University of Lisbon - Instituto Superior Técnico

Masters in Data Science and Engineering

Computational Methods in Finance

Submitted by: Ricardo Fraga Simões, nº 93674

To: Prof. Dr. Farid Bozorgnia

Lisbon, 5/03/2022

Index

General Comments

This report presents the results obtained for the Computational Methods in Finance project. For all 5 questions, discussion about the numerical results was included, and in some cases, also the linkage with the mathematical concepts that were used. Relevant images were also included to facilitate the results' interpretation.

Alongside this report, Python notebooks were also delivered (one for each question) with comments that help the reader to better understand the code. The notebooks were made using Google Collab, and the reader can obtain **exactly all** the same results here presented with these notebooks. It is advised to import these notebooks into Google Collab, since they were developed there.

It is worth mentioning that this project was developed solely by one person of the Computational Methods in Finance course, and all the questions were answered (note only the ones for Computational Methods in Finance course, but also the questions for the Numerical Analysis of Partial Differential Equations course) . Sometimes, even different ways to obtain a numerical solution were implemented.

All results were calculated on a computer with an Intel Core i3-7100U CPU, 2.4 GHz with 4096Mb of RAM.

Exercise 1

For this exercise, we will solve the following advection equation:

$$\begin{cases} u_t - u_x = 0, x \in (0,1) \\ u(0,x) = e^{\frac{-(x-0.2)^2}{0.01}} \\ and\ Periodic\ B.C. \end{cases}$$

We will use the FTCS, BTCS, and Leapfrog schemes to solve it for $t = 2$. The numerical solutions will be compared with the analytical solution. The last one can be obtained via the method of characteristics:

$$u(x,t=2) \underset{\textbf{MoC}}{=} u(x-(-1)*2,0) = u(x+2,0) \underset{\textbf{Per. B.C.}}{=} u(x,0)$$

The periodic Boundary conditions are:

$$\begin{cases} u(1+dx,t) = u(dx,t) \\ u(0-dx,t) = u(1-dx,t) \end{cases}, dx > 0$$

Please consult the notebook Exercise1.ipynb for details about the code.

FTCS Scheme

Describing the previous equation using the FTCS Scheme, we obtain:

$$u_t - u_x = 0 \ \rightarrow \ \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{u_{j+1}^n - u_{j-1}^n}{2\text{h}} = 0 \Leftrightarrow$$

$$\Leftrightarrow\rightarrow \ u_j^{n+1} = u_j^n + \frac{a * \Delta t}{2h} * (u_{j-1}^n - u_{j+1}^n)$$

Which is an explicit (in time) scheme.

We can start by applying Von Neumann's analysis to assess the stability of the scheme. Replacing $u_j^n$by $g^n e^{ij\theta}$:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{u_{j+1}^n - u_{j-1}^n}{2\text{h}} \to \frac{g^{n+1}e^{ij\theta} - g^n e^{ij\theta}}{\Delta t} - \frac{g^n e^{i(j+1)\theta} - g^n e^{i(j-1)\theta}}{2\text{h}} =$$

$$= g^n e^{ij\theta} * (\frac{g-1}{\Delta t} - \frac{e^{i\theta} - e^{-i\theta}}{2\text{h}})$$

The amplification factor of this method has a squared norm equal to:

$$|g(\theta)|^2 = 1 + (\frac{\Delta t}{h})^2 * \sin^2\theta$$

Having $|g(\theta)| > 1$, we can conclude that the scheme is unstable. Not having stability will also lead to the absence of convergence, so we are likely to encounter this in our solutions. Let's consider the plots from Fig. 1, using $h = 0.005$ and varying the Δt

