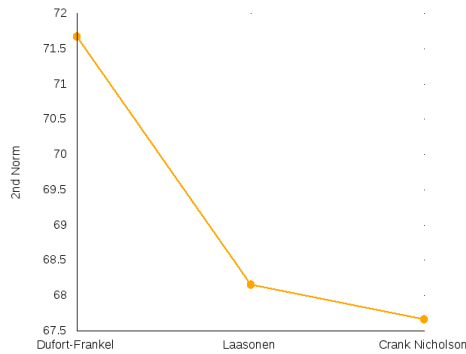
**Figure 7:** Laasonen's solution at  $t = 0.4$ .



**Figure 8:** Crank-Nicholson's solution at  $t = 0.4$ .

In other hand, when a quantitative analysis was done, it could be seen that the Crank Nicholson scheme is more accurate than the Laasonen and DuFort-Frankel methods. By

looking at **Figure 9**, it can be observed that the second norm value of the **Error matrix** of this scheme is smaller than the values obtained by the other methods **Error Matrices**.



**Figure 9:** 2nd norm values of Error Matrices. **Figure 10:** Computational times of stable methods.

### Laasonen Implicit Scheme: study of time step variation

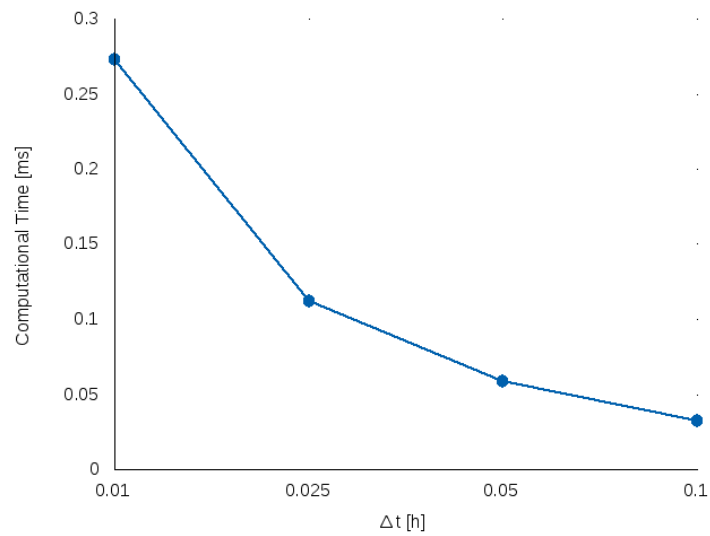
Laasonen Implicit Scheme is an unconditionally stable scheme to solve Parabolic Partial Differential Equations. Therefore, with the right time and space step, there's almost no error related to the development of its results throughout the time advancement.

A reduction on these steps led to a higher computational time, since there's more calculations to be made. Whereas steps with higher values led to more inaccurate results. This phenomenon could be explained with a concept that was introduced earlier, the **truncation error**. This error can only be avoided with exact calculations, but can be reduced by applying a larger number of smaller intervals or steps. As referred before, different results of this method were studied by changing the time step size. The space step was maintained,  $\Delta x = 0.05$ .

**Table 3:** Laasonen method error table for the several  $\Delta t$  at  $t = 0.5$

$\Delta t \backslash x$	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
0.01	0.288694	0.385764	0.255427	0.0405061	-0.0611721	0.0405061	0.255427	0.385764	0.288694
0.025	0.738044	1.0344	0.805442	0.368491	0.157551	0.368491	0.805442	1.0344	0.738044
0.05	1.53627	2.15669	1.71375	0.864487	0.457364	0.864487	1.71375	2.15669	1.53627
0.1	3.29955	4.49523	3.46045	1.7082	0.898726	1.7082	3.46045	4.49523	3.29955

**Table 3** and **figure 11** could support the previous affirmations. While observing **table 3**, it could be seen that the error is larger for bigger time steps, as it was expected. Whereas when observing **figure 11**, it can be identified a reduction in computational time as the **time step** becomes larger.



**Figure 11:** Laasonen method computational times for the several  $\Delta t$ .

# Conclusions

The obtained results could support the theoretical concepts. Unstable methods demonstrated an error growth through the time progress. The **Forward in Time, Central in Space** explicit scheme was stable with the given initial conditions, therefore it could support a good solution for the explicit stable scheme, **DuFort-Frankel**. As referred, the solution of the DuFort-Frankel method strongly depends on the first iteration solution.

It could be observed that smaller steps can lead to a time expensive solution, whereas larger steps lead to an error increase. Stable methods could give a good solution with the right time and space steps, but by analysing the second norm value, it was concluded that the Crank-Nicholson method is more accurate. This is due to the fact that this method has a better approximation order.

It is important to have a balance between the two problems (time and approximation), a method should be computed in an acceptable time, and still obtain a good result. In realistic scenarios the problem solution is not known, therefore error estimates are impractical. The used step size should be small as possible, as long as the solution is not dominated with round-off errors. The solution must be obtained with a number of steps that one has time to compute.



# References

Gilberto E. Urroz, July 2004, *Convergence, Stability, and Consistency of Finite Difference Schemes in the Solution of Partial Differential Equations*, Available at: <[http://ocw.usu.edu/Civil\\_and\\_Environmental\\_Engineering/Numerical\\_Methods\\_in\\_Civil\\_Engineering/StabilityNumericalSchemes.pdf](http://ocw.usu.edu/Civil_and_Environmental_Engineering/Numerical_Methods_in_Civil_Engineering/StabilityNumericalSchemes.pdf)> [Accessed 2 October 2017]

S. Scott Collis, April 26, 2005, *An Introduction to Numerical Analysis for Computational Fluid Mechanics*, [Accessed 2 October 2017]

Klaus A. Hoffman, Steve T. Chiang, August 2000, *Computational Fluid Dynamics, Volume 1* [Accessed 27 October 2017]

Richard H. Pletcher, Jhon C. Tannehill, Dale A. Anderson, 2013, *Computational Fluid Mechanics and Heat Transfer, Third Edition*, [Accessed 27 October 2017]

W. T. Lee, *Tridiagonal Matrices: Thomas Algorithm*, Available at: <[https://www3.ul.ie/wlee/ms6021\\_thomas.pdf](https://www3.ul.ie/wlee/ms6021_thomas.pdf)> [Accessed 28 October 2017]

*Error in Euler's Method*, Available at: <<http://www.math.unl.edu/~gledder1/Math447/EulerError>> [Accessed 2 November 2017]

John D. Cook, February 22 2008, *Step size for numerical differential equations*, Available at: <<https://www.johndcook.com/NumericalODEStepSize.pdf>> [Accessed 3 November 2017]

B.J.P. Kaus, *Explicit versus implicit Finite Difference Schemes*, Available at: <[https://www.geowiss.uni-mainz.de/Dateien/Finite\\_Difference\\_Timpl\\_expl.pdf](https://www.geowiss.uni-mainz.de/Dateien/Finite_Difference_Timpl_expl.pdf)> [Accessed 4 November 2017]

Markus Schmuck, *Numerical Methods for PDEs*, Available at: <[http://www.macs.hw.ac.uk/~ms713/lecture\\_9.pdf](http://www.macs.hw.ac.uk/~ms713/lecture_9.pdf)> [Accessed 10 November 2017]

# Appendices

## Richardson Method Analysis

### Accuracy

Every term can be developed with Taylor expansions:

$$f_i^{n+1} = f_i^n + \Delta t \left( \frac{\partial f}{\partial t} \right)_i^n + \frac{\Delta t^2}{2} \left( \frac{\partial^2 f}{\partial t^2} \right)_i^n + O(\Delta t^2)$$

$$f_i^{n-1} = f_i^n - \Delta t \left( \frac{\partial f}{\partial t} \right)_i^n + \frac{\Delta t^2}{2} \left( \frac{\partial^2 f}{\partial t^2} \right)_i^n + O(\Delta t^2)$$

$$f_{i+1}^n = f_i^n + \Delta x \left( \frac{\partial f}{\partial x} \right)_i^n + \frac{\Delta x^2}{2} \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n + O(\Delta x^2)$$

$$f_{i-1}^n = f_i^n - \Delta x \left( \frac{\partial f}{\partial x} \right)_i^n + \frac{\Delta x^2}{2} \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n + O(\Delta x^2)$$

By replacing every term expansion in the **Richardson's Method equation**, one should obtain the **Heat Conduction Equation**.

Starting by the left side of the **Richardson's Method equation**,

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t}$$

it can be converted into,

$$\frac{f_i^n + \Delta t \left( \frac{\partial f}{\partial t} \right)_i^n + \frac{\Delta t^2}{2} \left( \frac{\partial^2 f}{\partial t^2} \right)_i^n + O(\Delta t^2) - f_i^n + \Delta t \left( \frac{\partial f}{\partial t} \right)_i^n - \frac{\Delta t^2}{2} \left( \frac{\partial^2 f}{\partial t^2} \right)_i^n + O(\Delta t^2)}{2\Delta t}$$

which can be translated to,

$$\left( \frac{\partial f}{\partial t} \right)_i^n + O(\Delta t^2)$$

When looking at the right side of the equation,

$$D \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2}$$

one can replace the terms by their expansions as well,

$$D \frac{f_i^n + \Delta x \left( \frac{\partial f}{\partial x} \right)_i^n + \frac{\Delta x^2}{2} \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n + O(\Delta x^2) - 2f_i^n + f f_i^n - \Delta x \left( \frac{\partial f}{\partial x} \right)_i^n + \frac{\Delta x^2}{2} \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n + O(\Delta x^2)}{(\Delta x)^2}$$

which can be translated to,

$$D \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n + O(\Delta x^2)$$

Therefore the obtained equation is of order of  $O(\Delta x^2, \Delta t^2)$ ,

$$\left( \frac{\partial f}{\partial t} \right)_i^n - D \left( \frac{\partial^2 f}{\partial x^2} \right)_i^n = O(\Delta x^2, \Delta t^2)$$

### Stability

The Richardson Scheme is given by,

$$\frac{f_i^{n+1} - f_i^{n-1}}{2\Delta t} = D \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{(\Delta x)^2}$$

Using the von-Neumann analysis one can write:

$$f_i^n = \xi^n \times e^{j\omega i}$$

and by inserting this expression in the scheme, assuming that  $q = D \frac{\Delta t}{(\Delta x)^2}$ , one obtains,

$$\xi^2 + 8q \sin^2\left(\frac{1}{2}\omega\right)\xi - 1 = 0$$

which is a quadratic expression, thus it can be solved with,

$$\xi = -4q \sin^2 \frac{\omega}{2} \pm \sqrt{1 + 16q^2 \sin^4 \frac{\omega}{2}}$$

assuming that  $r = 4q \sin^2 \frac{\omega}{2}$ :

$$\xi = -r \pm \sqrt{1 + r^2}$$

one will obtain  $|\xi| > 1$  when  $r > 0$ , and because stability requires that  $|\xi| \leq 1$ , the Richardson scheme is **unconditionally unstable**.

Heat conduction equation

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	Analytical Class Reference . . . . .	7
4.1.1	Detailed Description . . . . .	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	<a href="#">one_norm() const</a>	49
4.9.4.5	<a href="#">operator*(const Matrix &amp;a) const</a>	50
4.9.4.6	<a href="#">operator*(const Vector &amp;v) const</a>	51
4.9.4.7	<a href="#">operator=(const Matrix &amp;m)</a>	52
4.9.4.8	<a href="#">operator==(const Matrix &amp;m) const</a>	52
4.9.4.9	<a href="#">set_row(int index, Vector v)</a>	53
4.9.4.10	<a href="#">transpose() const</a>	54
4.9.4.11	<a href="#">two_norm() const</a>	55
4.9.4.12	<a href="#">uniform_norm() const</a>	55
4.9.5	<a href="#">Friends And Related Function Documentation</a>	56
4.9.5.1	<a href="#">operator&lt;&lt;</a>	56
4.9.5.2	<a href="#">operator&lt;&lt;</a>	57
4.9.5.3	<a href="#">operator&gt;&gt;</a>	57
4.9.5.4	<a href="#">operator&gt;&gt;</a>	58
4.10	<a href="#">Method Class Reference</a>	58
4.10.1	<a href="#">Detailed Description</a>	61
4.10.2	<a href="#">Constructor &amp; Destructor Documentation</a>	62
4.10.2.1	<a href="#">Method()</a>	62
4.10.2.2	<a href="#">Method(Problem problem)</a>	62
4.10.3	<a href="#">Member Function Documentation</a>	62
4.10.3.1	<a href="#">compute()</a>	62
4.10.3.2	<a href="#">compute_norms(Matrix analytical_matrix)</a>	63
4.10.3.3	<a href="#">compute_solution()=0</a>	63
4.10.3.4	<a href="#">get_computational_time()</a>	63
4.10.3.5	<a href="#">get_deltat()</a>	64
4.10.3.6	<a href="#">get_name()</a>	64
4.10.3.7	<a href="#">get_solution()</a>	64
4.10.3.8	<a href="#">get_two_norm()</a>	65
4.10.3.9	<a href="#">get_xvalues()</a>	65
4.10.4	<a href="#">Member Data Documentation</a>	66

4.10.4.1	<code>computational_time</code>	66
4.10.4.2	<code>name</code>	66
4.10.4.3	<code>one_norm</code>	66
4.10.4.4	<code>problem</code>	66
4.10.4.5	<code>q</code>	66
4.10.4.6	<code>two_norm</code>	66
4.10.4.7	<code>uniform_norm</code>	66
4.11	Problem Class Reference	67
4.11.1	Detailed Description	69
4.11.2	Constructor & Destructor Documentation	69
4.11.2.1	<code>Problem()</code>	69
4.11.2.2	<code>Problem(double dt, double dx)</code>	69
4.11.3	Member Function Documentation	70
4.11.3.1	<code>get_deltat()</code>	70
4.11.3.2	<code>get_deltax()</code>	71
4.11.3.3	<code>get_first_row()</code>	71
4.11.3.4	<code>get_solution()</code>	72
4.11.3.5	<code>get_tsize()</code>	72
4.11.3.6	<code>get_tvalues()</code>	72
4.11.3.7	<code>get_xsize()</code>	73
4.11.3.8	<code>get_xvalues()</code>	73
4.11.3.9	<code>set_initial_conditions()</code>	74
4.11.3.10	<code>set_t_values()</code>	74
4.11.3.11	<code>set_time_step(Vector step, double time)</code>	75
4.11.3.12	<code>set_x_values()</code>	76
4.11.4	Member Data Documentation	76
4.11.4.1	<code>delta_t</code>	76
4.11.4.2	<code>delta_x</code>	76
4.11.4.3	<code>solution</code>	76
4.11.4.4	<code>t_size</code>	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 &amp;v)</code>	85
4.13.3.4	<code>Vector(std::vector&lt; double &gt; 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 &amp;v)</code>	87
4.13.4.5	<code>operator==(const Vector &amp;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&lt;&lt;</code>	90
4.13.5.2	<code>operator&lt;&lt;</code>	90
4.13.5.3	<code>operator&gt;&gt;</code>	91
4.13.5.4	<code>operator&gt;&gt;</code>	91

<b>5 File Documentation</b>	<b>93</b>
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:

<a href="#">Analytical</a>	An <a href="#">Analytical</a> class to compute the solution with standard procedures The implementation is derived from the <a href="#">Method</a> Object . . . . .	7
<a href="#">CrankNicolson</a>	A <a href="#">CrankNicolson</a> method class that contains a r vector builder . . . . .	11
<a href="#">DufortFrankel</a>	A <a href="#">DufortFrankel</a> method class that contains an iteration builder . . . . .	15
<a href="#">Explicit</a>	An explicit method class that contains default methods that only explicit methods use The implementation is derived from the <a href="#">Method</a> class . . . . .	19
<a href="#">FTCS</a>	A <a href="#">FTCS</a> method class that contains an iteration builder . . . . .	24
<a href="#">Implicit</a>	An implicit method class that contains default methods that only implicit methods use The implementation is derived from the <a href="#">Method</a> class . . . . .	27
<a href="#">IOManager</a>	An input/output manager class to handle plot exportations and future implementations of input handling . . . . .	33
<a href="#">Laasonen</a>	A <a href="#">Laasonen</a> method class that contains a r vector builder . . . . .	38
<a href="#">Matrix</a>	A matrix class for data storage of a 2D array of doubles The implementation is derived from the standard container vector std::vector We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [] . . . . .	42
<a href="#">Method</a>	A <a href="#">Method</a> class to structure information used to solve the problem . . . . .	58
<a href="#">Problem</a>	A <a href="#">Problem</a> class to structure relevant information related with the problem . . . . .	67
<a href="#">Richardson</a>	A <a href="#">Richardson</a> method class that contains an iteration builder . . . . .	77
<a href="#">Vector</a>	A vector class for data storage of a 1D array of doubles The implementation is derived from the standard container vector std::vector We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [] . . . . .	81





