COMPUTATIONAL METHODS & C++

Heat Conduction Equation

December 10, 2017

António Pedro Araújo Fraga

Student ID: 279654

Cranfield University

M.Sc. in Software Engineering for Technical Computing

Contents

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	Δt
Space step	Δx
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 **100ºF** and the surface temperatures at both sides were suddenly increased and maintained to **300ºF**. It is also known that the wall is composed of nickel steel (40% Ni) with a Diffusivity of **0.01** ft^2/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 [1, 2].

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** [3]. 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** [2].

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[3],

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

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[4]. 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 $\mathbf{O}(\Delta x^2, \Delta t^2)$ [3]. Following the heat conduction equation, the expression can be written as following:

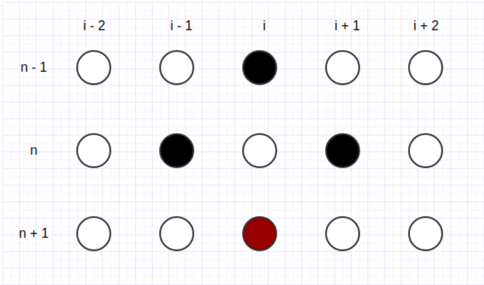


Figure 1: DuFort-Frankel's method stencil.

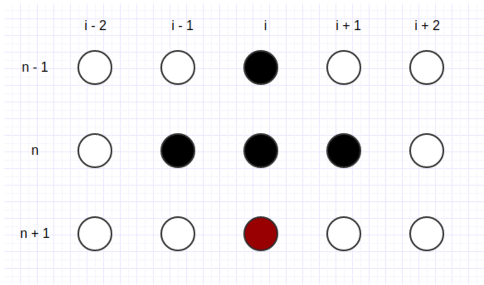


Figure 2: Richardson's method stencil.

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

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 $\mathbf{O}(\Delta x^2, \Delta t^2, (\frac{\Delta t}{\Delta x})^2)$ [3] 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, $Ax = r$[5]. 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 $A = LU$[5]. This algorithm consists of two steps, the downwards phase where the equation $Lp = r$ is solved and the upwards phase, solving $Ux = p$[5], obtaining a solution for x.

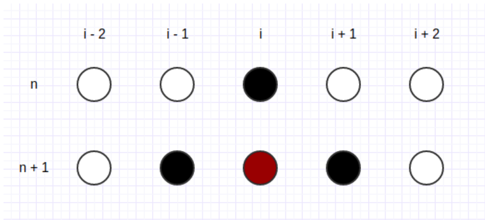


Figure 3: Laasonen's method stencil.

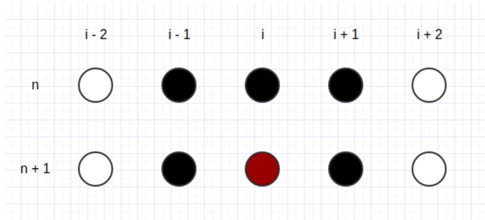


Figure 4: Crank-Nicholson's method stencil.

Laasonen Simple Implicit

The time derivative is considered forward in time. Central difference is used in space derivative, and the scheme is of order $\mathbf{O}(\Delta x, \Delta t^2)$[3], 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:

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

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:

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

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

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

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

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

Considering that the maximum number of space steps is **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 **n** and **n + 1**. The method is of order **O(**Δx^2**,** Δt^2**)** [3] 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,

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

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

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

, penultimate,

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

, and every other space steps with unknown values.

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

Thus, a tridiagonal matrix system is obtained,

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

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**.

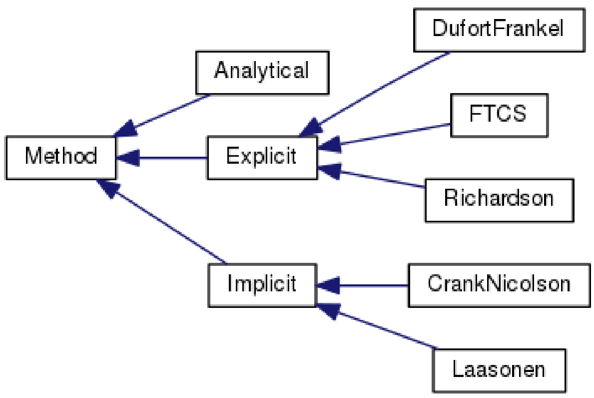


Figure 5: Method Class inheritance diagram.

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.

t \ x	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.

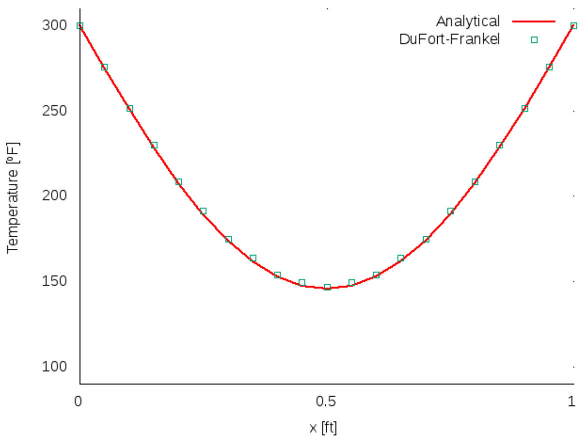


Figure 6: DuFort-Frankel's solution at $t = 0.5$.

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.

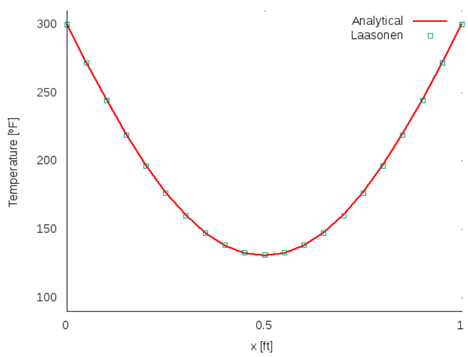


Figure 7: Laasonen's solution at $t = 0.4$.

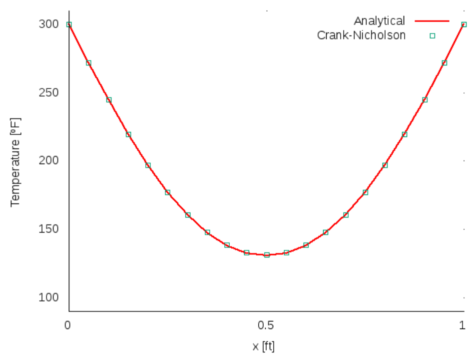


Figure 8: Crank-Nicholson's solution at $t = 0.4$.

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**.

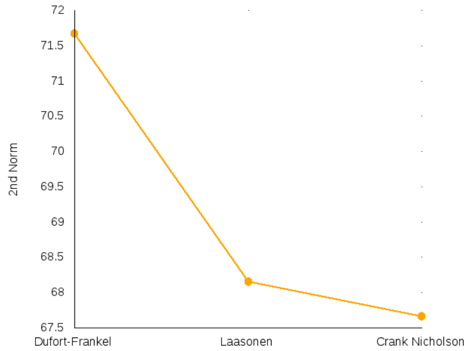


Figure 9: 2nd norm values of Error Matrices.

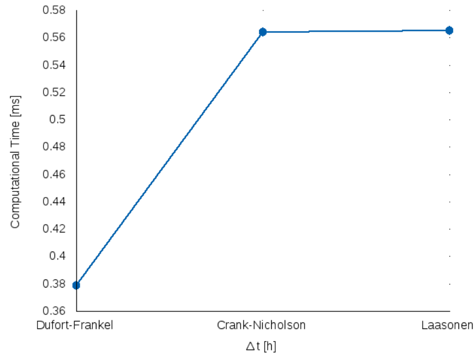


Figure 10: Computational times of stable methods.

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[7]. This phenomenon could be explained with a concept that was introduced earlier, the **truncation error**[3]. 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 Δt at $t = 0.5$

Δt \ 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.

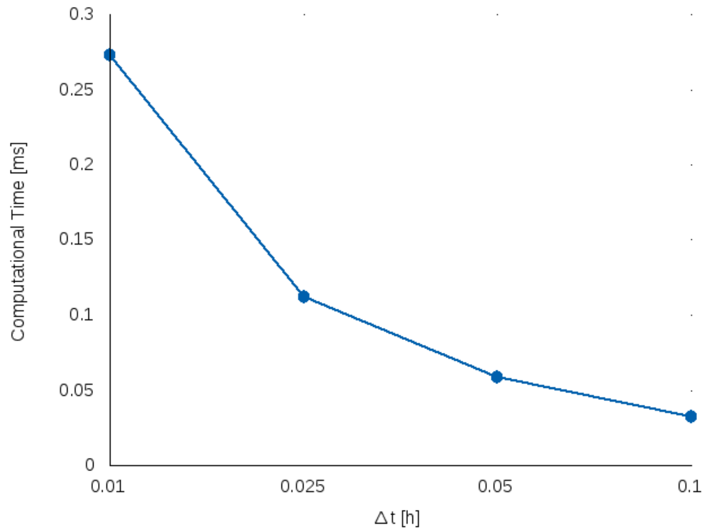


Figure 11: Laasonen method computational times for the several Δt.

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

[1] 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> [Accessed 2 October 2017]

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

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

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

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

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

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

[8] B.J.P. Kaus, *Explicit versus implicit Finite Difference Schemes*, Available at: <https://www.geowiss.uni-mainz.de/Dateien/Finite_Differerence_Timpl_expl.pdf> [Accessed 4 November 2017]

[9] Markus Schmuck, *Numerical Methods for PDEs,* Available at: <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:

$$f(x+\Delta t) = f_i^n + \Delta t \left(\frac{\partial f}{\partial t}\right)_i^n + \frac{\Delta t^2}{2}\left(\frac{\partial^2 f}{\partial t^2}\right)_i^n + O(\Delta t^3)$$

$$f(x-\Delta t) = f_i^n - \Delta t \left(\frac{\partial f}{\partial t}\right)_i^n + \frac{\Delta t^2}{2}\left(\frac{\partial^2 f}{\partial t^2}\right)_i^n + O(\Delta t^3)$$

$$f(x+\Delta x) = f_i^n + \Delta x \left(\frac{\partial f}{\partial x}\right)_i^n + \frac{\Delta x2}{2}\left(\frac{\partial^2 f}{\partial x^2}\right)_i^n + \frac{\Delta x3}{3}\left(\frac{\partial^3 f}{\partial x^3}\right)_i^n + O(\Delta x^4)$$

$$f(x-\Delta x) = f_i^n - \Delta x \left(\frac{\partial f}{\partial x}\right)_i^n + \frac{\Delta x^2}{2}\left(\frac{\partial^2 f}{\partial x^2}\right)_i^n - \frac{\Delta x3}{3}\left(\frac{\partial^3 f}{\partial x^3}\right)_i^n + O(\Delta x^4)$$

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,

$$\frac{f_i^n + \Delta t\left(\frac{\partial f}{\partial t}\right)_i^n + \frac{\Delta t^2}{2}\left(\frac{\partial^2 f}{\partial t^2}\right)_i^n + O(\Delta t^3) - f_i^n + \Delta t\left(\frac{\partial f}{\partial t}\right)_i^n - \frac{\Delta t^2}{2}\left(\frac{\partial^2 f}{\partial t^2}\right)_i^n + O(\Delta t^3)}{2\Delta t}$$

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,

$$D\frac{f_i^n+\Delta x\left(\frac{\partial f}{\partial x}\right)_i^n+\frac{\Delta x^2}{2}\left(\frac{\partial^2 f}{\partial x^2}\right)_i^n+\frac{\Delta x3}{3}\left(\frac{\partial^3 f}{\partial x^3}\right)_i^n+O(\Delta x^4)-2f_i^n+f_i^n-\Delta x\left(\frac{\partial f}{\partial x}\right)_i^n+\frac{\Delta x^2}{2}\left(\frac{\partial^2 f}{\partial x^2}\right)_i^n-\frac{\Delta x3}{3}\left(\frac{\partial^3 f}{\partial x^3}\right)_i^n+O(\Delta x^4)}{(\Delta x)^2}$$

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,

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

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

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

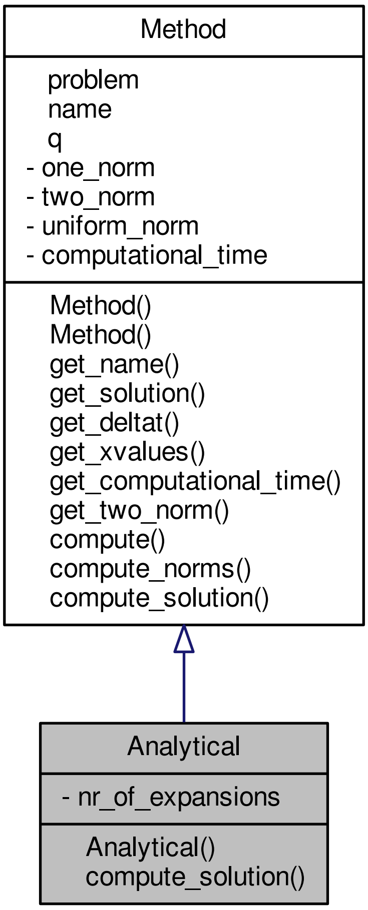
Class Documentation

4.1 Analytical Class Reference

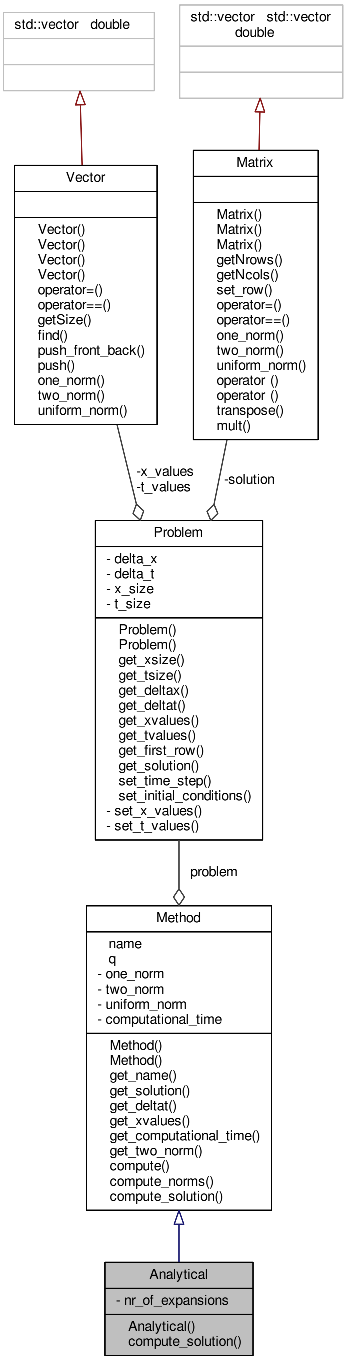
An Analytical class to compute the solution with standard procedures
The implementation is derived from the Method Object.

```
#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

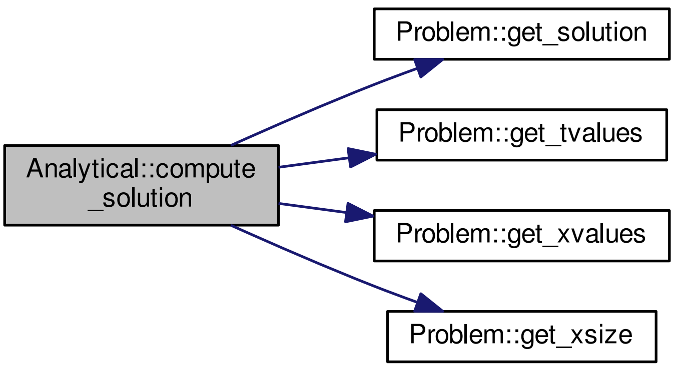
4.1.3.1 void Analytical::compute_solution ( ) [virtual]

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:

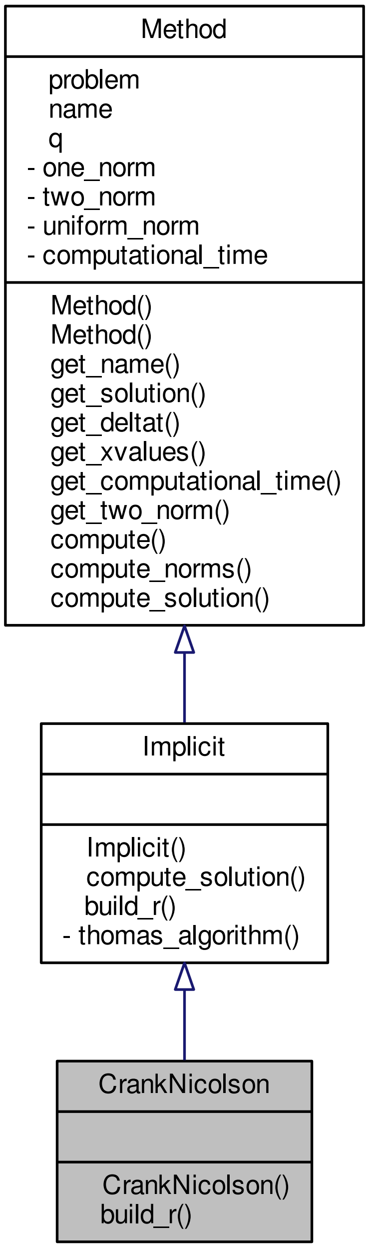
- methods/analytical.h
- methods/analytical.cpp

4.2 CrankNicolson Class Reference

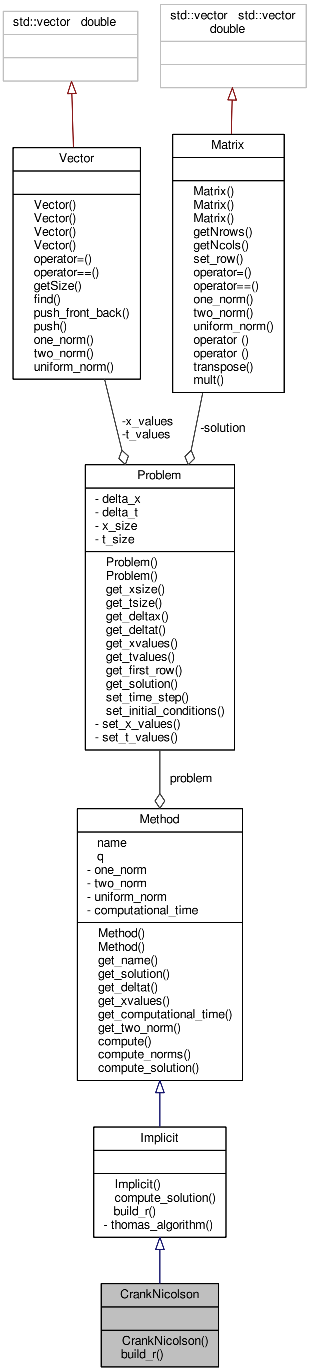
A CrankNicolson method class that contains a r vector builder.