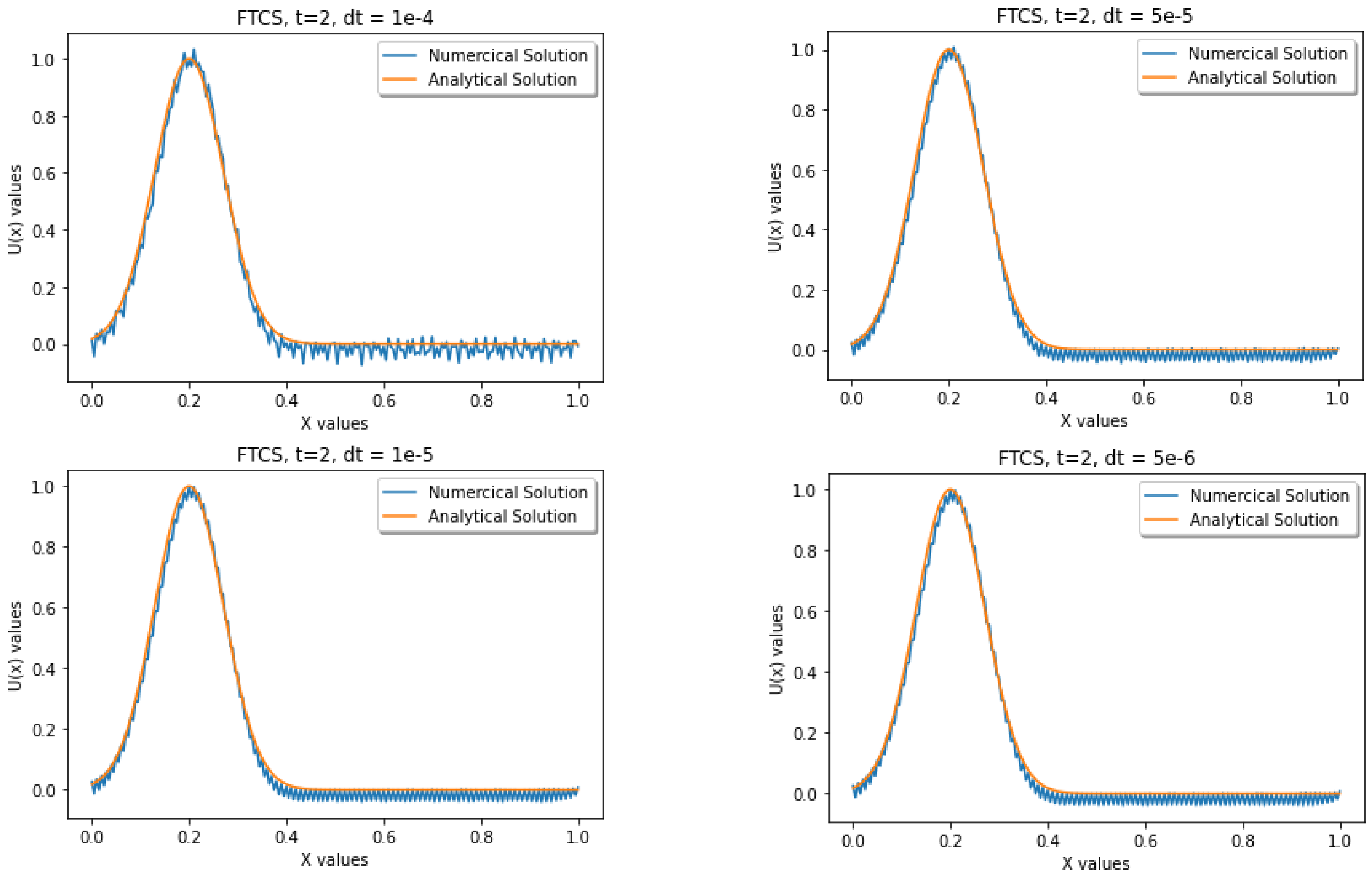


Figure 1 - FTCS scheme for different Δt values

As expected, there is a lot of instability around the analytical solution, even for a very thin Δt. We conclude that there is no convergence due to the lack of stability of the FTCS scheme.

Leapfrog Scheme

Describing the previous equation using the Leapfrog Scheme, we obtain:

$$u_t - u_x = 0 \rightarrow \frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} - \frac{u_{j+1}^n - u_{j-1}^n}{2\text{h}} = 0 \Leftrightarrow$$

$$\Leftrightarrow\rightarrow\ u_j^{n+1} = u_j^{n-1} + \frac{\Delta t}{h} * (u_{j+1}^n - u_{j-1}^n)$$

Which is an explicit (in time) scheme.

To assess the stability of the Leapfrog scheme, one can use the Fourier Inversion formula for $u^{n+1}, u^{n,}$ and u^{n-1} and achieve a condition for $\frac{\Delta t}{h}$ such that the amplification factor is less than one. The proof is quite extensive and it won't be presented here, but it can be found in references at the end of the report. For this particular PDE, the Leapfrog's stability condition is $\frac{\Delta t}{h} < 1$.

Having the stability condition met is not a sufficient condition for convergence. Nevertheless, it is worth investigating the numerical solutions for different values of Δt and fixed h, such that the stability condition is satisfied. Let's consider the plots of Fig. 2, where $h = 0.005$ and Δt is varying.

