Study of multiple Heat Diffusion schemes through a C++ implementation

Yoann Masson
Student in Software Engineering at Cranfield University

5th of december, 2017

Abstract

This report is about studying 4 schemes to implement a numerical solution of the heat diffusion equation in a 1D material through time. The equation is given by the following formula:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{1}$$

During this paper we will see that implicit methods are more accurate and more time consumming than explicit methods. The heat diffusion is one of the "solved" phenomena, meaning that we know how to model it and that is how we are able to determine the accuracy of a method.
Schemes relies on computing steps after steps that relie on each other making the time steps size a big matter in this subject. The more steps there is, the more accurate is the method, in condition that the method is stable. This matter will be treated by comparing the same schemes with differents time steps' size.

Contents

Introduction

The simulation of physics phenomena is one of the many uses of modern computer. Thoses simulations are based on the exact formula or on approximations based on observations of the phenomenas' behaviour.
Because the phenomena often depends on many parameters, the analytic equation is not often known. So that, the only way the model a phenomena is to try to approch it with differents schemes based on observations of the initial conditions. However, simulations has very different accuracy and time computing meaning that every phenomena have to be treated separatly.
In this report we will deal with the heat diffusion problem with the following constraints: a one foot long wall at an initial temperature of 100°F is being subject to a rise of its surface temperature to 300°F. The diffusivity D of the material is 0.1ft^2/hr. We will study the rise of the temperature of the material with time t = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5.
We will see four different schemes, each of them is providing a numerical solution. Those schemes are split into two categories:

- explicit schemes, schemes where there is only one unknown in the equation making it eazier to implement but less acurate
  - DuFort & Frankel Method
  - Richardson Method
- implicit schemes, schemes where there are mutiple unkowns that needs to be solved through a linear system, which is harder and more time consuming to solve, but are far more accurate.
  - Laasonen Method
  - Crank & Nicolson Method

The four methods will give four different solutions that we will compare with the known analytical solution. Firstly the discussion will be oriented toward the behaviour of the errors and secondly we will discuss about the C++ solution that I provided.

Chapter 1

The analytical solution

1.1 The expected behaviour

By chance, the heat diffusion equation has been solved and we know the temperature at the given time t and space x, with the given formula:

$$T = T_{sur} + 2(T_{in} - T_{sur}) \sum_{1}^{\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \tag{1.1}$$

If we plot the result for the wanted times, which are from time = 0 to time = 0.5 with a 0.1 time step we get the following plot:

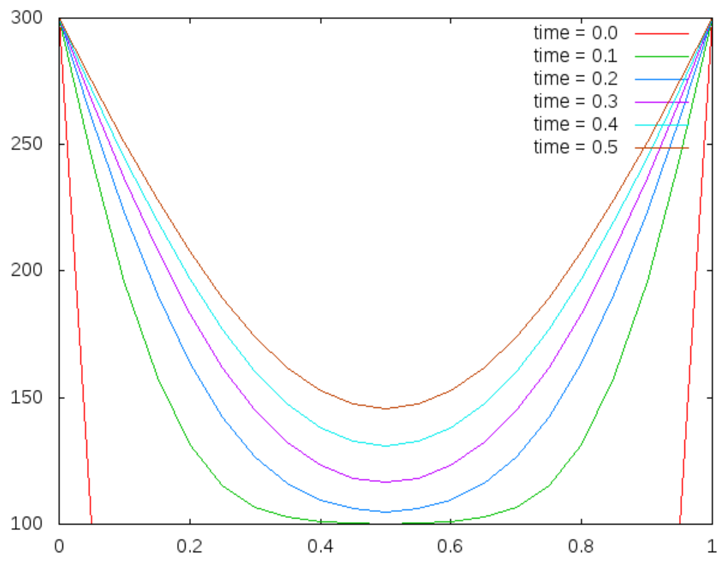


Figure 1.1: Analytical solution

On the horizontale axis is the space and on the vertical is the temperature. Each line represents the temperature of the wall at a given time. As expected as the time flows the temperature is rising from the surface to the center. This is usefull to understand the heat diffusion behaviour. We will later use the data, used to make this plot, to compare with the other solutions. This plot has been computed with a m value of 50, which we will discuss in the next section.

1.2 Study of m

$$\sum_{1}^{\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \tag{1.2}$$

When considering the analytical equation of the heat diffusion, we can see that there is a sum to the infinty involving the term m, which is of course not computable since we do not have an infinit time/space ressources to compute. In computer science, we can consider stopping the computation of the sum if the steps are becoming smaller and smaller, to the point where it is not use anymore to continue computing steps of the sum.

