FYS3150
Project 5

Antoine Hugounet & Ethel Villeneuve

December 2017
University of Oslo
`https://github.com/kryzar/Calypso.git`

Abstract

This project aims to resolve numerically the diffusion equation in both one and two spatial dimensions. We implement explicit, implicit and Crank-Nicolson schemes in one dimension, implicit scheme in two dimensions, investigate their stability properties and compare them. We also study the diffusion process with videos we have made and observe the differences in the convergences to the steady state when we vary the boundary conditions. The results are convincing and reliables, yet the errors of the schemes do not behave as expected. However the schemes, except the explicit scheme in one dimension, are very close to the analytical result, with a relative error inferior to 10^{-3} if we choose our parameters wisely.

Contents

Introduction

This project is another project dealing with finite-difference methods. But this time we gather several methods we have studied earlier (iterative solvers, gaussian elimination) to compute PDEs. The classical formulae for the first derivative of a function (forward formula, backward formula, average of the two previous formulae) however leads to very different mathematical methods : the explicit scheme, the implicit scheme and the Crank-Nicolson scheme.
However their use comes with many constraints regarding stability and mathematical errors. That's why we spend a lot of time discuting about how to choose the parameters for the solvers, how the Von Neumann stability analysis evaluates the theoritical errors and how it does not always match the informatic reality. Yet you will see plots full of colors and simulations that show very well how the diffusion of temperature occurs.

Chapter 1

Theory and algorithms

1.1 One space-dimension equation

We start with

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, \qquad t > 0,\, x \in [0, L] \tag{1}$$

We set the initial conditions to be :

$$\begin{aligned} u(x,0) &= 0, & 0 < x < L \\ u(0,t) &= 0, & t > 0 \\ u(L,t) &= 1, & t > 0 \end{aligned}$$

We scale the equations by choosing $L = 1$ and we will use the more compact notation $u_{xx} = u_t$. We will consider three methods for partial differential equations : the explicit forward Euler algorithm, the implicit backward Euler algorithm and the implicit Crank-Nicolson scheme.

1.1.1 Analytical solution

We introduce $\widetilde{u}(x,t)$ as $u(x,t) = \frac{x}{L} + \widetilde{u}(x,t)$ because the boundary conditions of u ($u(0,t) = 0$ and $u(L,t) = 1$) do not allow us to simplify enough the equations. Considering $\widetilde{u}$, the boundary conditions are

$$\widetilde{u}(0,t) = \widetilde{u}(L,t) = 0$$

Then :

$$u_t = u_{xx} \Longrightarrow \widetilde{u}_t = \widetilde{u}_{xx}$$

Let's solve the equation for $\widetilde{u}$. We assume that the solution takes the form of $\widetilde{u}(x,t) = F(x).G(x)$ by separation of variables.

$$\begin{cases} \widetilde{u}_t = F\dot{G} \\ \widetilde{u}_{xx} = F''G \end{cases} \Longrightarrow \frac{\dot{G}}{G} = \frac{F''}{F} = k \Longrightarrow \begin{cases} F'' - kF = 0 \\ \dot{G} - kG = 0 \end{cases}$$

We begin with the first equation $F'' - kF = 0$. The discriminant of this equation is $\Delta = k^2$. The only valid case $\sqrt{\Delta} = k < 0$, otherwise we end up with $F = 0$. In that case, we have a solution of the form : $F(x) = A\cos\Big(\sqrt{|k|}x\Big) + B\sin\Big(\sqrt{|k|}x\Big)$. Applying the boundary conditions we have $F(0) = A = 0$ and $F(L) = B\sin\Big(\sqrt{|k|}L\Big) = 0$. The latest equation implies $\sin\Big(\sqrt{|k|}L\Big) = 0 \Longrightarrow \sqrt{|k|}L = n\pi \Longleftrightarrow \sqrt{|k|} = n\frac{\pi}{L}$ for any integer n. Then :

$$F_n(x) = B_n \sin n\frac{\pi}{L}x \tag{2}$$

Now, we take the second equation : $\dot{G} - kG = 0$. This has solutions of the form $G_n(t) = C_n e^{-\lambda_n^2 t}$ with $\lambda_n = \frac{n\pi}{L}$. Then :

$$\widetilde{u}(x,t) = \widetilde{u}_n(x,t) = D_n \sin\left(\frac{n\pi x}{L}\right)e^{-\left(\frac{n\pi}{L}\right)^2 t}$$

with $D_n = B_n \times C_n$.
We have a solution $u_n(x,t)$ for the diffusion equation for all n given by $u_n(x,t) = \widetilde{u_n}(x,t) + \frac{x}{L}$. Because of the linearity of the solutions for $\widetilde{u}$, any linear combination of solutions will be a solution and can write $\widetilde{u}(x,t)$ as a Fourier series:

$$\widetilde{u}(x,t) = \sum_{n=1}^{\infty} D_n \sin(\lambda_n x)e^{-\lambda_n^2 t}$$

For $t = 0$, if we define :

$$f(x) := \widetilde{u}(x,0) = \sum_{n=1}^{\infty} D_n \sin(\lambda_n x)$$

we can compute the Fourier coefficients of this new series using the initial conditions and the fact that we have done some complex analysis and that we know very well our formulae :

$$D_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi x}{L}\right)dx$$

The initial condition is $u(x,0) = \widetilde{u}(x,0) + x = 0$ for $0 \leq x < L$. Choosing L=1 this gives us $f(x) = -x$. With an integration by parts we easily end up with :

$$D_n = \frac{2\cos(k\pi)}{k\pi}, \quad L = 1$$

$$u(x,t) = x + \sum_{n=1}^{\infty} \frac{2\cos(k\pi)}{k\pi}\sin(\lambda_n x)e^{-\lambda_n^2 t}, \quad L = 1$$

with $\lambda_n = n\pi$ when $L = 1$.