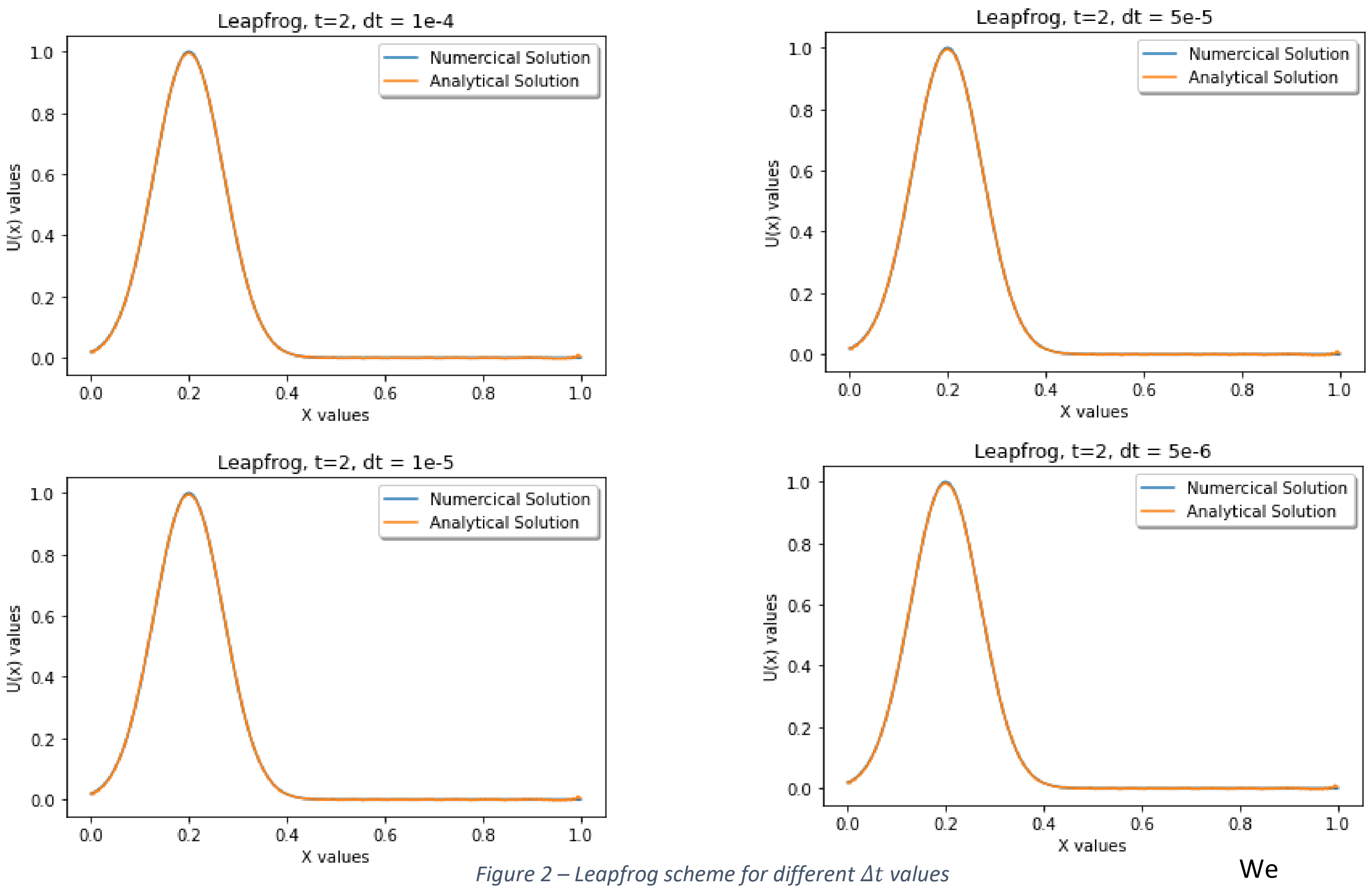
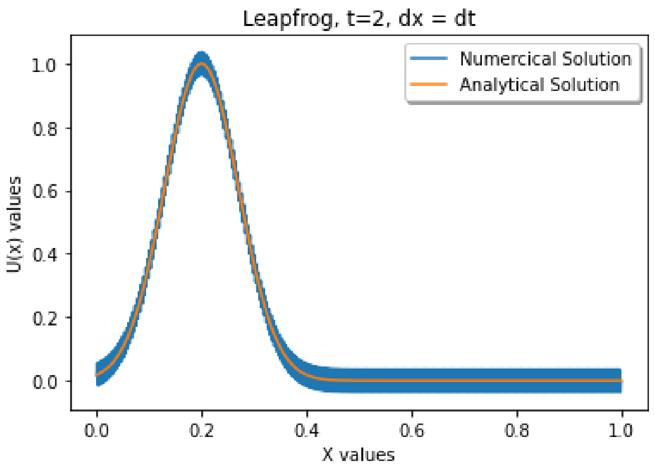


Figure 2 – Leapfrog scheme for different Δt values

We can observe that the numerical solution perfectly converged with the analytical solution when the stability condition was met. Nevertheless, let's observe in Fig. 3, what happens when $\frac{\Delta t}{h} = 1$ and $\frac{\Delta t}{h} > 1$. Respectively:



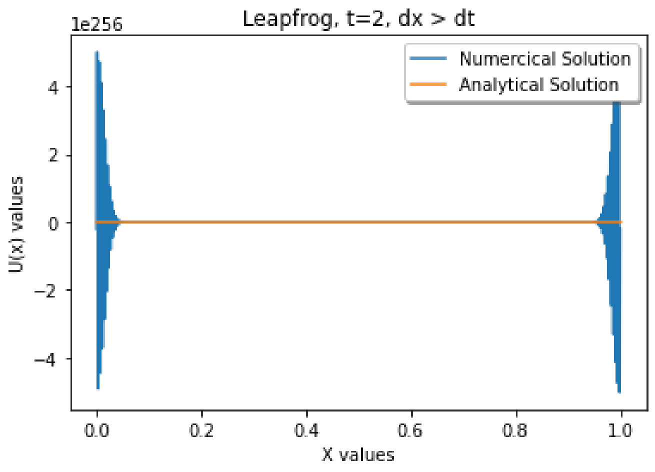


Figure 3- Leapfrog scheme for different Δt values (instable cases)

As expected, there is no convergence due to the lack of stability. When Δt = h the numerical solution oscillates around the analytical solution, and when h > Δt the numerical solution explodes.

BTCS Scheme

Describing the previous equation using the BTCS Scheme, we obtain:

$$u_t - u_x = 0 \;\rightarrow\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\text{h}} = 0 \Leftrightarrow$$

Which is an implicit (in time) scheme.

To assess the stability of the BTCS scheme, we can re-write the previous equations as:

$$u_j^{n+1}\left(\frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\text{h}}\right) = 0 \Leftrightarrow$$

$$\Leftrightarrow \left(u_j^{n+1}\right)^2 + \left(u_j^{n+1} - u_j^n\right)^2 = (u_j^n)^2 + \frac{\Delta t}{h}\left(u_{j+1}^{n+1} - u_{j-1}^n\right)^2 u_j^{n+1} \longrightarrow$$

$$\rightarrow \|u^{n+1}\|^2 = \|u^n\|^2 - h\sum_j (u_j^{n+1} - u_j^n)^2 \leq \|u^n\|^2$$

So the scheme is (strongly) stable with no restriction on Δt.