In this case, we are dealing with numbers in a range of 100 to 300, so I considered that an error less than 10^{-4} is acceptable. To know when the m value is to be stopped I printed out the temperature for the same time and space with a different m value. Time is t = 0.01,0.25,0.5 and space is x = 0.5.

m value	temperature at t = 0.5	temperature at t = 0.25	temperature at t = 0.01
1	144.5380	101.0325	47.8530
5	145.5377	110.1387	85.7275
10	145.5377	110.1389	95.4361
20	145.5377	110.1389	100.1123
49	145.5377	110.1389	100.0000
50	145.5377	110.1389	100.0000
100	145.5377	110.1389	100.0000

Table 1.1: Temperature by m value

The outcome of those results is that in this example, we don't need such a high value for m, we can see that values stop varying after $m > 49$. A too small m value is not accurate, at time $t = 0.01$ the temperature is equal to 47°F when it is supposed to be at least 100°F. If we want an error less than 10^{-4}, $m = 50$ is enough. Thanks to that, we know that we don't have to waste computer ressources and time on computing the analytical solution with a higher m value.

Chapter 2

Laasonen method, time step size and computation time

2.1 Theory

Laasonen method The Laasonen method is a way of solving the heat diffusion equation by computing a result step by step. The Laasonen method is part of the implicit scheme meaning that each time step not only relies on the previous one but also one the current one. When computing the result, the previous time step has already been computed but the current one will need to be solved through a linear system. Let's see the equation to get a clearer idea:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{2.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{\Delta x^2} \tag{2.2}$$

$$f\binom{n}{i} = -Cf\binom{n+1}{i+1} + (1+2C)f\binom{n+1}{i} - Cf\binom{n+1}{i-1} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{2.3}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \begin{pmatrix} f\binom{n}{0} \\ f\binom{n}{1} \\ ... \\ ... \\ f\binom{n}{k} \end{pmatrix} \tag{2.4}$$

With the above equations, we can see that we need to solve a linear system for each time step. Linear system can be really time and ressources consum-

ming for a computer. That's why we need a good algorithm to resolve this linear system, if we take a look at the left matrix we can see that it is a diagonal matrix.
A linear system with a diagonal matrix can be solved with Thomas' algorithm, which is a really good linear system solving algorithm that works with diagonal matrix, it's time complexity is of: *O(2n)* with n being the size of the matrix, so the number of space step. L=1 Δt=0.05 so we have 21 space steps. Meaning that a matrix twice larger will "only" take four time the requiered time. (A bit more explanation about Thomas' algorithm in the Appendix).

Since the method relies on computing different time with a constant time step, a good interrogation would be: what is a good time step ? In theory, considering a stable method we can expect a more accurate solution the smaller the time step is.

2.2 Pratice

Having smaller time steps allows us to have a better precision on each row, but if we decrease the time step we increase the number of linear system we need to solve. So I ran 4 simulations with various value of Δt, to compare the results. Let's see the norms of the errors matrices, the errors matrices is the Laasonen resulting matrix minus the analytical matrix:

Δt	Number of time step	One norm	Two norm	uniform norm	computing time
0.1	5	50.26	67.01	154.56	0.14ms
0.05	10	58.62	60.62	102.73	0.6ms
0.025	20	67.90	52.13	64.04	0.42ms
0.01	50	77.01	39.68	36.26	2.2ms

Table 2.1: Error norms according to Δt

Increasing the number of rows in the matrix by decreasing the time step size affects the one and two norms. Because the one norm is the highest error among the sums of error in each column and the two norm is just the sum of all errors. So more rows equals to possibly a worst one and two norms

one norm The one norm is about errors in one column, so for Δt = 0.1, we can say that on average the "worst" column has an error of 50.26/5=10.05.

Considering the fact that we are dealing with numbers in range of 100-300, this is almost a 10% error on average by row. Compared to the Δt = 0.01 that has an average of 77.01/50 = 1.5 error, that's about 1%error on average by row.
According to the one norm, decreazing the size of time steps is a good idea.

two norm The two norm is just the square root of the sum of the square of errors, so increasing the numbers of rows in the matrix should increase the two norm, which is not the case. Between Δt = 0.1 and Δt = 0.05, we double the matrix size but the sum of errors is even smaller meaning that we have a lot less errors overall.

uniform norm The uniform norm is the highest number among the sum of rows. Increasing the number of rows does not affect the behaviour of this number, so if this decreases with a smaller time step size, it means that the method is more and more accurate. By comparing the worst uniform norm 154.56 and the best 36.26, we can say that the worst row with a Δt = 0.1 is nearly six times less accurate than the worst row with a Δt = 0.01.

