Computational Methods & C++ Assignment

Augustin Reille

2 November 2017

Abstract

A one-space dimensional problem is considered, to examine the application of numerical schemes for the solution of partial differential equations.

Contents

List of Figures

List of Tables

1 Introduction

2 Methods

Several methods are used :

- Dufort-Frankel
- Richardson
- Laasonen simple implicit
- Crank-Nicholson

For all of the used schemes, the initialization of the result matrix is run the same way. First the study grid must be chosen. The initials conditions are:

- T_{int} of 100°F
- T_{sur} of 300°F
- $D = 0.1ft^2/hr$

First it was assumed that $\Delta x = 0.05$ and $\Delta t = 0.01$, so that the result matrix could be initialized. We set:

$$n_{space} = \frac{L}{\Delta x} + 1 \tag{1}$$

$$n_{time} = \frac{T}{\Delta t} \tag{2}$$

n_{space} is the number of iterations over the grid, according to the coordinate x. n_{time} is the number of iterations over the grid, according to the time t.

$$\begin{array}{ccccccc} & i=0 & i=1 & \cdots & n_{space}-1 & n_{space} & \\ \left(\right. & T_{ext} & T_{int} & \cdots & T_{int} & T_{ext} & \left.\right) j=0 \\ & T_{ext} & 0 & \cdots & 0 & T_{ext} & j=1 \\ & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ & T_{ext} & 0 & \cdots & 0 & T_{ext} & n_{time} \end{array}$$

This is how the matrix is initialized, according to initial conditions. For most of the used schemes, we need two previous time steps to calculate the current step. An order one scheme (FTCS) is used to find the second line of the matrix:

$$T_{i,1} = \frac{\alpha}{2}(T_{i+1,0} + T_{i-1,0}) + (1-\alpha)T_{i,0} \tag{3}$$

with

$$\alpha = 2\frac{D\delta t}{\delta x^2}$$

So we have:

$$\begin{array}{c} \begin{array}{ccccc} i=0 & i=1 & \cdots & n_{space}-1 & n_{space} \end{array} & \\ \begin{pmatrix} T_{ext} & T_{int} & \cdots & T_{int} & T_{ext} \\ T_{ext} & T_{i,1} & \cdots & T_{i,1} & T_{ext} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{ext} & 0 & \cdots & 0 & T_{ext} \end{pmatrix} & \begin{array}{c} j=0 \\ j=1 \\ \vdots \\ n_{time} \end{array} \end{array}$$

With this trick our matrix is ready to be used and filled up with every scheme.

2.1 Explicit schemes

For both of the explicit schemes used (Dufort-Frankel and Richardson schemes), the solving is run the same way : with a double for-loop, the matrix is filled up. The only thing that change is the way we advance through time and space.

2.1.1 Dufort-Frankel

Explicitly, for Dufort-Frankel method, we have:

$$T_j^{n+1} = (\frac{1-\alpha}{1+\alpha})T_j^{n-1} + (\frac{\alpha}{1+\alpha})(T_{j+1}^n + T_{j-1}^n) \tag{4}$$

with

$$\alpha = 2\frac{D\delta t}{\delta x^2}$$

2.1.2 Richardson

Explicitly, for Richardson method, we have:

$$T_j^{n+1} = T_j^{n-1} + \alpha(T_{j-1}^n - 2T_j^n + T_{j+1}^n) \tag{5}$$

with

$$\alpha = 2\frac{D\delta t}{\delta x^2}$$

2.2 Implicit schemes

For both implicit schemes, the vector solution is given by a matrix equation. A LU factorization is used once, before a for-loop, to resolve the system for each space step.

2.2.1 Laasonen

We have :

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

Written in a matrical way, the solution at space step $n+1$ is given by :

$$\begin{bmatrix} 1+2C & -C & 0 & \cdots & 0 \\ -C & 1+2C & -C & \ddots & \vdots \\ 0 & -C & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1+2C & -C \\ 0 & \cdots & 0 & -C & 1+2C \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} \end{bmatrix}_{n+1} = \begin{bmatrix} T_1 + CT_0 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} + CT_f \end{bmatrix}_{n} \tag{7}$$

with

$$C = \frac{D\Delta t}{\Delta x^2}$$

The first vector at space n is our boundaries conditions vector, so it is known. In this scheme we must be aware that the right-term vector is actually bigger than the other, of 2 values, which corresponds to the exterior temperatures, which stays the same all the time.
We apply the LU solve algorithm for each space step, at the reduced vector, and add the $n+1$ vector to our result matrix. The new right term vector is the one found with the LU solve algorithm.