$$u(x,t) = x + \sum_{n=1}^{\infty} \frac{2\cos(k\pi)}{k\pi}\sin(n\pi x)e^{-(n\pi)^2 t}, \quad L = 1 \tag{3}$$

□

1.1.2 Explicit scheme

By discretizing the first-order and second-order derivative formulae we define :

$$u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$
$$u_{xx} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

Using the original diffusion equation (1) this reads :

$$u_{xx} = u_t$$
$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t}$$
$$\implies u_{i,j+1} = \frac{\Delta t}{\Delta x^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + u_{i,j}$$

defining $\frac{\Delta t}{\Delta x^2} = \alpha$, we get

$$u_{i,j+1} = (1 - 2\alpha)u_{i,j} + \alpha(u_{i-1,j} + u_{i+1,j}) \tag{4}$$

This the explicit (or forward Euler) scheme. We can write this equation in matrix form. Let a vector V_j at a time $t_j = j\Delta t$ be :

$$V_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \dots \\ \dots \\ u_{n,j} \end{bmatrix}$$

Of course the boundary conditions give $u_{0,j} = 0$ and $u_{n+1,j} = 1$. Now rewriting (4), we end up with

$$V_{j+1} = \hat{A}V_j \iff V_{j+1} = \hat{A}^{j+1}V_0$$

with $\hat{A}$ a matrix given by

$$\hat{A} = \begin{bmatrix} 1-2\alpha & \alpha & 0 & \dots & \dots & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \alpha & 1-2\alpha & \alpha \\ 0 & \dots & \dots & 0 & \alpha & 1-2\alpha \end{bmatrix}$$

or by $\hat{A}$ as $\hat{A} = \hat{I} - \alpha\hat{B}$ with

$$\hat{B} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{bmatrix}$$

The latter expression allows us to find the eigenvalues of $\hat{A}$ using $\lambda_i = 1 - \alpha\mu_i$ with μ_i the eigenvalues of $\hat{B}$. This will allow us to find the mathematical criteria of stability for this scheme.

First we give the initializations that will be used for the three algorithms :

Algorithm 1: Inputs and initializations

Input: j_f
▷ the final time

Input: n_t
▷ the number of time-steps

Input: n
▷ the number of mesh-points

$\frac{j_f}{n_t} \to \Delta j$
$\frac{1}{n} \to \Delta x$
$\frac{\Delta j}{\Delta x^2} \to \alpha$

for $0 \leq i \leq n$ **do**
  $0 \to V_{i,0}$
end
$1 \to V_{n+1,0}$ ▷ Initialization of the vector V_0 at $t = 0$

Here is the explicit scheme algorithm :

Algorithm 2: Explicit scheme algorithm

inputs and initializations ▷ **Algorithm 1**

for $0 \leq j \leq j_f$ **do**
  for $1 \leq i < n$ **do**
    $(1 - 2\alpha)V_i + \alpha(V_{i-1} + V_{i+1}) \to V_i$
  end
end

1.1.3 Implicit scheme

Again by discretizing and using the original equation :

$$u_t \approx \frac{u_{i,j} - u_{i,j-1}}{\Delta t}$$
$$u_{xx} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

You notice here a different approach in the first order derivative with respect to time. This tiny difference will result in a very different final algorithm.

$$\begin{aligned} u_{xx} &= u_t \\ \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} &= \frac{u_{i,j} - u_{i,j-1}}{\Delta t} \\ \implies u_{i,j-1} &= \frac{\Delta t}{\Delta x^2}(-u_{i+1,j} + 2u_{i,j} - u_{i-1,j}) + u_{i,j} \\ &= (1+2\alpha)u_{i,j} - \alpha(u_{i-1,j} + u_{i+1,j}) \end{aligned} \tag{5}$$

This is the implicit backward Euler scheme. Similarly to the explicit scheme, we use the vector V_j to rewrite this equation in a matrix form.

$$\hat{A}V_j = V_{j-1} \iff V_j = \hat{A}^{-j}V_0$$

with

$$\hat{A} = \begin{bmatrix} 1+2\alpha & -\alpha & 0 & \dots & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -\alpha & 1+2\alpha & -\alpha \\ 0 & \dots & \dots & 0 & -\alpha & 1+2\alpha \end{bmatrix}$$

And using again the same matrix $\hat{B}$ we can rewrite $\hat{A}$ as $\hat{A} = \hat{I} + \alpha\hat{B}$ and find the eigenvalues of $\hat{A}$ saying that $\lambda_i = 1 - \alpha\mu_i$. Consequently the algorithm is not as simple as the explicit scheme because we have a negative power on the matrix instead of a positive power. Thus the algorithl is based on an a tridiagonal matrix inversion with the Gaussian elimination technic (this algorithm is meant to solve $\hat{A}u = y$ equations for u, with $\hat{A}$ a known tridiagonal matrix). We already gave a full description of this algorithm in the first project of this course, therefore here is the implicit scheme algorithm directly using the tridiagonal solver.

The idea is to perform the elimination $\hat{A}V_j = V_{j-1}$ in a loop over the time-steps, from the initial to the final time. At each run of the loop the function finds V_j and replace V_{j-1} by V_j. We must notice that despite a very small change in the time-derivative expression, the algorithm is very different than the explicit one. The explicit scheme computes the function at a latter time by knowing the current state, whereas the implicit scheme resolves an equation between the current time and the next times ; it is obvious that implicit methods require more computations.