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

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



## Chapter 4

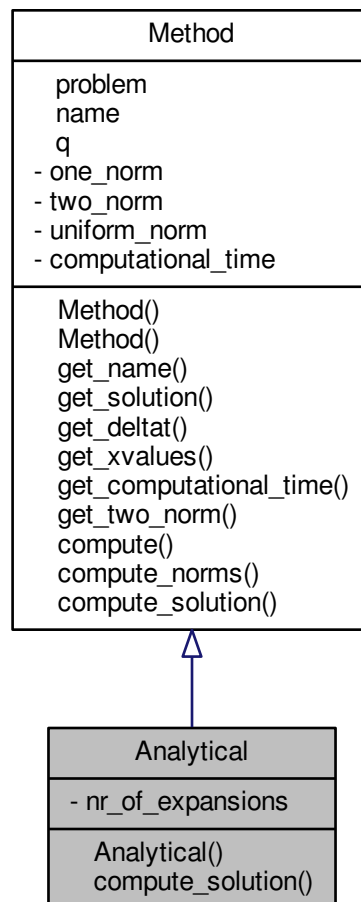
# Class Documentation

### 4.1 Analytical Class Reference

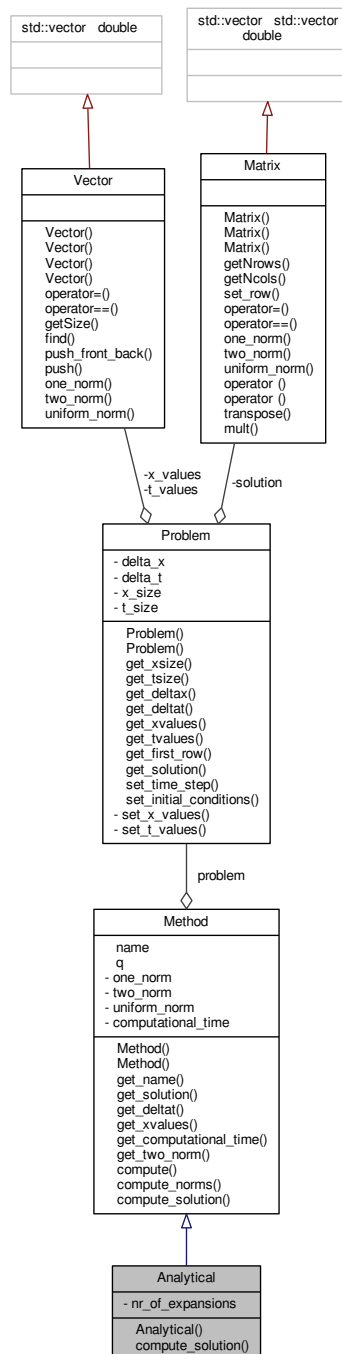
An [Analytical](#) class to compute the solution with standard procedures  
The implementation is derived from the [Method](#) Object.

```
#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

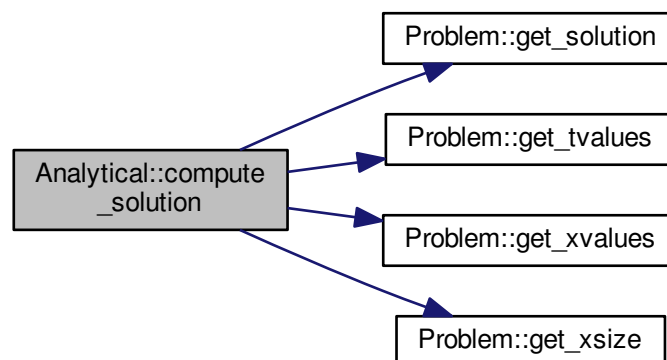
#### 4.1.3.1 void [Analytical](#)::[compute\\_solution](#) ( ) [[virtual](#)]

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](#)

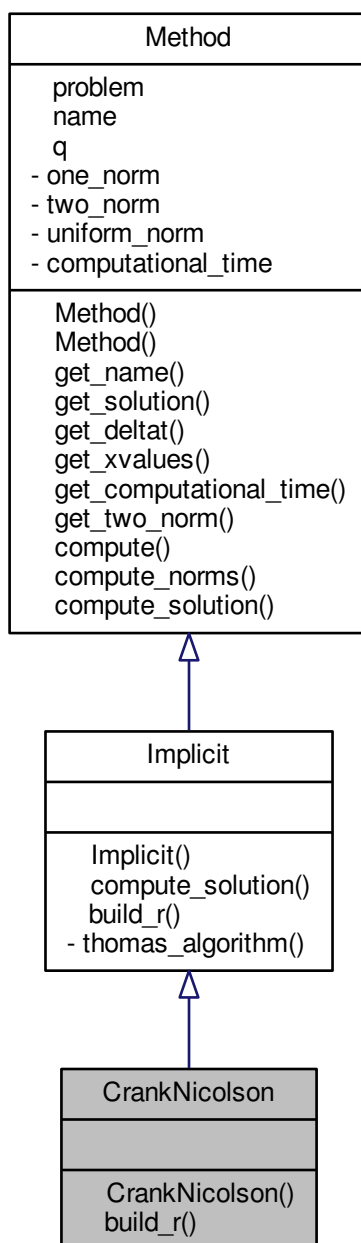
## 4.2 CrankNicolson Class Reference

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

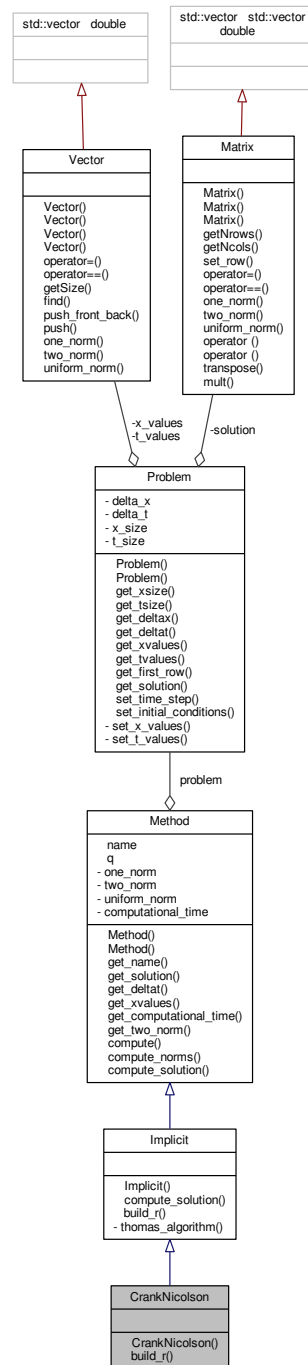
```
#include <crank_nicolson.h>
```



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

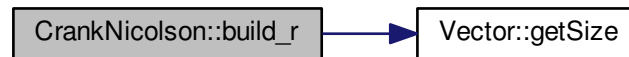
<i>previous_step</i>	<a href="#">Vector</a> representing the solution of the previous time step.
----------------------	---

#### Returns

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

Implements [Implicit](#).

Here is the call graph for this function:



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

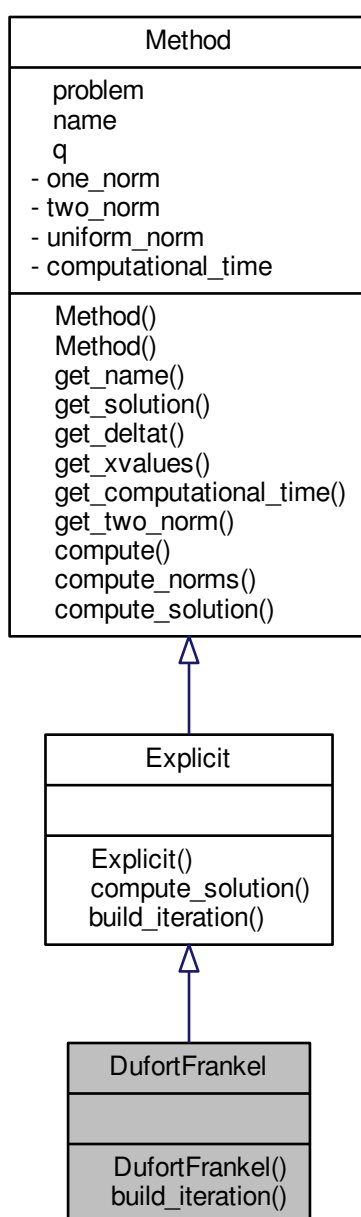
- [methods/implicit/crank\\_nicolson.h](#)
- [methods/implicit/crank\\_nicolson.cpp](#)

## 4.3 DufortFrankel Class Reference

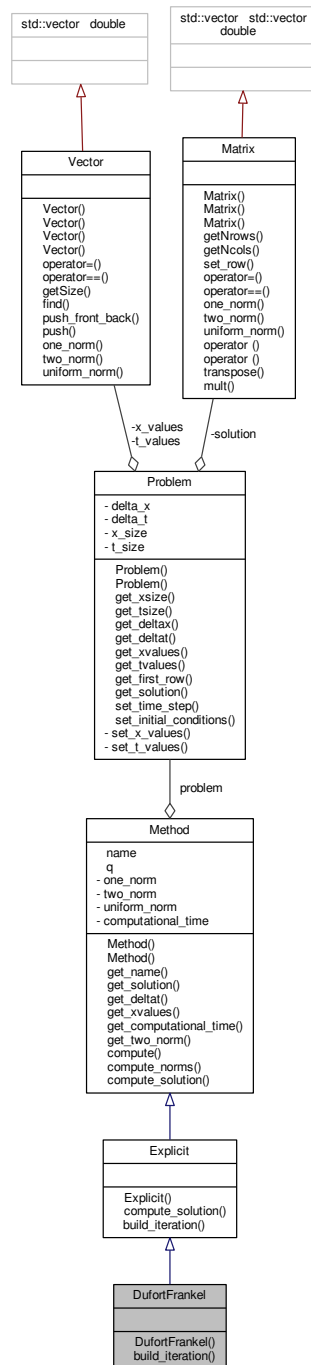
A [DufortFrankel](#) method class that contains an iteration builder.

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

Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

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

## Additional Inherited Members

### 4.3.1 Detailed Description

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

This builder is used to calculate a solution using the Dufort-Frankel method.

The [DufortFrankel](#) class provides:

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

### 4.3.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.3.3 Member Function Documentation

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

Normal protected method.

Calculate a next time step solution requiring a previous time step and a current time step solution.

#### Parameters

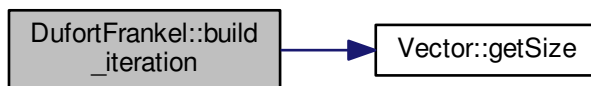
<i>current_step</i>	A vector representing the current time step solution.
<i>previous_step</i>	A vector representing the previous time step solution.

#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

Here is the call graph for this function:



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

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

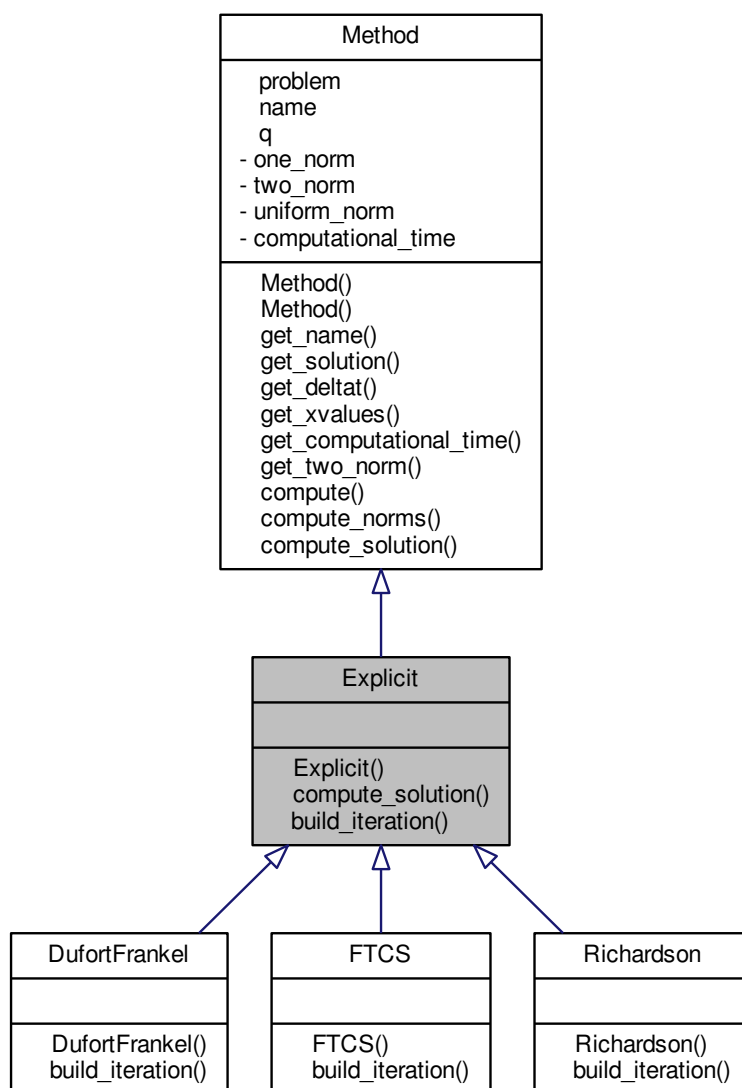
## 4.4 Explicit Class Reference

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

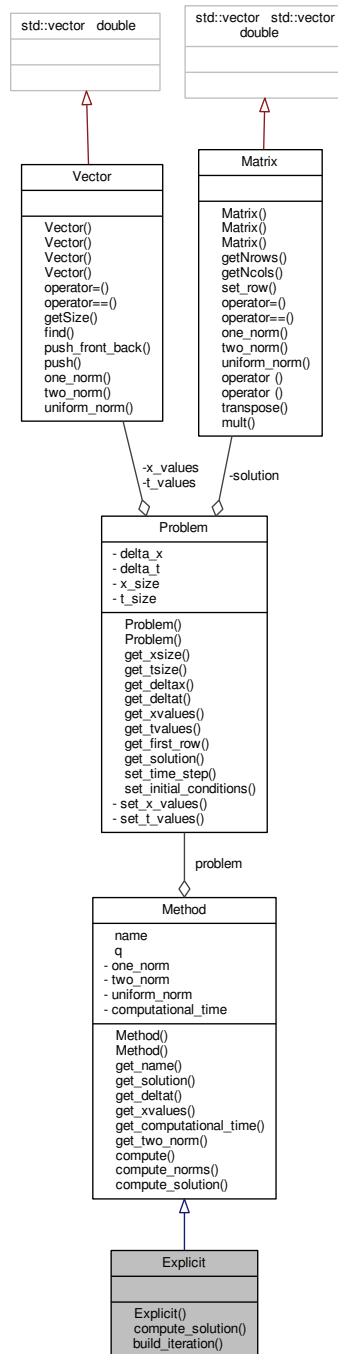
```
#include <explicit.h>
```



Inheritance diagram for Explicit:



Collaboration diagram for Explicit:



## Public Member Functions

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

## Protected Member Functions

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

## Additional Inherited Members

### 4.4.1 Detailed Description

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

The [Explicit](#) class provides:

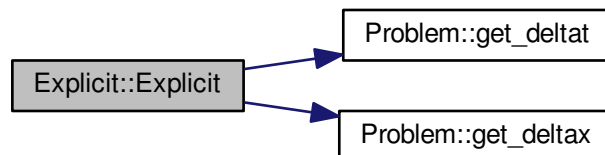
- a basic constructor for creating an explicit method object.
- a method to compute a solution following explicit methods rules

### 4.4.2 Constructor & Destructor Documentation

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

Default constructor.

Here is the call graph for this function:



### 4.4.3 Member Function Documentation

#### 4.4.3.1 virtual [Vector](#) [Explicit::build\\_iteration](#) ( [Vector](#) *current\_step*, [Vector](#) *previous\_step* ) [protected], [pure virtual]

A pure virtual member.

Build the solution of the next time step, using the previous time step and the next time step solutions

#### Parameters

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

## Returns

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

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

Here is the caller graph for this function:



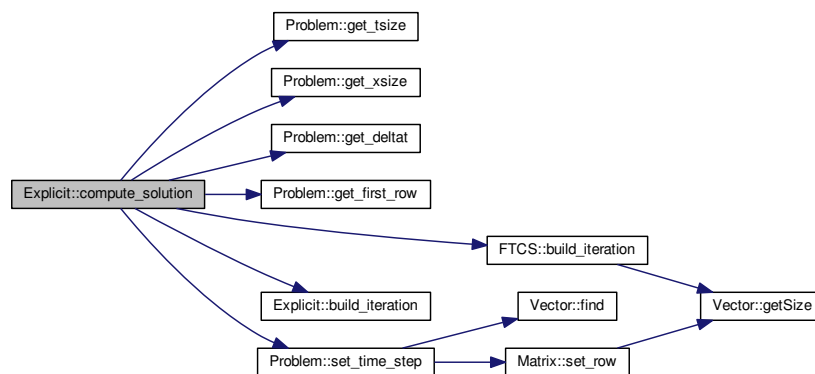
#### 4.4.3.2 void Explicit::compute\_solution ( ) [virtual]

Normal public method.

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

Implements [Method](#).

Here is the call graph for this function:



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

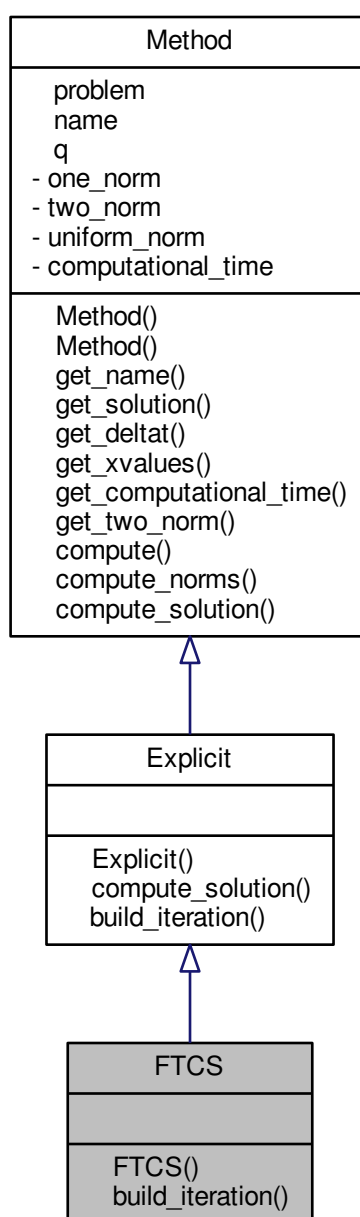
- [methods/explicit/explicit.h](#)
- [methods/explicit/explicit.cpp](#)

## 4.5 FTCS Class Reference

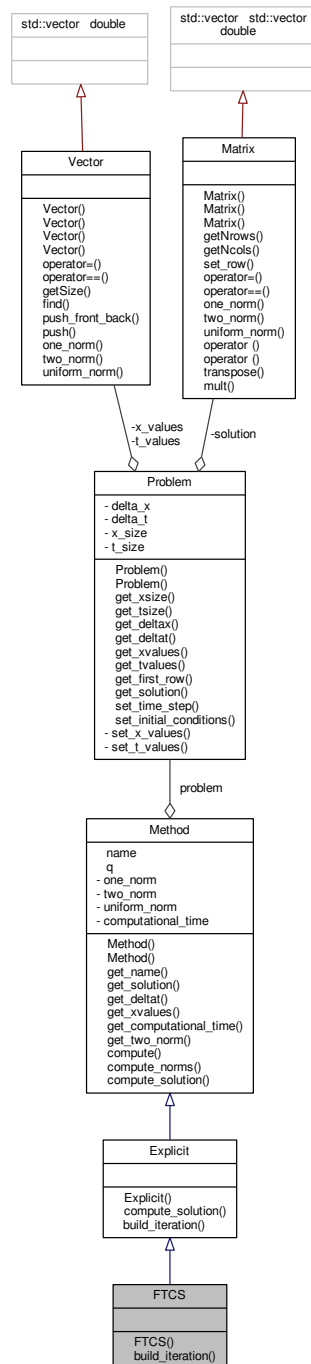
A [FTCS](#) method class that contains an iteration builder.

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

Inheritance diagram for FTCS:



Collaboration diagram for FTCS:



## Public Member Functions

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

## Additional Inherited Members

### 4.5.1 Detailed Description

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

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

The [FTCS](#) class provides:

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

### 4.5.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.5.3 Member Function Documentation

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

Normal public method.

Calculate a solution requiring only the previous time step solution.

##### Parameters

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

##### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

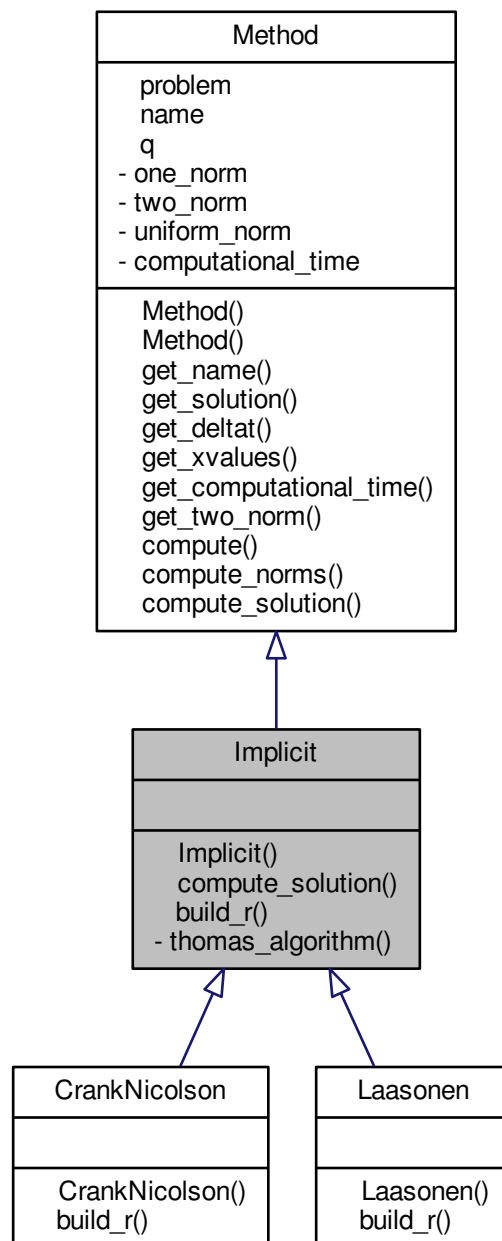
## 4.6 Implicit Class Reference

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

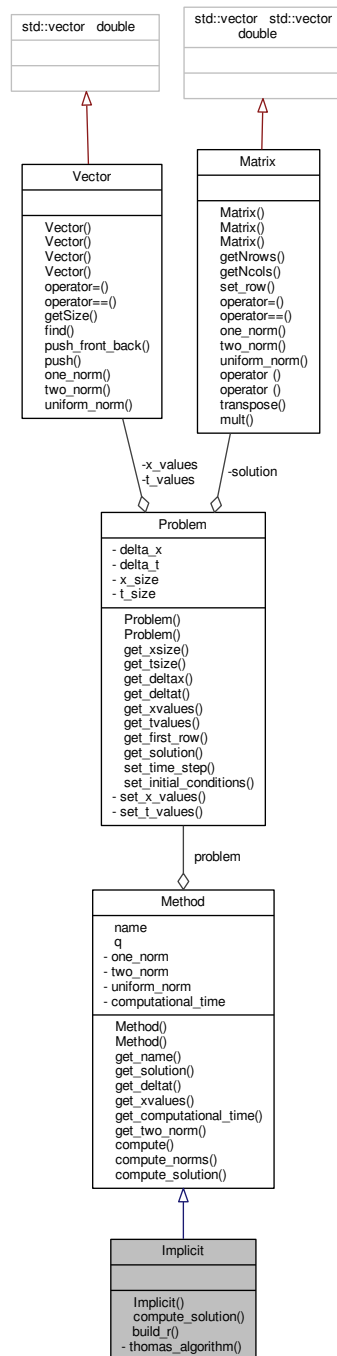
```
#include <implicit.h>
```



Inheritance diagram for Implicit:



Collaboration diagram for Implicit:



## Public Member Functions

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

## Protected Member Functions

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

## Private Member Functions

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

## Additional Inherited Members

### 4.6.1 Detailed Description

An implicit method class that contains default methods that only implicit methods use  
The implementation is derived from the `Method` class.

The `Implicit` class provides:

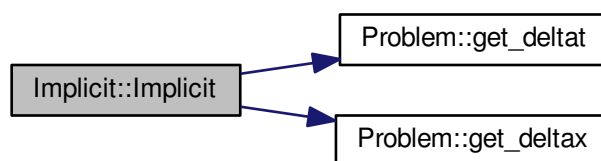
- a basic constructor for creating an implicit method object.
- a method to compute a solution following implicit methods rules

### 4.6.2 Constructor & Destructor Documentation

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

Default constructor.

Here is the call graph for this function:



### 4.6.3 Member Function Documentation

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

A pure virtual member.

Build the `r` vector in a linear system of  $A.x = r$  in which  $A$  is a matrix, whereas  $b$  and  $r$  are vectors.  
This method is used to compute a solution using the thomas algorithm, which can be used in a triadiagonal matrix.

## Parameters

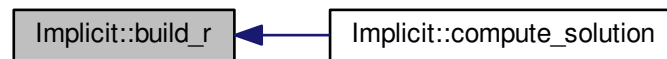
<i>previous_step</i>	A vector containing the previous time step solution.
----------------------	--

## Returns

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

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

Here is the caller graph for this function:



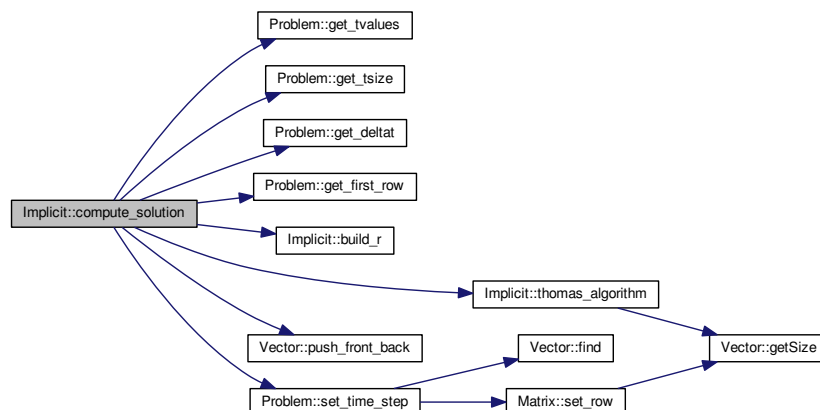
## 4.6.3.2 void Implicit::compute\_solution ( ) [virtual]

Normal public method.

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

Implements [Method](#).

Here is the call graph for this function:



#### 4.6.3.3 `Vector Implicit::thomas_algorithm ( Vector r, double a, double b, double c )` `[private]`

Normal private method.

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

See also

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

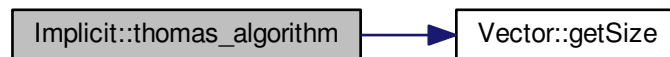
##### Parameters

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

##### Returns

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

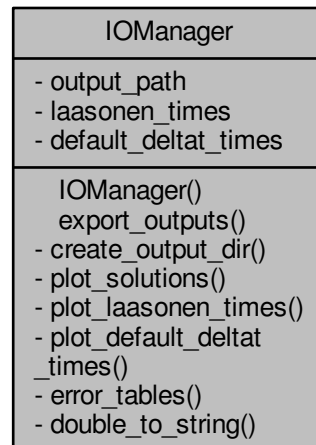
- [methods/implicit/implicit.h](#)
- [methods/implicit/implicit.cpp](#)

## 4.7 IOManager Class Reference

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

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