Since the scheme is implicit, solving it is not a trivial task as solving the equations of the previous schemes. Nevertheless, we can re-write the scheme as:

$$u_{j+1}^{n+1}\left(-\frac{\Delta t}{2h}\right) + u_j^{n+1} + u_{j-1}^{n+1}\left(\frac{\Delta t}{2h}\right) = u_j^n \Leftrightarrow U^{n+1}A = U^n$$

Where,

$$A = \begin{bmatrix} 1 & \left(-\frac{\Delta t}{2h}\right) & 0 & \cdots & \cdots & \left(\frac{\Delta t}{2h}\right) & 0 \\ \left(\frac{\Delta t}{2h}\right) & 1 & \left(-\frac{\Delta t}{2h}\right) & 0 & \dots & 0 & 0 \\ 0 & \left(\frac{\Delta t}{2h}\right) & 1 & -\left(\frac{\Delta t}{2h}\right) & \dots & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & \left(\frac{\Delta t}{2h}\right) & 1 & \left(-\frac{\Delta t}{2h}\right) \\ 0 & \left(-\frac{\Delta t}{2h}\right) & \cdots & \cdots & \cdots & \left(\frac{\Delta t}{2h}\right) & 1 \end{bmatrix}$$

Knowing the solution at $t = 0$, we can sequentially solve this system for the next time-step until we arrive at $t = 1$. Instead of finding the inverse of the matrix A, we decided to use a more efficient method. Therefore, we used function *linalg.solve*, from the *NumPy* package. It first factorizes the matrix using LU decomposition and then solves for our unknowns using forward and backward substitution.

Although this scheme is consistent and stable for any values of Δt/h, oddly, the scheme presented some oscillations, even for values of Δt/h very small. In Fig. 4 we present some examples obtained for, $h = 0.005$ and Δt is varying.

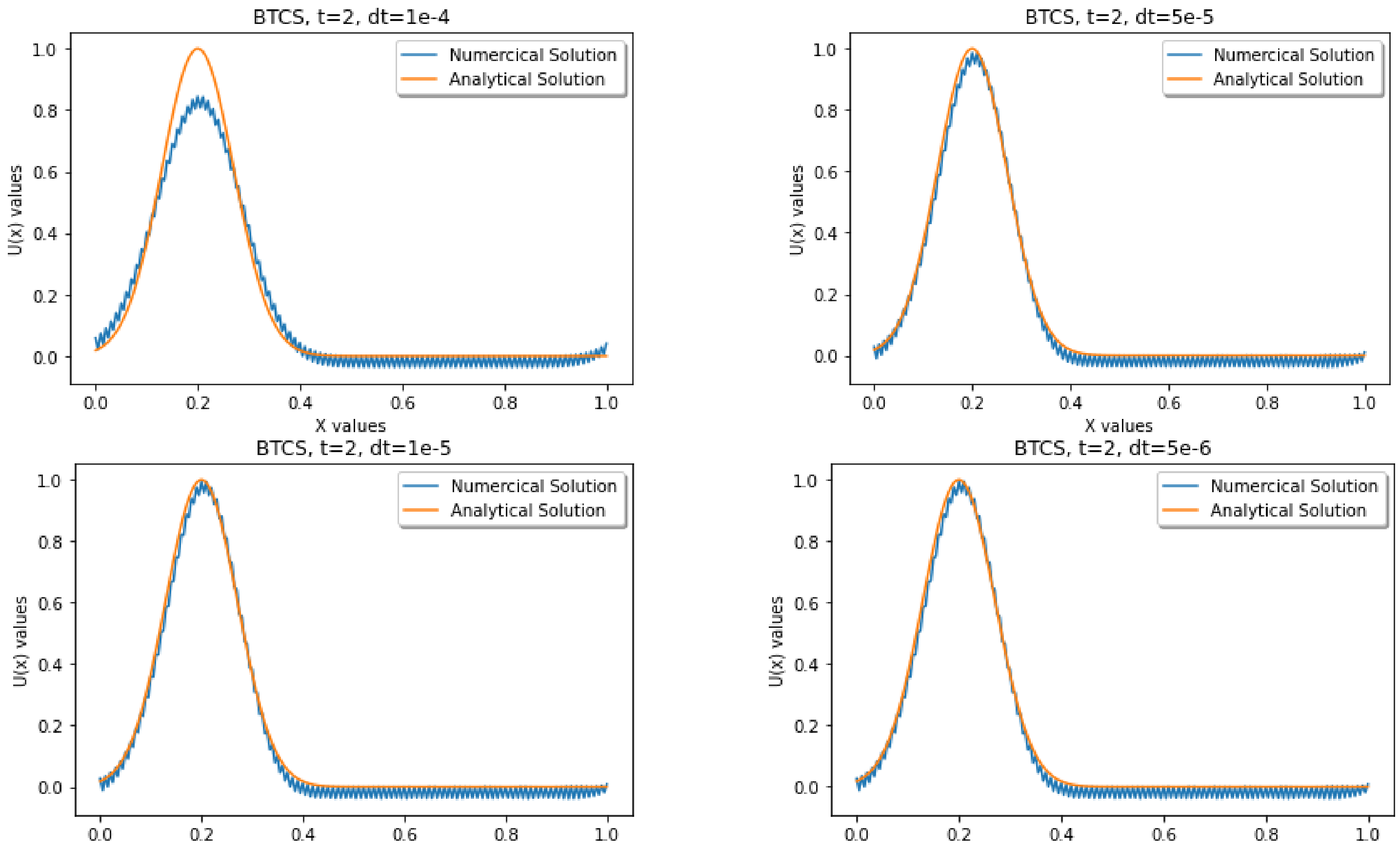


Figure 4 – BTCS scheme for different Δt values

Exercise 2

For this exercise, we will solve the following 1D equation:

$$\begin{cases} (\beta(x)u_x)_x - \gamma(x)u = f(x), x \in (0,1) \\ u(0) = 1 \\ au(1) + bu'(1) = c \end{cases}$$