On a stable method such as Laasonen we have clearly seen that having a smaller Δt is giving the method a better accuracy, going from 10% to nearly 1% error with the first norm. In an ideal case we should find the best deal between time computing and errors, not to forget that If we want to be very accurate we would hurt ourself against errors implied by the numerical 64-bit system.

Chapter 3

Methods Result

3.1 Unstable Method

As we said already, thoses numerical schemes are based on computing steps after steps using the principe of decomposing $\frac{\partial T}{\partial t}$ into something that looks like this $\frac{f\binom{n+1}{i}-f\binom{n-1}{i}}{2\Delta t}$. Doing this implies errors because in theory this is correct for the smallest Δt possible . That is why, the more steps we have the more accurate results we have because the Δ in $f(x+\Delta x)$is smaller. This is introducing errors and we must be sure that the errors are not comulating. In order to do this, we study the stability of the methods on the given problem. This means that every method does not work on every problem. Methods can be stable under certains conditions mostly relying on a Δx-Δt ratio. They can also be unconditionally stable/unstable meaning that for any Δx-Δt, the method will (or will not) work at any case.

Richardson case Let's take a look at Richardson method, it consists in derivating centrally the time and space.

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{3.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i}}{\Delta x^2} \tag{3.2}$$

$$f\binom{n+1}{i} = Cf\binom{n}{i+1} - C2f\binom{n}{i} + Cf\binom{n}{i-1})+f\binom{n-1}{i} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{3.3}$$

Now that we have only one unknown we can figure it out. Since Richardson's method is unstable (see Appendix for proof) considering this problem, the results are not coherent at all.

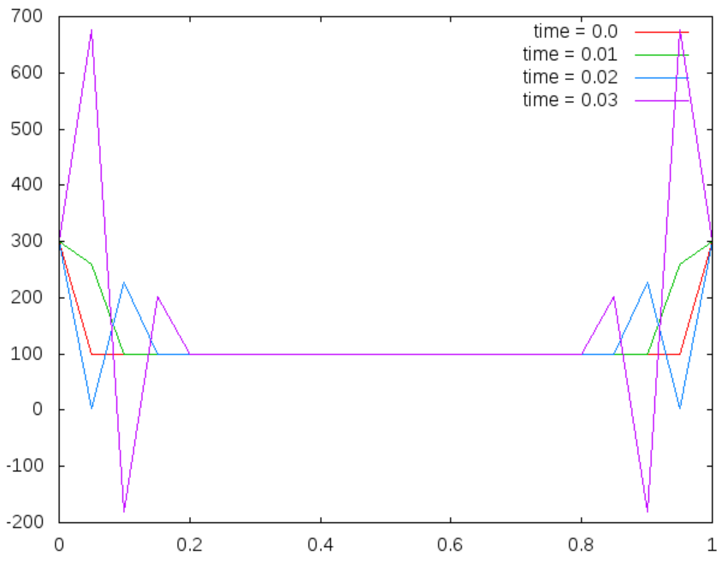


Figure 3.1: Richardson solution

I only plotted the three first steps because it shows that with only three steps we can clearly see that the methods is incoherent. Temperatures are supposed to go in a range of 100°F to 300°F. Which is clearly not the case here with only three time steps computed.

3.2 Stable Methods

Within the conditions defined in the original problem, Dufort-Frankel Laasonen and Crank-Nicholson methods are stable, which means that we do not have to fear for the errors to grow exponentially. We can even expected a better accuracy with time.

Mathematic The derivation of the original equation into the equations used in those schemes can be found in the appendix.

results The best way to compare the results are to compare the norms of their error matrix. For that, we can take a look at table 3.1 that compares the one norm , two norm, uniform norm and the computation time of the three methods.

Method	One norm	Two norm	Uniform norm	Computation time
Crank Nicolson	14.95	14.48	20.77	2.04ms
Laasonen	77.01	39.68	36.26	3.17ms
Dufort-Frankel	85.61	74.47	82.82	0.93ms

Table 3.1: Accuracy of stable methods

Looking at those results, we can clearly see the superiority of the implicit methods (Crank-Nicolson & Laasonen) over the Dufort-Frankel method. The norms in this table represents the sum of errors, so the fewer errors the best a method is, regarding the considered norm. However, the computation time for Dufort-Frankel is far better. Crank-Nicolson seems to be the better deal here: the sum of the error of the least accurate time step is 20°F. So considering 21 space step, that's a 1°F error by number in average, which is acceptable. The second norm of Crank-Nicolson (14.48), that somehow represents the error in the all result matrix, is also the best.
The outcome of this table is that if we have enough time we should consider using Crank-Nicolson since it is the more accurate method but if if we don't have enough computation time we should consider using Dufort Frankel. Dufort Frankel is a bit less accurate but is way faster because its computation only requires solving one equation and not a linear system.