Collaboration diagram for IOManager:



### Public Member Functions

- [IOManager](#) ()  
*Default constructor.*
- void [export\\_outputs](#) ([Method](#) \*analytical, std::vector< [Method](#) \* > methods)  
*Exports outputs regarding plots images and error tables for each computed solution, comparing them to the analytical solution.*

### Private Member Functions

- bool [create\\_output\\_dir](#) ()  
*[Method](#) to create 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

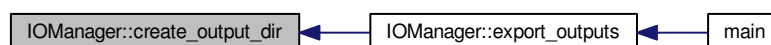
#### 4.7.3.1 `bool IOManager::create_output_dir ( )` `[private]`

[Method](#) to create ouput 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.

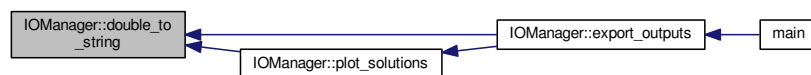
## Parameters

<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:



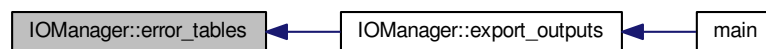
#### 4.7.3.3 void IOManager::error\_tables ( std::string output\_name, std::vector< Method \* > method ) [private]

Exports a plot that compares the norms of each solution.

## Parameters

<i>string</i>	output_name File name to be exported
<i>vector&lt; Method*&gt;</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 )

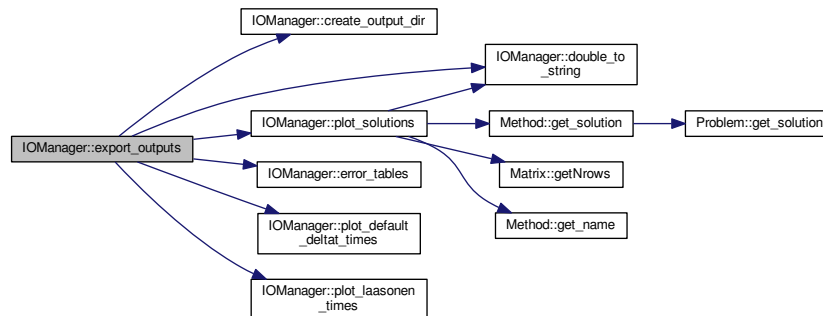
Exports outputs regarding plots images and error tables for each computed solution, comparing them to the analytical solution.

## Parameters

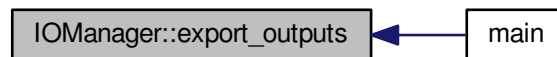
<i>Method*</i>	analytical The analytical solution
<i>vector&lt; Method*&gt;</i>	methods <a href="#">Vector</a> containing the solutions



Here is the call graph for this function:



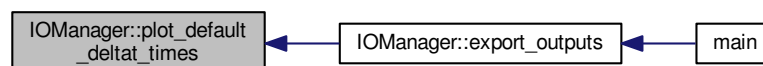
Here is the caller graph for this function:



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

Exports a plot with four methods computational times.

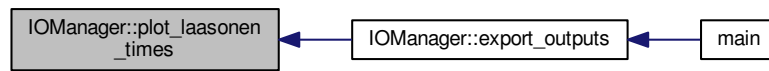
Here is the caller graph for this function:



#### 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:



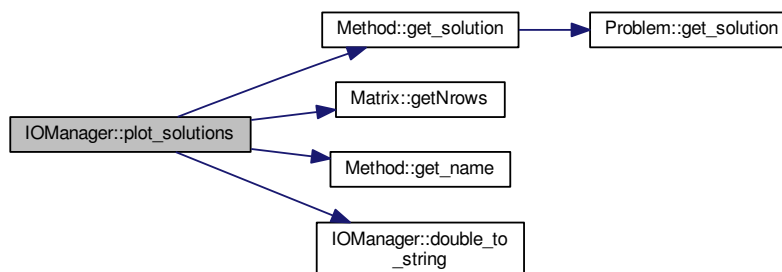
**4.7.3.7** `void IOManager::plot_solutions ( std::string output_name, Method * analytical, Method * method )`  
`[private]`

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:

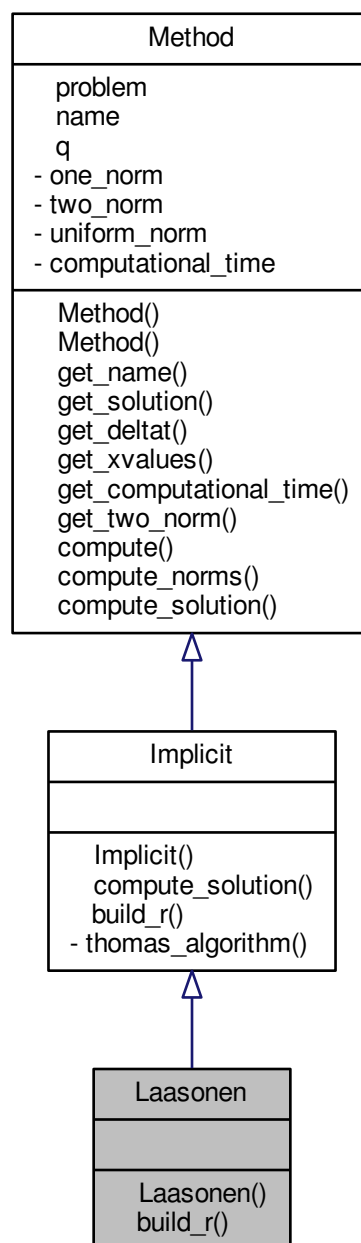
- [io/iomanager.h](#)
- [io/iomanager.cpp](#)

## 4.8 Laasonen Class Reference

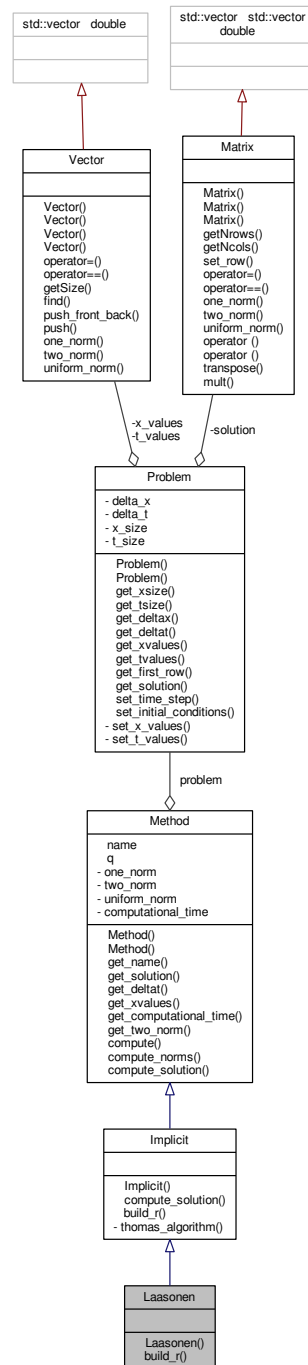
A [Laasonen](#) method class that contains a r vector builder.

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

#### Returns

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

Implements [Implicit](#).

Here is the call graph for this function:



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

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

## 4.9 Matrix Class Reference

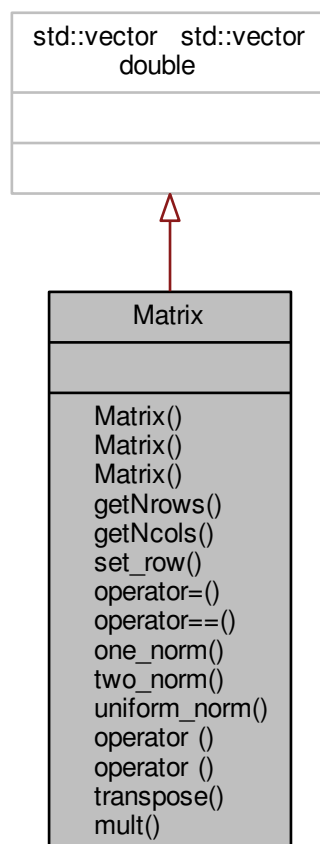
A matrix class for data storage of a 2D array of doubles

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

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

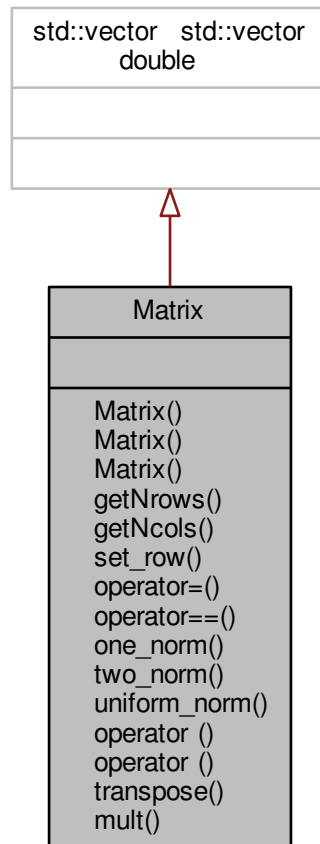
```
#include <matrix.h>
```

Inheritance diagram for Matrix:





Collaboration diagram for Matrix:



## Public Member Functions

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

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

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

## Private Types

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

## Friends

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

### 4.9.1 Detailed Description

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

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

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

The `Matrix` class provides:

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

## 4.9.2 Member Typedef Documentation

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

## 4.9.3 Constructor & Destructor Documentation

4.9.3.1 `Matrix::Matrix ( )`

Default constructor.

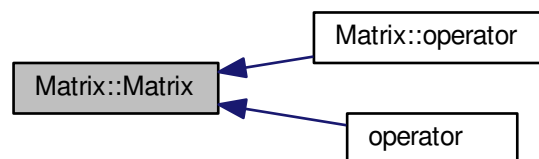
Initialize an empty [Matrix](#) object

See also

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

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

Here is the caller graph for this function:



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

Alternate constructor.

build a matrix *Nrows* by *Ncols*

See also

[Matrix\(\)](#)

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

### Exceptions

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

## Parameters

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

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

Copy constructor.

build a matrix from another matrix

## See also

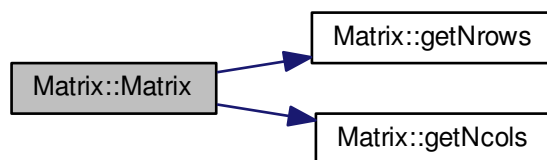
[Matrix\(\)](#)

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

## Parameters

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

Here is the call graph for this function:



## 4.9.4 Member Function Documentation

4.9.4.1 `int Matrix::getNcols ( ) const`

Normal public get method.

get the number of columns

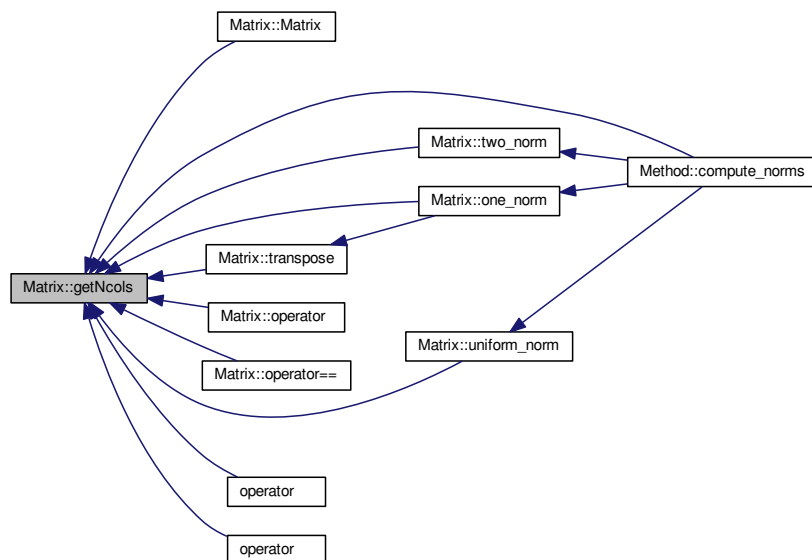
## See also

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

**Returns**

int. number of columns in matrix

Here is the caller graph for this function:



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

Normal public get method.

get the number of rows

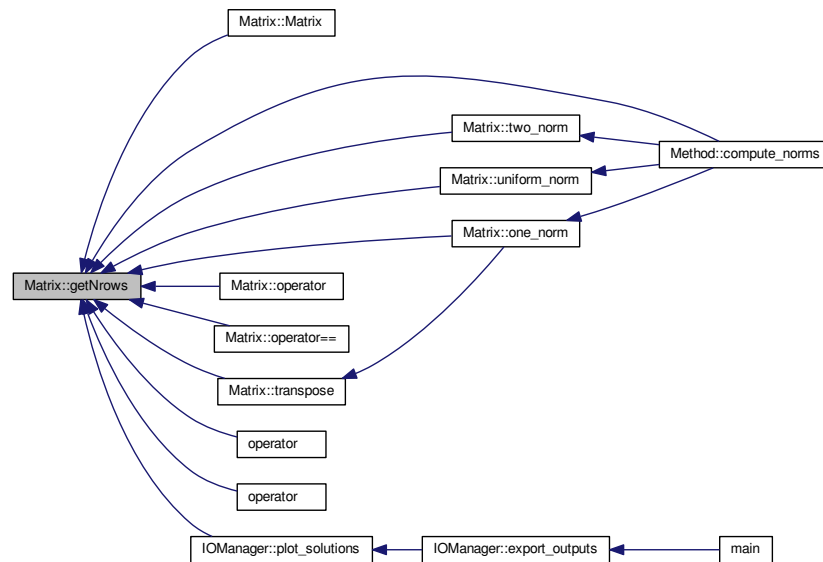
**See also**

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

## Returns

int. number of rows in matrix

Here is the caller graph for this function:



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

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

Normal public method that returns a double.

It returns L1 norm of matrix

## See also

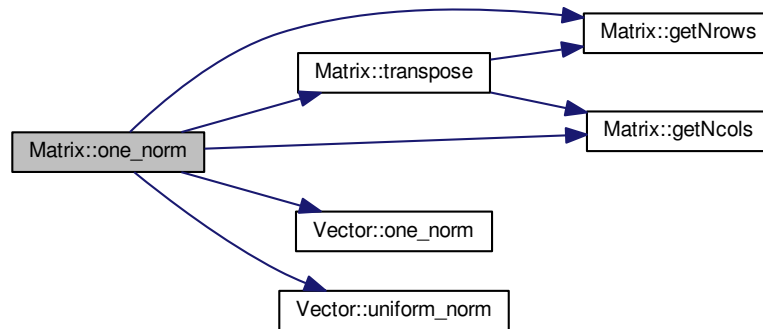
[two\\_norm\(\)const](#)

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

**Returns**

double. matrix L1 norm

Here is the call graph for this function:



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

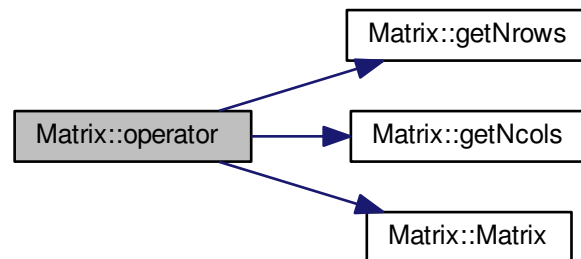
## Returns

[Matrix](#). matrix-matrix product

## Parameters

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

Here is the call graph for this function:



#### 4.9.4.6 Vector `Matrix::operator*( const Vector & v ) const`

Overloaded `*operator` that returns a [Vector](#).

It Performs matrix by vector multiplication.

## See also

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

## Exceptions

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

## Returns

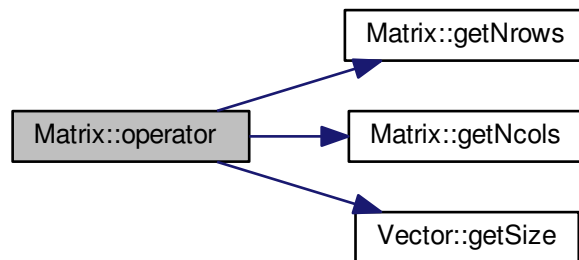
[Vector](#). matrix-vector product

## Parameters

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



Here is the call graph for this function:



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

Overloaded assignment operator.

See also

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

Returns

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

Parameters

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

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

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

See also

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

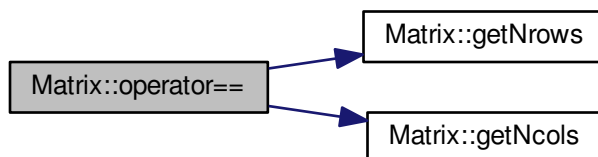
Returns

bool. true or false

## Parameters

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

Here is the call graph for this function:

4.9.4.9 void Matrix::set\_row ( int *index*, Vector *v* )

Normal public set method.

replace a row with a given vector

## Parameters

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

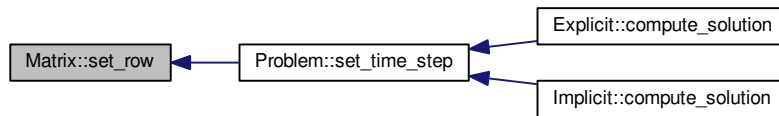
## Exceptions

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

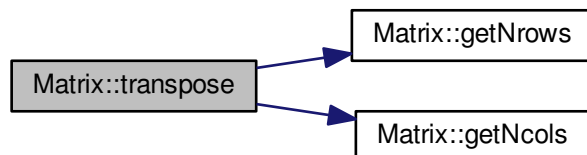
public method that returns the transpose of the matrix.

It returns the transpose of matrix

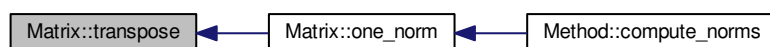
##### Returns

[Matrix](#). matrix transpose

Here is the call graph for this function:



Here is the caller graph for this function:



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

Normal public method that returns a double.

It returns L2 norm of matrix

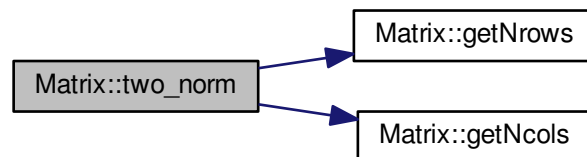
See also

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

Returns

double. matrix L2 norm

Here is the call graph for this function:



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

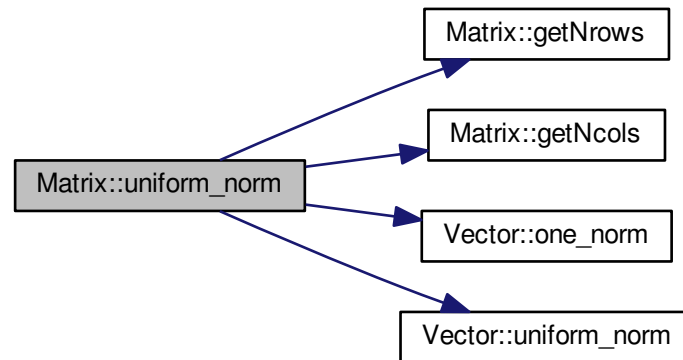
See also

[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>	<a href="#">Matrix</a> to read from