Where:

- $\beta(x) = 1 + x^2$
- $\gamma(x) = x$
- $a = 2$
- $b = -3$

And using the following central scheme:

$$\frac{\beta_{i+1/2}\frac{u_{i+1} - u_i}{h} - \beta_{i-1/2}\frac{u_i - u_{i-1}}{h}}{h} - \gamma_i u_i = f_i$$

Knowing that the exact solution is:

$$u(x) = e^{-x}(x-1)^2$$

We can find the expression of $f(x)$ by plugging in $u(x)$ into the PDE. Using the well-known derivation rules, and simplifying the expression, we obtain:

$$f(x) = e^{-x} * (x^4 - 9x^3 + 18x^2 - 13x + 7)$$

As for the constant c we can use, again, $u(x)$ to find it. We must compute $u'(x)$ first, and then plug $x = 1$. Since $a = 2$ and $b = -3$, then

$$c = 0$$

Please consult the notebook *Exercise2.ipynb* for details about the code.

Explicit Scheme (Using LU Decomposition)

Using the provided explicit scheme, two different ways of obtaining the numerical solution were used.

We can start by re-writing the central scheme as:

$$u_{i+1} * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_i * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_{i-1} * (\frac{\beta_{i-1/2}}{h^2}) = f_i \Leftrightarrow$$

$$\Leftrightarrow MU^n = F, n \in \{1, \dots, 100\}$$

So solving this equation with this finite difference scheme is equivalent to solving a system of linear equations for $n = 100$ interior points. Nevertheless, we have to impose the boundary conditions. The *Dirichlet* boundary condition is easier to impose; since $u(x = 0) = u_0 = 1$, the previous equation is slightly different for updating $u(x = 0 + h) = u_1$:

$$u_2 * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_1 * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_0 * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_1 \Leftrightarrow$$

$$u_2 * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_1 * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) = f_1 - u_0 * \left(\frac{\beta_{i-1/2}}{h^2}\right) \Leftrightarrow$$

Since we have a Robin boundary condition on the other end, the equation for updating $u(x = 1 - h) = u_n = u_{100}$ will also be different. Recovering the boundary condition, we will re-write it using finite differences. We can approximate the term $u'(1)$ using central or backward differences. In theory, the central is preferable because it produces an error of $O(h^2)$, while the other produces an $O(h)$. Both where implemented to assess if one is preferable to the other.

<u>Backward:</u>

$$au(1) + bu'(1) = 0 \Leftrightarrow au_{n+1} + b\frac{u_{n+1} - u_n}{h} = 0 \Leftrightarrow u_{n+1} * \left(\frac{ah + b}{b}\right) = u_n \Leftrightarrow u_{n+1} = u_n\left(\frac{b}{ah + b}\right)$$

$$u_{n+1} = u_n\left(\frac{3}{3 - 2h}\right)$$

<u>Central:</u>

$$au(1) + bu'(1) = 0 \Leftrightarrow au_{n+1} + b\frac{u_{n+2} - u_n}{2h} = 0 \Leftrightarrow u_{n+2} = \frac{-2ah}{b}u_{n+1} + u_n$$

$$\Leftrightarrow u_{n+2} = \frac{4h}{3}u_{n+1} + u_n$$

We can see that for the central approximation of the derivative, we need a ghost point u_{n+2}. This ultimately leads to having u_{n+1}in the system of equations, something that does not happen when we make the backward approximation.

So last equation when we make the backward approximation is:

$$u_{n+1} * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_n * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_{n-1} * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_n \Leftrightarrow$$

$$u_n * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i + \left(\frac{3}{3 - 2h}\right) * \left(\frac{\beta_{i+1/2}}{h^2}\right)\right) + u_{n-1} * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_n \Leftrightarrow$$

While the last equation of the system when we make the central approximation is:

$$u_{n+2} * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_{n+1} * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_n * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_{n+1} \Leftrightarrow$$

$$\Leftrightarrow \left(\frac{-2ah}{b}u_{n+1} + u_n\right) * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_{n+1} * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_n * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_{n+1} \Leftrightarrow$$

$$\Leftrightarrow u_{n+1} * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i - \frac{-2ah}{b}\frac{\beta_{i+1/2}}{h^2}\right) + u_n * \left(\frac{\beta_{i-1/2}}{h^2} + \frac{\beta_{i+1/2}}{h^2}\right) = f_{n+1} \Leftrightarrow$$

We highly advise the reader to analyze the code provided to see how these conditions are imposed and the equations are created. After creating the M matrix and the F vector, there were many ways to solve this linear system of equations. A more efficient procedure was used instead of inverting an $\mathbb{R}^{100x100}$ matrix (which is not a good practice). Again, we decided to use the function *linalg.solve*, from the *NumPy* package.

In Fig 5 we can see that for both scenarios the numerical solution converged to the analytical one. Regarding the errors obtained, they are presented below on the sub-section regarding grid refinement analysis.