Algorithm 3: Implicit scheme algorithm

inputs and initializations ▷ **Algorithm 1**

$-\alpha \to a$

$1 + 2\alpha \to b$

$-\alpha \to c$

▷ Initialization of the matrix

$V_0 \to y$

▷ We work with $\hat{A} \times u = y$

for $0 \leq j \leq j_f$ **do**

  gaussian elimination(a, b, c, u, y_i)

  $u \to y$

end

1.1.4 Crank-Nicolson scheme

We can generalize these two methods by combining their approaches in a more general one. We introduce a parameter θ and a new equation for the second-order derivative with respect to the space :

$$\frac{\theta}{\Delta x^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + \frac{1-\theta}{\Delta x^2}(u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}) = \frac{1}{\Delta t}(u_{i,j} - u_{i,j-1}) \quad (6)$$

If we set $\theta = 0$ then we find the forward formula, if we set $\theta = 1$ this is the backward formula we end with. If we set $\theta = \frac{1}{2}$ we obtain a new formula : the Crank-Nicolson formula.

$$\frac{1}{2\Delta x^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j} + u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}) = \frac{1}{\Delta t}(u_{i,j} - u_{i,j-1})$$

$$\iff \frac{1}{2}(u_{xx,\text{implicit scheme}} + u_{xx,\text{explicit scheme}}) = u_{tt}$$

$$\implies \frac{\Delta t}{2\Delta x^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - u_{i,j} = \frac{\Delta t}{2\Delta x^2}(-u_{i-1,j-1} + 2u_{i,j-1} - u_{i+1,j-1}) - u_{i,j-1}$$

$$\implies (1+\alpha)u_{i,j} - \frac{\alpha}{2}(u_{i-1,j} + u_{i+1,j}) = (1-\alpha)u_{i,j-1} + \frac{\alpha}{2}(u_{i-1,j-1} + u_{i+1,j-1})$$

$$\implies (2+2\alpha)u_{i,j} - \alpha(u_{i-1,j} + u_{i+1,j}) = (2-2\alpha)u_{i,j-1} + \alpha(u_{i-1,j-1} + u_{i+1,j-1}) \quad (7)$$

which can be written in matrix form as

$$(2\hat{I} + \alpha\hat{B})V_j = (2\hat{I} - \alpha\hat{B})V_{j-1} \iff V_j = (2\hat{I} + \alpha\hat{B})^{-1}(2\hat{I} - \alpha\hat{B})V_{j-1}$$

with

$$\hat{B} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{bmatrix}$$

As you see it in the formula, there are two operations to compute : a matrix-vector multiplication $\widetilde{V}_{j-1} := (2\hat{I} - \alpha\hat{B})V_{j-1}$ and a matrix inversion with the new vector $(2\hat{I} + \alpha\hat{B})^{-1}\widetilde{V}_{j-1}$. The first operation is similar to the explicit scheme but with a different matrix, and the second operation is similar to the gaussian elimination of the implicit scheme. That is the results can be seen as averages of the two methods (after all we use the averages of the second-space derivatives of the implicit and the explicit schemes), and you will see in the next section that the precision is far better here.

Like in the other schemes we initialize the vector u with the boundary conditions and the matrix a. We write $u = V_{j-1}$ and $y = \widetilde{V}_{j-1}$. We perform this operations at each loop by puting the computed vector y to the vector u. That is, at the next time-step, we perform the same operation with vectors updated.

Algorithm 4: Crank-Nicolson scheme algorithm

inputs and initializations
$-\alpha \to a$
$2 + 2\alpha \to b$
$-\alpha \to c$
$V_0 \to y$

Beginning of the algorithm:
for $0 \leq j < j_f$ **do**
  for $1 \leq i < n$ **do**
    $(2 - 2\alpha) \times u_i + \alpha(u_{i-1} + u_{i+1}) \to y_i$
  end
  gaussian elimination(a, b, c, u, y_i)
  $u \to y$
end

1.2 Two space-dimensions equations

$$\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} = \frac{\partial u(x,y,t)}{\partial t}, \qquad t > 0,\ x \in [0, L],\ y \in [0, L] \tag{8}$$

By discretizing the first-order and second-order derivative formulae we define :

$$u_t \approx \frac{u_{i,j}^{l+1} - u_{i,j}^{l}}{\Delta t}$$
$$u_{xx} \approx \frac{u_{i+1,j}^{l} - 2u_{i,j}^{l} + u_{i-1,j}^{l}}{\Delta x^2}$$
$$u_{yy} \approx \frac{u_{i,j+1}^{l} - 2u_{i,j}^{l} + u_{i,j-1}^{l}}{\Delta y^2}$$

We have $\Delta x^2 = \Delta y^2 = h^2$ with $h = \frac{L}{n+1}$ a step. The index i is related to x where $x_i = x_0 + ih$, j is related to y where $y_j = y_0 + jh$ and l is related to t with $t_l = t_0 + lh$ Using the equation (8) :

$$u_t = u_{xx} + u_{yy}$$
$$\frac{u_{i,j}^{l+1} - u_{i,j}^{l}}{\Delta t} = \frac{u_{i+1,j}^{l} - 2u_{i,j}^{l} + u_{i-1,j}^{l}}{h^2} + \frac{u_{i,j+1}^{l} - 2u_{i,j}^{l} + u_{i,j-1}^{l}}{h^2}$$

and we get

$$u_{i,j}^{l+1} = u_{i,j}^{l} + \alpha[u_{i+1,j}^{l} + u_{i-1,j}^{l} + u_{i,j+1}^{l} + u_{i,j-1}^{l} - 4u_{i,j}^{l}] \tag{8}$$

Algorithm 5: Explicit scheme algorithm in two dimensions

Input: t_f ▷ the final time

Input: n ▷ the number of mesh-points

Input: n_t ▷ the number of time-steps

$\frac{t_f}{n_t} \to \Delta t$
$\frac{1}{n} \to h$
$\frac{\Delta j}{h^2} \to \alpha$

for $0 \leq t \leq t_f$ **do**
  for $1 \leq i < n$ **do**
    for $1 \leq j < n$ **do**
      $u_{i,j} + \alpha \times [u_{i+1,j} + u_{i-1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}] \to u_{i,j}$
    end
  end
end

The explicit scheme in two-dimensions is very similar in the approach to the one-dimension one. We just come up with two loops corresponding to the two spatial directions instead of just one. This family of explicit solvers is very convenient to implement. However we will see that it has also lot of restrictions.