```
#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

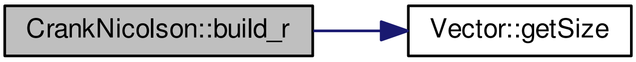
previous_step	Vector representing the solution of the previous time step.

Returns

Vector. r vector to be used in A.x = r

Implements Implicit.

Here is the call graph for this function:



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

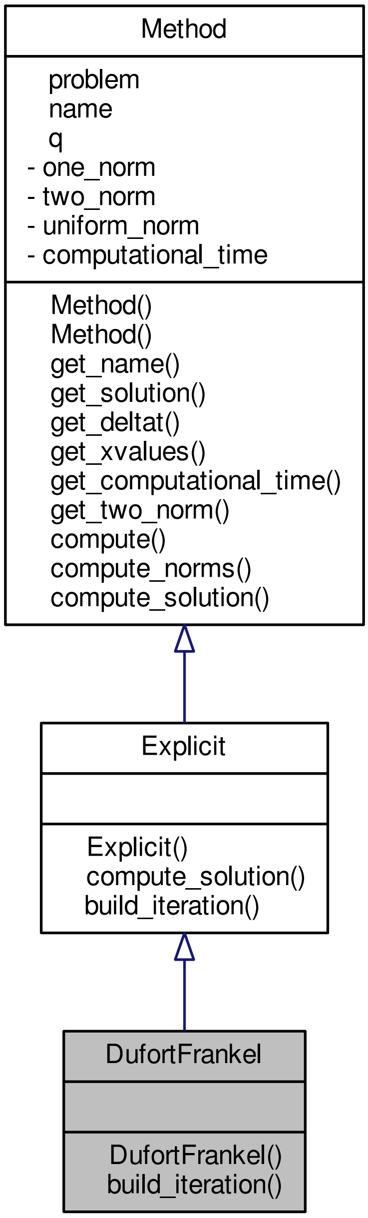
- methods/implicit/crank_nicolson.h
- methods/implicit/crank_nicolson.cpp

4.3 DufortFrankel Class Reference

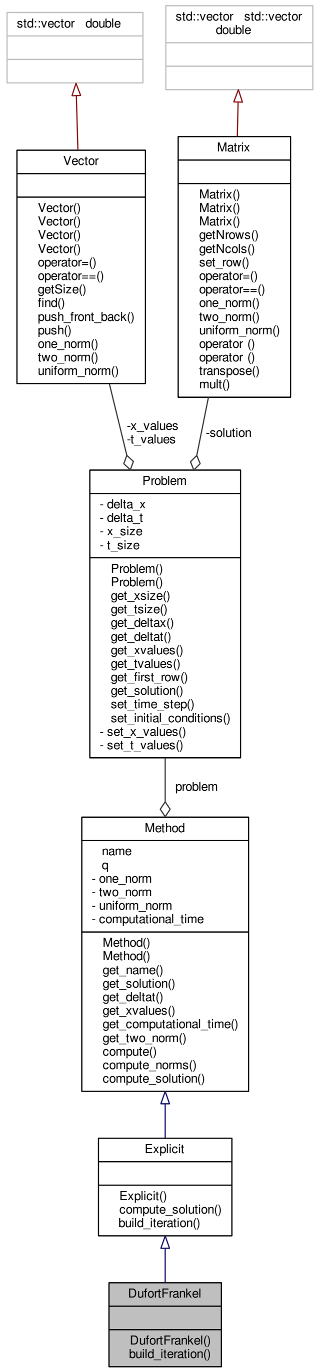
A DufortFrankel method class that contains an iteration builder.

`#include <dufort_frankel.h>`

Inheritance diagram for DufortFrankel:



Collaboration diagram for DufortFrankel:



Public Member Functions

- DufortFrankel (Problem problem)

  Default constructor.

Protected Member Functions

- Vector build_iteration (Vector current_step, Vector previous_step)

  Normal protected method.

Additional Inherited Members

4.3.1 Detailed Description

A DufortFrankel method class that contains an iteration builder.

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

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

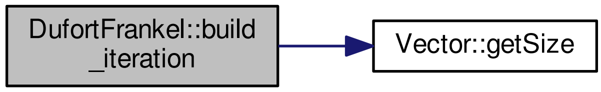
current_step	A vector representing the current time step solution.
previous_step	A vector representing the previous time step solution.

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:

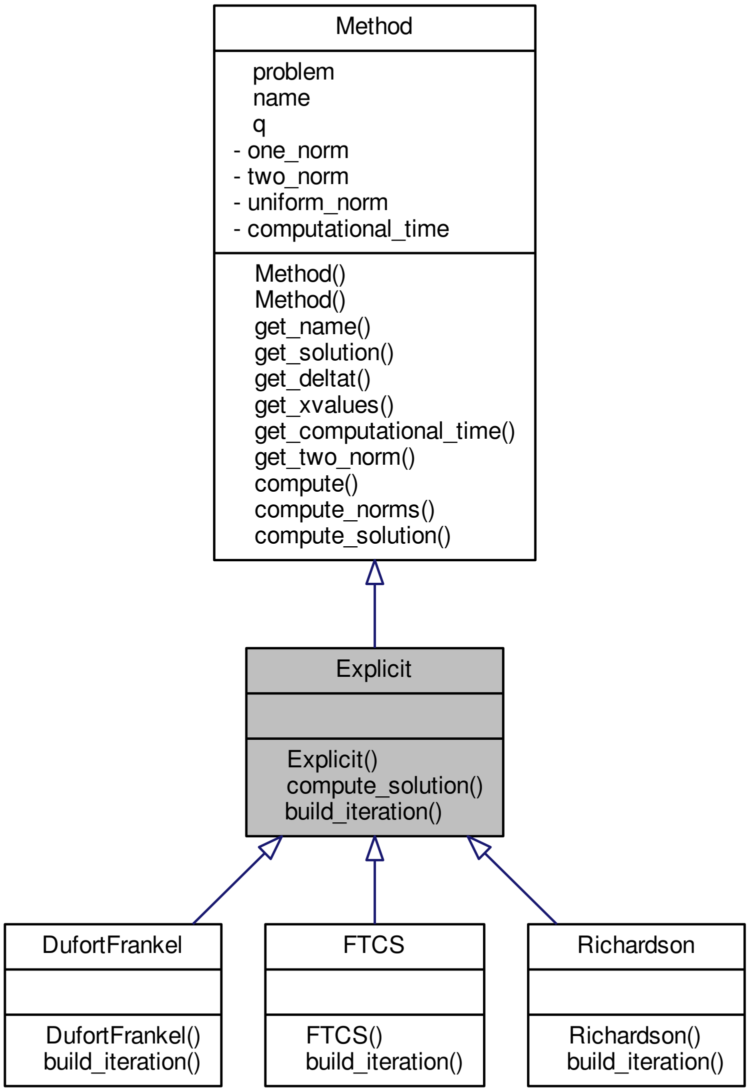
- methods/explicit/dufort_frankel.h
- methods/explicit/dufort_frankel.cpp

4.4 Explicit Class Reference

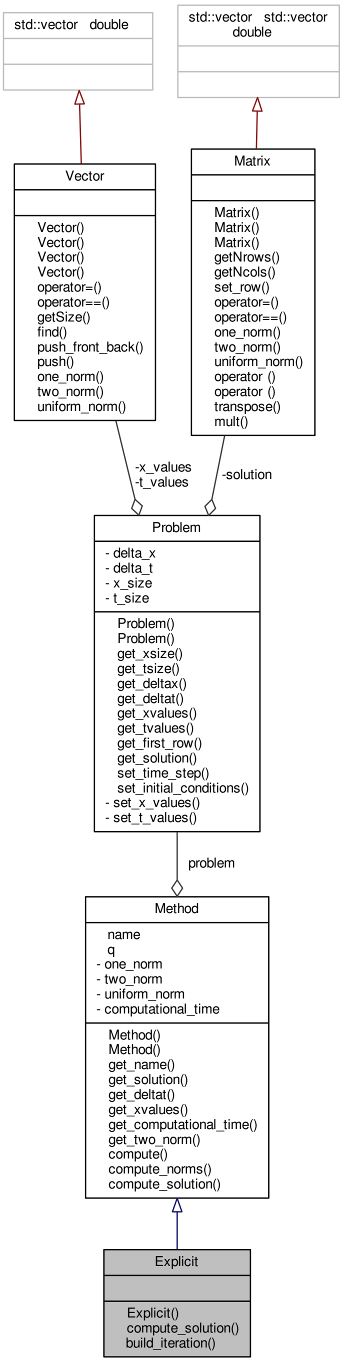
An explicit method class that contains default methods that only explicit methods use
The implementation is derived from the Method class.

```
#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.

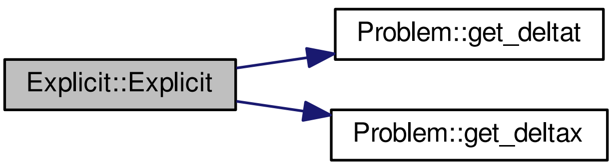
The Explicit class provides:
-a basic constructor for creating an explicit method object.
-a method to compute a solution following explicit methods rules

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

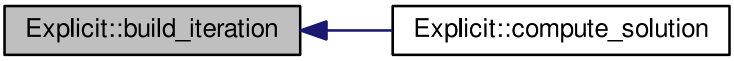
previous_step	A vector containing the previous time step solution.
current_step	A vector containing the current time step solution.

Returns

Vector. A vector representing the next time step solution.

Implemented in FTCS, DufortFrankel, and Richardson.

Here is the caller graph for this function:



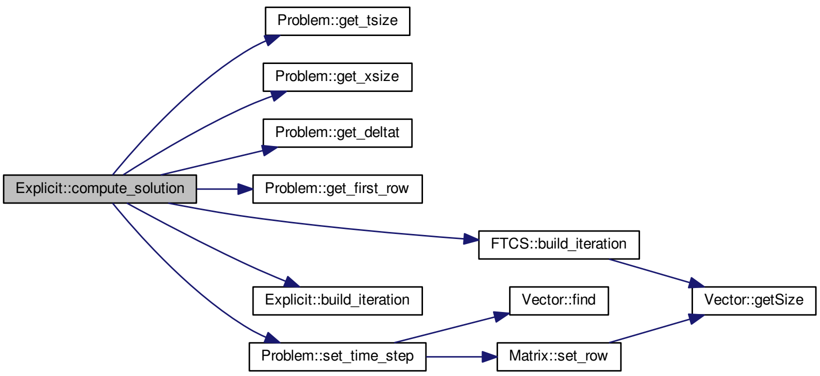
4.4.3.2 void Explicit::compute_solution ( ) [virtual]

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:

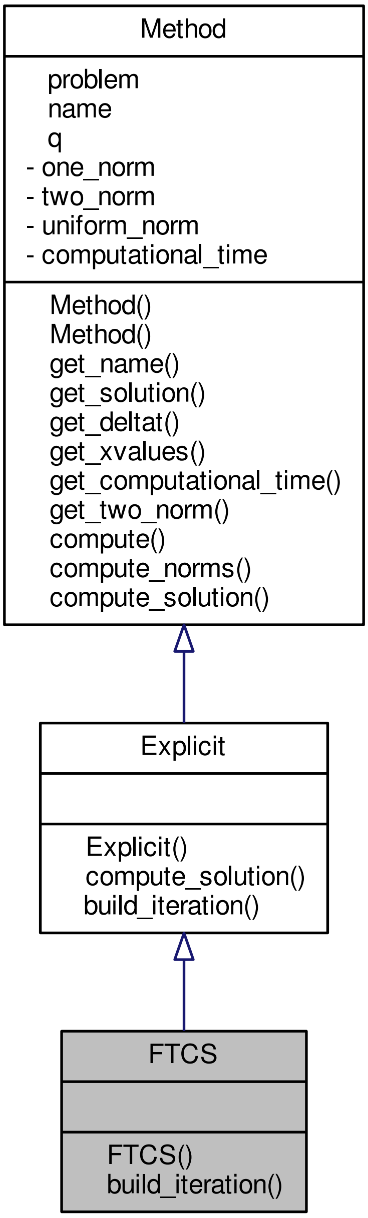
- methods/explicit/explicit.h
- methods/explicit/explicit.cpp

4.5 FTCS Class Reference

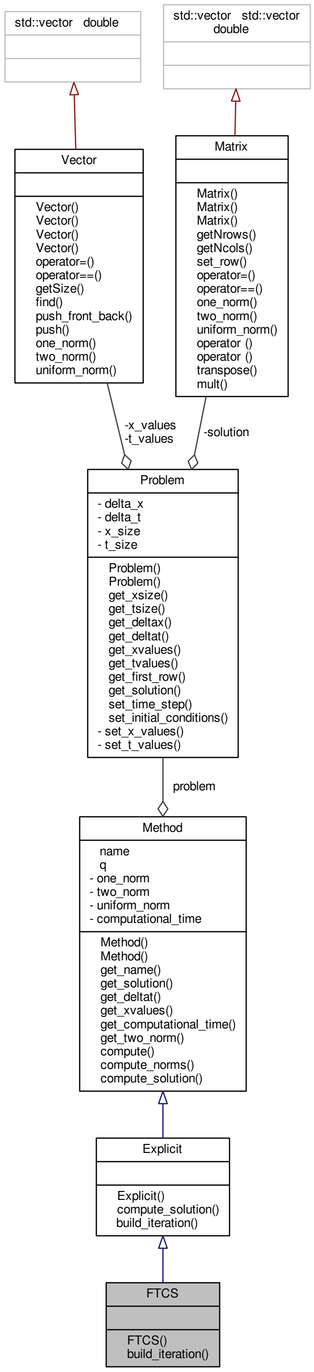
A FTCS method class that contains an iteration builder.

```
#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

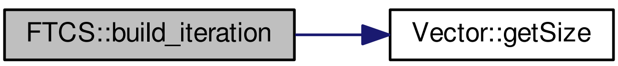
current_step	A vector with size 0, it's not required in this method.
previous_step	A vector representing the previous time step solution

Returns

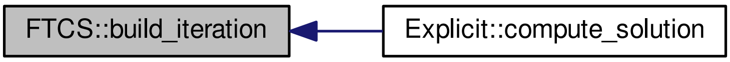
Vector. The computed solution.