2.2.2 Crank-Nicholson

We have :

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

Written in a matrical way, the solution at space step $n+1$ is given by :

$$\begin{bmatrix} 1+C & -C/2 & 0 & \cdots & 0 \\ -C/2 & 1+C & -C/2 & \ddots & \vdots \\ 0 & -C/2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1+C & -C/2 \\ 0 & \cdots & 0 & -C/2 & 1+C \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} \end{bmatrix}_{n+1} \tag{9}$$

$$= \begin{bmatrix} 1-C & C/2 & 0 & \cdots & 0 \\ C/2 & 1-C & C/2 & \ddots & \vdots \\ 0 & C/2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1-C & C/2 \\ 0 & \cdots & 0 & C/2 & 1-C \end{bmatrix} \begin{bmatrix} T_1 + CT_0 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_{ntime} + CT_f \end{bmatrix}_{n}$$

with

$$C = \frac{D\Delta t}{\Delta x^2}$$

The method is the same as Laasonen scheme, except that the left term vector found with the LU solve algorithm is multiplied by the right term matrix before the next space step.

2.3 Analytical solution

The analytical solution of the problem considered will be used to compare our results and to calculate the errors. The result of the analytical solution for a time t and a position x is given by :

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

2.4 Object Oriented Design

The object oriented classes for the C++ code are designed the following way :

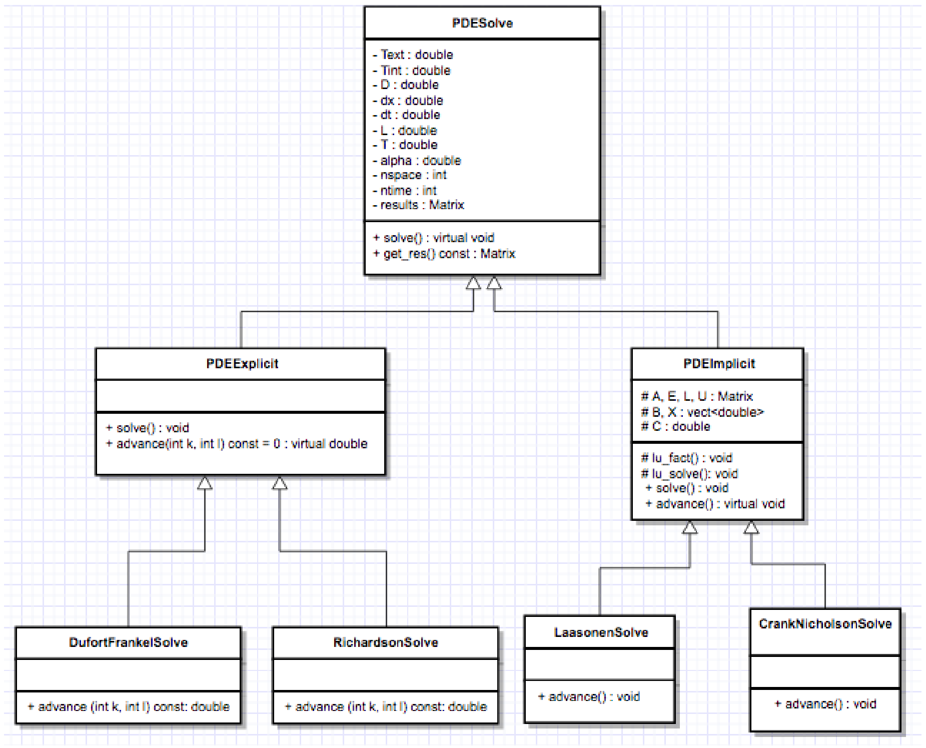

Figure 1: UML Diagram of the C++ object oriented code

There is a main base class called PDESolve, which initializes all the variables, and two functions :

- get_res() : returns the results Matrix
- solve() : virtual function which will be overwritten in subclasses

There is two subclasses of PDESolve : PDEExplicit and PDEImplicit. Each one has a solve() function, and a virtual advance function, which corresponds at the way the space and time steps are incrementded. The PDEImplicit class also initializes all the matrixes and functions needed for the LU factorization and solve. Each subsubclass, which corresponds to each method, has its own particular advance method.

3 Results

3.1 Schemes study

Note that results from Richardson scheme are not displayed on the following plots, due to its unstability [1]: the incorrect values polluted our charts.