Remark 1.1. *The implicit scheme has been implemented two so that you can compare it with our explicit scheme. Their outputs are very similar.*

Chapter 2

Results and analysis

Beside using some standard library functions of the class std::vector instead of old tricks (for example to fill a vector), the implementation is extremely simple. The explicit scheme is just a loop, and the implicit scheme was made easy by reusing the code for project 1. We just had to transform it into a function. The rest was just copying the paper algorithm into C++ language.

2.1 One dimension

The results are pretty convincing for all the schemes. In any case we observe that we begin with a non-differentiable function which gets more smooth and linear as time goes. The equilibrium state is reached after $t = 0.3$ roughly and we observe $u(x,t) = x$. Here we compare the algorithms for different space-steps sizes.

2.1.1 For a time $t = 0.00001$

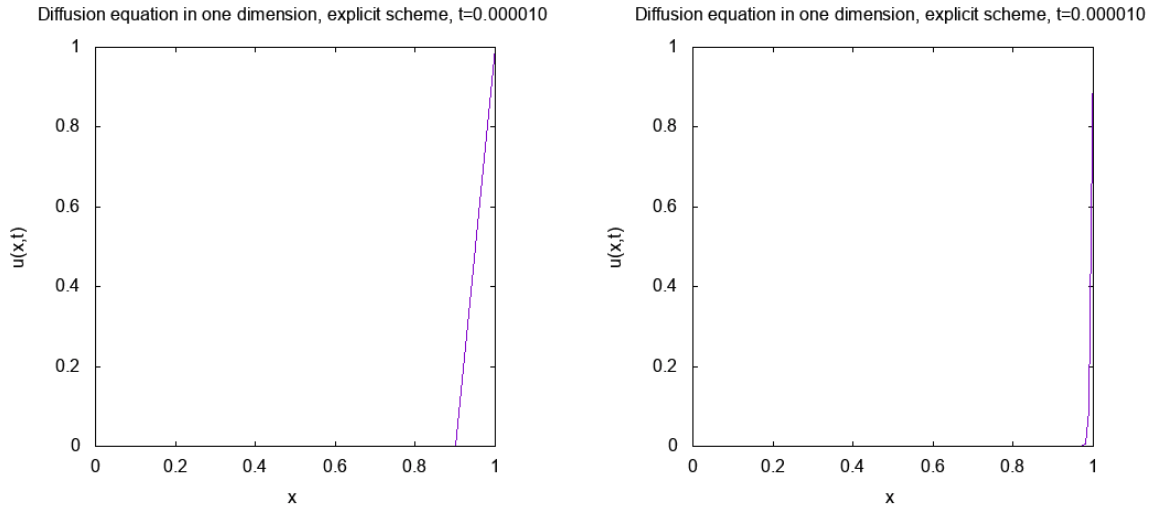

Figure 2.1: Explicit scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

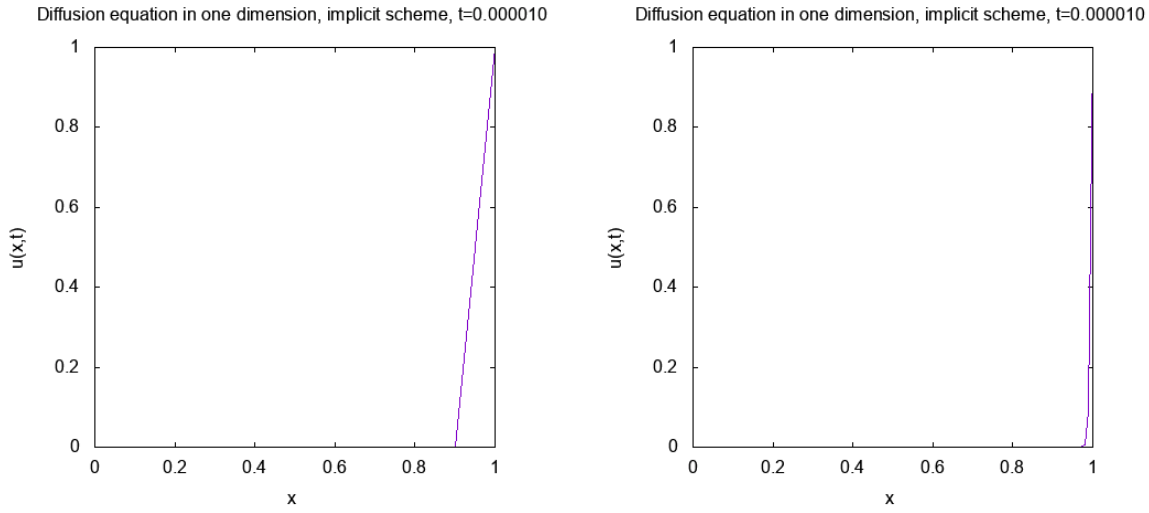

Figure 2.2: Implicit scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

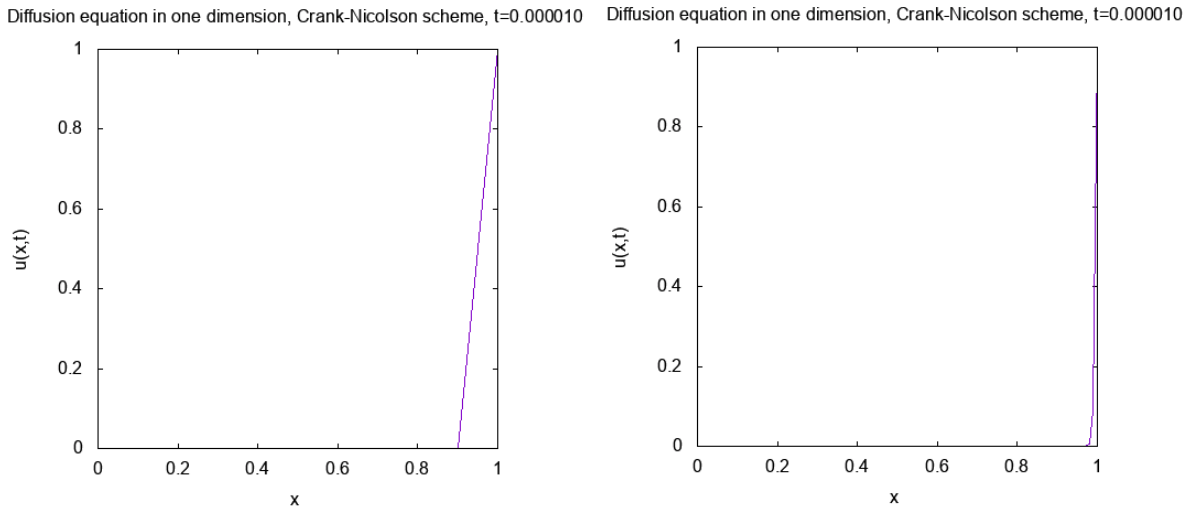

Figure 2.3: Crank-Nicolson scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

As you can observe it, choosing a low precision like $\Delta x = \frac{1}{10}$ can be very dangerous for the results, even if the requirements on α are fulfiled. That being said, it is not clear at this time which scheme is the best.