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:

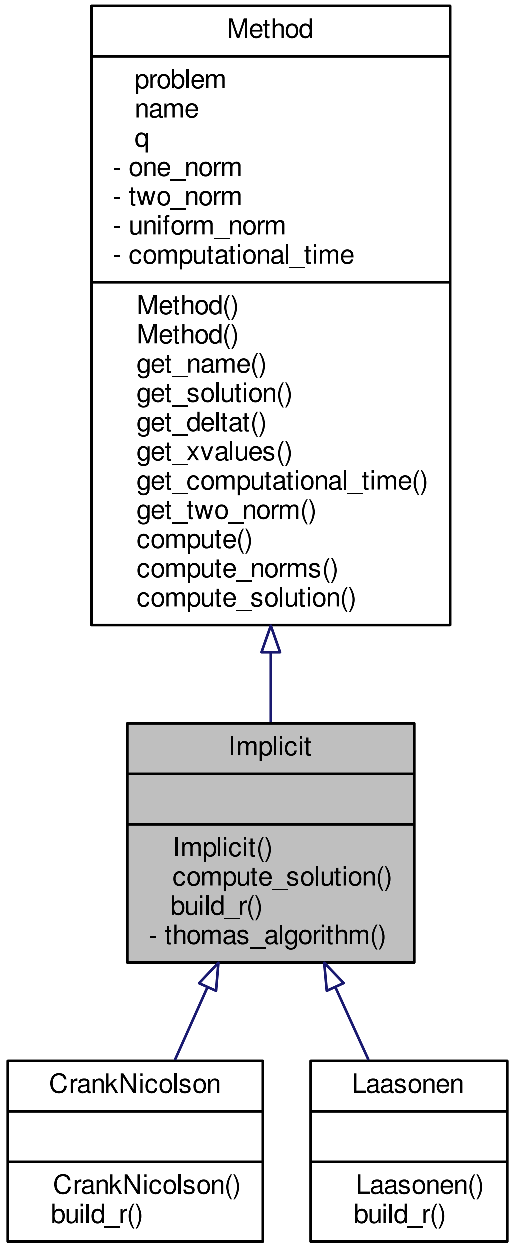
- methods/explicit/forward_t_central_s.h
- methods/explicit/forward_t_central_s.cpp

4.6 Implicit Class Reference

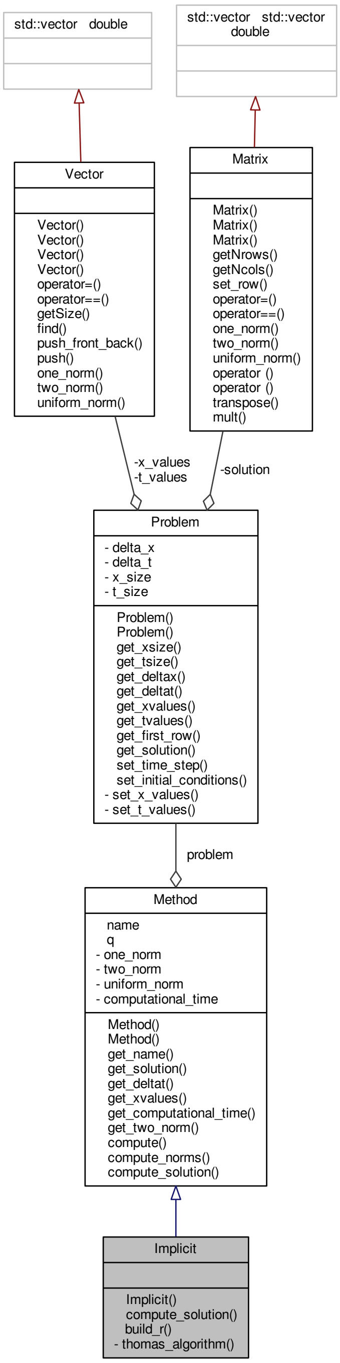
An implicit method class that contains default methods that only implicit methods use
The implementation is derived from the Method class.

```
#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.

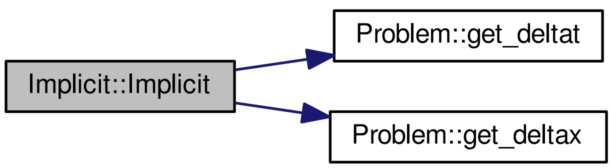
The Implicit class provides:
-a basic constructor for creating an implicit method object.
-a method to compute a solution following implicit methods rules

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 triadiogonal matrix.

Parameters

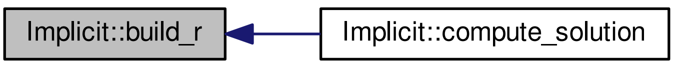
previous_step	A vector containing the previous time step solution.

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:



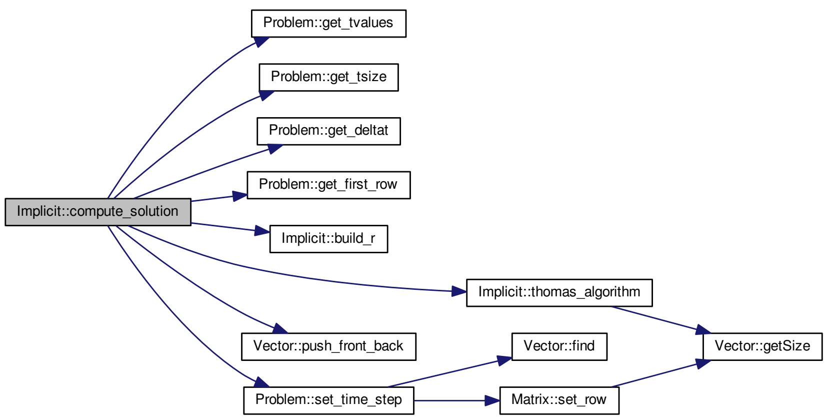
4.6.3.2 void Implicit::compute_solution ( ) [virtual]

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)

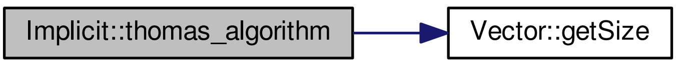
Parameters

r	Vector calculated by the build_r method.
a	Lower diagonal value of the tridiagonal matrix
b	Center diagonal value of the tridiagonal matrix
c	Upper diagonal value of the tridiagonal matrix

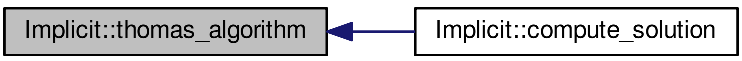
Returns

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:

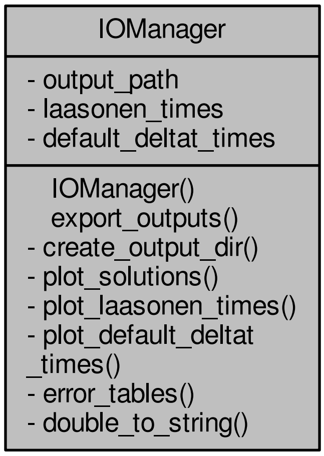
- methods/implicit/implicit.h
- methods/implicit/implicit.cpp

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, ploting 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

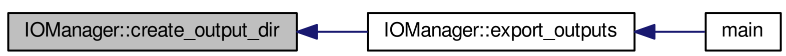
4.7.3.1 bool IOManager::create_output_dir ( ) [private]

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 precison of 2 decimal places.

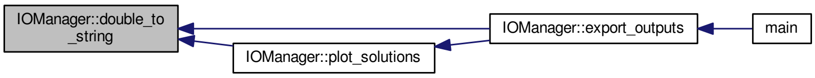
Parameters

double	value Number to be converted
int	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

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

Here is the caller graph for this function:

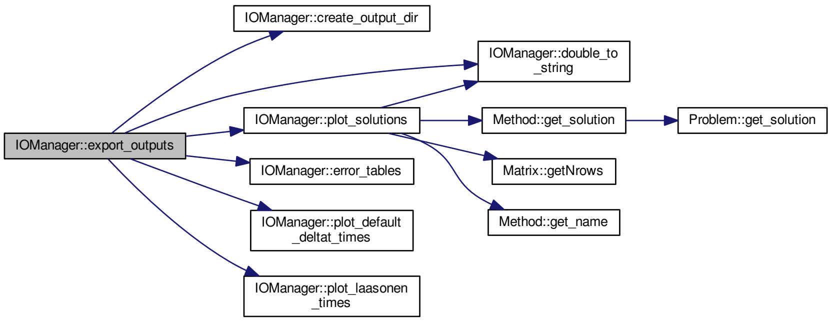
4.7.3.4 void IOManager::export_outputs (**Method** * *analytical*, std::vector< **Method** * > *methods*)

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

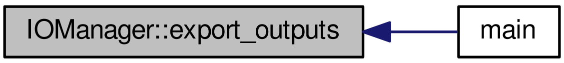
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.5 void IOManager::plot_default_deltat_times ( ) [private]

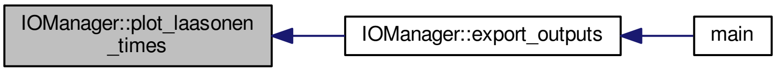
Exports a plot with four methods computational times.

Here is the caller graph for this function:

4.7.3.6 void IOManager::plot_laasonen_times ( ) [private]

Exports a plot with Laasonen delta t variation computational times.

Here is the caller graph for this function:



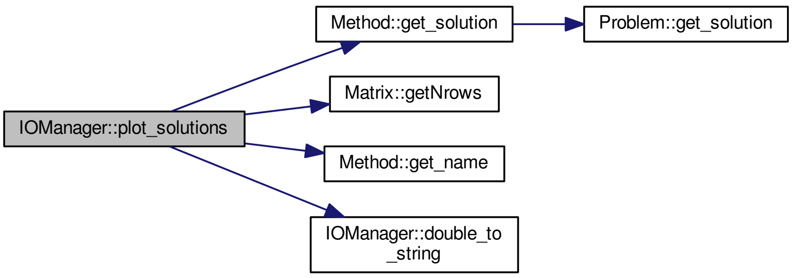
4.7.3.7 void IOManager::plot_solutions (std::string *output_name,* Method ∗ *analytical,* Method ∗ *method*) [private]

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

Parameters

string	output_name File name to be exported
Method∗	analytical The analytical solution
Method∗	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 ouput directory path name.

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

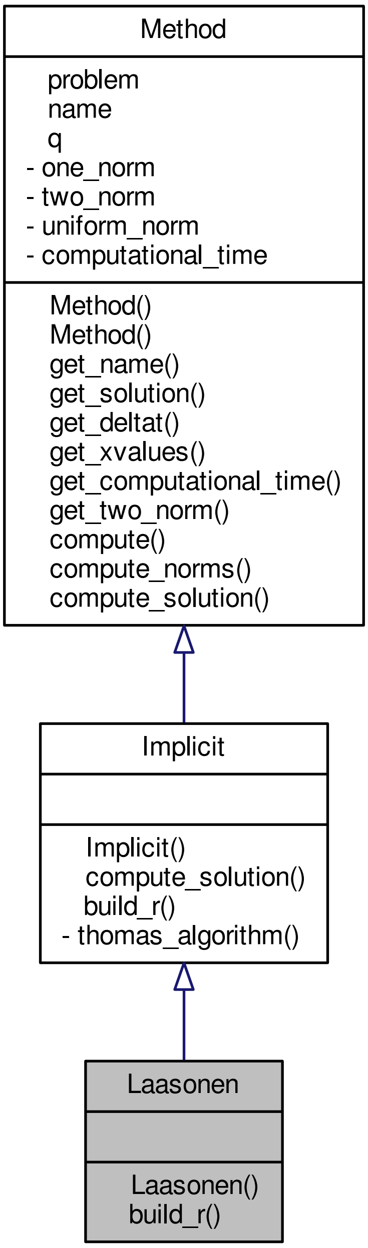
- io/iomanager.h
- io/iomanager.cpp

4.8 Laasonen Class Reference

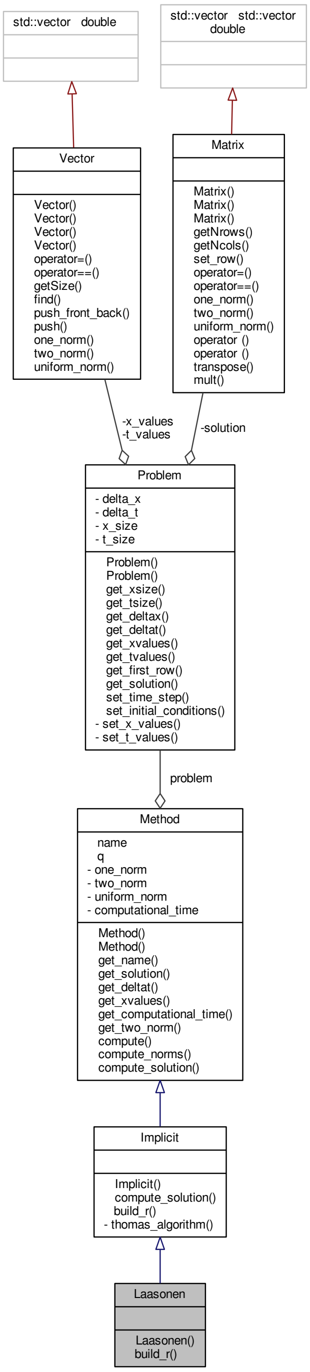
A Laasonen method class that contains a r vector builder.

`#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

previous_step	Vector representing the solution of the previous time step.

Returns

Vector. r vector to be used in A.x = r

Implements Implicit.

Here is the call graph for this function:

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

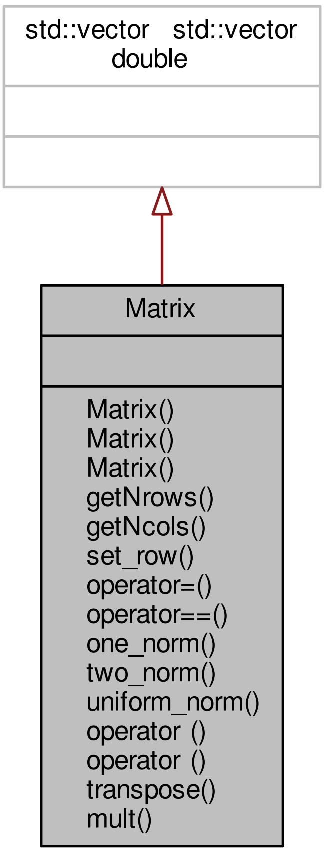
- methods/implicit/laasonen.h
- methods/implicit/laasonen.cpp

4.9 Matrix Class Reference

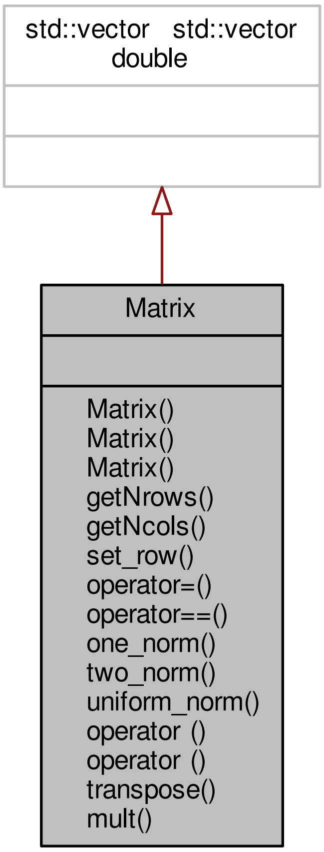
A matrix class for data storage of a 2D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

```
#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 usto 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 oput operation via >> and << operators using keyboard or file
-basic operations like access via [] operator, assignment and comparision

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 ( )

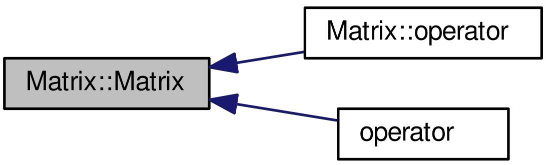
Default constructor.

Intialize 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

invalid_argument	("matrix size negative or zero")

Parameters

Nrows	int. number of rows in matrix
Ncols	int. number of columns in matrix

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

Copy constructor.

build a matrix from another matrix

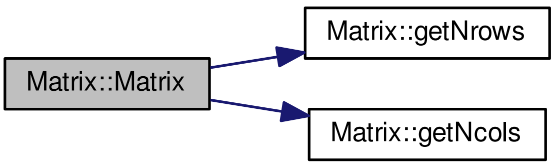
See also

Matrix()
Matrix(int Nrows, int Ncols)

Parameters

m	Matrix&. matrix to copy from

Here is the call graph for this function:



4.9.4 Member Function Documentation

4.9.4.1 int Matrix::getNcols ( ) const

Normal public get method.

get the number of columns

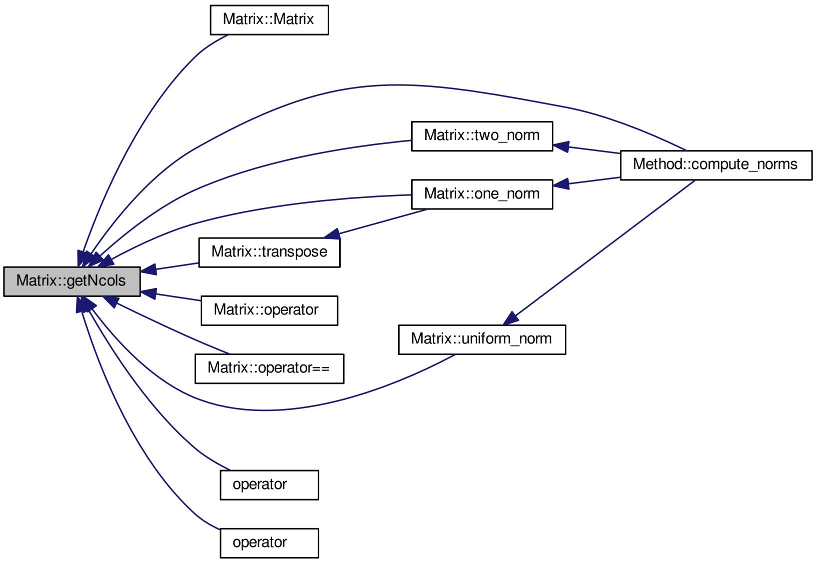
See also

int getNrows()const

Returns

int. number of columns in matrix

Here is the caller graph for this function:



4.9.4.2 int Matrix::getNrows ( ) const

Normal public get method.

get the number of rows

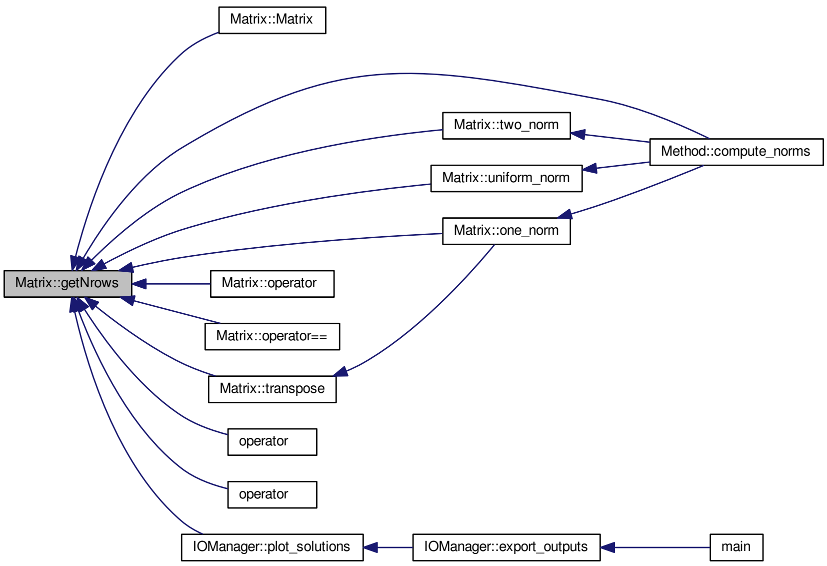
See also

int getNcols()const

Returns

int. number of rows in matrix

Here is the caller graph for this function:



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

4.9.4.4 double Matrix::one_norm ( ) const

Normal public method that returns a double.