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

Overloaded ofstream << operator.

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

## See also

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

## Exceptions

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

## Returns

`std::ofstream&`. The ofstream object

## Parameters

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

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

Overloaded istream >> operator.

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

## See also

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

## Exceptions

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

**Returns**

std::istream&. The istream object

**Parameters**

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

#### 4.9.5.4 std::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>	<a href="#">Matrix</a> to write into

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

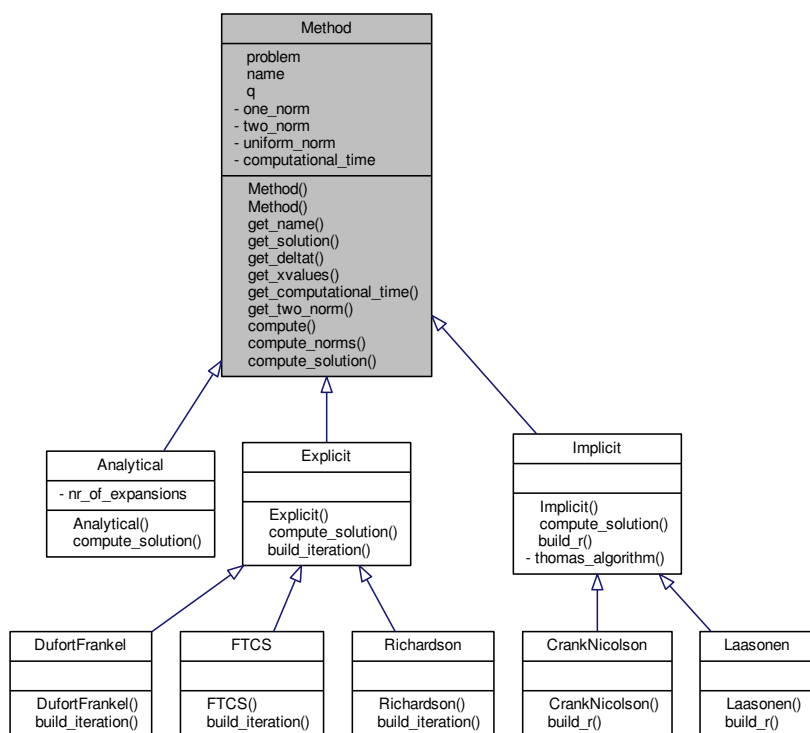
- [grid/matrix.h](#)
- [grid/matrix.cpp](#)

## 4.10 Method Class Reference

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

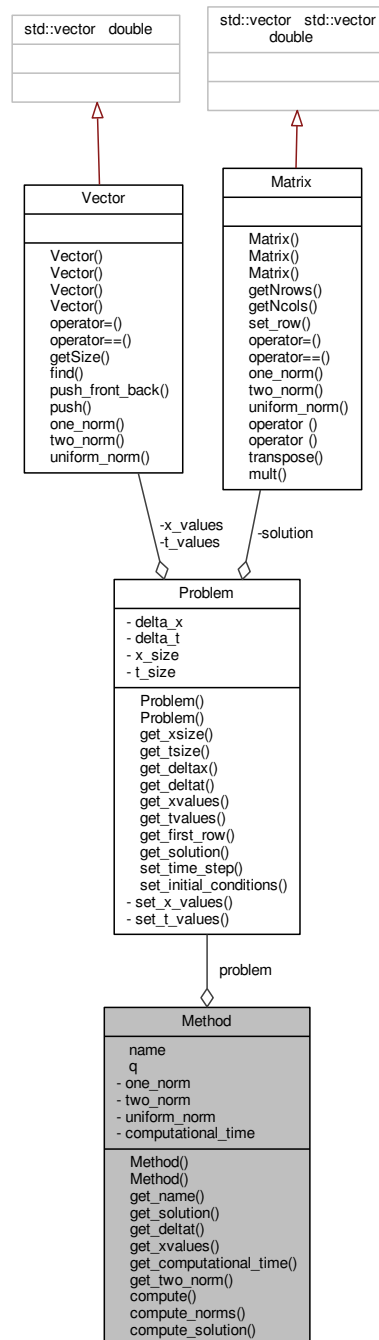
```
#include <method.h>
```

Inheritance diagram for Method:





Collaboration diagram for Method:



## Public Member Functions

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

- Normal public get method.*
- [Matrix get\\_solution \(\)](#)
- Normal public get method.*
- double [get\\_deltat \(\)](#)
- Normal public get method.*
- [Vector get\\_xvalues \(\)](#)
- Normal public get method.*
- double [get\\_computational\\_time \(\)](#)
- Normal public get method.*
- 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.

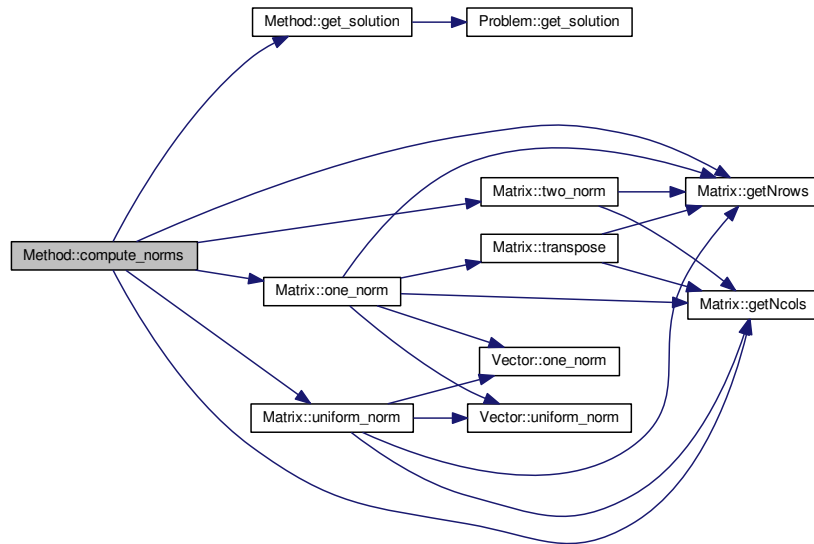
Keeps track of the time to compute a solution

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 ( )

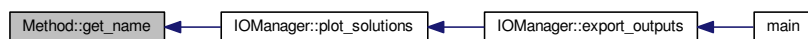
Normal public get method.

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

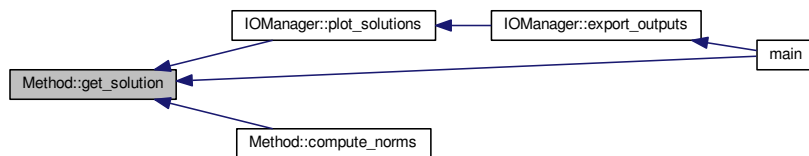
**Returns**

[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:

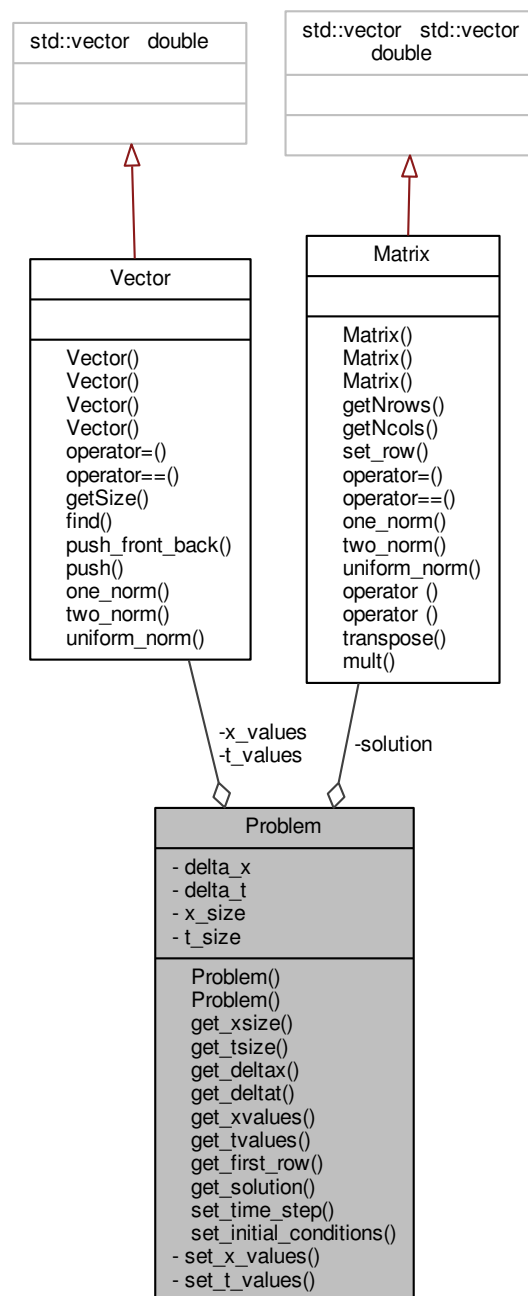
- [methods/method.h](#)
- [methods/method.cpp](#)

## 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\(\)](#)

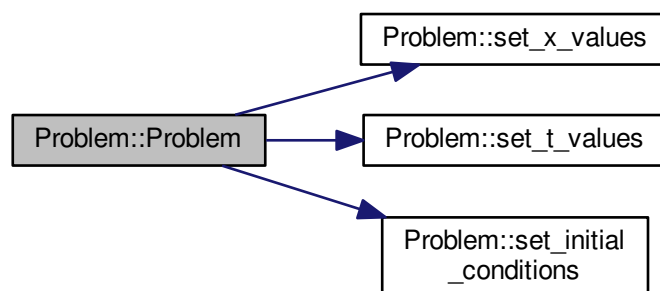
#### Parameters

<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

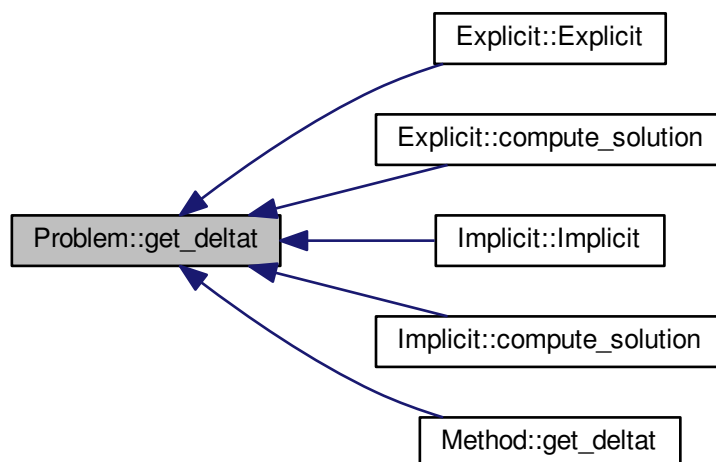
#### 4.11.3.1 `double Problem::get_deltat ( )`

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:



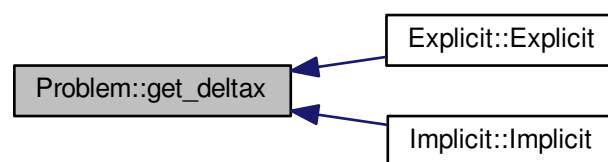
#### 4.11.3.2 double Problem::get\_deltax ( )

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:



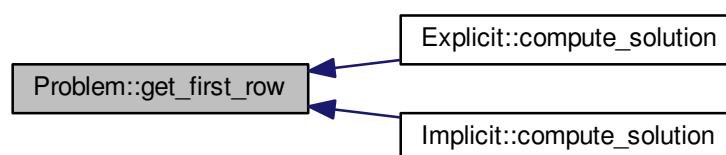
#### 4.11.3.3 Vector Problem::get\_first\_row ( )

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:



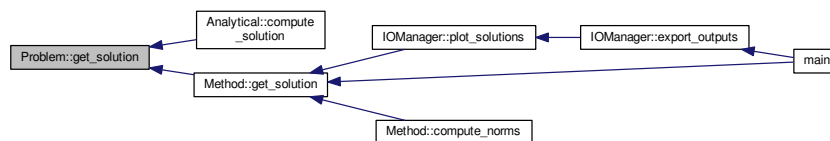
#### 4.11.3.4 **Matrix \*** Problem::get\_solution ( )

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:



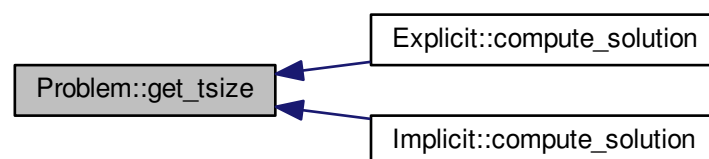
#### 4.11.3.5 **unsigned int** Problem::get\_tsize ( )

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:



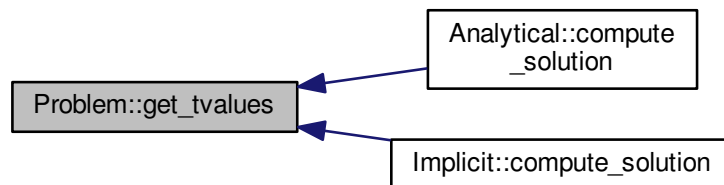
#### 4.11.3.6 **Vector** Problem::get\_tvalues ( )

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:

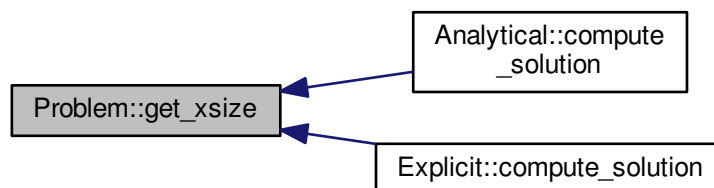
**4.11.3.7 unsigned int Problem::get\_xsize ( )**

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:

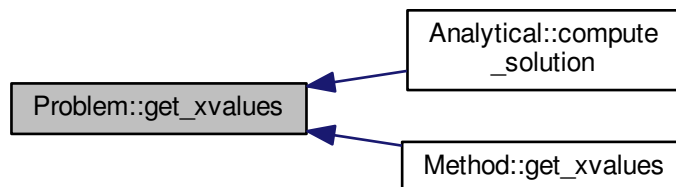
**4.11.3.8 Vector Problem::get\_xvalues ( )**

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 )

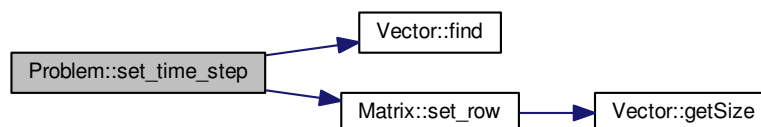
Normal public set method.

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

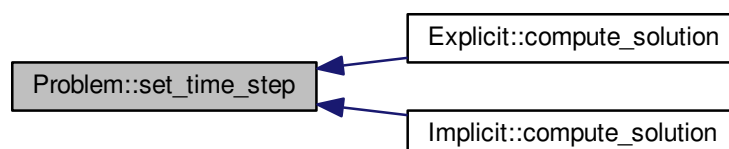
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:





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

Normal private set method.

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

See also

[x\\_values](#)

Here is the caller graph for this function:



### 4.11.4 Member Data Documentation

#### 4.11.4.1 double Problem::delta\_t [private]

Private double delta\_t.

Time step of the solution.

#### 4.11.4.2 double Problem::delta\_x [private]

Private double delta\_x.

Space step of the solution.

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

Private [Matrix](#) solution.

[Matrix](#) containing the computed solution.

#### 4.11.4.4 unsigned int Problem::t\_size [private]

Private unsigned int t\_size.

Time size of the solution.

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

Private [Vector](#) `t_values`.

Time correspondent value for each row index.

#### 4.11.4.6 `unsigned int Problem::x_size` [private]

Private unsigned int `x_size`.

Space size of the solution.

#### 4.11.4.7 `Vector Problem::x_values` [private]

Private [Vector](#) `x_values`.

Space correspondent value for each column index.

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

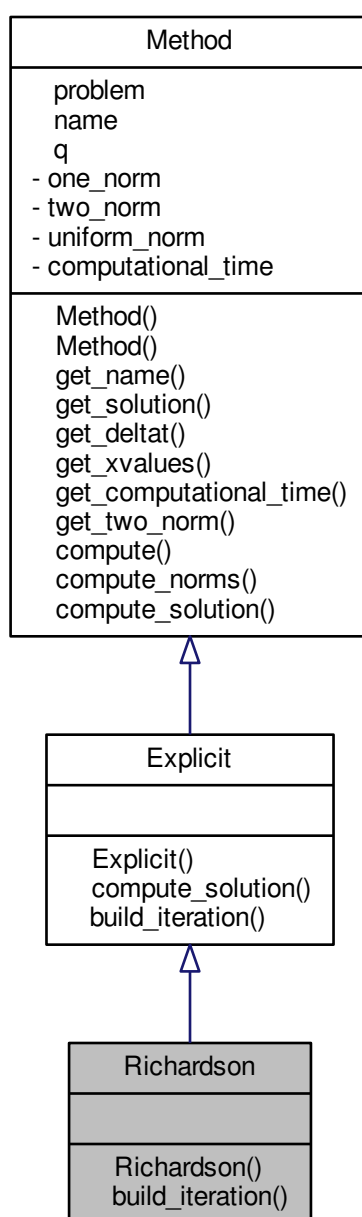
- [variants/problem.h](#)
- [variants/problem.cpp](#)

## 4.12 Richardson Class Reference

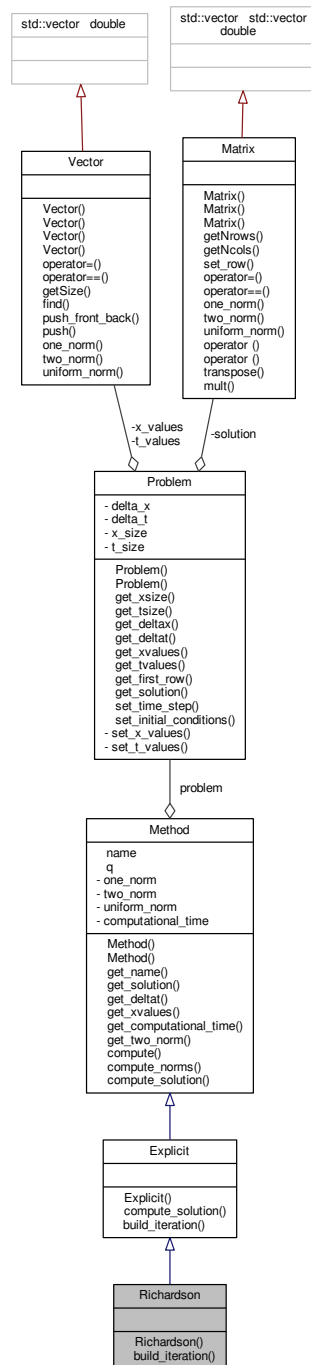
A [Richardson](#) method class that contains an iteration builder.

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

Inheritance diagram for Richardson:



Collaboration diagram for Richardson:



## Public Member Functions

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

*Default constructor.*

## Protected Member Functions

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

## Additional Inherited Members

### 4.12.1 Detailed Description

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

This builder is used to calculate a solution using the [Richardson](#) method.

The [Richardson](#) class provides:

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

### 4.12.2 Constructor & Destructor Documentation

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

Default constructor.

### 4.12.3 Member Function Documentation

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

Normal protected method.

Calculate a next time step solution requiring a previous time step and a current time step solution.

#### Parameters

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

#### Returns

[Vector](#). The computed solution.

Implements [Explicit](#).

Here is the call graph for this function:



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

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

## 4.13 Vector Class Reference

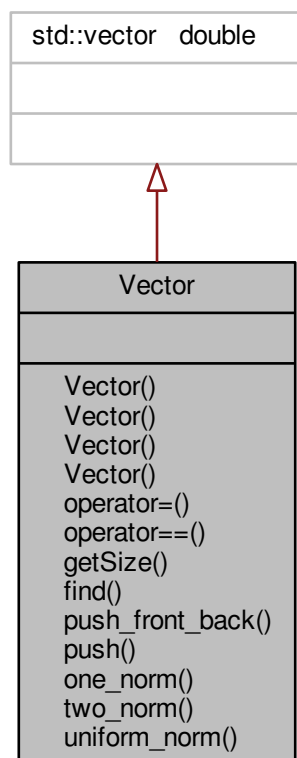
A vector class for data storage of a 1D array of doubles

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

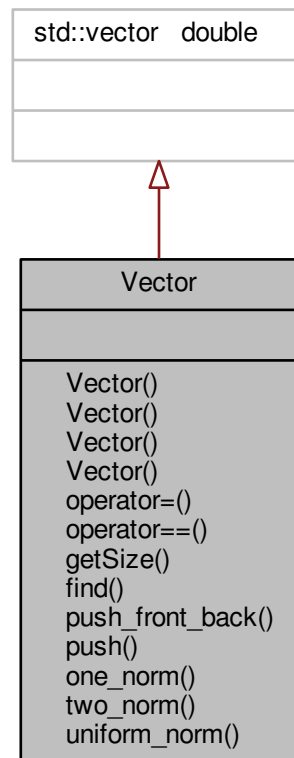
We use private inheritance to base our vector upon the library version whilst expose only those base class functions we wish to use - in this the array access operator `[]`.

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

Inheritance diagram for Vector:



Collaboration diagram for Vector:



## Public Member Functions

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



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

## Private Types

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

## Friends

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

### 4.13.1 Detailed Description

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

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

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

The [Vector](#) class provides:

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

### 4.13.2 Member Typedef Documentation

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

### 4.13.3 Constructor & Destructor Documentation

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

Default constructor.

Initialize an empty [Vector](#) object

See also

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

Here is the caller graph for this function:



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

[Explicit](#) alternative constructor takes an integer.

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

See also

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

#### Exceptions

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

#### Parameters

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

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