Chapter 4

Implementation

Now that's we saw the theory and the results, let's take a look at the C++ implementation and the design of the application.

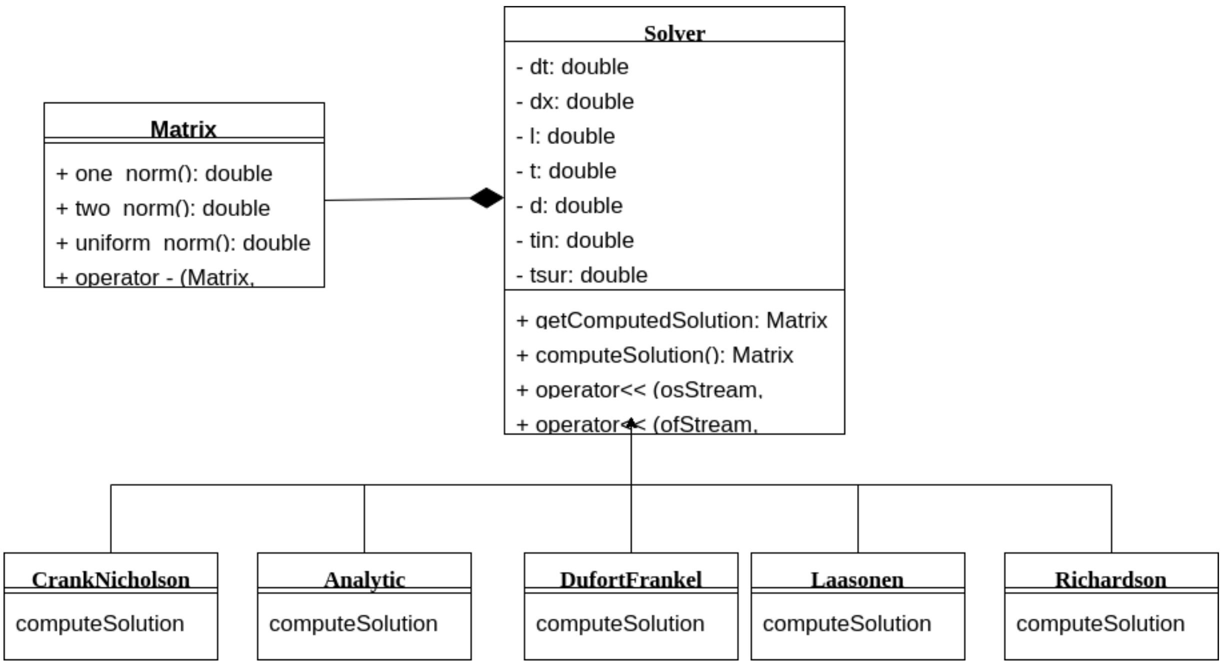


Figure 4.1: Class Diagramm

Class Diagram This is the class diagram of the application implementing the four schemes and the analytic solution. The*Solver* class uses *Matrix* to store the solution of the problem where rows are time steps and columns are space steps. *Solver* declares the virtual method *computeSolution()* so that

classes that inherit from it need to give an implementation of *computeSolution()* if they wish to be instanciated in a *main* programm.
The base class *Solver* class holds all of the informations regarding the initial problem. Since it is attribut and not just hard-coded value in the code, all the child class are bounded to those values. Let's say that we want to see the solution with a surface temperature of 250°F instead of 300°F, we just have to instanciate the object with a different *Tsur* value.

functions To print out the solution both in screen or in a file, I added the redefinition of the " << " operator. On screen, the matrix will print the solution value for time [0.1, 0.2, 0.3, 0.4, 0.5] and will write on a file in a way that would be exploitable by GNUPlot. I added the minus operator in the matrix class to be able to substract matrices in order to get the error matrix.
When instanciated an object of a Solver child class, you have to respect some rules in order not to get exceptions. For example, it is impossible to have a *dx* larger than a *L* it would not make any sense, so the programm would raise an *invalid_argument* exception

Conclusion

We had a problem that requiered a numerical approximation and four methods that would provide differents solutions with differents computation times. As we have seen the first thing to look at, is wether the method is stable or not to be sure to get "correct results". An other important aspect is the accuracy that we want to have, often the more accurate is a method, the more time consuming it will be, so we should look at the complexity of the method to know wether we can compute it in a reasonable time. Because all of the methods are numerical approch there will always be errors either implied by the method or by the limit of the machine, not being able to store the correct value of a number.
In the scope of the valid solutions, there is not a better solution above the others for all cases, it all depends on the computation time and on the accuracy wanted.