It returns L1 norm of matrix

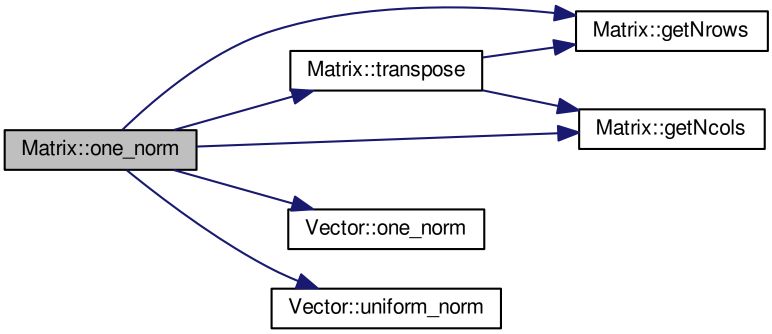
See also

two_norm()const
uniform_norm()const

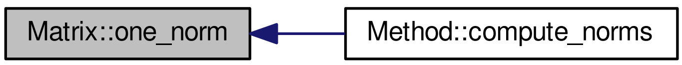
Returns

double. matrix L1 norm

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.4.5 Matrix Matrix::operator∗ (const Matrix & *a*) const

Overloaded ∗operator that returns a Matrix.

It Performs matrix by matrix multiplication.

See also

operator∗(const Matrix & a) const

Exceptions

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

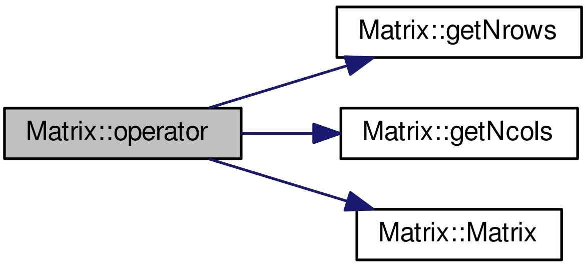
Returns

Matrix. matrix-matrix product

Parameters

a	Matrix. matrix to multiply by

Here is the call graph for this function:



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

Overloaded *operator that returns a Vector.

It Performs matrix by vector multiplication.

See also

operator*(const Matrix & a)const

Exceptions

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

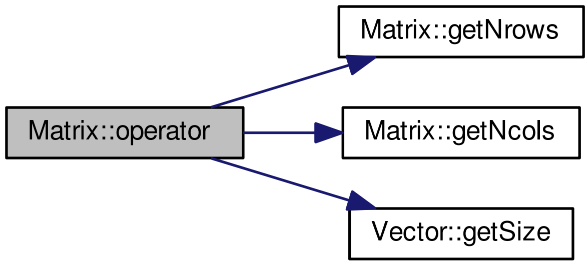
Returns

Vector. matrix-vector product

Parameters

v	Vector. Vector to multiply by

Here is the call graph for this function:



4.9.4.7 Matrix & Matrix::operator= (const Matrix & *m*)

Overloaded assignment operator.

See also

operator==(const Matrix& m)const

Returns

Matrix&. the matrix on the left of the assignment

Parameters

m	Matrix&. Matrix to assign from

4.9.4.8 bool Matrix::operator== (const Matrix & *m*) const

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

See also

operator=(const Matrix& m)

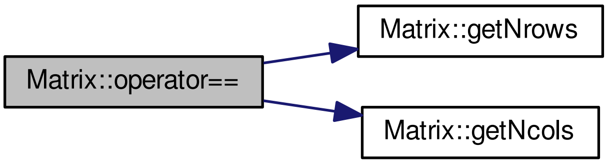
Returns

bool. true or false

Parameters

m	Matrix&. Matrix to compare to

Here is the call graph for this function:



4.9.4.9 void Matrix::set_row (int *index*, Vector *v*)

Normal public set method.

replace a row with a given vector

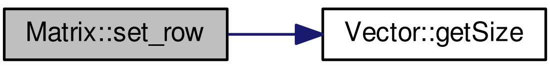
Parameters

index	Index of row to mutate
v	New vector

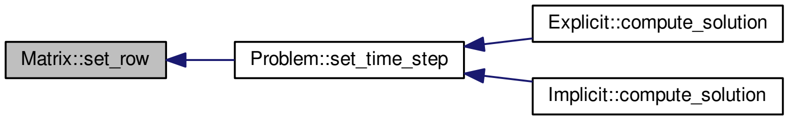
Exceptions

out_of_range	("index out of range.\n")
out_of_range	("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

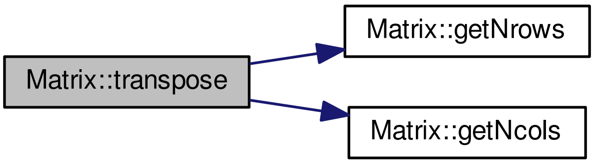
public method that returns the transpose of the matrix.

It returns the transpose of matrix

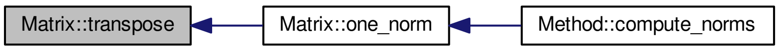
Returns

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

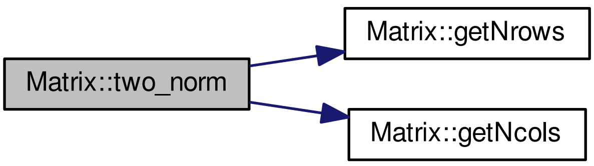
See also

one_norm()const
uniform_norm()const

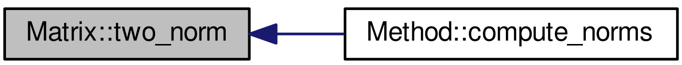
Returns

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

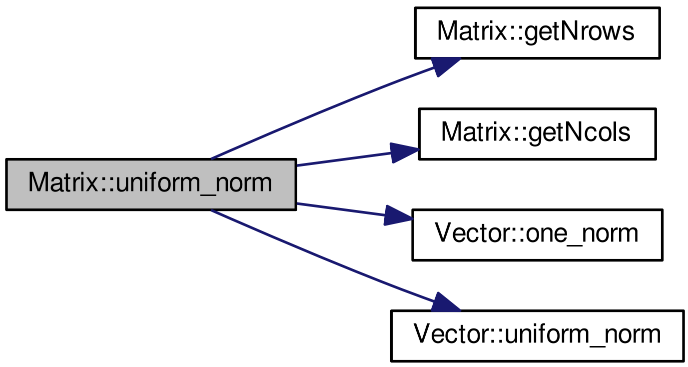
See also

one_norm()const
two_norm()const

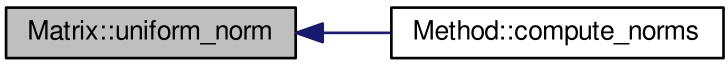
Returns

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

os	Display output stream
m	Matrix to read from

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

Overloaded ofstream << operator.

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

See also

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

Exceptions

std::invalid_argument	("file read error - negative matrix size");

Returns

std::ofstream&. The ofstream object

Parameters

m	Matrix to read from

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

Overloaded istream >> operator.

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

See also

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

Exceptions

std::invalid_argument	("read error - negative matrix size");

Returns

std::istream&. The istream object

Parameters

is	Keyboard input stream
m	Matrix to write into

4.9.5.4 std::ifstream& operator>> (std::ifstream & *ifs*, Matrix & *m*) [friend]

Overloaded ifstream >> operator.

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

See also

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

Returns

std::ifstream&. The ifstream object

Parameters

ifs	Input file stream with opened matrix file
m	Matrix to write into

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

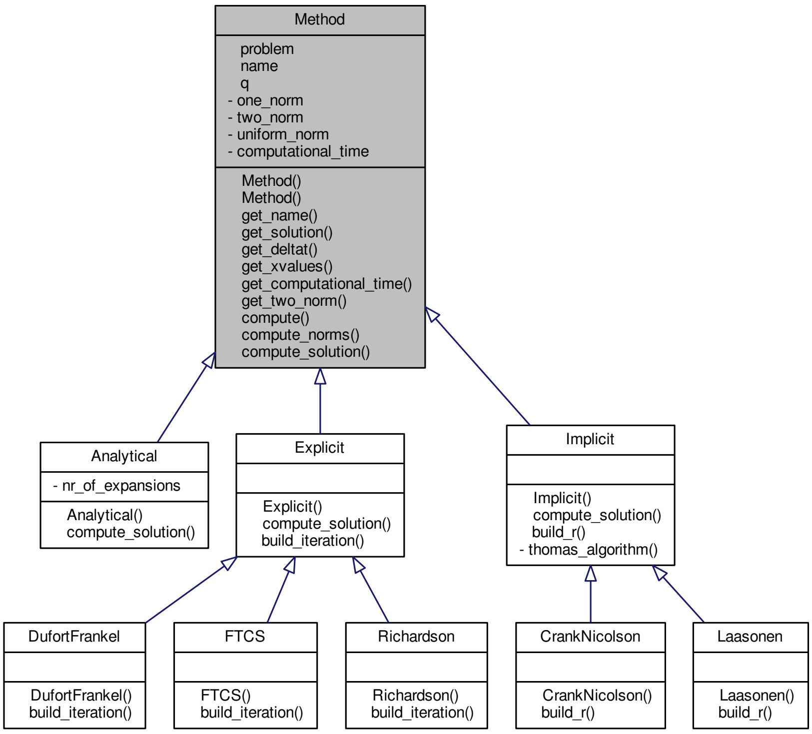
- grid/matrix.h
- grid/matrix.cpp

4.10 Method Class Reference

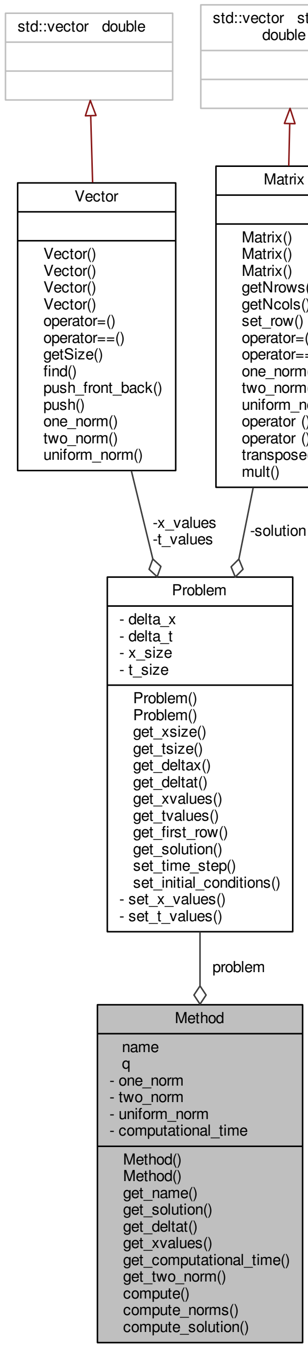
A Method class to structure information used to solve the problem.

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

Inheritance diagram for Method:



Collaboration diagram for Method:



Public Member Functions

- Method ()

    Default constructor.
- Method (Problem problem)

    Alternate constructor.
- std::string get_name ()

Normal public get method.
- Matrix get_solution ()

  Normal public get method.
- double get_deltat ()

  Normal public get method.
- Vector get_xvalues ()

  Normal public get method.
- double get_computational_time ()

  Normal public get method.
- double get_two_norm ()

  Normal public get method.
- void compute ()

  Normal public method.
- void compute_norms (Matrix analytical_matrix)
- virtual void compute_solution ()=0

  A pure virtual member.

Protected Attributes

- Problem problem

  Protected Problem problem.
- std::string name

  Protected string name.
- double q

  Protected double q.

Private Attributes

- double one_norm
- double two_norm
- double uniform_norm
- double computational_time

  Private double computational_time.

4.10.1 Detailed Description

A Method class to structure information used to solve the problem.

The Method class provides:
-basic constructors for creating a Method object.
-acessor methods to retrieve valuable information
-mutator methods to change the problem grid system

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Method::Method ( )

Default constructor.

Intialize a Method object

See also

Method(Problem problem)

4.10.2.2 Method::Method (Problem *problem*)

Alternate constructor.

Initializes a Method with a given parabolic problem.

See also

Method()

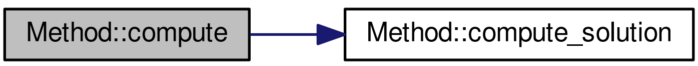
4.10.3 Member Function Documentation

4.10.3.1 void Method::compute ( )

Normal public method.

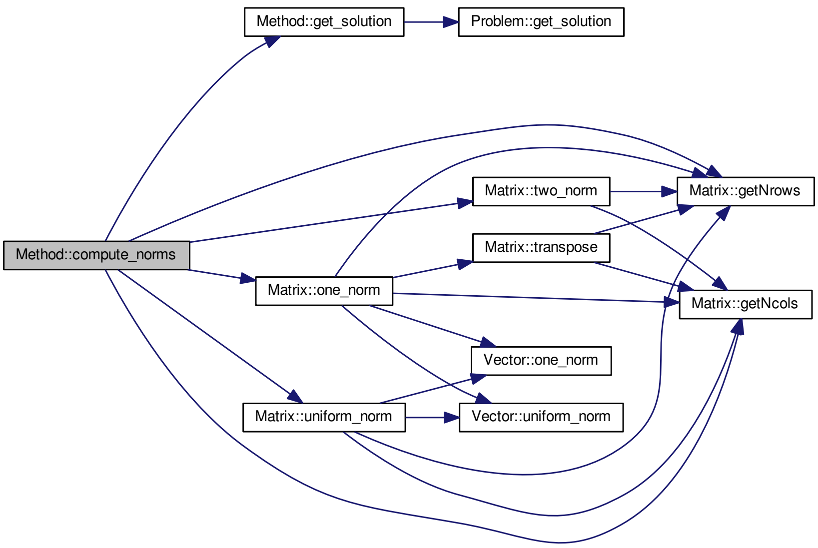
Keeps track of the time to compute a solution

Here is the call graph for this function:



4.10.3.2 void Method::compute_norms (**Matrix** *analytical_matrix*)

Here is the call graph for this function:



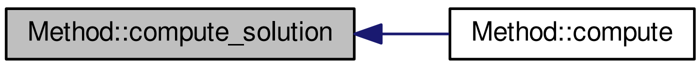
4.10.3.3 virtual void Method::compute_solution ( ) `[pure virtual]`

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 ( )

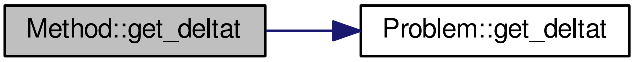
Normal public get method.

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 ( )

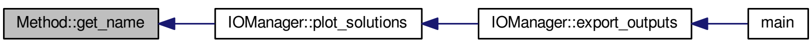
Normal public get method.

get the method name

Returns

string. Method name.

Here is the caller graph for this function:



4.10.3.7 Matrix Method::get_solution ( )

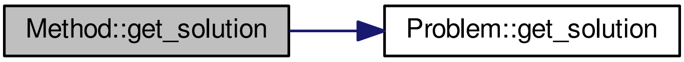
Normal public get method.

get the solution grid

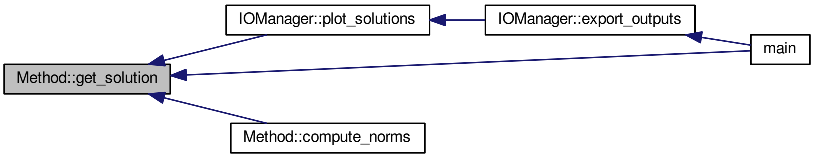
Returns

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 ( )

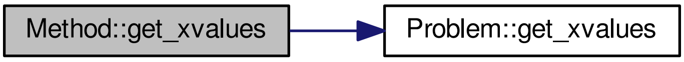
Normal public get method.