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

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

See also

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

#### 4.13.3.4 Vector::Vector ( std::vector< double > vec )

Copy constructor takes an vector<double> object reference.

Initialize [Vector](#) object with an vector<double> object

See also

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

### 4.13.4 Member Function Documentation

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

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

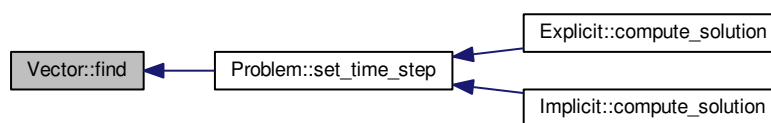
Parameters

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

Returns

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

Here is the caller graph for this function:



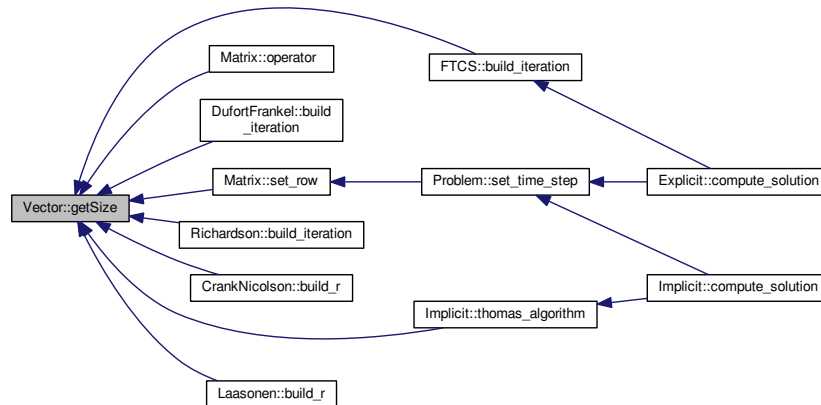
#### 4.13.4.2 int Vector::getSize ( ) const

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

**Returns**

int. the size of the vector

Here is the caller graph for this function:

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

Normal public method that returns a double.

It returns L1 norm of vector

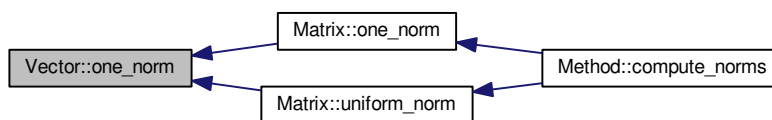
See also

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

**Returns**

double. vectors L1 norm

Here is the caller graph for this function:

**4.13.4.4 Vector & Vector::operator= ( const Vector & v )**

Overloaded assignment operator.

See also

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

## Parameters

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

## Returns

the object on the left of the assignment

## Parameters

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

4.13.4.5 `bool Vector::operator== ( const Vector & v ) const`

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

## See also

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

## Returns

bool. true or false

## Exceptions

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

## Parameters

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

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

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

## Parameters

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

4.13.4.7 void Vector::push\_front\_back ( double *value* )

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

## Parameters

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

Here is the caller graph for this function:



## 4.13.4.8 double Vector::two\_norm ( ) const

Normal public method that returns a double.

It returns L2 norm of vector

## See also

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

## Returns

double. vectors L2 norm

## 4.13.4.9 double Vector::uniform\_norm ( ) const

Normal public method that returns a double.

It returns L\_max norm of vector

## See also

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

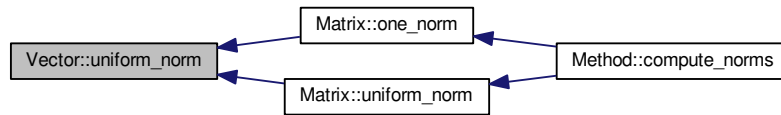
## Exceptions

<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>	<a href="#">Vector</a> &. vector to read from

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

Overloaded istream >> operator.

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

**See also**

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

**Returns**

std::istream&. the input stream object is

**Exceptions**

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

**Parameters**

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

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

Overloaded ifstream >> operator.

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

**See also**

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

**Returns**

ifstream&. the input ifstream object ifs



### Exceptions

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

### Parameters

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

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

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

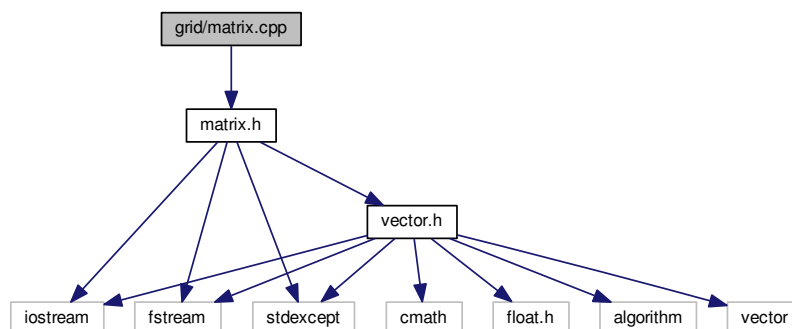
## Chapter 5

# File Documentation

### 5.1 grid/matrix.cpp File Reference

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

Include dependency graph for matrix.cpp:



### Functions

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

#### 5.1.1 Function Documentation

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

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

## See also

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

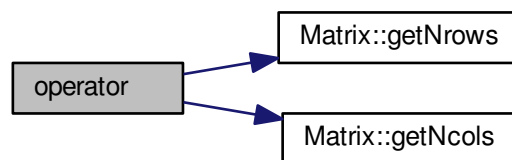
## Returns

std::ostream&. The ostream object

## Parameters

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

Here is the call graph for this function:



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

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

## See also

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

## Exceptions

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

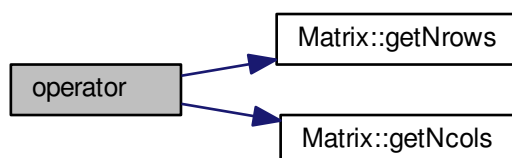
## Returns

std::ofstream&. The ofstream object

## Parameters

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

Here is the call graph for this function:

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

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

## See also

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

## Exceptions

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

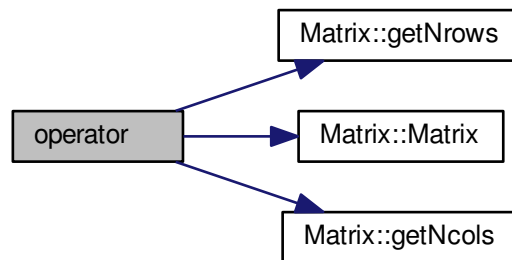
## Returns

`std::istream&`. The istream object

## Parameters

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

Here is the call graph for this function:



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

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

See also

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

Returns

`std::ifstream&`. The ifstream object

Parameters

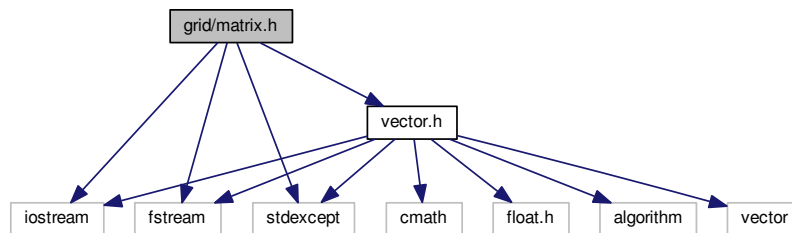
<i>ifs</i>	Input file stream with opened matrix file
<i>m</i>	<a href="#">Matrix</a> to write into

Here is the call graph for this function:

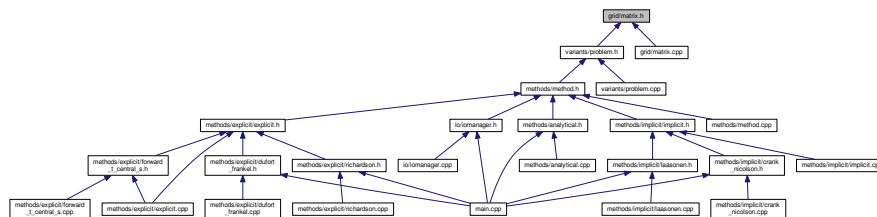


## 5.2 grid/matrix.h File Reference

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



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



## Classes

- class **Matrix**

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

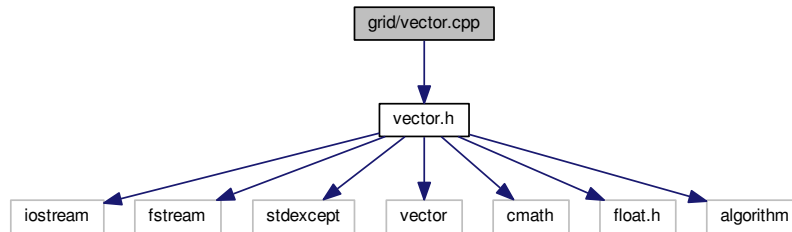
The implementation is derived from the standard container vector `std::vector`

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

### 5.3 grid/vector.cpp File Reference

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

Include dependency graph for vector.cpp:



## Functions

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

### 5.3.1 Function Documentation

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

Display output.

See also

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

#### Returns

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

#### Parameters

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

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

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

## See also

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

## Returns

std::ofstream&. the output ofstream object ofs

## Parameters

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

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

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

## See also

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

## Returns

std::istream&. the input stream object is

## Exceptions

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

## Parameters

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

Here is the call graph for this function:





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

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

See also

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

Returns

ifstream&. the input ifstream object ifs

Exceptions

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

Parameters

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

Here is the call graph for this function:

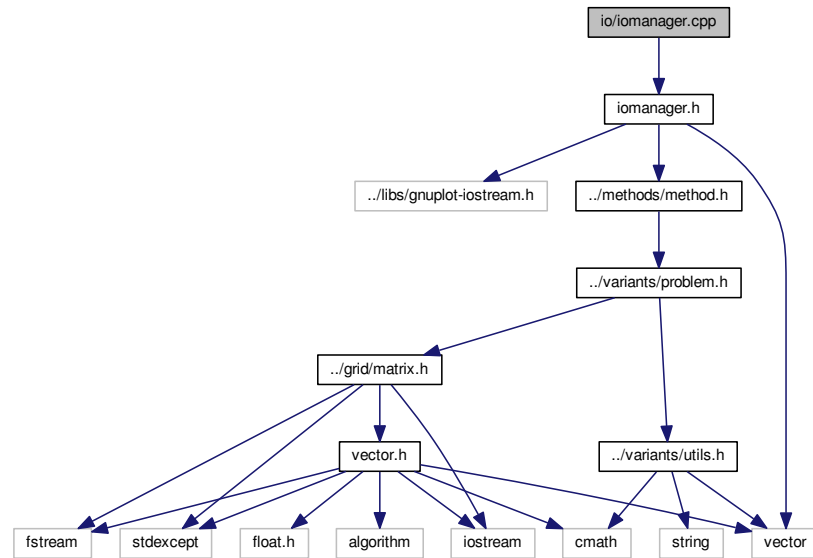


## 5.4 grid/vector.h File Reference

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



Include dependency graph for iomanager.cpp:



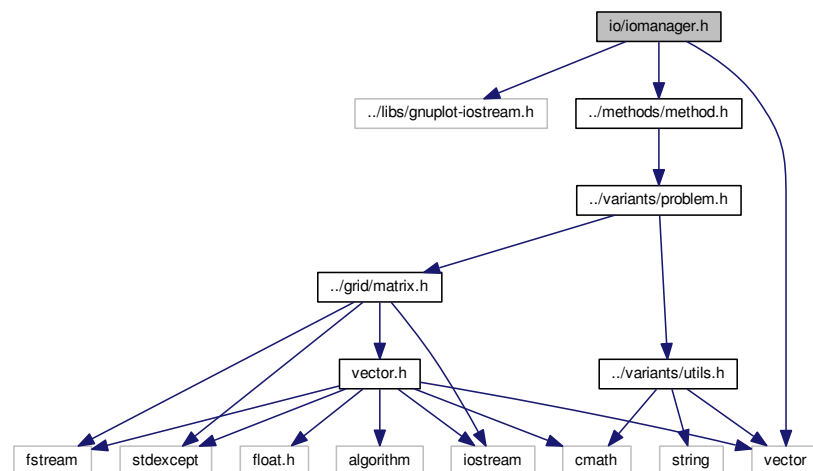
## 5.6 io/iomanager.h File Reference