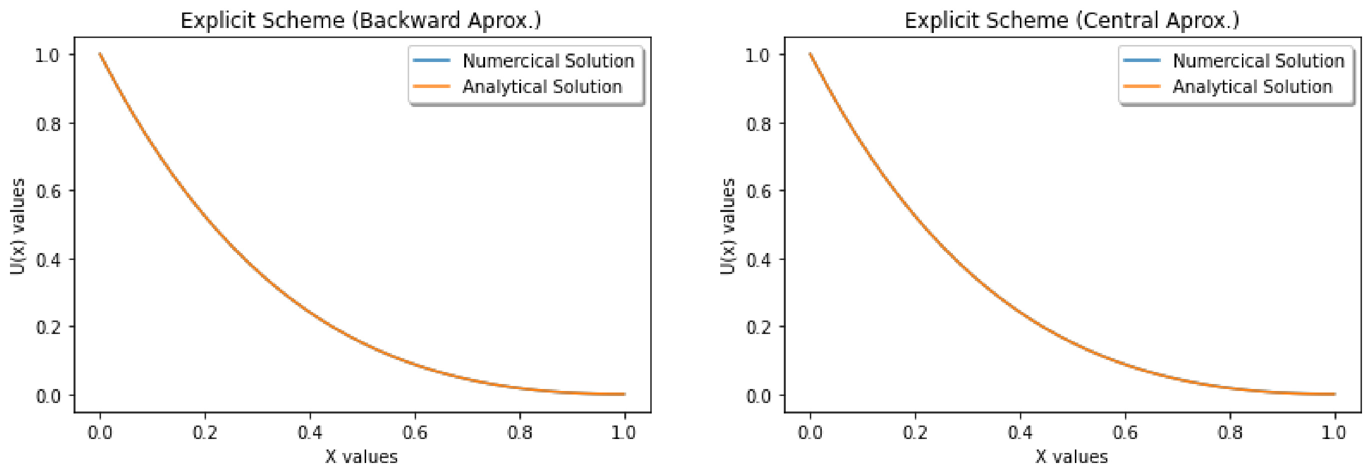


Figure 5 - Numerical results for the Explicit Scheme (for twi different ways of approximating derivative)

We can see that both the analytical and numerical solutions are completely identical. In particular, using the LU Decomposition method to solve the system of equations was a good choice since the solution was almost instantly computed. The CPU times were 5.43ms and 5.53ms, respectively.

Explicit Scheme (solving the system of equation with Iterative Method)

Moving on to the second method for solving the system of equations, this one is another plausible alternative, but more computationally demanding.

Recapping the finite-difference scheme equation:

$$u_{i+1} * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_i * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_{i-1} * (\frac{\beta_{i-1/2}}{h^2}) = f_i$$

First, we will re-write the equation to u_i, using u_{i+1}, $u_{i-1,}$ and f_i,

$$u_i = \frac{f_i - u_{i+1} * \left(\frac{\beta_{i+1/2}}{h^2}\right) - u_{i-1} * \left(\frac{\beta_{i-1/2}}{h^2}\right)}{\left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right)}$$

Now the rationale is to initialize $u(x) = 0$ for all points in the grid and update from left to right using this previous equation. The two exceptions are the last point of the grid and the boundary point $x = 1$.

For a particular iteration after updating u_1, all the way up to u_{n-1}, the last two updates are given by:

1. $u_n = \dfrac{f_n - u_{n-1} * \left(\frac{\beta_{i-1/2}}{h^2}\right)}{\left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i + \left(\frac{3}{3-2h}\right) * \left(\frac{\beta_{i+1/2}}{h^2}\right)\right)}$
2. $u_{n+1} = u_n\left(\frac{3}{3-2h}\right)$, by this order.

Here we are using the boundary conditions given when we approximate $u'(1)$ by a backward scheme.

After the updates, we compute de distance between the new solution vector with the solution vector of the previous equation. If it decreased less than a specific amount ($1e-5$ was the used value), the cycle ends.

Below, in Fig. 6, we present the analytical and numerical solution for this problem, using this iterative procedure.

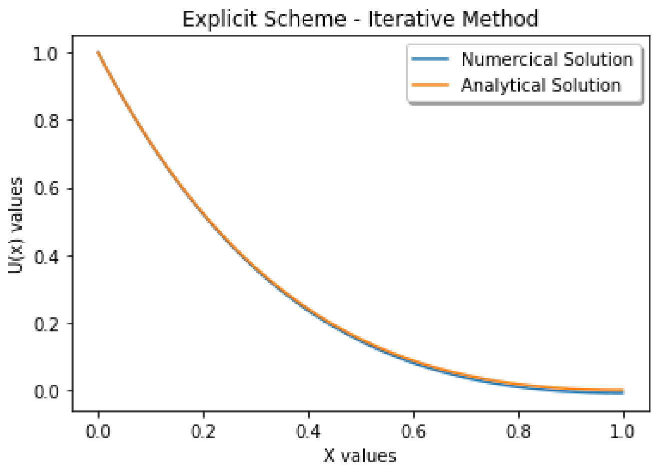


Figure 6 - Explicit Scheme solution using Iterative Method

Like we expected, this procedure was by far more demanding than the previous one. The CPU time was 38.2 seconds, which is much worse than the other method using LU decomposition.

Error and Grid Refinement Analysis

We will now proceed with the grid refinement analysis and the analysis of the errors. In the previous exercise, we use 100 interior points. Now we will start with a grid of 25 interior points and double this number successively and verify how much the error drop for each refinement step. The error is given by the infinity norm of the difference between the analytical solution and numerical solution, given by the method where LU Decomposition was used to solve the system. We did this analysis for both cases where the derivative was approximated differently. The results of this procedure are presented in table 1.