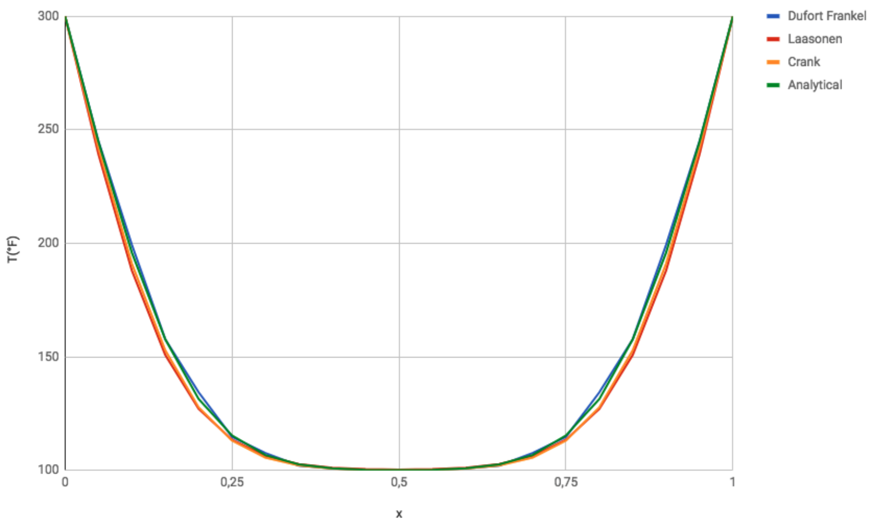

Figure 2: Results with differents schemes at t = 0,1h

We can see that except for the Richardson scheme, the results are quite the same. The errors had been calculated to evaluate the accuracy of each method.

Table 1: Errors for different schemes at t = 0,1h

Errors	Dufort-Frankel	Laasonen	Crank-Nicholson
1-norm	0,56%	1,72%	1,26%

We observe that Dufort-Frankel method is the most accurate, followed by the Crank-Nicholson method then by the Laasonen method. Let see the evolution of the error while time is increased.

Table 2: Errors for different schemes at different times

Errors	Dufort frankel	Laasonen	Crank-Nicholson
t=0,2h	0,37%	1,05%	0,80%
t=0,3h	0,27%	0,77%	0,60%
t=0,4h	0,22%	0,64%	0,49%
t=0,5h	0,18%	0,55%	0,42%

We observe that the error is globally decreasing as time increase. The most accurate method appears to be the Dufort-Frankel method, then the Crank-Nicholson method, then the Laasonen method.

3.2 Laasonen method : effect of the step time

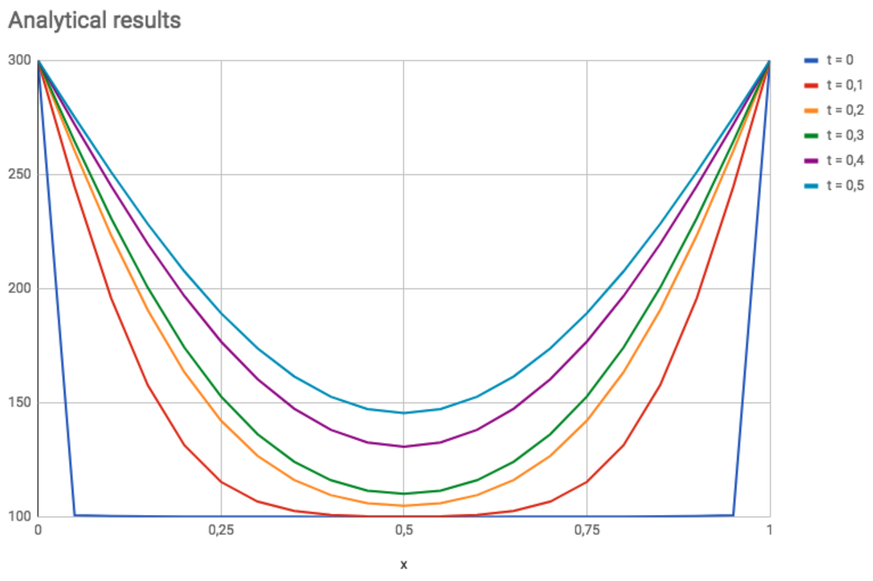

Figure 3: Analytical results at different times

We can see that the shape of the results is more accurate as Δt is little. It may be assumed that increasing Δt gives less accurate results. Let's calculate the error between these values and the analytical values.