2.1.2 For a time $t = 0.2$

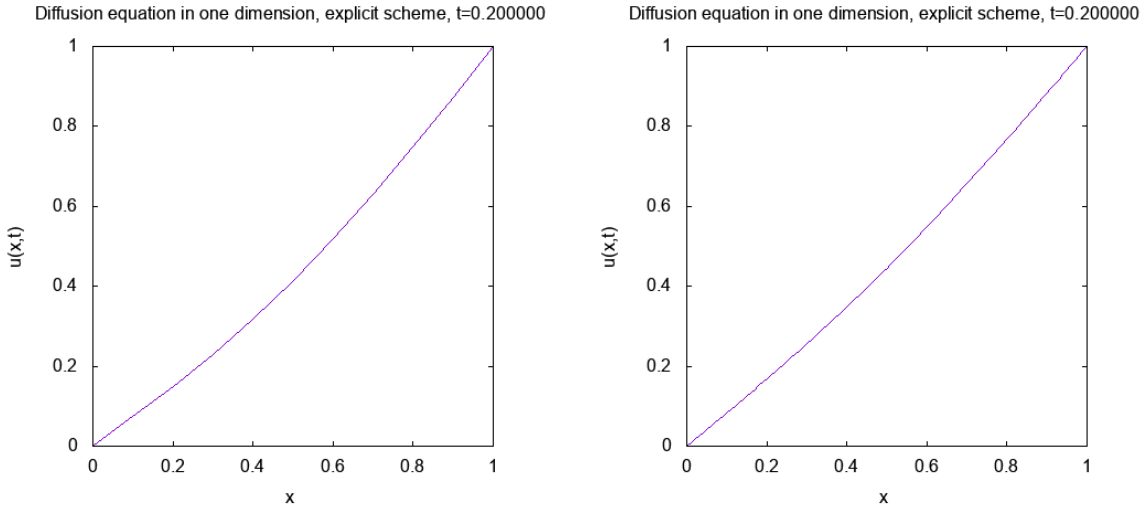

Figure 2.4: Explicit scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

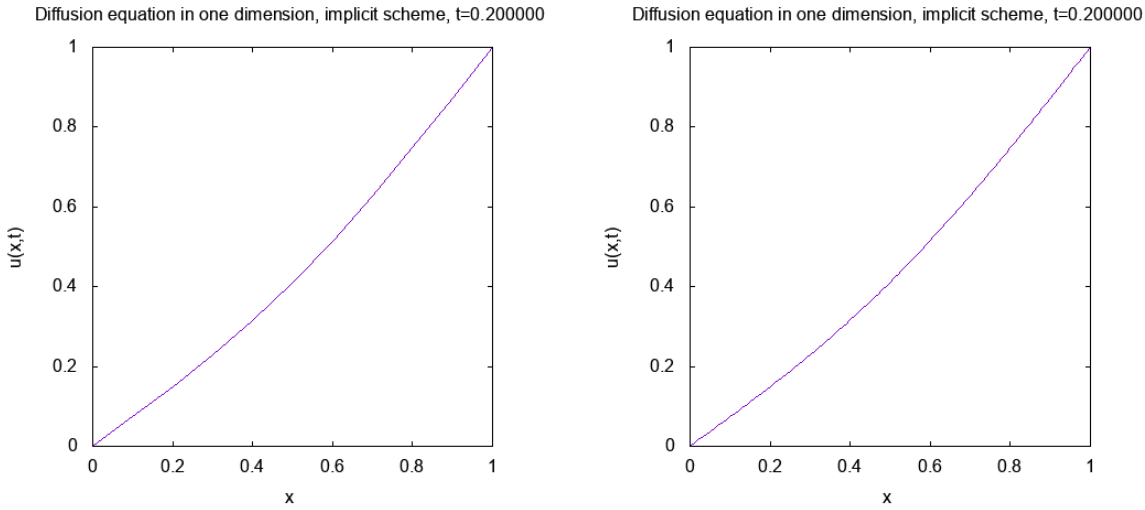

Figure 2.5: Implicit scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

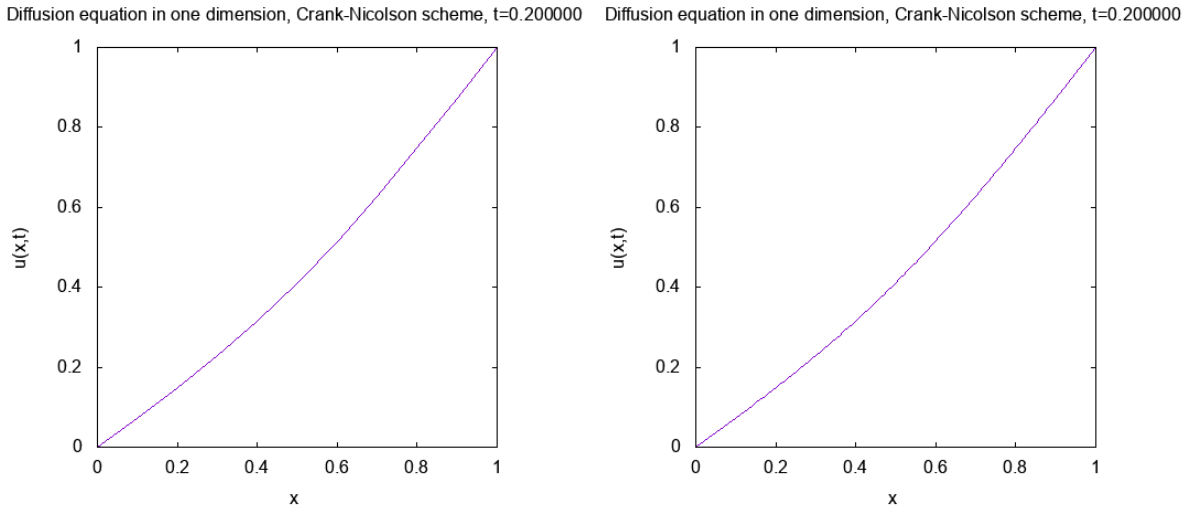

Figure 2.6: Crank-Nicolson scheme for $\Delta x = \frac{1}{10}$ and $\Delta x = \frac{1}{100}$

When the space-steps are increased, the differences between the schemes are not obvious and they all seem efficient. Eventually when we choose $\Delta x = \frac{1}{100}$ the curve is smoother and does not look like a set of straight lines.

2.1.3 Comparison

On this picture it is clear that the explicit scheme is way less efficient than the others. We computed the simulations with 200 space-steps, 20000 time-steps and a final time $t = 0.2$, that is we had $\alpha = 0.4$, which fulfills the stability requirements for the explicit scheme. Both the implicit scheme and the Crank-Nicolson scheme are cofounded with the analytical solution[1] and they would clearly be the best choice.

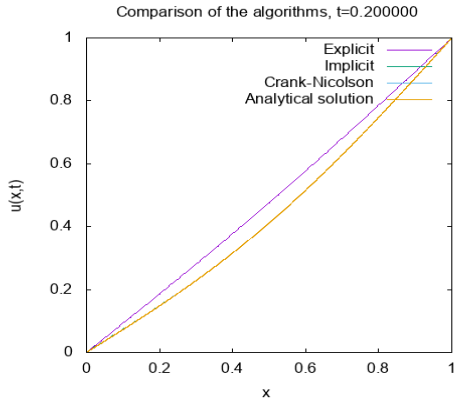

Figure 2.7: Comparison of the three schemes for the same values of Δx and Δt.

[1]It was plotted using the first 228 terms of the Fourier-Series and 5000 space-steps.

Even for $\Delta x = \frac{1}{100}$ they are very similar. Here are some benchmarks for those one-dimension solvers with 500 space-points, 100000 time-points and a final time $t = 0.2$, which is a precision way more than sufficient. The explicit scheme is without any doubt the fastest, the Crank-Nicolson scheme is roughly 5.7 times slower, but the precision is way more accurate. Once again, it is a question of priority between speed and precision, but we would advice to choose the implicit scheme even if it is on the paper less efficient than the Crank-Nicolson scheme.

Explicit scheme	Implicit scheme	Crank-Nicolson scheme
0.1424s	0.767s	0.8231s

Table 2.1: Operation times in average for the three schemes in one dimension with the above parameters.

The latter Von Neumann stability analysis will prove us that both the Implicit and the Crank-Nicolson scheme have a precision of the order 10^{-3} if we compare them with the exact solution.

2.2 Two dimensions and discussion on the diffusion process

The two-dimensions solvers and the free boundary conditions allow to understand how diffusion of a process occur. We made short videos with this program and some bash. They show the diffusion process in time in one-dimension and two-dimensions with various boundary conditions. They are available here : `https://www.youtube.com/playlist?list=PL9Bkzl2Vcy4sJMAbtl1KsfRhMv7KhHTp6`.