Appendix A

Mathematic derivation

Richardson

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i}}{\Delta x^2} \tag{A.2}$$

$$f\begin{pmatrix} n+1 \\ i \end{pmatrix} = Cf\begin{pmatrix} n \\ i+1 \end{pmatrix} - C2f\begin{pmatrix} n \\ i \end{pmatrix} + Cf\begin{pmatrix} n \\ i-1 \end{pmatrix}) + f\begin{pmatrix} n-1 \\ i \end{pmatrix} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{A.3}$$

Dufort-Frankel

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.4}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - (f\binom{n+1}{i} + f\binom{n-1}{i}) + f\binom{n}{i-1}}{\Delta x^2} \tag{A.5}$$

$$f\begin{pmatrix} n+1 \\ i \end{pmatrix} = \frac{f\binom{n-1}{i} + 2C(f\binom{n}{i+1} - f\binom{n-1}{i} + f\binom{n}{i-1}))}{(1+2C)} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{A.6}$$

Laasonen

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.7}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{\Delta x^2} \tag{A.8}$$

$$f\begin{pmatrix} n \\ i \end{pmatrix} = -Cf\begin{pmatrix} n+1 \\ i+1 \end{pmatrix} + (1+2C)f\begin{pmatrix} n+1 \\ i \end{pmatrix} - Cf\begin{pmatrix} n+1 \\ i-1 \end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.9}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \begin{pmatrix} f\binom{n}{0} \\ f\binom{n}{1} \\ ... \\ ... \\ f\binom{n}{k} \end{pmatrix} \tag{A.10}$$

Crank Nicolson

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.11}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i-1}}{2\Delta x^2} + D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{2\Delta x^2} \tag{A.12}$$

$$-Cf\begin{pmatrix} n+1 \\ i+1 \end{pmatrix} + (1+2C)f\begin{pmatrix} n+1 \\ i \end{pmatrix} - Cf\begin{pmatrix} n+1 \\ i-1 \end{pmatrix} = Cf\begin{pmatrix} n \\ i+1 \end{pmatrix} + (1-2C)f\begin{pmatrix} n \\ i \end{pmatrix} + Cf\begin{pmatrix} n \\ i-1 \end{pmatrix} \tag{A.13}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \tag{A.14}$$

$$\begin{pmatrix} Cf\binom{n}{1} + (1-2C)f\binom{n}{0} + C * Tsurface \\ Cf\binom{n}{2} + (1-2C)f\binom{n}{1} + Cf\binom{n}{0} \\ ... \\ ... \\ C * Tsurface + (1-2C)f\binom{n}{k} + Cf\binom{n}{k-1} \end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.15}$$

Appendix B

Thomas' Algorithm

Thomas' algorithm is an algorithm capable of solving linear system that can be stored in a diagonal matrix. If we take an example:

$$\begin{pmatrix} b & c & ... & & & 0 \\ a & b & c & ... & ... & 0 \\ & & & & & \\ & & & & & \\ 0 & & ... & a & b & c \\ 0 & & & ... & a & b \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ ... \\ ... \\ ... \\ x_k \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ ... \\ ... \\ ... \\ d_k \end{pmatrix}$$

then we changed the matrix to look like this, by substracting rows, having $c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}$:

$$\begin{pmatrix} 1 & c' & ... & & & 0 \\ 0 & 1 & c' & ... & ... & 0 \\ & & & & & \\ & & & & & \\ 0 & & ... & 0 & 1 & c \\ 0 & & & ... & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ ... \\ ... \\ ... \\ x_k \end{pmatrix} = \begin{pmatrix} d'_0 \\ d'_1 \\ ... \\ ... \\ ... \\ d'_k \end{pmatrix}$$

At this point, we have a one unknown by equation, so the method is just working backward on the equation with $x_i = d'_i x_{i+1}$

Complexity About the complexity, if there is N equations then the matrix will be of size N. It's N operations to transform the matrix and N operations again to go backward. So the complexity is of *O(2N)*

Appendix C

Documentation

C.1 documentation

Here you can see the latex-pdf documentation generated by Doxygen. With the source code, there is the html documentation that you might find more attractive.