Table 3: Laasonen - effect of time step increasing

Errors	t = 0,1h	t = 0,2h	t = 0,3h	t = 0,4h	t = 0,5h
Δt=0,01	1,72%	1,05%	2,83%	0,64%	0,55%
Δt=0,025	4,53%	2,76%	0,81%	1,69%	1,45%
Δt=0,05	9,75%	5,84%	4,95%	3,50%	2,98%
Δt=0,1	53,96%	12,80%	16,30%	7,35%	6,19%

We observe that as time grows, the error decrease. Also, as time step grows, the error is increasing. Previous asumption is correct, according to the error calculation.

4 Discussion

4.1 Analytical solution

First thing that should be considered is that the analytical solution should not be considered as the perfect exact solution, because of several things.
First, the analytical solution is calculated by the computer, so there may be some numerical error.
Secondly, we have to consider the calculation of the sum inside equation (10). Indeed, as m should be infinite, the biggest it is is the best. However, as I choose to calculate all the results for each time step, increasing m also increase run time. I choosed a compromise where values of the analytical solution at $t = 0$ were as correct as possible, with an m not too big, to reduce run time.

Though, it may have been a good idea to calculate the analytical solutions only at studied times : it would have reduced the memory usage, and we could have afforded to increase the value of m. This would have given us better values for our errors calculation.

4.2 Computational study

4.2.1 LU factorization and solving

LU factorization and solving is used in the C++ code. However, Thomas algorithm could have been used for several reasons.
First, our matrix is tridiagonal, which is a condition to use Thomas algorithm.
Secondly, Thomas algorithm has a smaller complexity ($O(n)$) than LU solving ($O(n^3)$), it is also faster.
This way, we would have reduced the total complexity of our code and the computational time.

4.2.2 Vectors

Another way to reduce the complexity of our code and the used memory would have been to use vectors instead of matrixes. Indeed, for the implicit methods, most of our matrixes are filled up with zeros, and every line is the same, only the position of the values change. We could have used only one vector of size 3 instead of a quite big matrix. It is quite all right for this study because we are working with only two dimensions, and with a quite reduced space. By moving in a more complex problem, delays would have appeared.

4.3 Schemes study

4.3.1 Richardson scheme

We saw that Richardson scheme was unstable [1]. Also, when we change the T_i^n term in (5) by the average $\frac{(T_i^{n+1}+T_i^{n-1})}{2}$, we obtain the following :

$$T_i^{n+1} = T_i^{n-1} + \frac{2D\delta t}{\delta x^2}[T_{i+1}^n - (T_i^{n+1} + T_i^{n-1}) + T_{i-1}^n] \tag{11}$$

Which is equivalent as (4), the Dufort-Frankel scheme, which is a stable scheme [2].
So we can see that by replacing the T_i^n term in an unstable scheme by its average in time, we obtain a stable scheme. This is a good thing to know as majority of explicit schemes are unstable. However, they are easy to test, so that we can easily find some way to make them stable if they are not.

4.3.2 Dufort-Frankel versus other schemes

It appears that Dufort-Frankel scheme is the most accurate of the studied schemes. One explanation to those results is that with this method, accurate solutions are obtained if $\Delta t << \Delta x$ [2]. The two other schemes, Laasonen and Crank Nicholson seem to have the same accuracy.
Crank Nicholson scheme needs a low D coefficient for numerical accuracy [3]. That's why we have a quite good accuracy (less than 1%) in our results.

4.4 Laasonnen method : effect of the step time

As we see on Table 3, for a fixed Δt, the error tends to decrease as time grows. However, for a fixed time, we can see that increasing the time step make the error bigger. We can assume that we need a quite small time step to have good results with this scheme. This can be explained by the fact that Laasonen has a good error response, however it has the disadvantage of a poor accuracy [4].

5 Future work

6 Conclusion

7 Aknowledgments

8 References

References

[1] Markus Schmuck - Numerical Methods for PDEs,
`http://www.macs.hw.ac.uk/m̃s713/lecture_9.pdf`.
Last visit date : 18/11/2017

[2] A. Salih - Finite Difference Method for Parabolic PDE,
`https://www.iist.ac.in/sites/default/files/people/parabolic.pdf`.
Last visit date : 18/11/2017

[3] Crank, J. Nicolson, P. Adv Comput Math (1996) 6: 207.
`https://doi.org/10.1007/BF02127704`

[4] D. Britz & Jorg Strutwolf, *Digital Simulation in Electrochemistry*. Springer, 2006

9 Appendices