Table 1 - Grid Refinement Analysis

i	Interior Points	$\|u - \widehat{u_i}\|_\infty$ (Backward Aprox.)	$\|u - \widehat{u_i}\|_\infty$ (Central Aprox.)
1	25	2.958e-04	7.146e-03
2	50	7.380e-05	1.859e-03
3	100	1.842e-05	4.740 e-04
4	200	4.602e-06	1.197e-04
5	400	1.150e-06	3.007e-05

So we can verify, empirically, that doubling the number of interior points will reduce the global error by four, for both cases. We can re-write this as:

$$4 \approx \frac{\|u - \widehat{u_\iota}\|_\infty}{\|u - \widehat{u_{\iota+1}}\|_\infty} \approx \frac{Ch^p}{C\left(\frac{h}{2}\right)^p} \approx 2^p \Rightarrow p = 2$$

Where $p = 2$ is the order of accuracy of the global solution. Oddly, when we make the central approximation for the derivative the error is slightly larger, but difference is meaningless.

Changing Boundary Conditions

In this sub-section, we will analyze what happens if one of the constants a and b of the $Robin$ boundary condition is set to 0 and investigate if the code produced for this exercise will work. We only did this analysis for the case where the derivative of the boundary condition is approximated by a central difference. Since the Python function built to solve the exercise receives a and b as inputs, then it will calculate a solution for the values the user wants

For $b = 0$, we immediately obtained an error saying that that vectors can't contain infinite values, and this makes sense since we are dividing by 0 on the boundary condition for $x = 1$ (see last equation in Page 9). Now if we consider $a = 0$,

$$au(1) + bu'(1) = c \;\Leftrightarrow 0 * u(1) - 3 * u'(1) = 0 \;\Leftrightarrow u'(1) = 0$$

This means that at the right end we have a *Neumann* boundary condition.

When we compute the solution for this scenario (see Fig. 7), we can see that it coincides with the solution of the initial scenario.

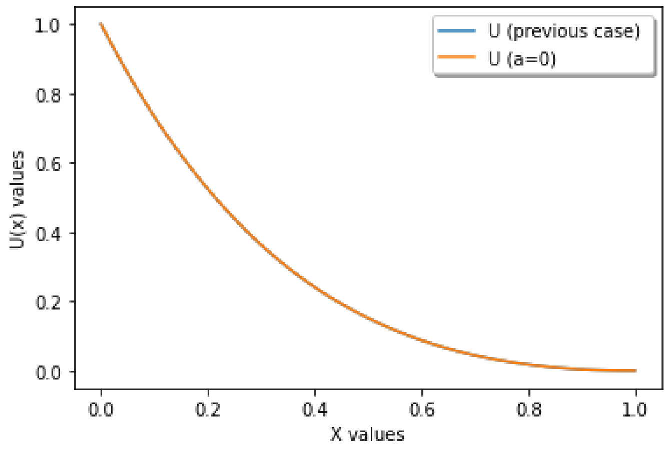


Figure 7 - Explicit scheme with a = 0

This is an expected outcome, because the analytical solution of the initial scenario also satisfies the boundary condition $u'(1) = 0$. Using the analytical solution.

$$u'(x) = -e^{-x}(x-3)(x-1) \qquad \Longrightarrow \qquad \mathrm{u}'(1) = 0$$

However, notice that if we approximate the boundary condition in $x = 1$ using backward differences for the derivative, we can have a or b equal to zero (see the equation before the last one in Page 9), because we are not dividing by zero.

Alternative PDE Equation

We will now investigate whether it's better to consider an alternative PDE, where the rule of the derivative of the product is applied to the first term of the equation, and using central finite differences. Let's first recover the matrix M, for the original PDE, the defined the linear system of equations.

$$u_{i+1} * \left(\frac{\beta_{i+1/2}}{h^2}\right) + u_i * \left(-\frac{\beta_{i+1/2}}{h^2} - \frac{\beta_{i-1/2}}{h^2} - \gamma_i\right) + u_{i-1} * \left(\frac{\beta_{i-1/2}}{h^2}\right) = f_i \Leftrightarrow$$

$$\Leftrightarrow MU^n = F\text{, where}$$

$$M = \begin{bmatrix} \left(-\frac{\beta_{1+1/2}}{h^2} - \frac{\beta_{1-1/2}}{h^2} - \gamma_1\right) & \left(\frac{\beta_{1+1/2}}{h^2}\right) & 0 & \dots & 0 \\ \left(\frac{\beta_{2-1/2}}{h^2}\right) & \left(-\frac{\beta_{2+1/2}}{h^2} - \frac{\beta_{2-1/2}}{h^2} - \gamma_2\right) & \left(\frac{\beta_{2+1/2}}{h^2}\right) & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \left(\frac{\beta_{n-1-1/2}}{h^2}\right) & \left(-\frac{\beta_{n-1+1/2}}{h^2} - \frac{\beta_{n-1-1/2}}{h^2} - \gamma_{n-1}\right) & \left(\frac{\beta_{n-1+1/2}}{h^2}\right) \\ 0 & \dots & 0 & \left(\frac{\beta_{n-1/2}}{h^2}\right) & \left(-\frac{\beta_{n+1/2}}{h^2} - \frac{\beta_{n-1/2}}{h^2} - \gamma_n\right) \end{bmatrix}$$