Study of multiple Heat Diffusion schemes

1.0.0

Generated by Doxygen 1.8.14

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

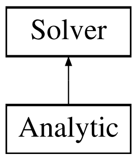
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 Analytic Class Reference

Inheritance diagram for Analytic:



Public Member Functions

- Analytic (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- virtual Matrix computeSolution ()

Additional Inherited Members

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Analytic()

```
Analytic::Analytic (
             double dx,
             double dt,
             double L,
             double T,
             double D,
             double Tsur,
             double Tin )
```

Construcs an analyic object

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider
T	double. Total time of the considerated problem
D	double. Diffusion coefficient of the material
Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.1.2 Member Function Documentation

3.1.2.1 computeSolution()

```
Matrix Analytic::computeSolution ( ) [virtual]
```

Compute the solution and return it. This method is the analytical solution of the heat diffusion equation problem

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

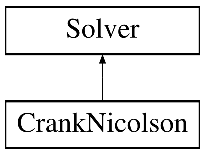
Implements Solver.

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

- Analytic.h
- Analytic.cpp

3.2 CrankNicolson Class Reference

Inheritance diagram for CrankNicolson:



Public Member Functions

- CrankNicolson (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- virtual Matrix computeSolution ()

Additional Inherited Members

3.2.1 Constructor & Destructor Documentation

3.2.1.1 CrankNicolson()

```
CrankNicolson::CrankNicolson (
             double dx,
             double dt,
             double L,
             double T,
             double D,
             double Tsur,
             double Tin )
```

Construcs a solver of the problem using Crank-Nicolson method

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider

Parameters

T	double. Total time of the considerated problem
D	double. Diffusion coefficient of the material
Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.2.2 Member Function Documentation

3.2.2.1 computeSolution()

```
Matrix CrankNicolson::computeSolution ( )  [virtual]
```

Compute the solution and return it. This method is the Crank-Nicolson method applied to the heat diffusion equation problem

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

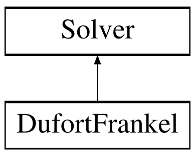
Implements Solver.

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

- CrankNicolson.h
- CrankNicolson.cpp

3.3 DufortFrankel Class Reference

Inheritance diagram for DufortFrankel:



Public Member Functions

- DufortFrankel (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- virtual Matrix computeSolution ()

Additional Inherited Members

3.3.1 Constructor & Destructor Documentation

3.3.1.1 DufortFrankel()

```
DufortFrankel::DufortFrankel (
            double dx,
            double dt,
            double L,
            double T,
            double D,
            double Tsur,
            double Tin )
```

Construcs a solver of the problem using DuFort-Frankel method

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider
T	double. Total time of the considerated problem
D	double. Diffusion coefficient of the material
Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.3.2 Member Function Documentation

3.3.2.1 computeSolution()

```
Matrix DufortFrankel::computeSolution ( ) [virtual]
```

Compute the solution and return it. This method is the Dufort-Frankel method applied to the heat diffusion equation problem

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

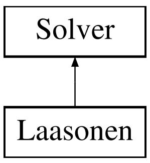
Implements Solver.

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

- DufortFrankel.h
- DufortFrankel.cpp

3.4 Laasonen Class Reference

Inheritance diagram for Laasonen:



Public Member Functions

- Laasonen (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- virtual Matrix computeSolution ()

Additional Inherited Members

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Laasonen()

```
Laasonen::Laasonen (
             double dx,
             double dt,
             double L,
             double T,
             double D,
             double Tsur,
             double Tin )
```

Construcs a solver of the problem using Laasonen method

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider
T	double. Total time of the considered problem
D	double. Diffusion coefficient of the material
Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.4.2 Member Function Documentation

3.4.2.1 computeSolution()

```
Matrix Laasonen::computeSolution ( )  [virtual]
```

Compute the solution and return it. This method is the Laasonen method applied to the heat diffusion equation problem

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

Implements Solver.

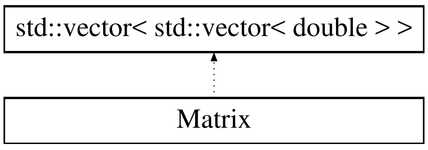
The documentation for this class was generated from the following files:

- Laasonen.h
- Laasonen.cpp

3.5 Matrix Class Reference

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

Inheritance diagram for Matrix:



Public Member Functions

- Matrix ()
- Matrix (int Nrows, int Ncols)
- Matrix (const Matrix &m)
- int getNrows () const
- int getNcols () const
- Matrix & operator= (const Matrix &m)
- bool operator== (const Matrix &m) const
- double one_norm () const
- double two_norm () const
- double uniform_norm () const
- Matrix operator∗ (const Matrix &a) const
- Matrix operator- (const Matrix &a) const
- Vector operator∗ (const Vector &v) const
- Matrix transpose () const

Friends

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

3.5.1 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

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Matrix() [1/3]

```
Matrix::Matrix ( )
```

Default constructor. Intialize an empty Matrix object

See also

Matrix(int Nrows, int Ncols)
Matrix(const Matrix& m)