In both one-dimension and two-dimensions we observe that the heating process is very fast at the beggining and the convergence is very slow after $t = 0.001$. This convergence is a uniform convergence to a stable function which represent the steady state. We explain the slowness of the convergence by the fact that the closed form solution contains a temperature term $e^{-{\lambda_n}^2 t}$ which goes to 0 as t goes to infinity, resulting in a slow convergence. Furthermore, the maximum principle for partial differential equations is verifies here because the minimum value for $u(x, y, t)$ is always the minimum of the boundary and the maximum value for $u(x, y, t)$ is always the maximum of the boundary. However the convergence occurs way before for $t < 1$ for all the tested boundary conditions. The fastest convergence occurs in the *Simulation 1*, and the other take longer time as they have different boundary conditions. When we apply only one heating process on one side on the lattice (*Simulation 4*) instead of applying the same heating process on every side of the lattice (*Simulation 1*), the convergence is also much slower.

Chapter 3

Stability properties of those algorithms

3.1 Theory

Some linear algebra tricks (again) allow us to evaluate the stability of the schemes we built. Those mathematical considerations are presented in the lecture notes, in *Introduction to partial differential equations* by Tveito and Winther and in lectures by T. Lakoba from the University of Vermont. We present thereafter their results :

Scheme	Truncation Error	Stability requirements
Cranck-Nicolson	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t^2)$	Stable for all Δt and Δx
Backward Euler	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t)$	Stable for all Δt and Δx
Forward Euler	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t)$	$\Delta t \leq \frac{1}{2}\Delta x^2$

Table 3.1: Comparison of the different schemes

To resume, the truncator error of the explicit scheme are laws of Δx whereas the truncator error of the Crank-Nicolson scheme is a law of Δx^2. The best algorithm would therefore be the Crank-Nicolson scheme since it is "undoncitionnaly stable" and that it can be computed for any α[1]. Intuitively we mat say that for very low times (roughly $t < 0.00001$) the derivated tangent coefficient u_{xx} undergoes jumps and irregularities since the function is hardly differentiable. An explicit scheme based on the current state and extrapolating the next step is therefore necessarely less accurate than a scheme based on more precise equations. But the Crank-Nicolson being a mix of the two paradigms is necessarely more precise.

Those mathematics can easilly be extended to a two-dimensions equation. The stability requirements for α are $\alpha < \frac{1}{4}$.

[1] Again it is reasonnable not to choose too few space-points.

3.2 Observations

3.2.1 One dimension

The plots for the diffusion function u are very convincing, and from a human eye, they all give the same results[2], and those results seem very close to the analytical solution for every time t as long as the value for α is ok. However our investigation do not match with the theory. Here is a plot of the relative error $\epsilon = \log|\frac{u_{comp}(x)-u_{anal}(x)}{u_{anal}(x)}|$ as a function of Δx. We fixed $x = 0.5$, $t = 0.2$ and $\Delta t = 500000$ so that we could compute the explicit scheme for many different Δx from $\frac{1}{10}$ to $\frac{1}{1000}$.

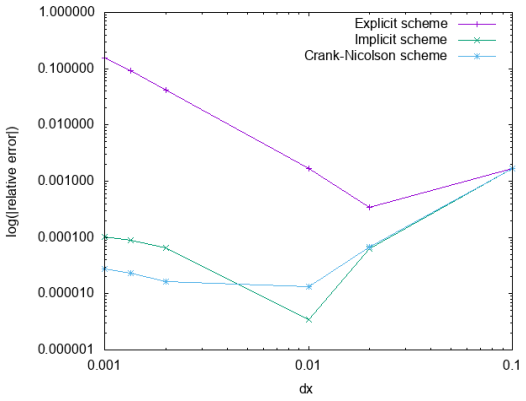

Figure 3.1: Relative error of the three schemes as a function of Δx

Note also that the relative error of the Crank-Nicolson scheme does not grow as a function of Δx^2 as expected. The result is linear after a certain dx is high enough. However, the fact that in all three cases the relative error grows with Δx decreasing is mathematically irrelevant.

This situation is not confortable for two reasons. Firstly this is hard to interpret because except for the explicit scheme the results of all the schemes seem relevant and reliable, secondly because this behavior is mathematically not logical and that it must be due to an implementation error. However the implementation is considered valid because the results are valid, homogeneous and reproducible, this implies that the problem is a problem of numerical precision (unlikely though) or a problem of adapting the mathematical method to an informatic concrete algorithm. In both cases this is hard to fix without spending some time investigating the code, the language and the math.

Moreover the stability of the Crank-Nicolson scheme is not "unconditional" as expected. Yet the informatic reality shows that there can be unconvenient oscillations on the curve due to the approximation of the Fourier series. The curve have the same aspect but undergoes a multiplicative factor. This can occur from times to times. Yet the error remains very

[2]Well maybe not for the explicit scheme.

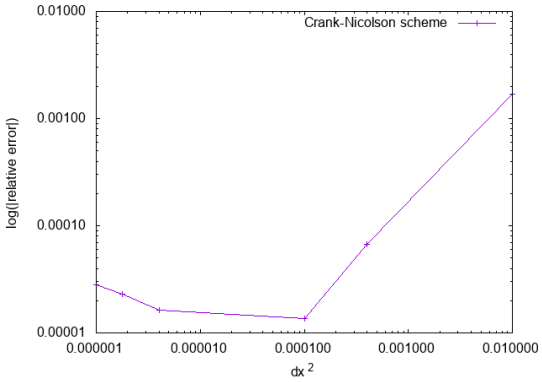

Figure 3.2: Relative error of the Crank-Nicolson scheme as a function of Δx^2

acceptable for both the implicit and explicit scheme (inferior to an order of magnitude of 10^{-3}) and the computation times are very fast, which lead us to consider that those solver fulfil what we expect from them, which is fast computations, reliable results and easyness to use.

3.2.2 Two dimensions

In this part we observe that even though the alpha requirement $\alpha < \frac{1}{4}$ is fulfilled, the explicit scheme in two dimensions is not very stable and precise when α is greater than 0.1. Here are some plots to prove it.