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 coeficient 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:

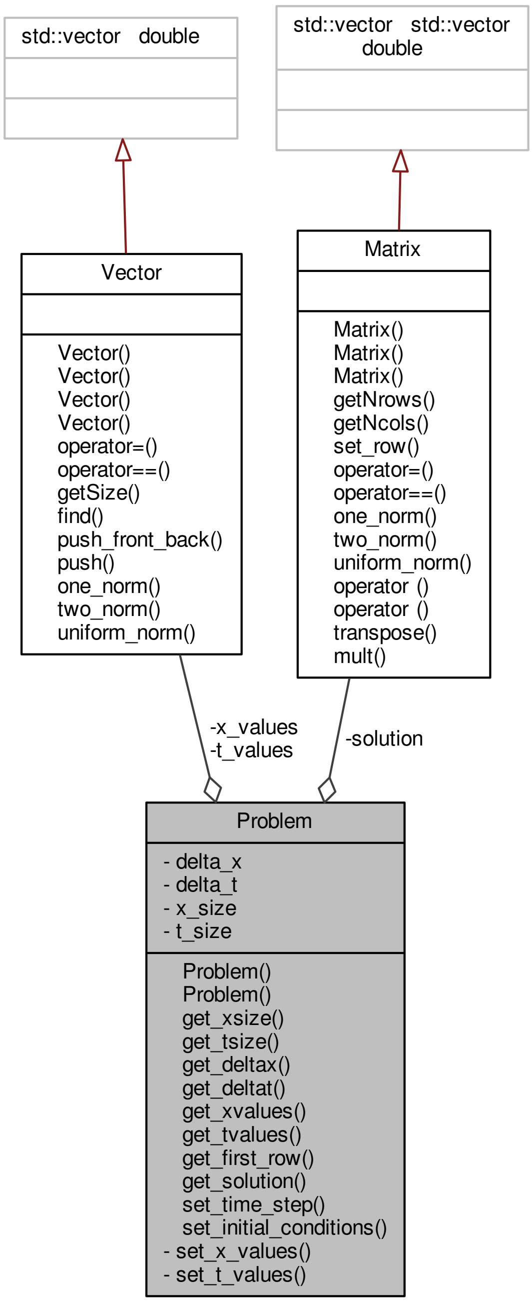
- methods/method.h
- methods/method.cpp

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)
  Intialize 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()

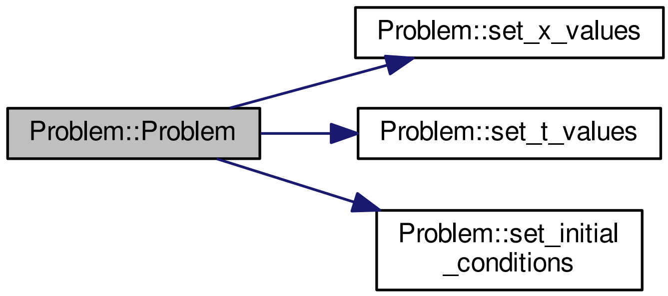
Parameters

dt	Time step to assign
dx	Space step to assign

Exceptions

out_of_range	("space step can't be negative or zero")
out_of_range	("time step can't be negative or zero")

Here is the call graph for this function:



4.11.3 Member Function Documentation

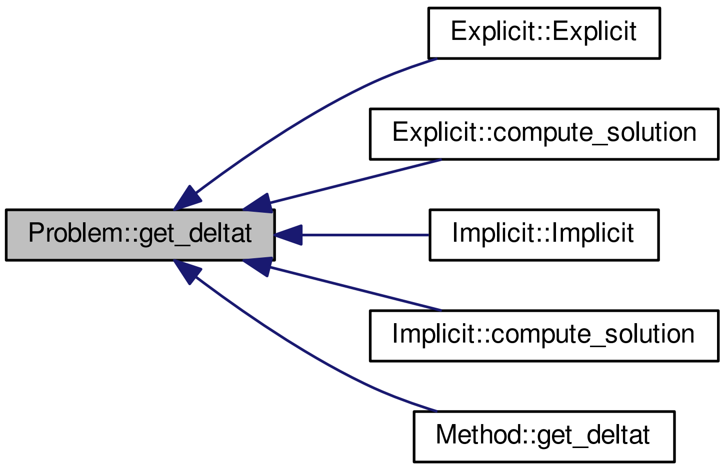
4.11.3.1 double Problem::get_deltat ( )

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:



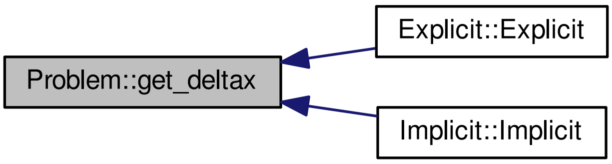
4.11.3.2 double Problem::get_deltax ( )

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:



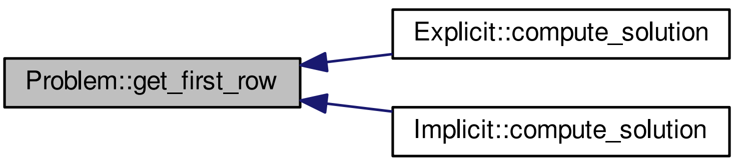
4.11.3.3 Vector Problem::get_first_row ( )

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:



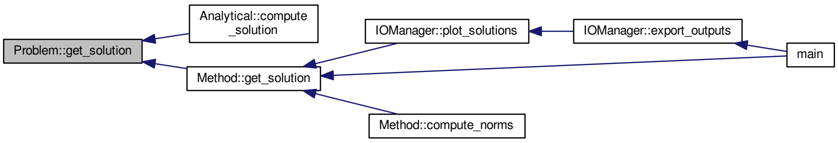
4.11.3.4 Matrix * Problem::get_solution ( )

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:



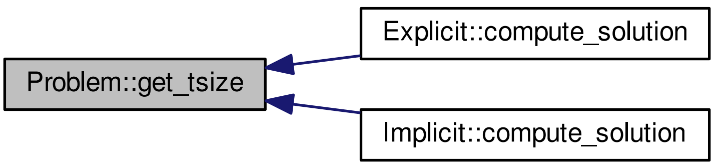
4.11.3.5 unsigned int Problem::get_tsize ( )

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:



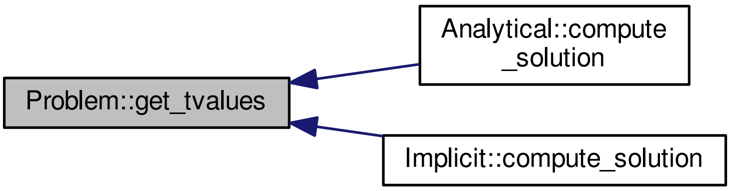
4.11.3.6 Vector Problem::get_tvalues ( )

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:



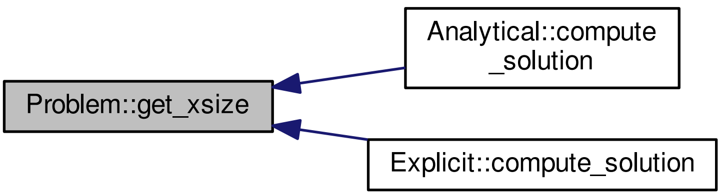
4.11.3.7 unsigned int Problem::get_xsize ( )

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:



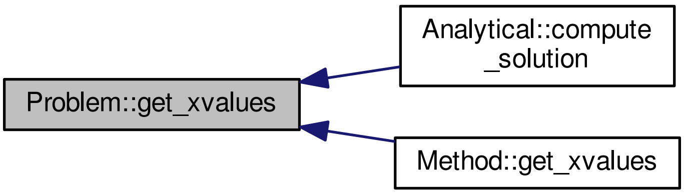
4.11.3.8 Vector Problem::get_xvalues ( )

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:

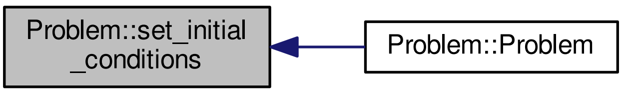


4.11.3.9 void Problem::set_initial_conditions ( )

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]

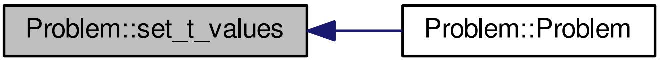
Normal private set method.

Intialize 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*)

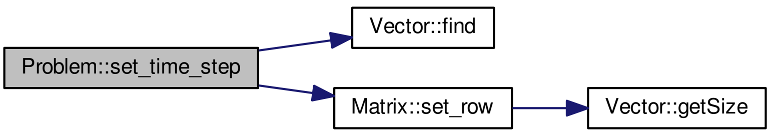
Normal public set method.

replace a row of the solution for a given Vector.

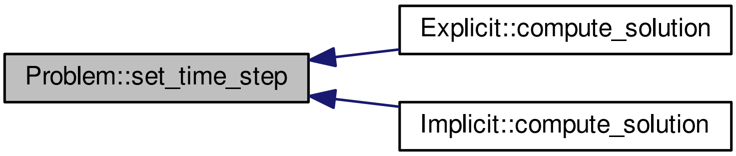
Parameters

step	Vector conatining the new values.
time	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]

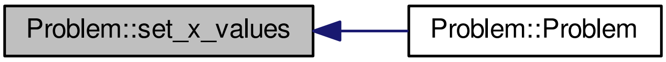
Normal private set method.

Intialize 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:

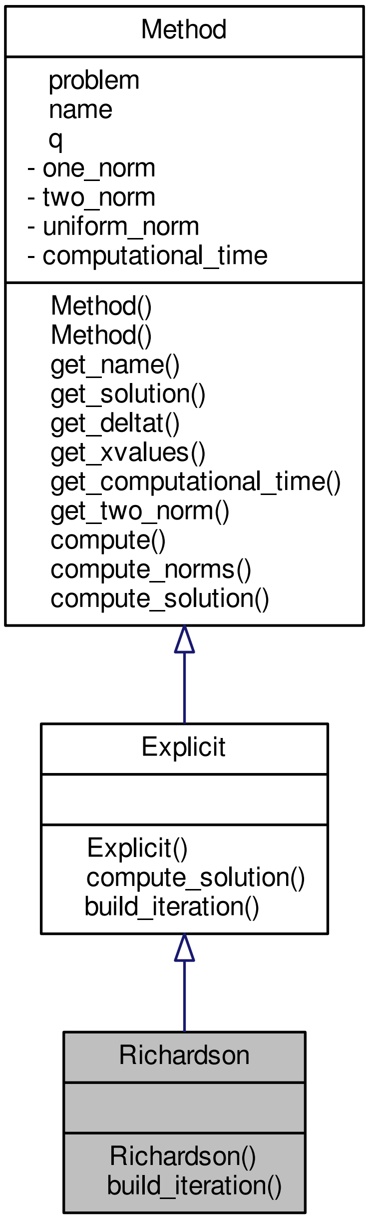
- variants/problem.h
- variants/problem.cpp

4.12 Richardson Class Reference

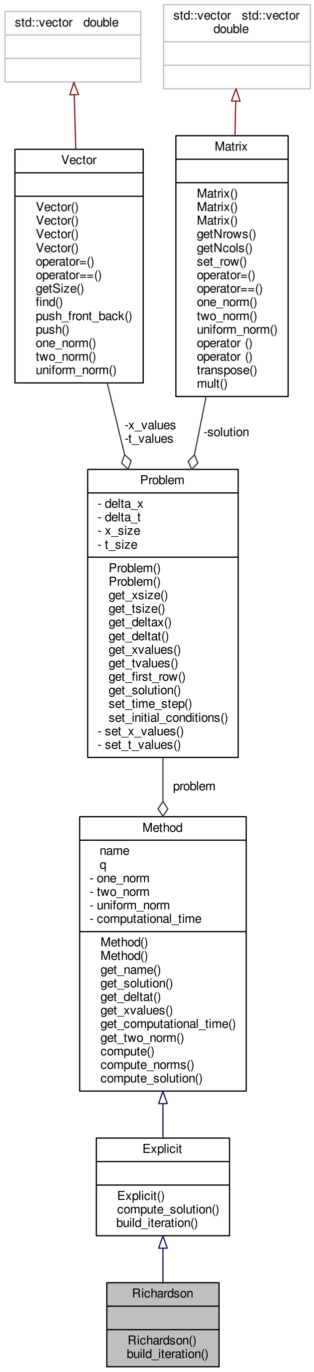
A Richardson method class that contains an iteration builder.

`#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

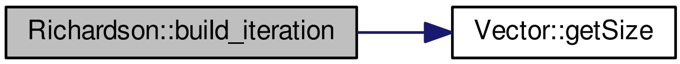
current_step	A vector representing the current time step solution.
previous_step	A vector representing the previous time step solution.

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:

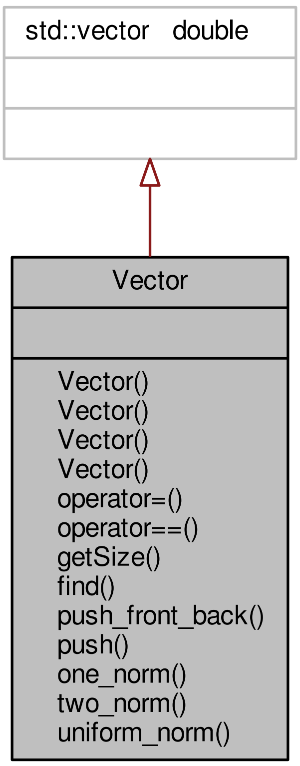
- methods/explicit/richardson.h
- methods/explicit/richardson.cpp

4.13 Vector Class Reference

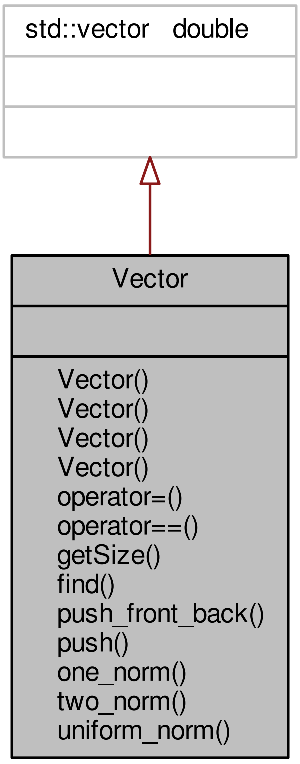
A vector class for data storage of a 1D array of doubles
The implementation is derived from the standard container vector std::vector
We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

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

Inheritance diagram for Vector:



Collaboration diagram for Vector:



Public Member Functions

- Vector ()

    Default constructor.
- Vector (int Num)

    Explicit alterative constructor takes an intiger.
- 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 ifstream << 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 usto 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 obcjet from other vector object, or by creating empty vector of a given size,
-input and oput operation via >> and << operators using keyboard or file
-basic operations like access via [] operator, assignment and comparision

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 ( )

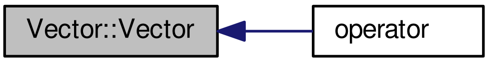
Default constructor.

Intialize 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 alterative constructor takes an intiger.

it is explicit since implicit type conversion int -> vector doesn't make sense Intialize Vector object of size Num

See also

Vector()
Vector(const Vector& v)

Exceptions

invalid_argument	("vector size negative")

Parameters

Num	int. Size of a vector

4.13.3.3 Vector::Vector (const Vector & *v*)

Copy constructor takes an Vector object reference.

Intialize 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.

Intialize 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.

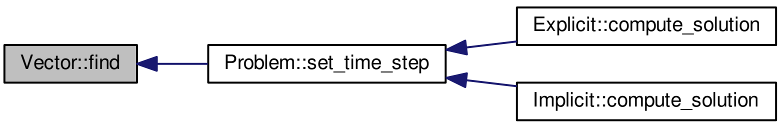
Parameters

value	Value to find

Returns

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

Here is the caller graph for this function:



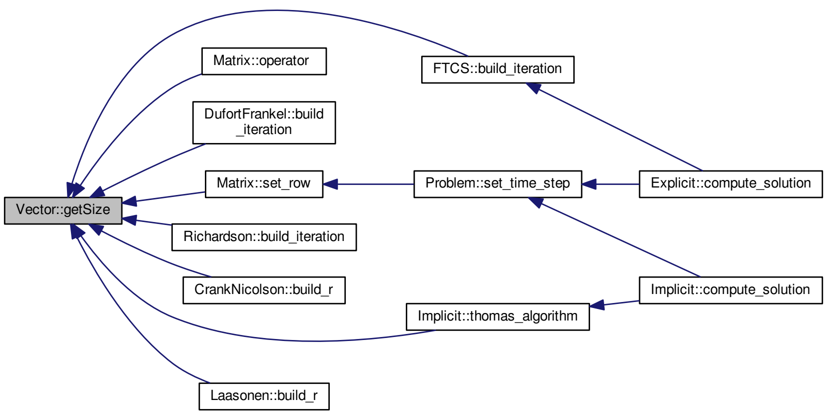
4.13.4.2 int Vector::getSize ( ) const

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

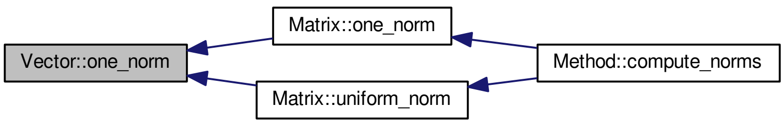
See also

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

v	Vector to assign from

Returns

the object on the left of the assignment

Parameters

v	Vecto&. Vector to assign from

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

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

See also

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

Returns

bool. true or false

Exceptions

invalid_argument	("incompatible vector sizes\n")

Parameters

v	Vector&. vector to compare

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

Method to push a value to the last position of a Vector.

