# Computtional Methods and C++ Assigment

Vincent Richard

02/11/17

**Abstract**

Sample text

# Contents



# List of Figures



# List of Tables

# 1 Introduction

In this assignment we are asked to examine the application of numerical schemes for the solution of partial diferential equations. In order to do so we will consider the following probem.

A wall 1 ft. thick and infinite in other directions has an initial uniform temperature Tin of 100°F. The surface temperatures Tsur at the two sides are suddenly increased and maintained at 300°F. The wall is composed of nickel steel (40% Ni) with a diffusivity of D = 0:1 ft2=hr. Please compute the temperature distribution within the wall as a function of time. The governing equation to be solved is the unsteady one-space dimensional heat conduction equation, which in Cartesian coordinates is:

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

## 1.1 Presentation of the different methods used

### 1.1.1 DuFort-Frankel

The DuFort-Frankel scheme is an explicit scheme unconditionnaly stable the parabolic PDE is:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = D\frac{T_{i+1}^n - (T_i^{n+1} + T_i^{n-1}) + T_{i-1}^n)}{\Delta x^2} \tag{2}$$

This equation leads to an explicit form which is:

$$T_i^{n+1}(1+2r) = T_i^{n-1} + 2r(T_{i+1}^n - T_i^{n-1} + T_{i-1}^n), r = \frac{D\Delta t}{\Delta x^2} \tag{3}$$

### 1.1.2 Richardson

The Richardson scheme is an explixit scheme, unconditionnaly unstable:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = D\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2} \tag{4}$$

This equation leads to an explicit form which is:

$$T_i^{n+1} = 2r(T_{i+1}^n - 2T_i^n + T_{i-1}^n) + T_i^{n-1}, r = \frac{D\Delta t}{\Delta x^2} \tag{5}$$

### 1.1.3 Laasonen

The Laasonen scheme is an implicit scheme, that as for equation:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = D\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2} \tag{6}$$

This equation leads to a form that result in asystem of linear equation:

$$-rT_{i+1}^{n+1} + (1+2r)T_i^{n+1} - rT_{i-1}^{n+1} = T_i^n, r = \frac{D\Delta t}{\Delta x^2} \tag{7}$$

### 1.1.4 Cranck-Nicholson

The Crank-Nicholson scheme is an implicit scheme, that as for equation:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{D}{2}\left(\frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{\Delta x^2} + \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{\Delta x^2}\right) \tag{8}$$

This equation leads to a form that result in asystem of linear equation:

$$-\frac{r}{2}T_{i+1}^{n+1} + (1+r)T_i^{n+1} - \frac{r}{2}T_{i-1}^{n+1} = \frac{r}{2}T_{i+1}^n + (1-r)T_i^n + \frac{r}{2}T_{i-1}^n \tag{9}$$

# 2 Methods and Procedures

To resolve the problem we decided to create a Oriented Object program base on those figures (for readibility reason I divided the structure in three graphics):

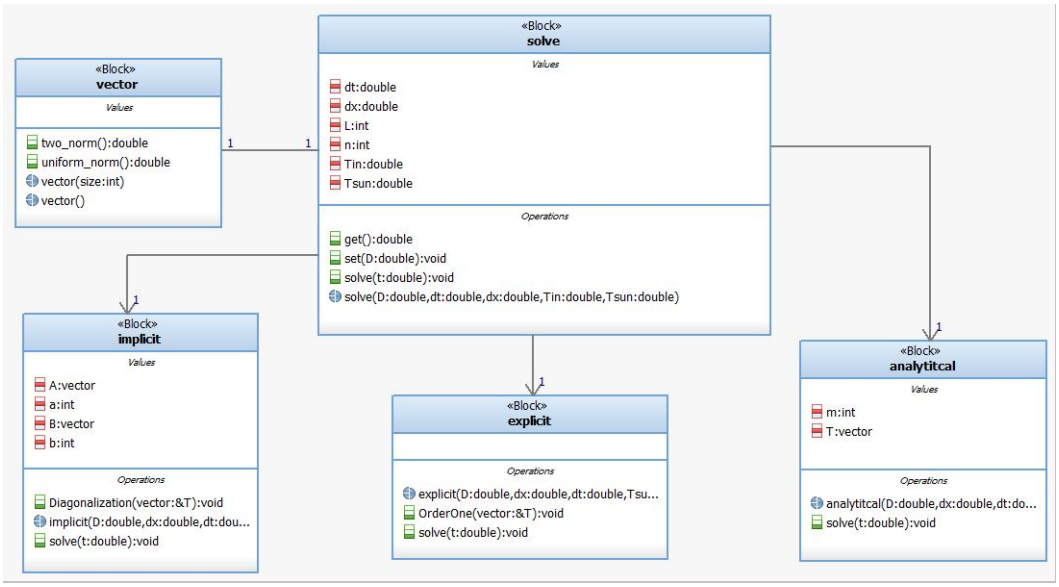


Figure 1: General Architecture

The based class is **Solve**. It has a number of argument:

- D the diffusivity which is in the assigment of $D = 0.1\ ft^2/hr$
- dt the gap in time between n and n+1
- dx the gap in space between i and i+1
- Tin the initial temperature of the wall here $Tin = 100°F$
- Tsun the temperature on the two sides that are maintained at $Tsun = 300°F$
- L the lenght of the wall which is fix to 1 ft. in this exercice

- n the number of possible position x for a fixed t, it calculated with $n = \frac{L}{dx}$

It also has couples of methods:

- Solve(double D, double dx, double dt, double Tsun, double Tin), the only constructor of this class and initialize the different value depending on the user input
- solve(double t) a virtual methods that is created to be use in the derived class it will be the function that solve and print the result of the problem.
- get() / set() methods, basic accessor methods

The **Solve** class also have three derived class **Implicit**, **Explicit**, and **Analytical**. The **Implicit** and **Explicit** class are further details below. We will talk about the **Analytical** class.

The class **Analytical** calculate the analytical solution of the input problem. The class provided two new arguments:

- m it is an integer used to simplify the expression of T since the anatical value of T is :

$$T = T_{sur} + 2(T_{in} - T_{sur}) \sum_{m=1}^{m=\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \quad (10)$$

  So we need to get a smaller m for the upper border of the sum. We choosed to fix it at 10 000, the result obtain was close enought to make this simplification.

- T it is a vector defined as an argument to initialize it in the constructor

The **Analytical** class has also two methods :

- The constructor analytical(...) it is based on the constructor of the solve class, it also initialize the value of m and the vector T
- The solve function that is here defined to print the value of the analytical solution T of the problem with the approximation as explain above. It will print the value of T at each 0.1hr until the argument t is reach

The **Vector** class is associated with the **Solve** class, it is use to defined vector in the derived class. The methods of the vector class:

- one_norm
- two_norm
- uniform_norm

are used to compare the analytical and the different schemes methods.

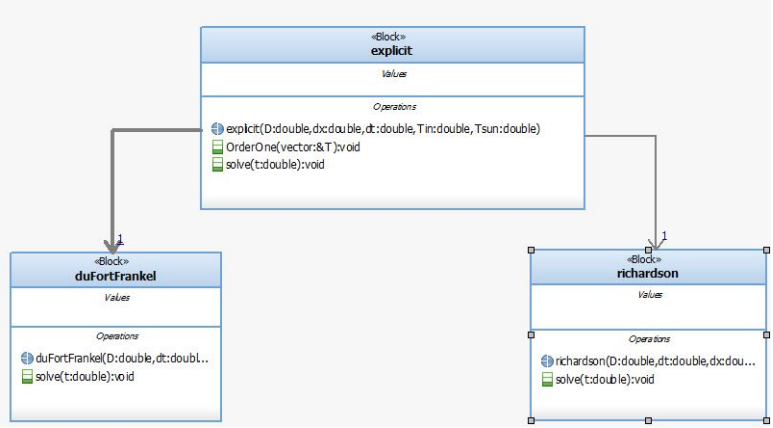


Figure 2: Explicit Architecture

The derived **Explicit** class is also a based class for two other class: the **DuFortFrankel** class and the **Richardson** class. Each of them solving the problem with the scheme of the same name.

The **Explicit** class doesn't define any new argument and as the same methods has the **Solving** class. But it does define an aditionnal method.

The **OrderOne** method was created because in the two explicit schemes, DuFort-Frankel and Richardson, we encounter the issue of not being able to start the scheme at $n = 1$. To find a the value of the temperature T at a time n we need the value of T at n and n-1 (See equation (3) and (5))¿. The problem set the values of T at $n = 0$ but we still need to find T at $n = 1$ to use those schemes. So we first need the method **OrderOne** to find T at $n = 1$ with a forward time central space scheme, and then apply the Richardson and DuFort-Frankel scheme to find the other values of T.