3.5.2.2 Matrix() [2/3]

```
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

3.5.2.3 Matrix() [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

3.5.3 Member Function Documentation

3.5.3.1 getNcols()

```
int Matrix::getNcols ( ) const
```

Normal public get method. get the number of columns

See also

int getNrows()const

Returns

int. number of columns in matrix

3.5.3.2 getNrows()

```
int Matrix::getNrows ( ) const
```

Normal public get method. get the number of rows

See also

int getNcols()const

Returns

int. number of rows in matrix

3.5.3.3 one_norm()

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

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

See also

two_norm()const
uniform_norm()const

Returns

double. matrix L1 norm

3.5.3.4 operator*() [1/2]

```
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

3.5.3.5 operator*() [2/2]

```
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

3.5.3.6 operator-()

```
Matrix Matrix::operator- (
            const Matrix & a ) const
```

Overloaded -operator that returns a Matrix. It Performs matrix by matrix substraction.

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/rows in first matrix do not match number of columns or/and rows in second matrix

Returns

Matrix. matrix-matrix product

Parameters

a	Matrix. matrix to multiply by

3.5.3.7 operator=()

```
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

3.5.3.8 operator==()

```
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

3.5.3.9 transpose()

```
Matrix Matrix::transpose ( ) const
```

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

Returns

Matrix. matrix transpose

3.5.3.10 two_norm()

```
double Matrix::two_norm ( ) const
```

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

See also

one_norm()const
uniform_norm()const

Returns

double. matrix L2 norm

3.5.3.11 uniform_norm()

```
double Matrix::uniform_norm ( ) const
```

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

See also

one_norm()const
two_norm()const

Returns

double. matrix L_max norm

3.5.4 Friends And Related Function Documentation

3.5.4.1 operator<< [1/2]

```
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

3.5.4.2 operator<< [2/2]

```
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

3.5.4.3 operator>> [1/2]

```
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

3.5.4.4 operator>> [2/2]

```
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

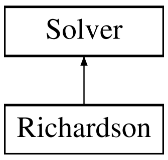
ifs	Input file stream with opened matrix file
m	Matrix to write into

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

- matrix.h
- matrix.cpp

3.6 Richardson Class Reference

Inheritance diagram for Richardson:



Public Member Functions

- Richardson (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- virtual Matrix computeSolution ()

Additional Inherited Members

3.6.1 Constructor & Destructor Documentation

3.6.1.1 Richardson()

```
Richardson::Richardson (
             double dx,
             double dt,
             double L,
             double T,
             double D,
             double Tsur,
             double Tin )
```

Construcs a solver of the problem using Richardson method

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider
T	double. Total time of the considerated problem
D	double. Diffusion coefficient of the material
Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.6.2 Member Function Documentation

3.6.2.1 computeSolution()

```
Matrix Richardson::computeSolution ( )  [virtual]
```

Compute the solution and return it. This method is the Richardson method applied to the heat diffusion equation problem

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

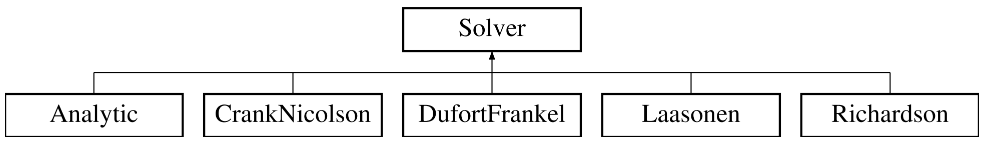
Implements Solver.

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

- Richardson.h
- Richardson.cpp

3.7 Solver Class Reference

Inheritance diagram for Solver:



Public Member Functions

- Solver (double dx, double dt, double L, double T, double D, double Tsur, double Tin)
- Matrix getComputedSolution ()
- double getDT ()
- double getDX ()
- double getL ()
- double getT ()
- double getD ()
- double getTsur ()
- double getTin ()
- virtual Matrix computeSolution ()=0
- virtual ~Solver ()

Protected Attributes

- Matrix **computedSolution**
- double **dx**
- double **dt**
- double **L**
- double **T**
- double **D**
- double **Tsur**
- double **Tin**

Friends

- std::ostream & operator<< (std::ostream &os, Solver &m)
- std::ofstream & operator<< (std::ofstream &ifs, Solver &m)

3.7.1 Constructor & Destructor Documentation

3.7.1.1 Solver()