Parameters

value	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

value	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

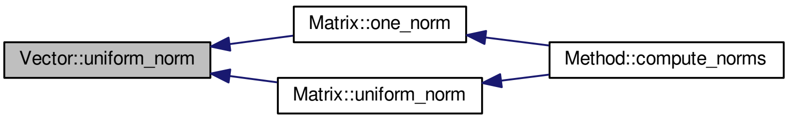
Exceptions

out_of_range	("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

os	output file stream
v	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

ofs	outputfile stream. With opened file
v	Vector&. vector to read from

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

Overloaded istream >> operator.

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

See also

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

Returns

std::istream&. the input stream object is

Exceptions

std::invalid_argument	("read error - negative vector size");

Parameters

is	keyboard input straem. For user input
v	Vector&. vector to write to

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

Overloaded ifstream >> operator.

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

See also

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

Returns

ifstream&. the input ifstream object ifs

Exceptions

std::invalid_argument	("file read error - negative vector size");

Parameters

ifs	input file straem. With opened matrix file
v	Vector&. vector to write to

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

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

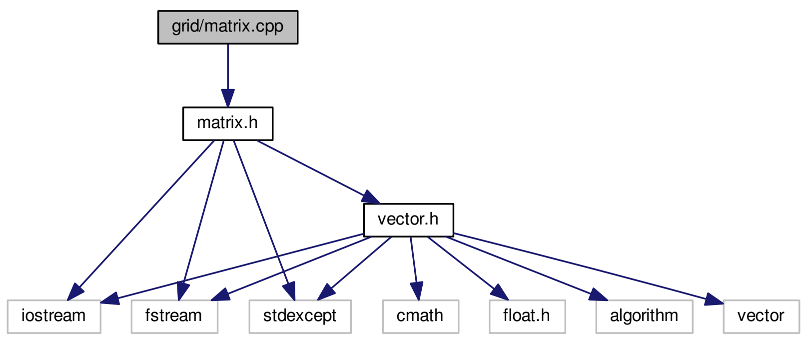
Chapter 5

File Documentation

5.1 grid/matrix.cpp File Reference

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

Include dependency graph for matrix.cpp:



Functions

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

5.1.1 Function Documentation

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

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

See also

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

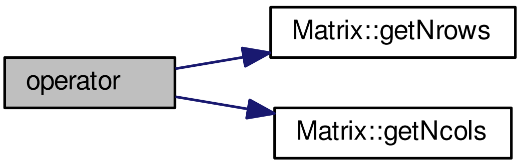
Returns

std::ostream&. The ostream object

Parameters

os	Display output stream
m	Matrix to read from

Here is the call graph for this function:



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

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

See also

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

Exceptions

std::invalid_argument	("file read error - negative matrix size");

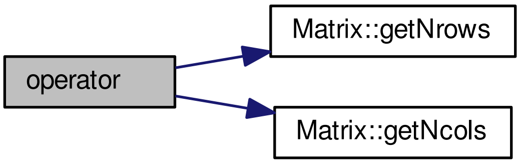
Returns

std::ofstream&. The ofstream object

Parameters

m	Matrix to read from

Here is the call graph for this function:



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

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

See also

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

Exceptions

std::invalid_argument	("read error - negative matrix size");

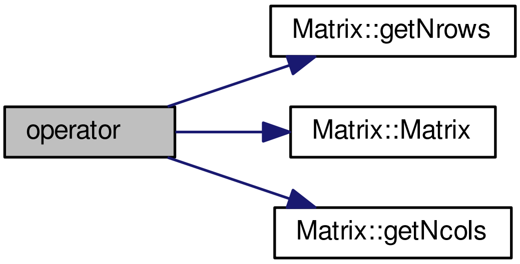
Returns

std::istream&. The istream object

Parameters

is	Keyboard input stream
m	Matrix to write into

Here is the call graph for this function:



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

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

See also

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

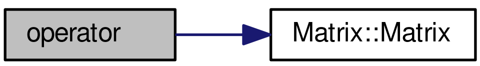
Returns

std::ifstream&. The ifstream object

Parameters

ifs	Input file stream with opened matrix file
m	Matrix to write into

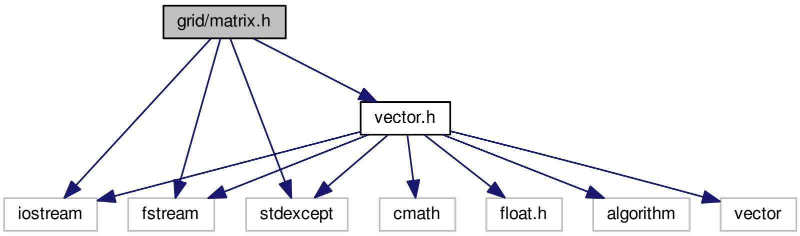
Here is the call graph for this function:



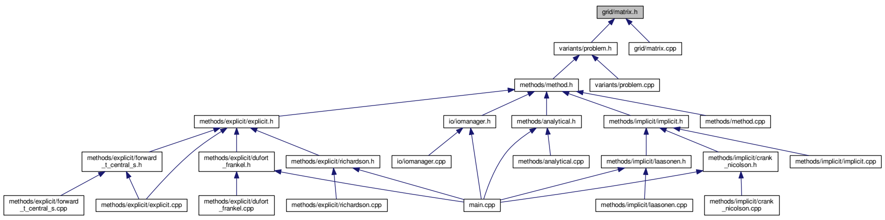
5.2 grid/matrix.h File Reference

```
#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

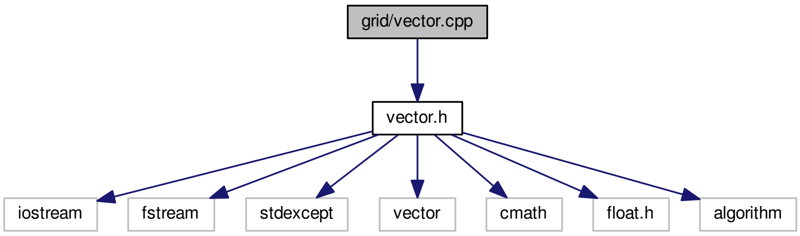
- class Matrix

  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 usto 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

os	output file stream
v	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

ofs	outputfile stream. With opened file
v	Vector&. vector to read from

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

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

See also

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

Returns

std::istream&. the input stream object is

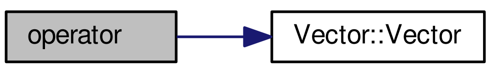
Exceptions

std::invalid_argument	("read error - negative vector size");

Parameters

is	keyboard input straem. For user input
v	Vector&. vector to write to

Here is the call graph for this function:



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

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

See also

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

Returns

ifstream&. the input ifstream object ifs

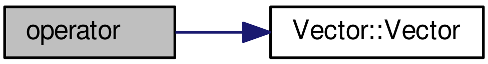
Exceptions

std::invalid_argument	("file read error - negative vector size");

Parameters

ifs	input file straem. With opened matrix file
v	Vector&. vector to write to

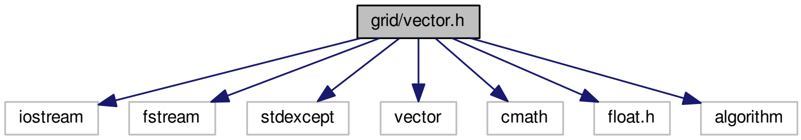
Here is the call graph for this function:



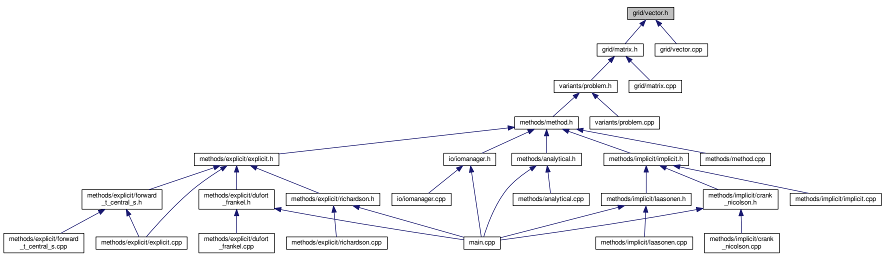
5.4 grid/vector.h File Reference

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

Include dependency graph for vector.h:



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



Classes

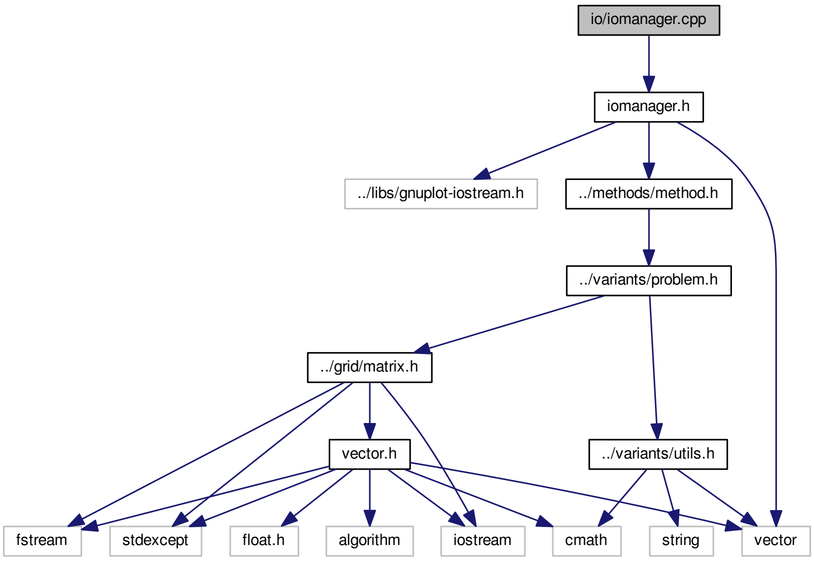
- class Vector

  A vector class for data storage of a 1D array of doubles
  The implementation is derived from the standard container vector std::vector
  We use private inheritance to base our vector upon the library version whilst usto expose only those base class functions we wish to use - in this the array access operator [].

5.5 io/iomanager.cpp File Reference

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

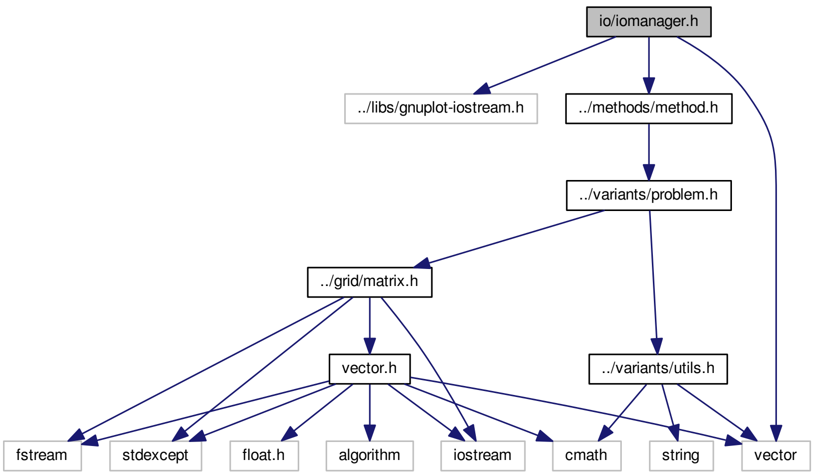
Include dependency graph for iomanager.cpp:



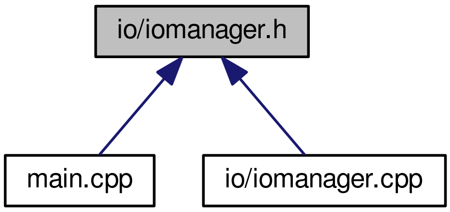
5.6 io/iomanager.h File Reference

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

Include dependency graph for iomanager.h:



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



Classes

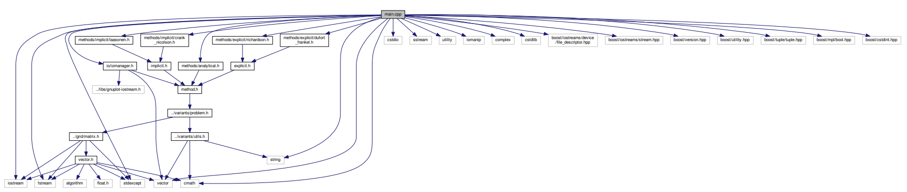
- class IOManager

  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:



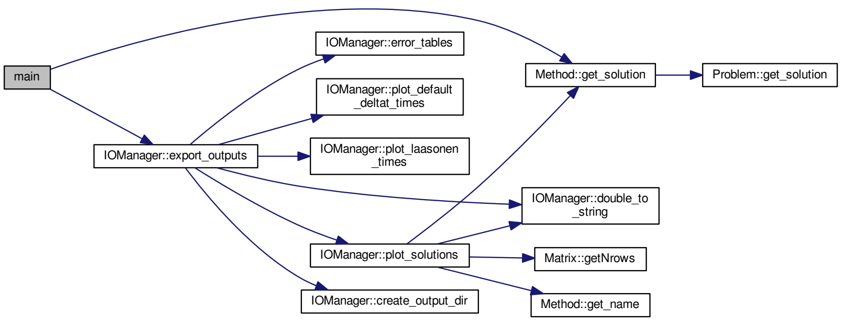
Functions

- int main ()

5.7.1 Function Documentation

5.7.1.1 int main ( )

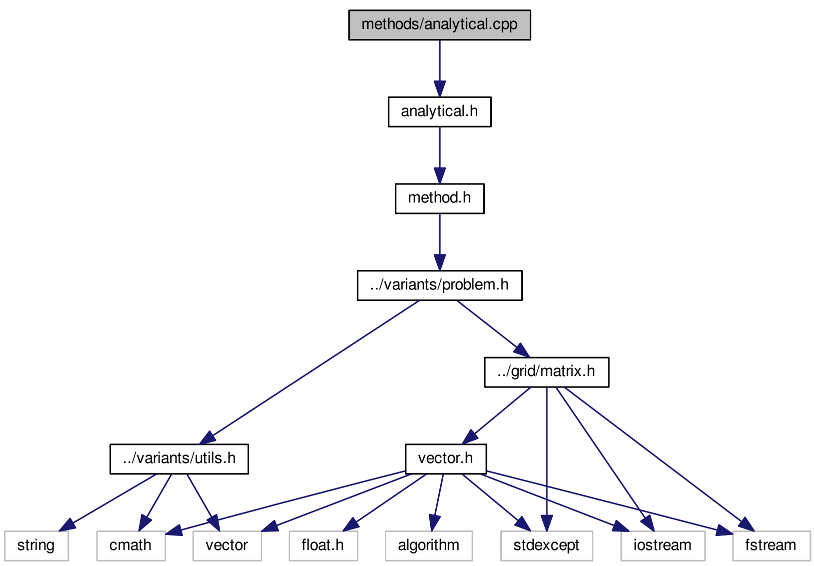
Here is the call graph for this function:



5.8 methods/analytical.cpp File Reference

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

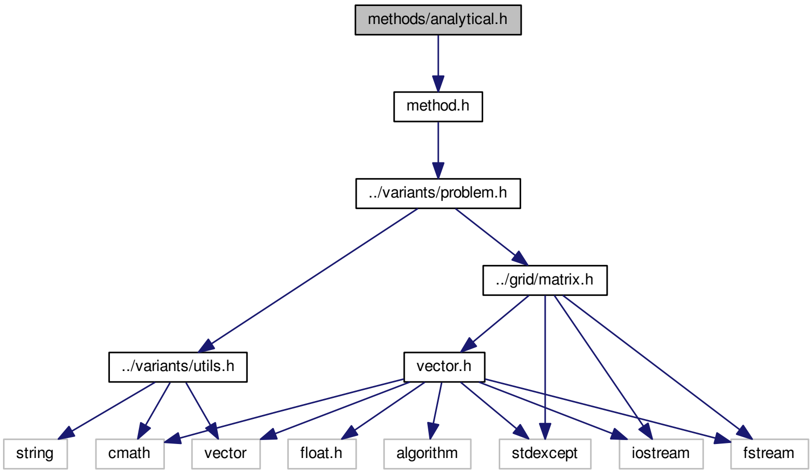
Include dependency graph for analytical.cpp:



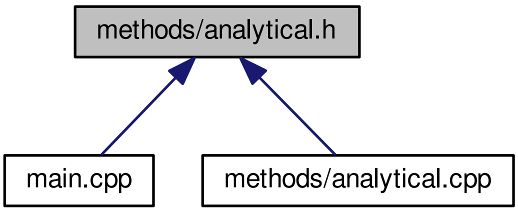
5.9 methods/analytical.h File Reference

`#include "method.h"`

Include dependency graph for analytical.h:



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



Classes

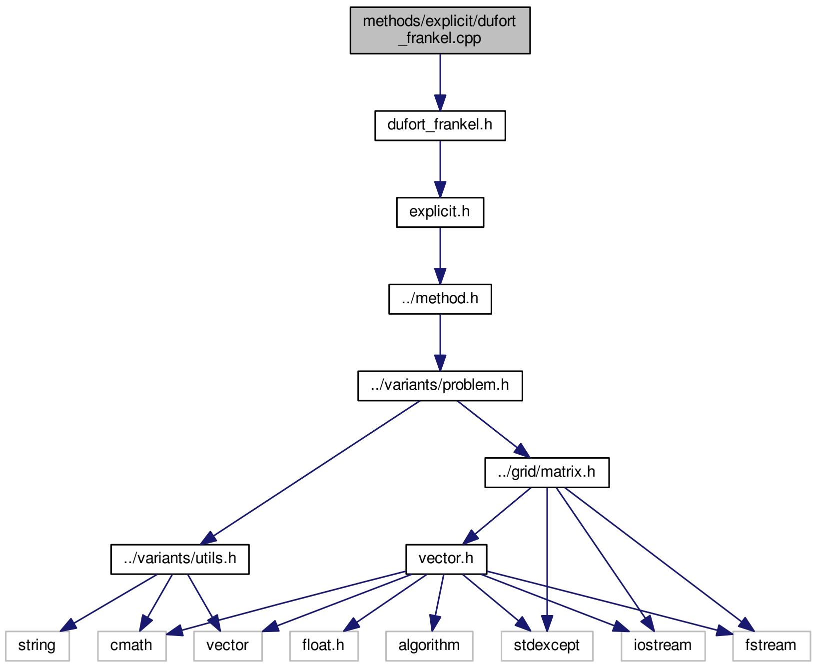
- class Analytical

  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"
```