Since this method is useful for both derived class, we decided to declare it in the **Explicit** class.

The two derived classes **DuFortFrankel** and **Richardson** work the same way.

They have the same constructor method based on the one defined in the **Explicit** class and they both defined the method **solve** in a very similar way.

Once the initialisation of T at $n = 0$ and $n = 1$ (with the **OrderOne** method) is done, we use a loop on depending on the scheme the equation (3) or (5) until we reach the value of t wanted by the user.

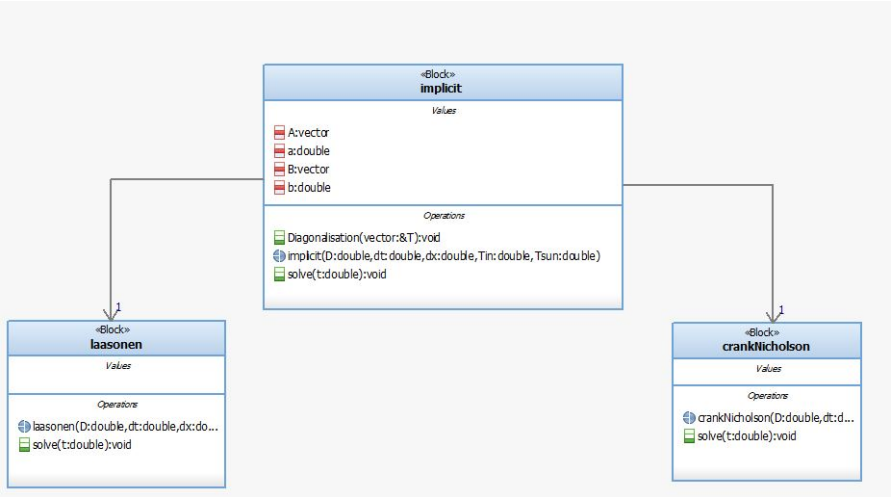


Figure 3: Implicit Architecture

The derived **Implicit** class is also a based class for two other class: the **Laasonen** class and the **Cranck-Nicholson** class. Each of them solving the problem with the scheme of the same name.

The **Implicit** class just like the **Explicit** as a particular method to simplify the solving of the problem in the two derived classes. To understand what the **Diagonalization** method is used for we need to talk about the resolution of the problem for the two implicit schemes.

The two system of equation that we wrote in equation (7) and (8) can be written like this:

$$\begin{pmatrix} b & a & 0 & \cdots & 0 \\ a & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a \\ 0 & \cdots & 0 & a & b \end{pmatrix} = \begin{pmatrix} T_0^{n+1} \\ \vdots \\ T_{imax}^{n+1} \end{pmatrix} R \tag{11}$$

Where a, b are doubles and P a vector, all of their values change depending on which scheme we choose. Since the problem is a tri-diagonal matrix and for each scheme, we have (in the special case of the data given by the assigment) $2\|a\| \leq \|b\|$ so we can use Thomas algorithm to solve those equation.

Thomas Algorithm is an efficient way of solving tridiagonal matrix systems. It is based on LU decompositionin which the matrix system $Mx = r$ is rewritten as $LUx = r$ where L is a lower triangular matrix and U is an upper triangular matrix. The system can be efficiently solved by setting $Ux = p$ and then solving first $Lp = r$ for p and then $Ux = p$ for x. The Thomas algorithm consist of two steps. In step one decomposing the matrix into $M = LU$ and solving $Lp = r$ are accomplished in a single downwards sweep, taking us straight from $Mx = r$ to $Ux = p$. In step two the equation $Ux = p$ is solved for x in an upward sweep.

In the **Diagonalization** we are taking care of the first step of the Thomas algorithm. We will have this kind of equation at the end of the method:

$$\begin{pmatrix} 1 & B_0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & B_{imax-1} \\ 0 & \cdots & 0 & 1 \end{pmatrix} = \begin{pmatrix} T_0^{n+1} \\ \vdots \\ T_{imax}^{n+1} \end{pmatrix} P \tag{12}$$

The values of $B_i$ and P will be stored respectively in the B vector and the A vector defined in the **Implicit** class.

The function **solve** in both **Cranck-Nicholson** and **Laasonen** is a loop that initialize the vector R (with the help of the equation (7) and (9), call the method **Diagonalization(R)** and then calculate the second step of the Thomas algorithm. It will print for each 0.1hr the value of T until the argument t of solve is reached.

# 3 Result