```

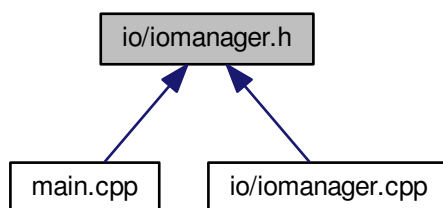
#include "../libs/gnuplot-iostream.h"
#include "../methods/method.h"
#include <vector>

```

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.*

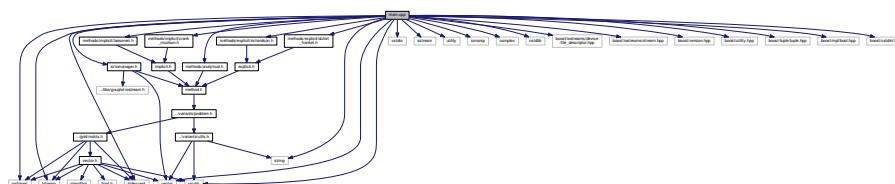
## 5.7 main.cpp File Reference

```

#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:



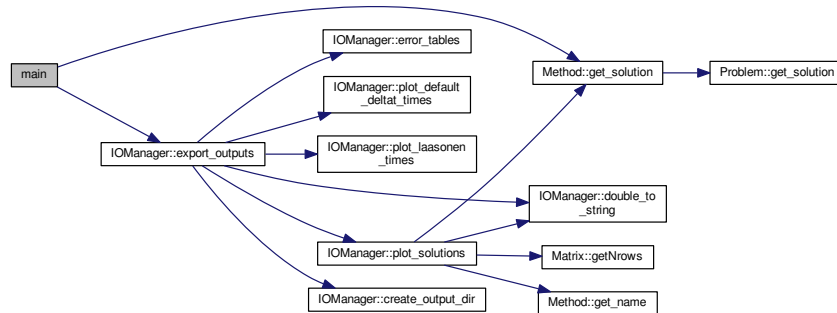
## Functions

- int `main()`

## 5.7.1 Function Documentation

### 5.7.1.1 `int main ( )`

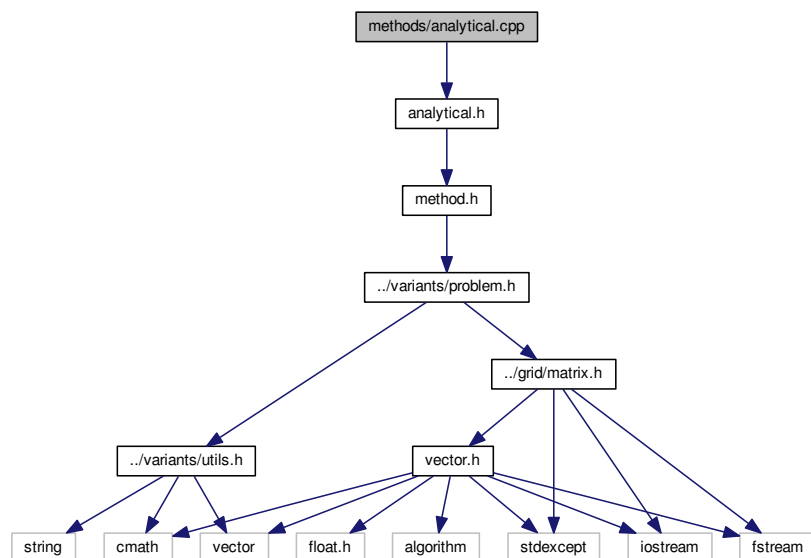
Here is the call graph for this function:



## 5.8 methods/analytical.cpp File Reference

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

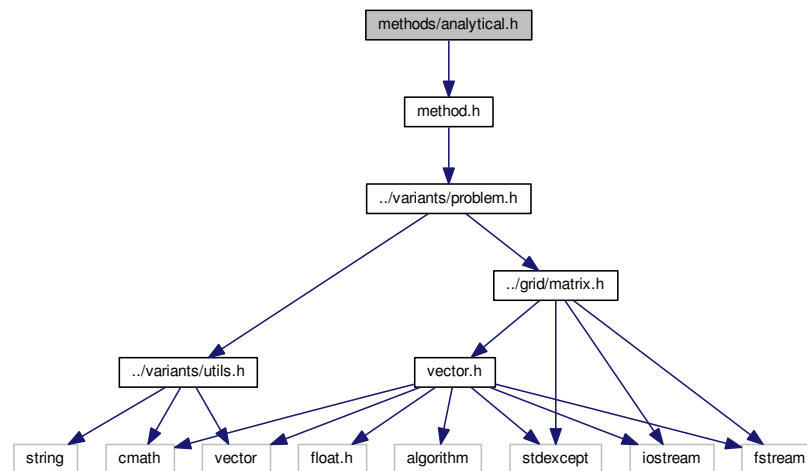
Include dependency graph for analytical.cpp:



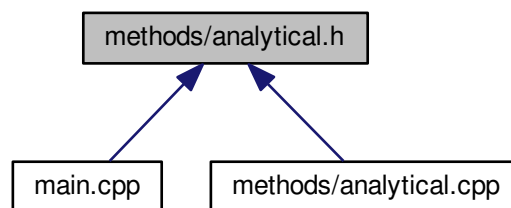
## 5.9 methods/analytical.h File Reference

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

Include dependency graph for analytical.h:



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



## Classes

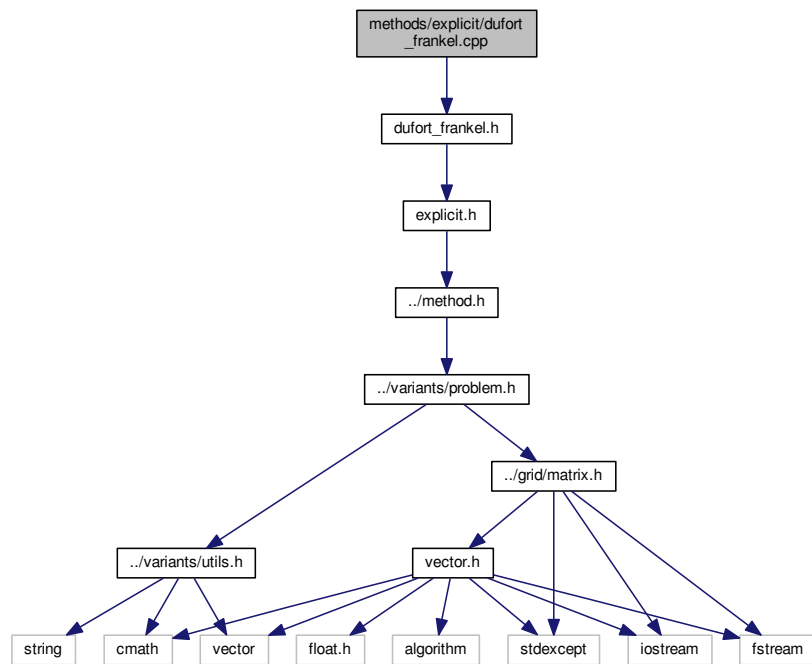
- class [Analytical](#)