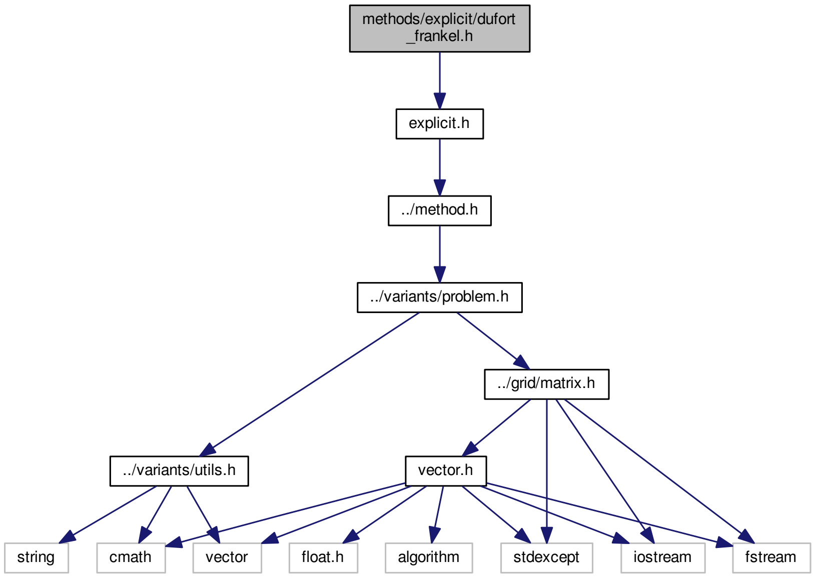
Include dependency graph for dufort_frankel.cpp:



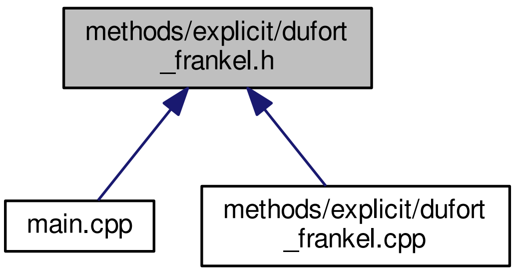
5.11 methods/explicit/dufort_frankel.h File Reference

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

Include dependency graph for dufort_frankel.h:



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



Classes

- class DufortFrankel

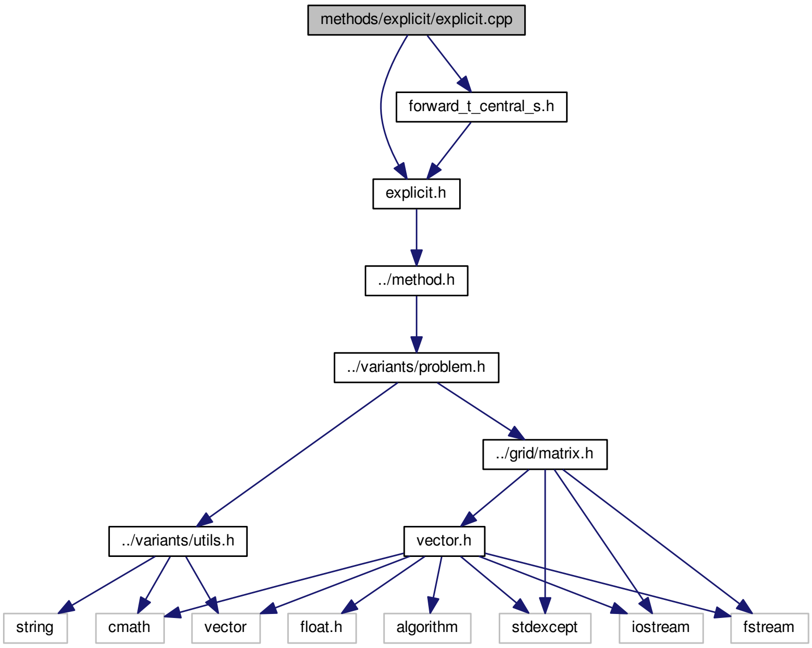
    A DufortFrankel method class that contains an iteration builder.

5.12 methods/explicit/explicit.cpp File Reference

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

`#include "forward_t_central_s.h"`

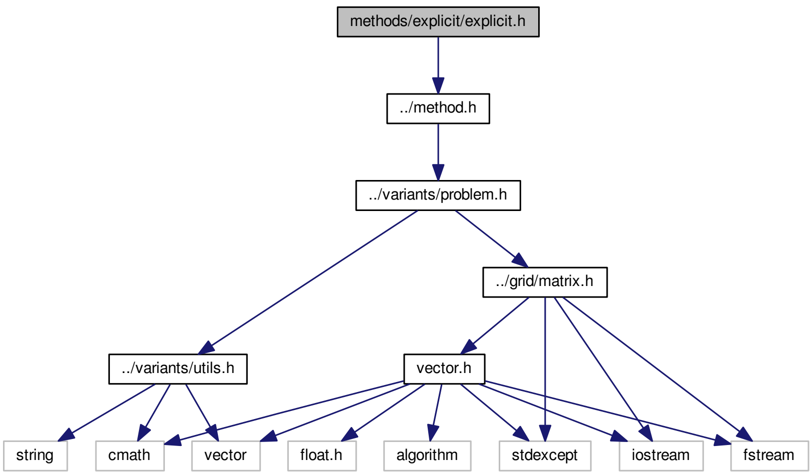
Include dependency graph for explicit.cpp:



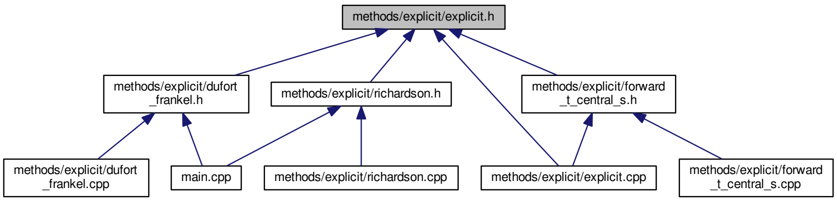
5.13 methods/explicit/explicit.h File Reference

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

Include dependency graph for explicit.h:



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



Classes

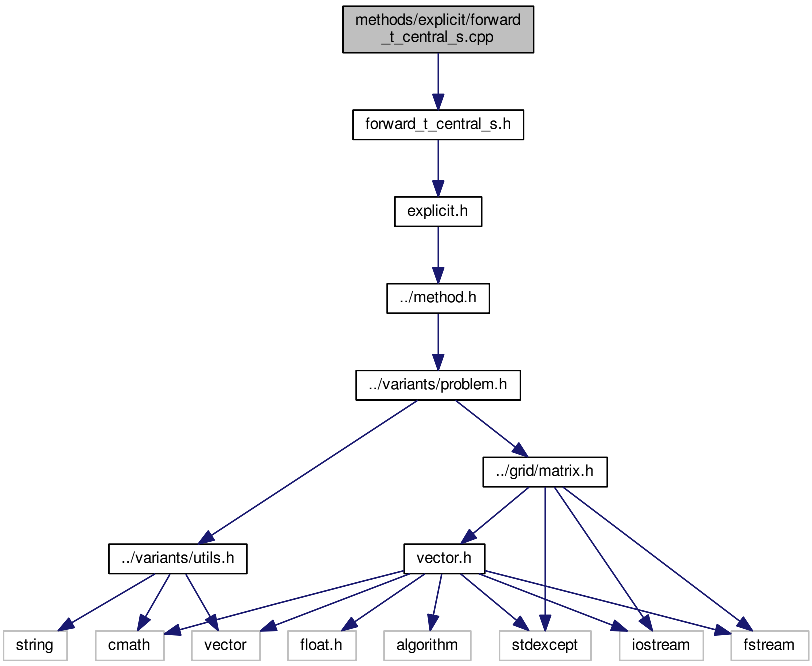
- class Explicit

  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"`

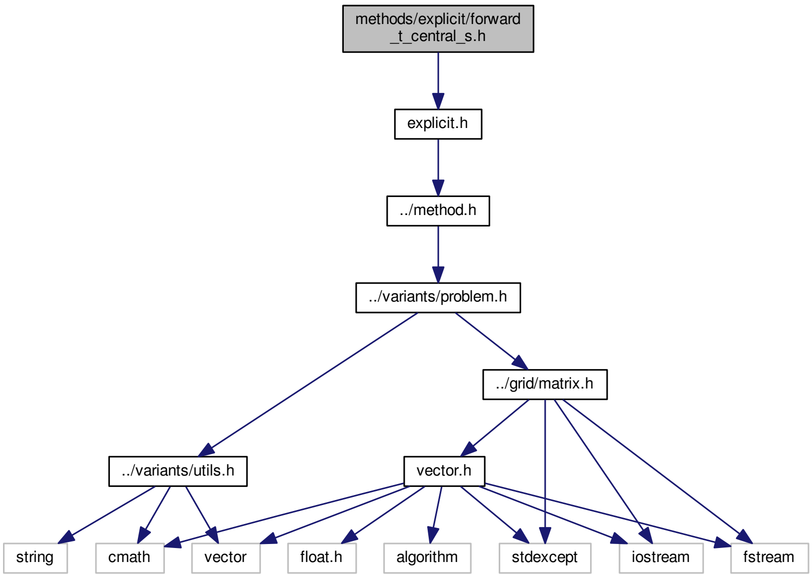
Include dependency graph for forward_t_central_s.cpp:



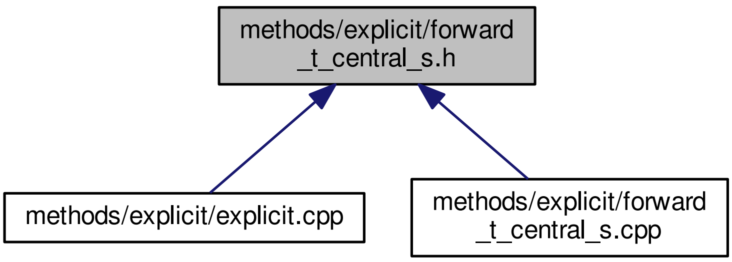
5.15 methods/explicit/forward_t_central_s.h File Reference

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

Include dependency graph for forward_t_central_s.h:



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



Classes

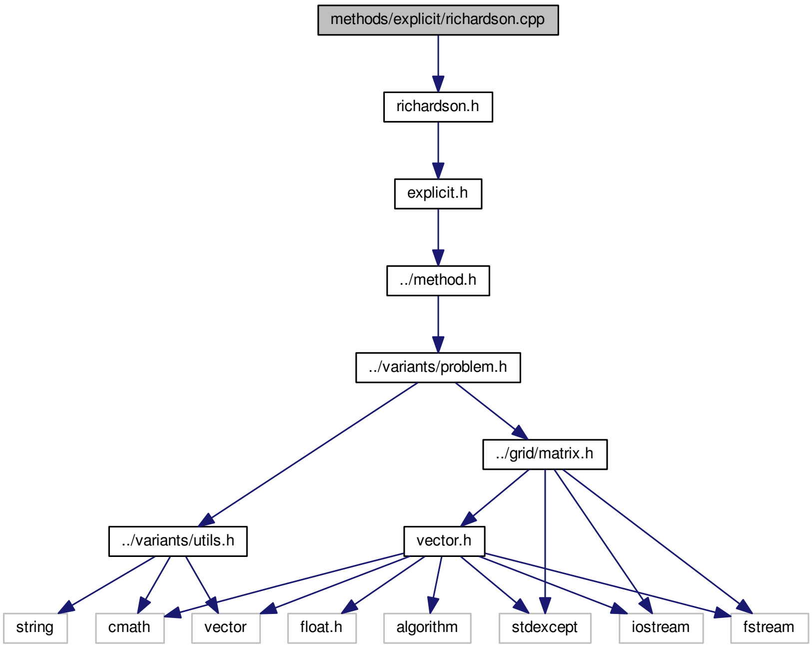
- class FTCS

  A FTCS method class that contains an iteration builder.

5.16 methods/explicit/richardson.cpp File Reference

`#include "richardson.h"`

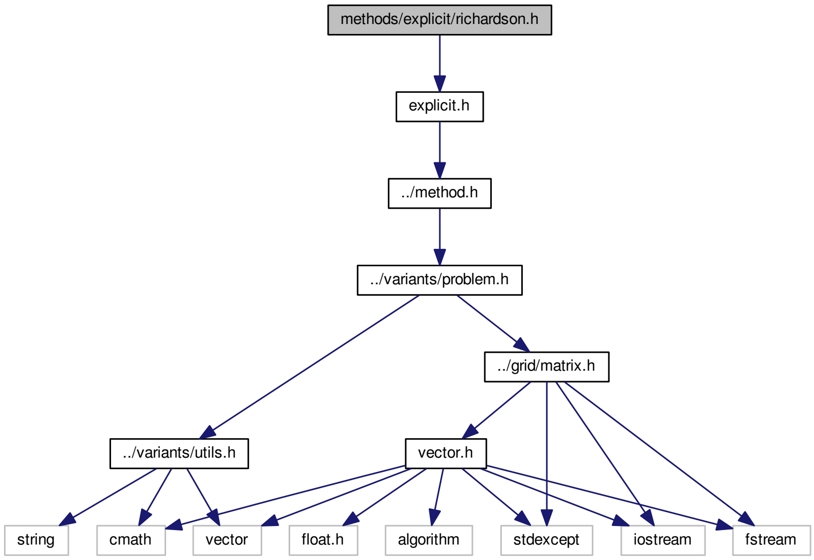
Include dependency graph for richardson.cpp:



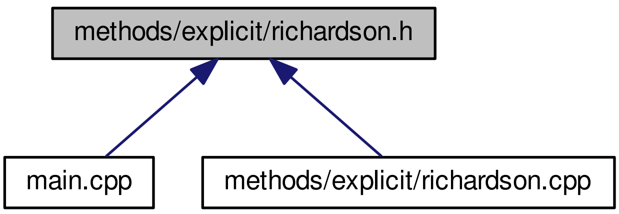
5.17 methods/explicit/richardson.h File Reference

`#include "explicit.h"`

Include dependency graph for richardson.h:



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



Classes

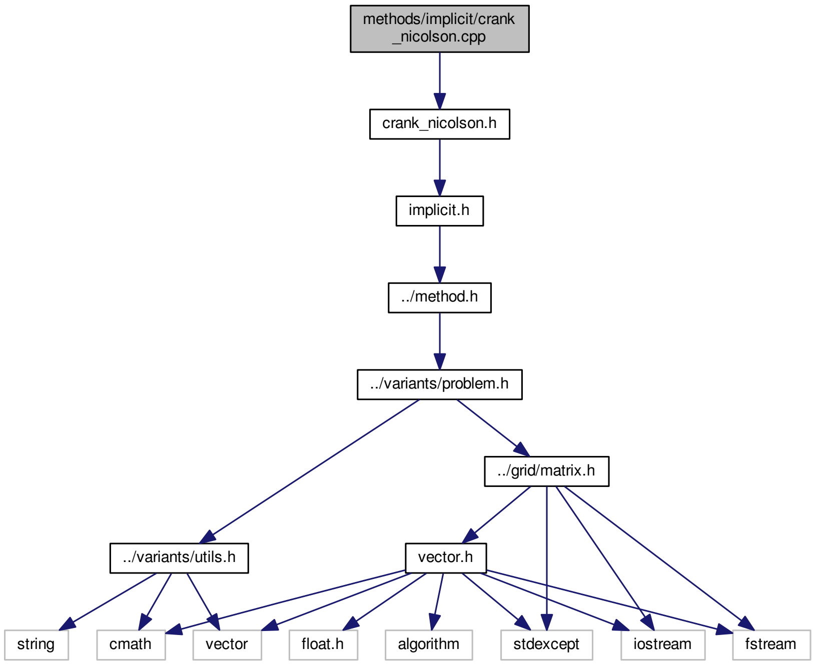
- class Richardson

  A Richardson method class that contains an iteration builder.