```
Solver::Solver (
            double dx,
            double dt,
            double L,
            double T,
            double D,
            double Tsur,
            double Tin )
```

Construcs a solver of the problem, can not be instanciated as an object since it is a virtual base class

Exceptions

invalid_argument	("dx should be positive")
invalid_argument	("dt should be positive")
invalid_argument	("L should be positive")
invalid_argument	("T should be positive")
invalid_argument	("L should be equal or larger than dx")
invalid_argument	("T should be equal or larger than dt")

Parameters

dx	double. distance between two space steps
dt	double. time between two time steps
L	double. width of the 1D material to consider
T	double. Total time of the considerated problem
D	double. Diffusion coefficient of the material

Parameters

Tsur	double. The temperature that will be applied on the boundaries of the material
Tin	double. The initial temperature of the material

3.7.1.2 ∼Solver()

```
virtual Solver::∼Solver ( ) [inline], [virtual]
```

Destroys the object

3.7.2 Member Function Documentation

3.7.2.1 computeSolution()

```
virtual Matrix Solver::computeSolution ( ) [pure virtual]
```

Compute the solution and return it. This method must be implemented in the child class if you want it not to be virtual

Returns

Matrix. The computed matrix, can also be accesed through getComputedSolution()

See also

getComputedSolution()

Implemented in CrankNicolson, Analytic, DufortFrankel, Laasonen, and Richardson.

3.7.2.2 getComputedSolution()

```
Matrix Solver::getComputedSolution ( )
```

Return the computed matrix, computeSolution() has to be called first to get the solution matrix.

Returns

Matrix, the computed matrix

3.7.2.3 getD()

```
double Solver::getD ( )
```

get the diffusion coefficient

Returns

- double. the diffusion coefficient considered

3.7.2.4 getDT()

```
double Solver::getDT ( )
```

get the time step

Returns

- double. the time step considered

3.7.2.5 getDX()

```
double Solver::getDX ( )
```

get the space step

Returns

- double. the space step considered

3.7.2.6 getL()

```
double Solver::getL ( )
```

get the width of the material

Returns

- double. the width considered

3.7.2.7 getT()

```
double Solver::getT ( )
```

get the overall time

Returns

- double. the overall time considered

3.7.2.8 getTin()

```
double Solver::getTin ( )
```

get the initial temperature

Returns

- double. the initial temperature considered

3.7.2.9 getTsur()

```
double Solver::getTsur ( )
```

get the temperature at the surface

Returns

- double. the temperature applied on the surface

3.7.3 Friends And Related Function Documentation

3.7.3.1 operator<< [1/2]

```
std::ostream& operator<< (
             std::ostream & os,
             Solver & m )  [friend]
```

redifinition of the << operator to the screen, displays the time every 0.1seconde from 0 to 0.5 seconde.

Returns

the generated stream

3.7.3.2 operator<< [2/2]

```
std::ofstream& operator<< (
            std::ofstream & ifs,
            Solver & m )  [friend]
```

redifinition of the << operator to a file, displays the time every 0.1seconde from 0 to 0.5 seconde. Especially mafe for GNUPlot usage.

Returns

the generated stream

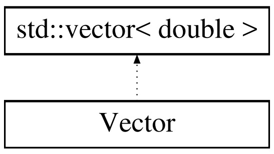
The documentation for this class was generated from the following files:

- Solver.h
- Solver.cpp

3.8 Vector Class Reference

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

Inheritance diagram for Vector:



Public Member Functions

- Vector ()
- Vector (int Num)
- Vector (const Vector &v)
- Vector & operator= (const Vector &v)
- bool operator== (const Vector &v) const
- int getSize () const
- double one_norm () const
- double two_norm () const
- double uniform_norm () const

Friends

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

3.8.1 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

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Vector() [1/3]

```
Vector::Vector ( )
```

Default constructor. Intialize an empty Vector object

See also

Vector(int Num)
Vector(const Vector& v)

3.8.2.2 Vector() [2/3]

```
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

3.8.2.3 Vector() [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)

3.8.3 Member Function Documentation

3.8.3.1 getSize()

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

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

Returns

int. the size of the vector

3.8.3.2 one_norm()

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

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

See also

two_norm()const
uniform_norm()const

Returns

double. vectors L1 norm

3.8.3.3 operator=()

```
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

3.8.3.4 operator==()

```
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

3.8.3.5 two_norm()

```
double Vector::two_norm ( ) const
```

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

See also

one_norm()const
uniform_norm()const

Returns

double. vectors L2 norm

3.8.3.6 uniform_norm()

```
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

3.8.4 Friends And Related Function Documentation

3.8.4.1 operator<< [1/2]

```
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

3.8.4.2 operator<< [2/2]

```
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

3.8.4.3 operator>> [1/2]

```
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

3.8.4.4 operator>> [2/2]

```
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:

- vector.h
- vector.cpp

Index