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"
```

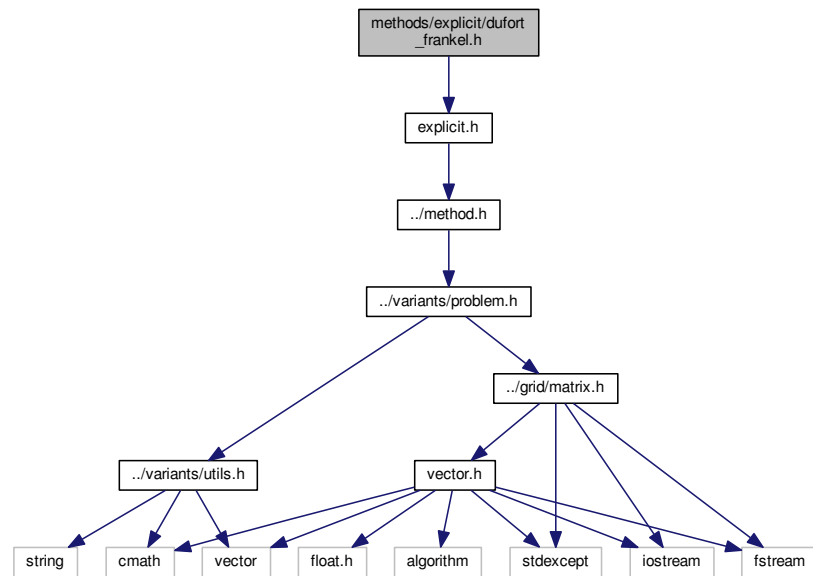
Include dependency graph for dufort\_frankel.cpp:



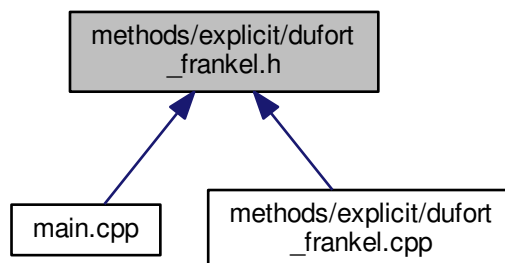
## 5.11 methods/explicit/dufort\_frankel.h File Reference

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

Include dependency graph for dufort\_frankel.h:



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



## Classes

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

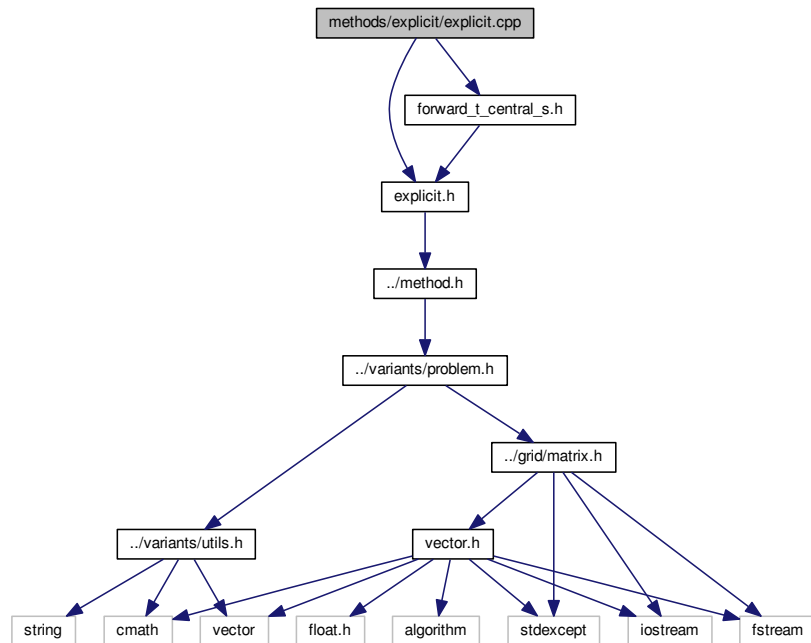
## 5.12 methods/explicit/explicit.cpp File Reference

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



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

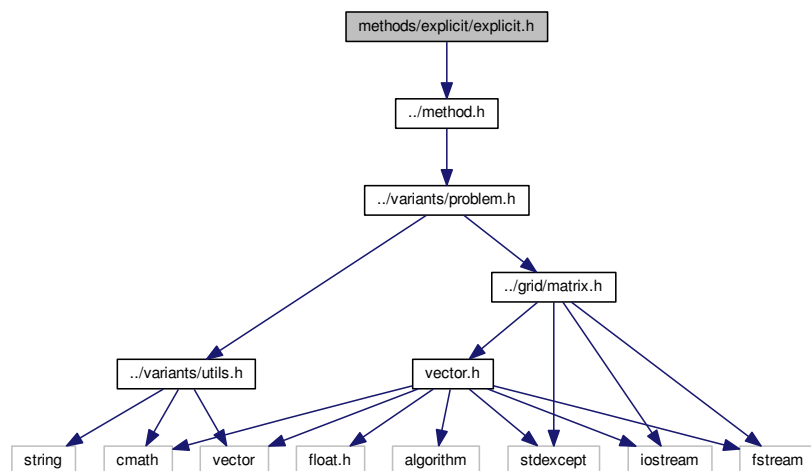
Include dependency graph for explicit.cpp:



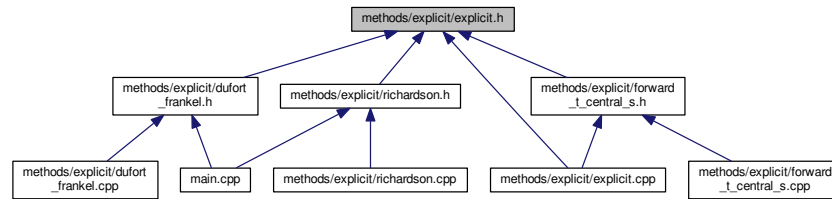
### 5.13 methods/explicit/explicit.h File Reference

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

Include dependency graph for explicit.h:



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



## Classes

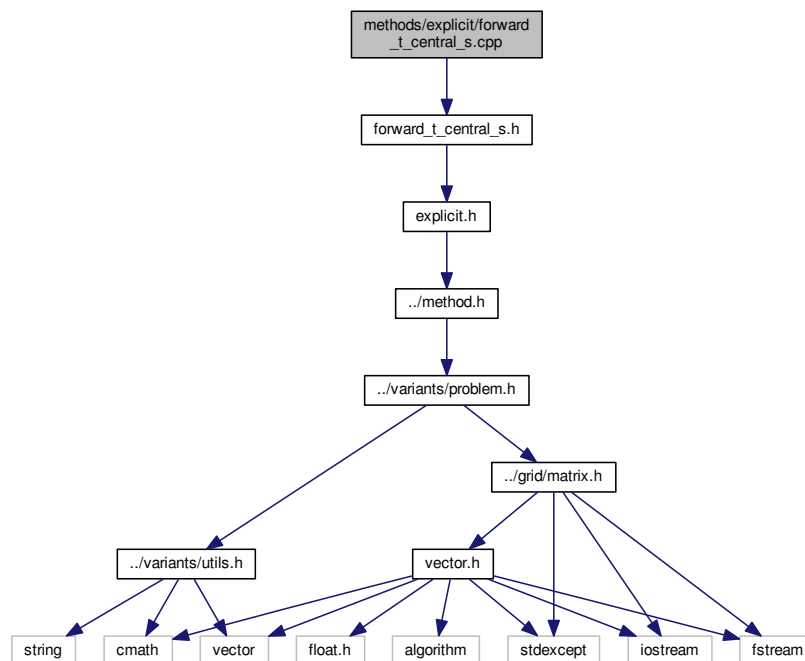
- class [Explicit](#)

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

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

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

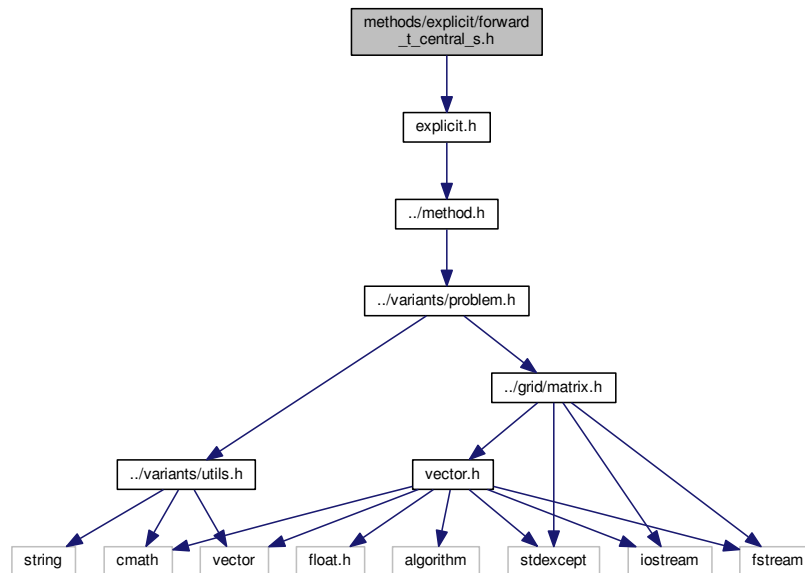
Include dependency graph for forward\_t\_central\_s.cpp:



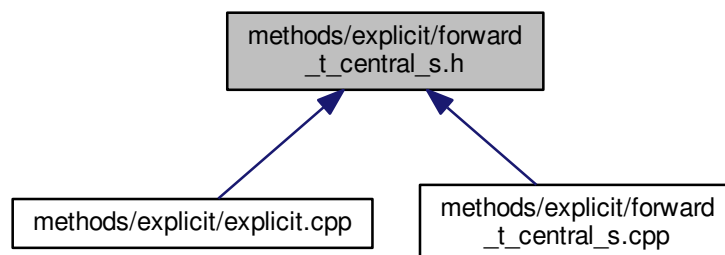
## 5.15 methods/explicit/forward\_t\_central\_s.h File Reference

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

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



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



## Classes

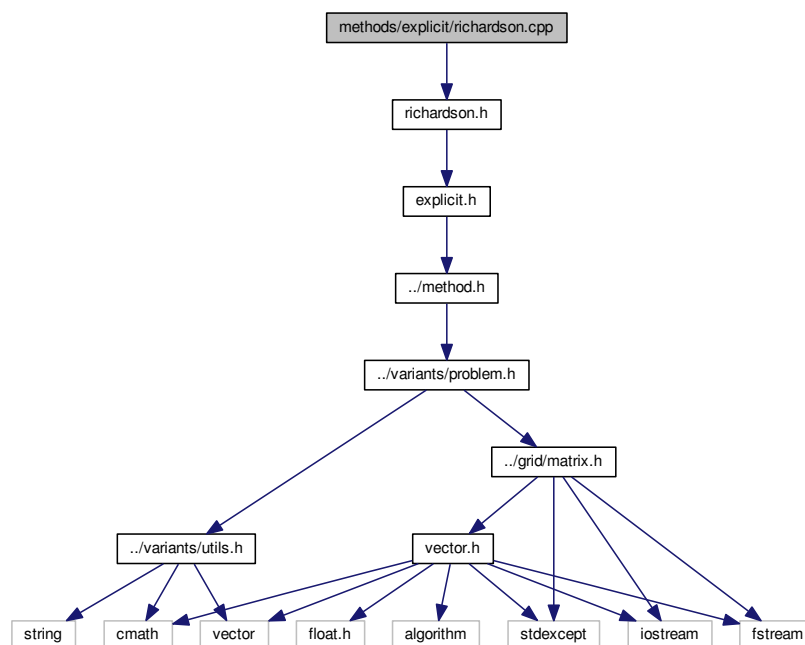
- class [FTCS](#)

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

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

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

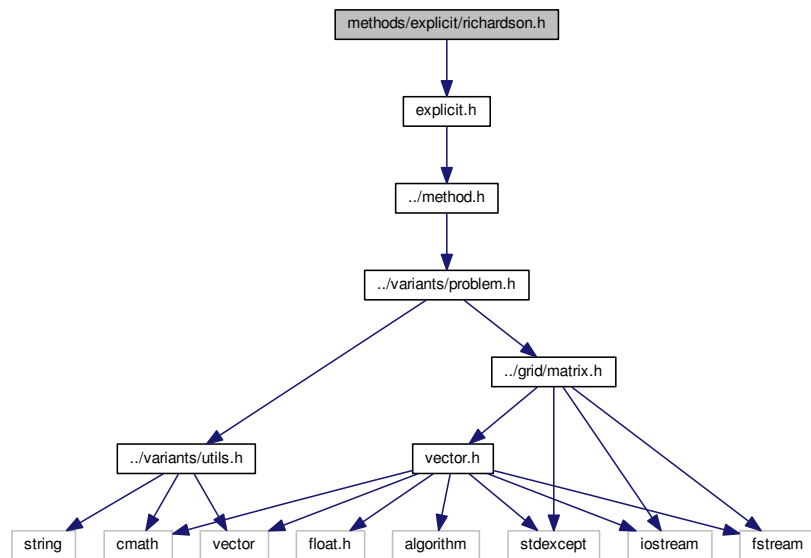
Include dependency graph for richardson.cpp:



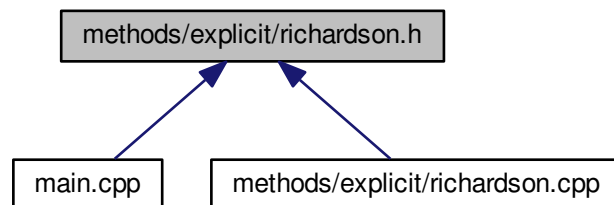
## 5.17 methods/explicit/richardson.h File Reference

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

Include dependency graph for richardson.h:



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



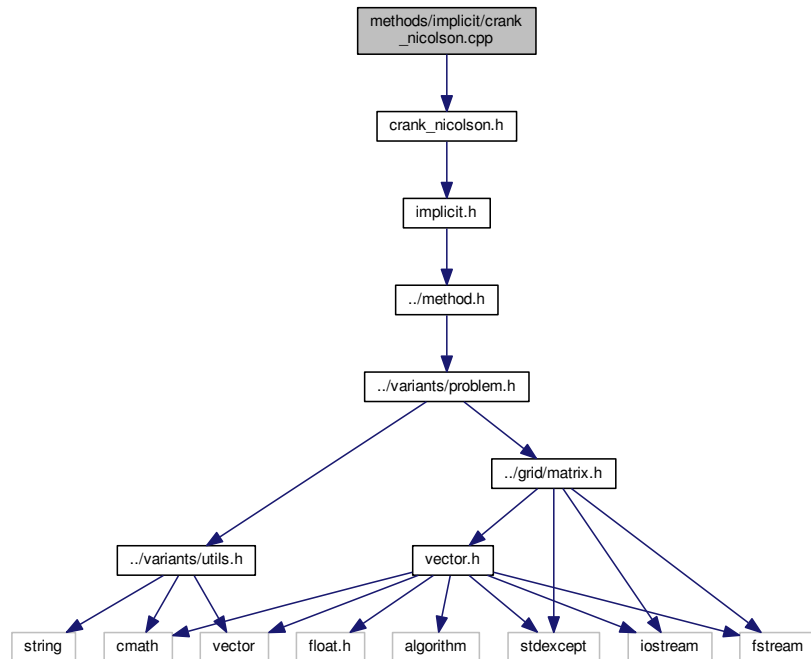
## Classes

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

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

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

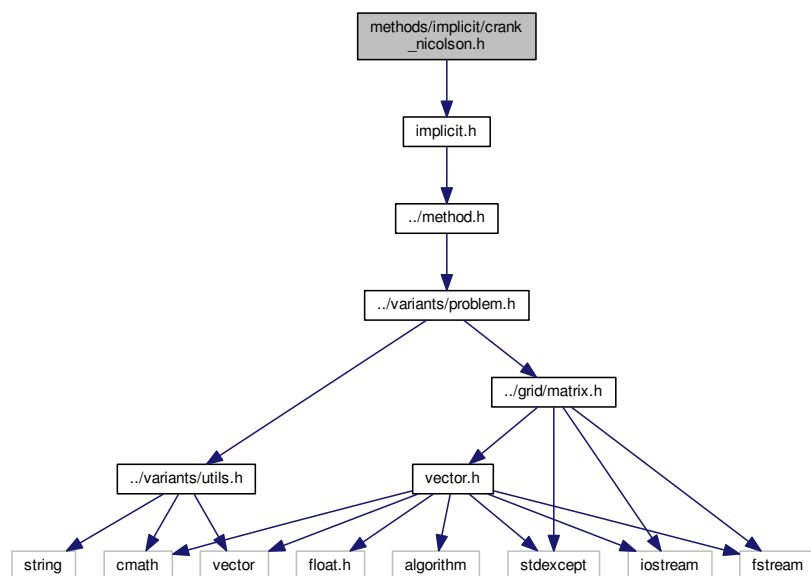
Include dependency graph for krank\_nicolson.cpp:



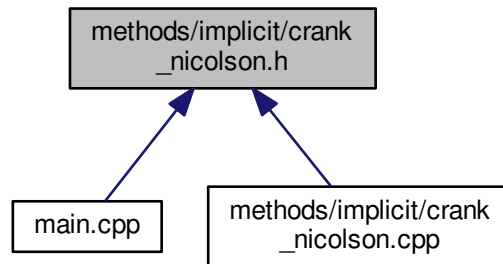
## 5.19 methods/implicit/krank\_nicolson.h File Reference

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

Include dependency graph for krank\_nicolson.h:



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



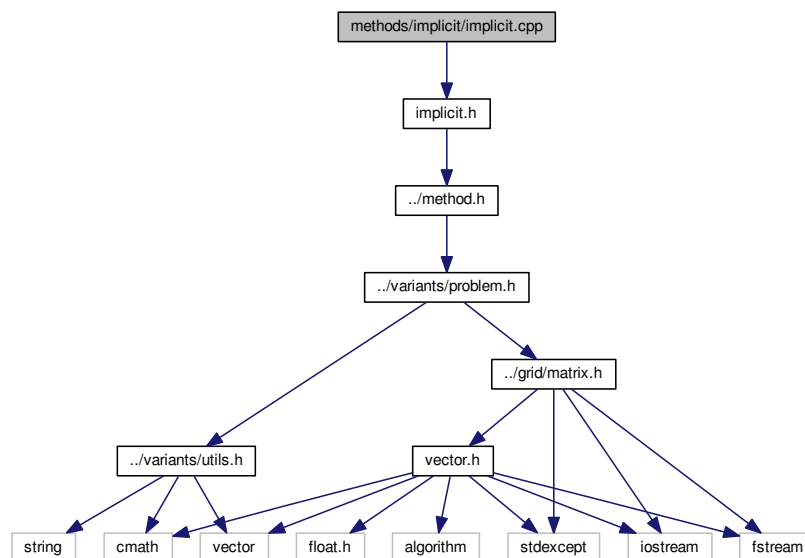
## Classes

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

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

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

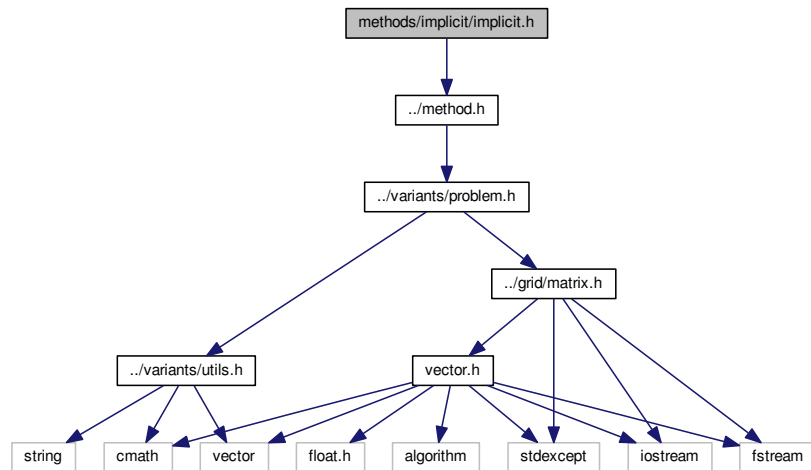
Include dependency graph for `implicit.cpp`:



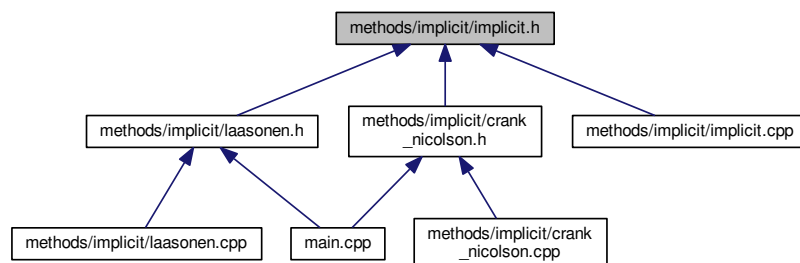
## 5.21 methods/implicit/implicit.h File Reference

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

Include dependency graph for implicit.h:



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



### Classes

- class [Implicit](#)

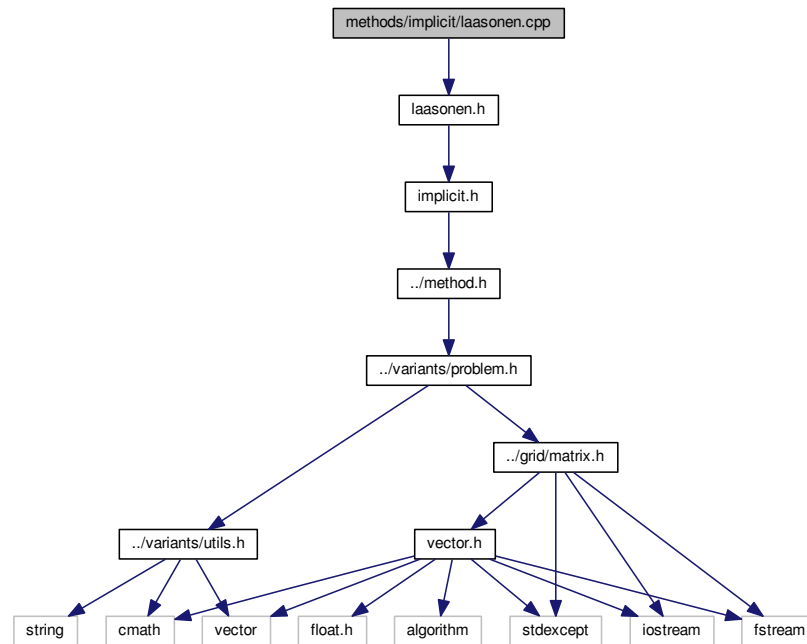
An implicit method class that contains default methods that only implicit methods use  
The implementation is derived from the [Method](#) class.

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

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



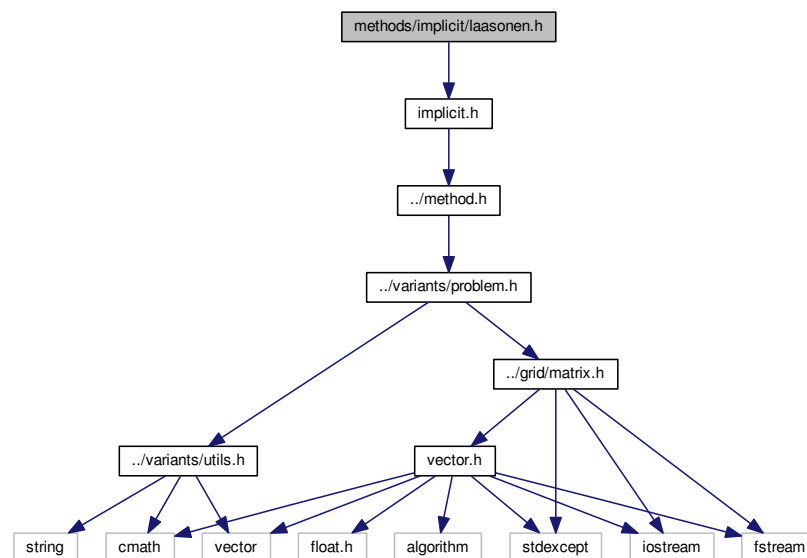
Include dependency graph for laasonen.cpp:



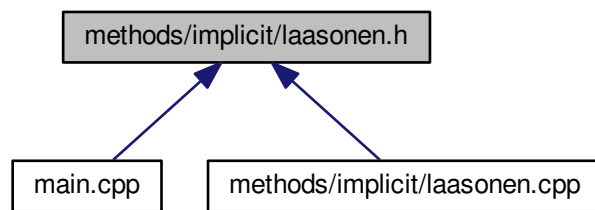
## 5.23 methods/implicit/laasonen.h File Reference

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

Include dependency graph for laasonen.h:



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



## Classes

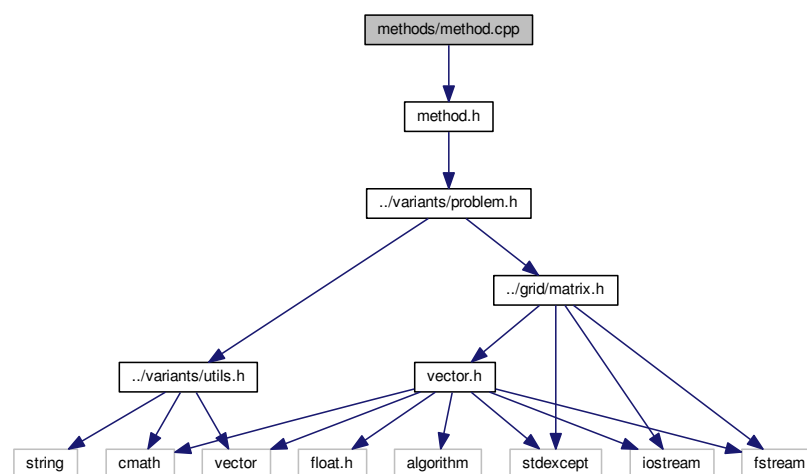
- class [Laasonen](#)

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

## 5.24 methods/method.cpp File Reference

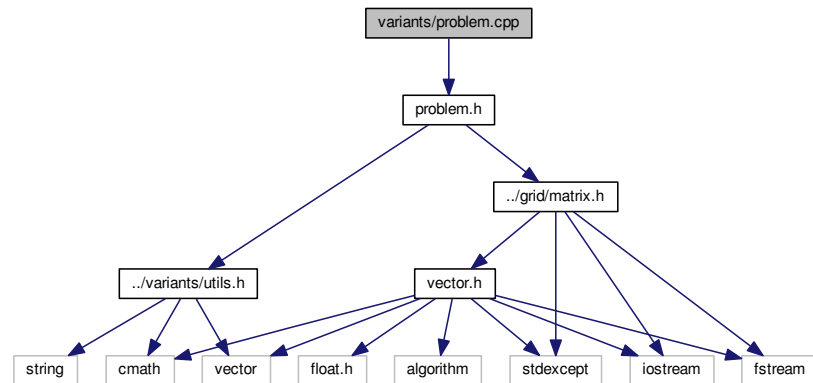
```
#include "method.h"
```

Include dependency graph for `method.cpp`:





Include dependency graph for problem.cpp:

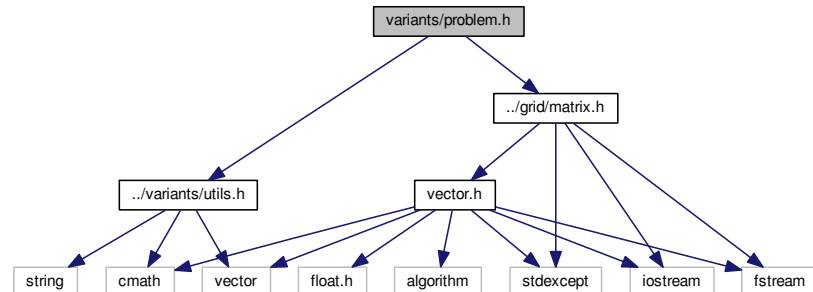


## 5.27 variants/problem.h File Reference

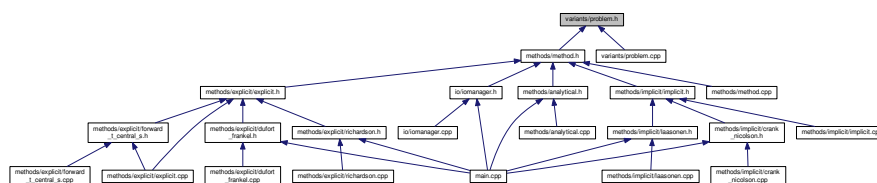
```
#include "../variants/utlis.h"
```

```
#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](#)