5.18 methods/implicit/crank_nicolson.cpp File Reference

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

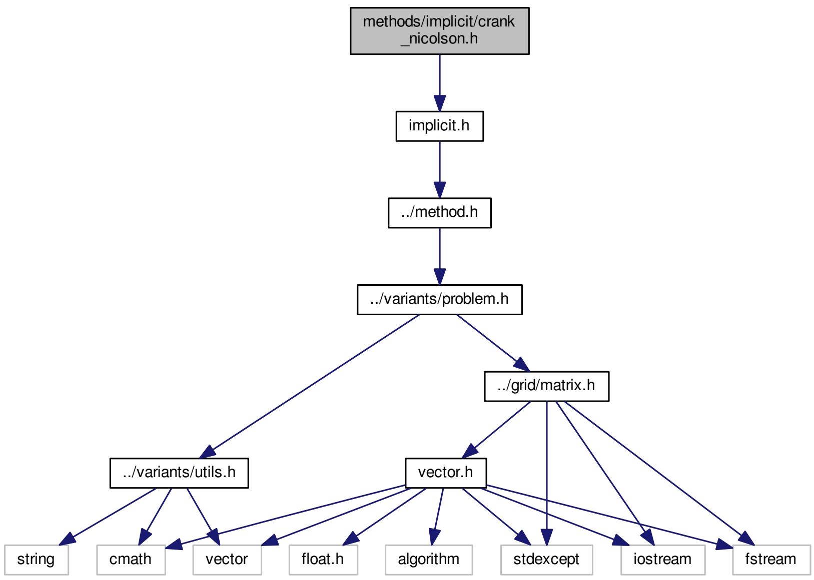
Include dependency graph for crank_nicolson.cpp:



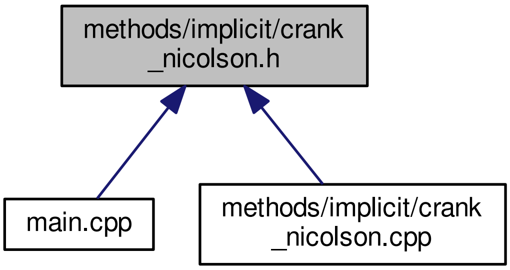
5.19 methods/implicit/crank_nicolson.h File Reference

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

Include dependency graph for crank_nicolson.h:



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



Classes

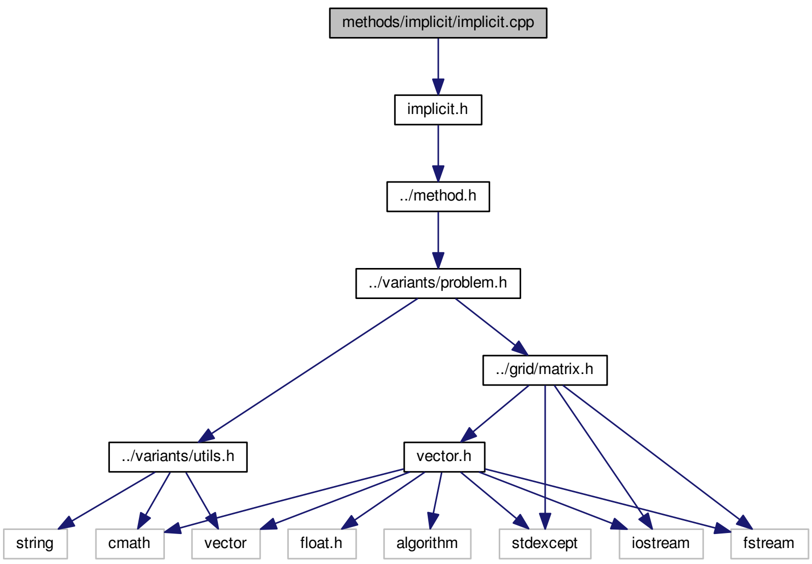
- class CrankNicolson

  A CrankNicolson method class that contains a r vector builder.

5.20 methods/implicit/implicit.cpp File Reference

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

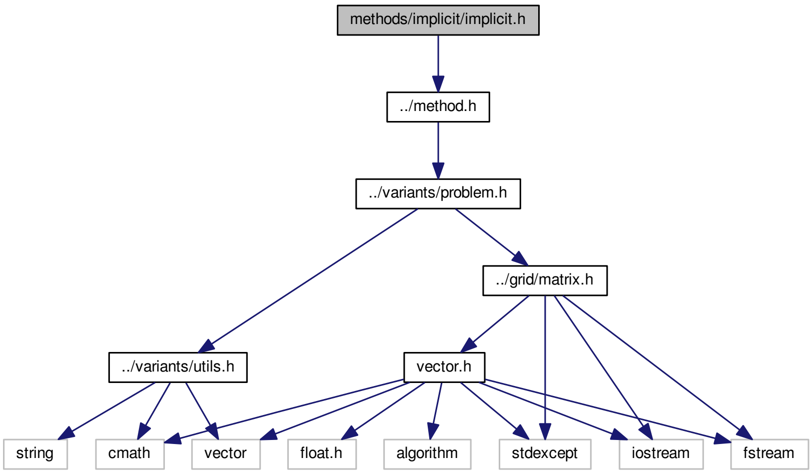
Include dependency graph for implicit.cpp:



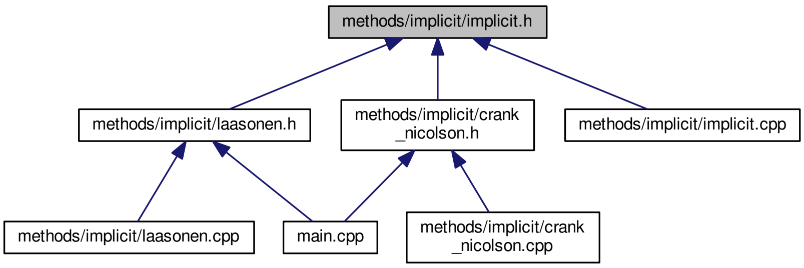
5.21 methods/implicit/implicit.h File Reference

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

Include dependency graph for implicit.h:



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



Classes

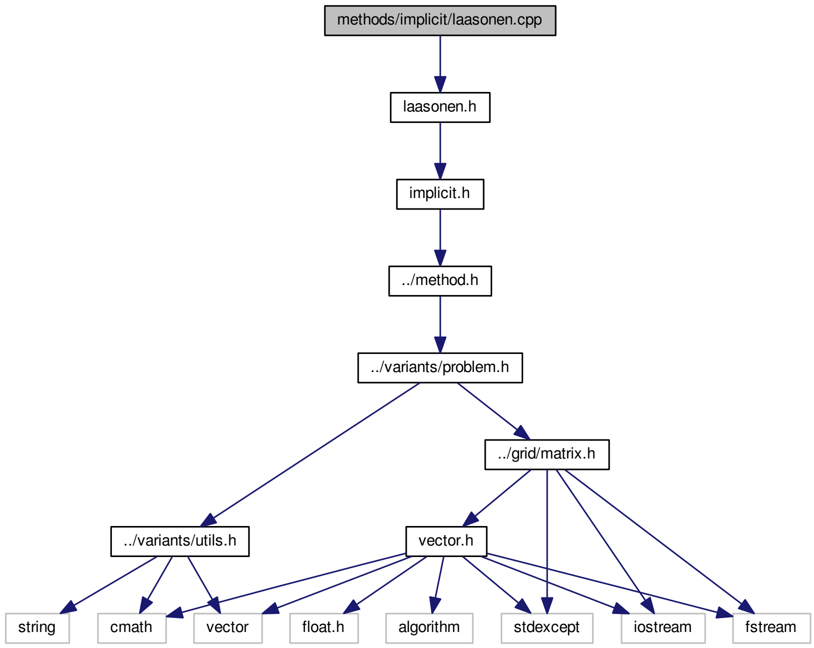
- class Implicit

    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"
```

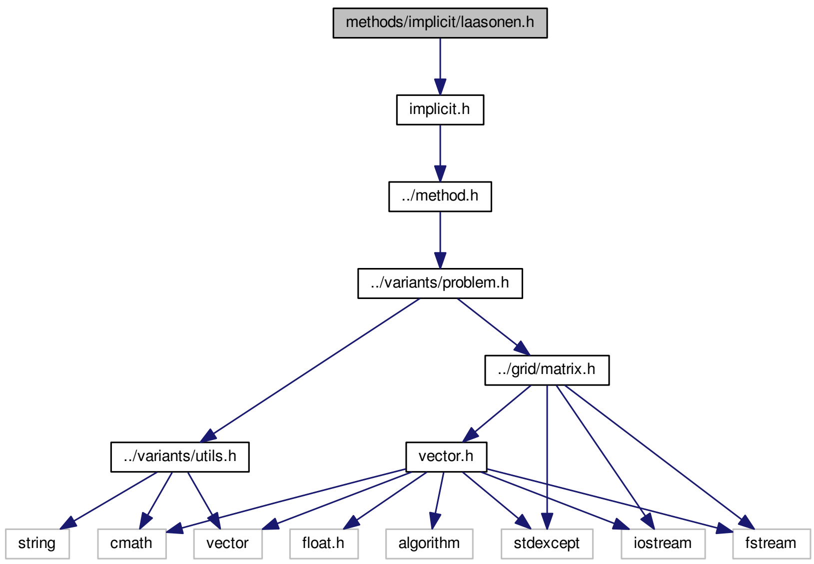
Include dependency graph for laasonen.cpp:



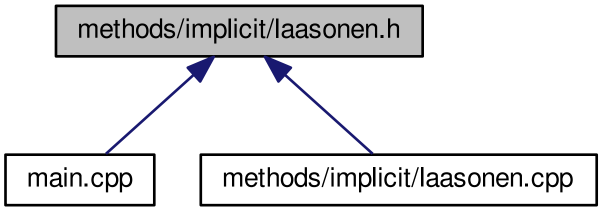
5.23 methods/implicit/laasonen.h File Reference

`#include "implicit.h"`

Include dependency graph for laasonen.h:



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



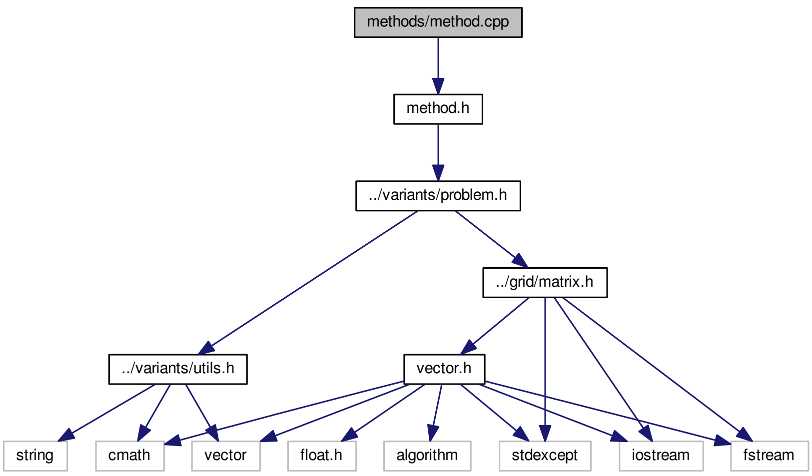
Classes

- class Laasonen

    A Laasonen method class that contains a r vector builder.

5.24 methods/method.cpp File Reference

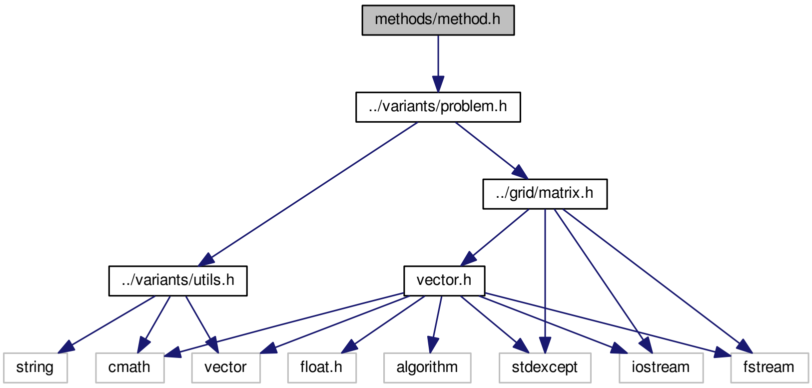
`#include "method.h"`
Include dependency graph for method.cpp:



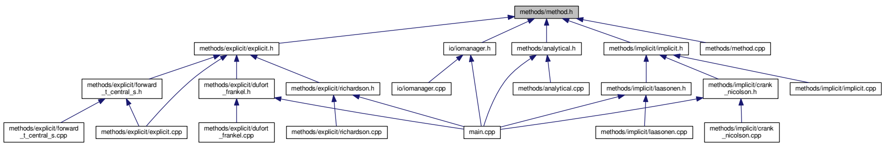
5.25 methods/method.h File Reference

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

Include dependency graph for method.h:



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



Classes

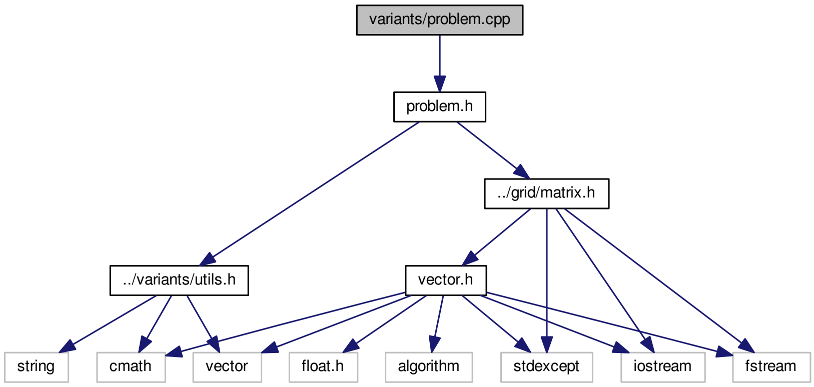
- class Method

  A Method class to structure information used to solve the problem.

5.26 variants/problem.cpp File Reference

`#include "problem.h"`

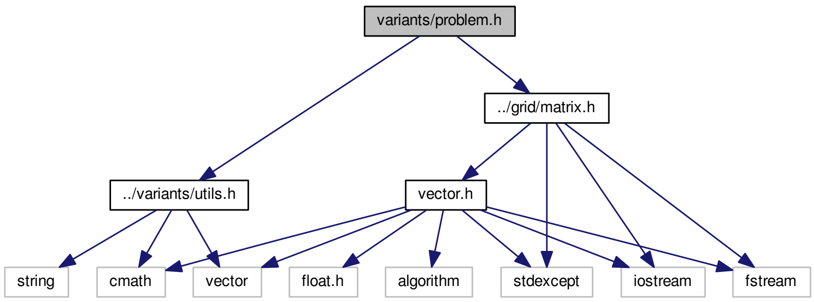
Include dependency graph for problem.cpp:



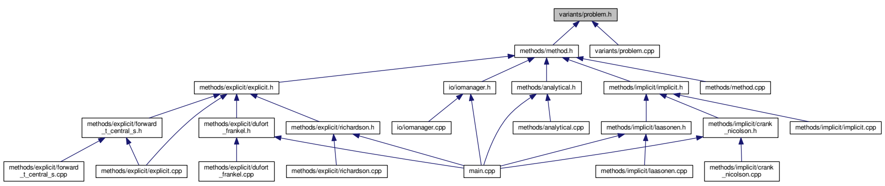
5.27 variants/problem.h File Reference

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

Include dependency graph for problem.h:



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



Classes

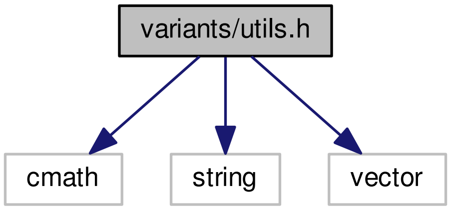
- class Problem

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

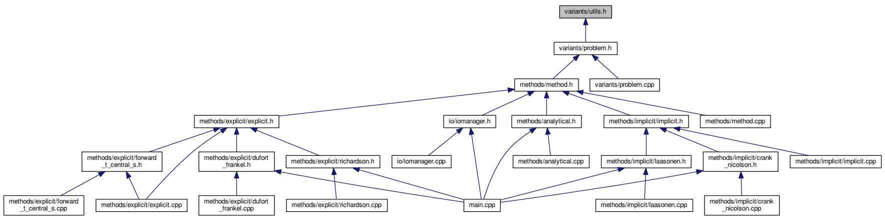
5.28 variants/utils.h File Reference

```
#include <cmath>
#include <string>
#include <vector>
```

Include dependency graph for utils.h:



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



Variables

- const double DELTA_T = 0.01

  Macro double.
- const double DELTA_X = 0.05

  Macro double.
- const std::vector< double > DELTA_T_LASSONEN = {0.01, 0.025, 0.05, 0.1}

  Macro double.
- const double DIFUSIVITY = 0.1

  Macro double.
- const double THICKNESS = 1.0

  Macro double.
- const double TIMELIMIT = 0.5

  Macro double.
- const double SURFACE_TEMPERATURE = 300.0

  Macro double.
- const double INITIAL_TEMPERATURE = 100.0

  Macro double.

- 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_LASSONEN = {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