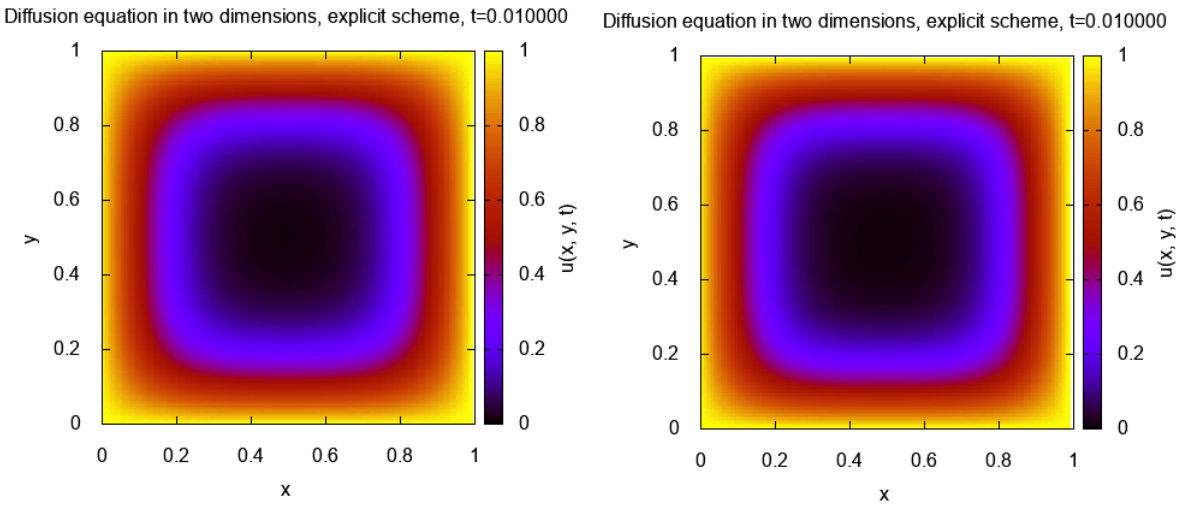

Figure 3.3: Implicit scheme comparison for $\alpha = 0.125$ and $\alpha = 0.08$

We observe that for the biggest α the diffusion process looks more advanced than in the other case. This proves that the stability analysis unfortunately remains very theoritical, and can not always match the reality. The best thing to do is therefore to compare the results with other algorithms (we implemented an implicit scheme as well) and to take many precautions, such as choosing stronger requirements for α. In our case, $\alpha < 0.1$ seems a good choice for the two-dimensions explicit solver.

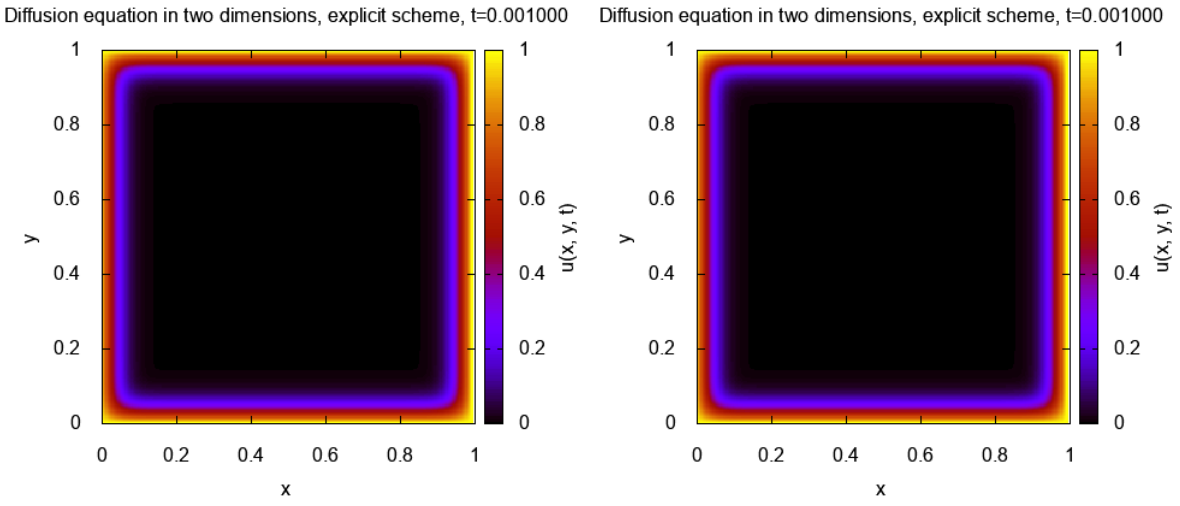

Figure 3.4: Implicit scheme comparison for $\alpha = 0.05$ and $\alpha = 0.0125$

This time the plots are very similar.

Conclusion

The most important thing to keep in mind after this project is that stability models should not be taken for granted. The error-predictions did not match the reality even if the results were very good with wisely chosen. Since those schemes are very similar and easy to implement, it would be wise to run the implicit and the Crank-Nicolson scheme in parallel and to output the average of their results. These methods are admittedly less fast than computing the Fourier-series which is solution of the equation, but they require very basic algorithmic knowledge to be built and they can stand on previously written codes.
That being said, those algorithms have the advantage of being very simple and to allow one to solve partial differential equations without requiring the mathematical background to solve those equations. After having studied those algorithms, we are know able to build larger algorithms which will need those kind of solvers.

Bibliography

- Introduction to Partial Differential Equations : A computational Approach - Aslak Tveito & Ragnar Winther
- The Heat equation in 2 and 3 spatial dimensions - T. Lakoba