This matrix is symmetric, negative defined, weakly diagonal, dominant, and an M-matrix, so it's a non-singular matrix for sure. In consequence, stability of the scheme is guaranteed. Let's now consider the modified PDE, and find the equivalent matrix for a central scheme applied to it.

$$(\beta(x)u_x)_x - \gamma(x)u = f(x) \Leftrightarrow \beta'(x)u_x + \beta(x)u_{xx} - \gamma(x)u = f(x) \longrightarrow$$

$$\longrightarrow \frac{\beta_{i+1} - \beta_{i-1}}{2h} * \frac{u_{i+1} - u_{i-1}}{2h} + \beta_i \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - \gamma_i u_i = f_i \Leftrightarrow$$

$$\Leftrightarrow u_{i+1} * \left(\frac{\beta_i}{h^2} + \frac{\beta_{i+1} - \beta_{i-1}}{4h^2}\right) + u_i * \left(-2\frac{\beta_i}{h^2} + \gamma_i\right) + u_{i-1} * \left(\frac{\beta_i}{h^2} - \frac{\beta_{i+1} - \beta_{i-1}}{4h^2}\right) = f_i \Leftrightarrow$$

$$\Leftrightarrow MU^n = F$$

$$M = \begin{bmatrix} \left(-2\frac{\beta_1}{h^2} + \gamma_1\right) & \left(\frac{\beta_1}{h^2} + \frac{\beta_2 - \beta_0}{4h^2}\right) & 0 & \dots & 0 \\ \left(\frac{\beta_2}{h^2} - \frac{\beta_3 - \beta_1}{4h^2}\right) & \left(-2\frac{\beta_2}{h^2} + \gamma_2\right) & \left(\frac{\beta_2}{h^2} + \frac{\beta_3 - \beta_1}{4h^2}\right) & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \left(\frac{\beta_{n-1}}{h^2} - \frac{\beta_n - \beta_{n-2}}{4h^2}\right) & \left(-2\frac{\beta_{n-1}}{h^2} + \gamma_{n-1}\right) & \left(\frac{\beta_{n-1}}{h^2} + \frac{\beta_n - \beta_{n-2}}{4h^2}\right) \\ 0 & \dots & 0 & \left(\frac{\beta_n}{h^2} - \frac{\beta_{n+1} - \beta_{n-1}}{4h^2}\right) & \left(-2\frac{\beta_n}{h^2} + \gamma_n\right) \end{bmatrix}$$

This matrix is neither symmetric, nor negative defined, nor negative positive definite, nor diagonally dominant. So it's harder to prove the scheme is stable for this PDE. We conclude that it's better to use the original PDE rather than the modified version.

Exercise 3

The presented exercise is a particular case of the obstacle problem, which has the following general equation,

$$G[u] = \varphi(F[u] - f, u - g) = 0 \Leftrightarrow \begin{cases} G[u] = 0 \ on \ \Omega \\ u = h \ on \ \partial\Omega \end{cases}$$

In our case, we have,

$$G[u] = min(-u_{xx} - f, u - g) = 0 \Leftrightarrow \begin{cases} G[u](x) = 0 \ on \ \Omega \\ u(x) = h(x) on \ \partial\Omega \end{cases}$$

$$\text{Where } g(x) = \begin{cases} 0, \ -2 \le x \le -\frac{7}{4} \\ -\left(x + \frac{7}{4}\right)\left(x + \frac{1}{2}\right), -\frac{7}{4} \le x \le -\frac{1}{2} \\ 0, \ -\frac{1}{2} \le x \le 0 \\ -2x^3 + 3x, \ 0 \le x \le \frac{3}{2} \\ 0, \frac{3}{2} \le x \le 2 \end{cases}$$

We will solve it for $f = 0, 10$ and $n = 50, 100, 500, 1000$ interior points. Please consult the notebook *Exercise3.ipynb* for details about the code.

Forward Euler Iterative Method (FEIM)

The first method that we will implement is the FEIM. As the name suggests, the solutions will be obtained with an iterative algorithm, until a stopping criteria is met. More formally,

1. $Counter = 0$
2. $u^{k+1} = u^k - dt \, G[u^k]$
3. $\|u^{k+1} - u^k\| = \varepsilon$
4. If $\varepsilon \ge \ Stopping \ Criteria$, then repeat ($Counter += 1$). Else, stop

FEIM method used to solve the PDE $G[u] + u_t = 0$, instead of $G[u] = 0$. When we apply discretization, we obtain:

$$\frac{u^{k+1} - u^k}{\Delta t} + G[u^k] = 0 \Rightarrow u_j^{k+1} = u_j^k - \Delta t \min(-\frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} - f, u_j^k - g_j)$$

Where Δt should satisfy $\Delta t \le \frac{h^2}{2}$. In fact, since we have a non-linear equation to solve, we cannot solve this system using a solver like the one already presented above, so there is no other option but to use an iterative procedure. In table 2, we can find the number of iterations needed for stopping criteria $\epsilon = 10^{-12}$ to be met, when we use an initial solution that coincides with the obstacle.

Table 2 Forward Euler Iterative Method results

	50	100	500	1000
$f = 0$	74 600	557 000	62 798 500	477 175 000
$f = 10$	1700	8500	1 104 000	8 292 000

The number of iterations until the stopping criterion is met is quite large, especially for a larger number of interior points. Nevertheless, the images from Fig. 8 show that the solution is larger or equal to the obstacle. In particular, for the larger f, the number of iterations is lower because the force f that "pushes" u towards the object is more intense, so we can expect that the final solution will be closer to the object g that coincides with the initial solution.

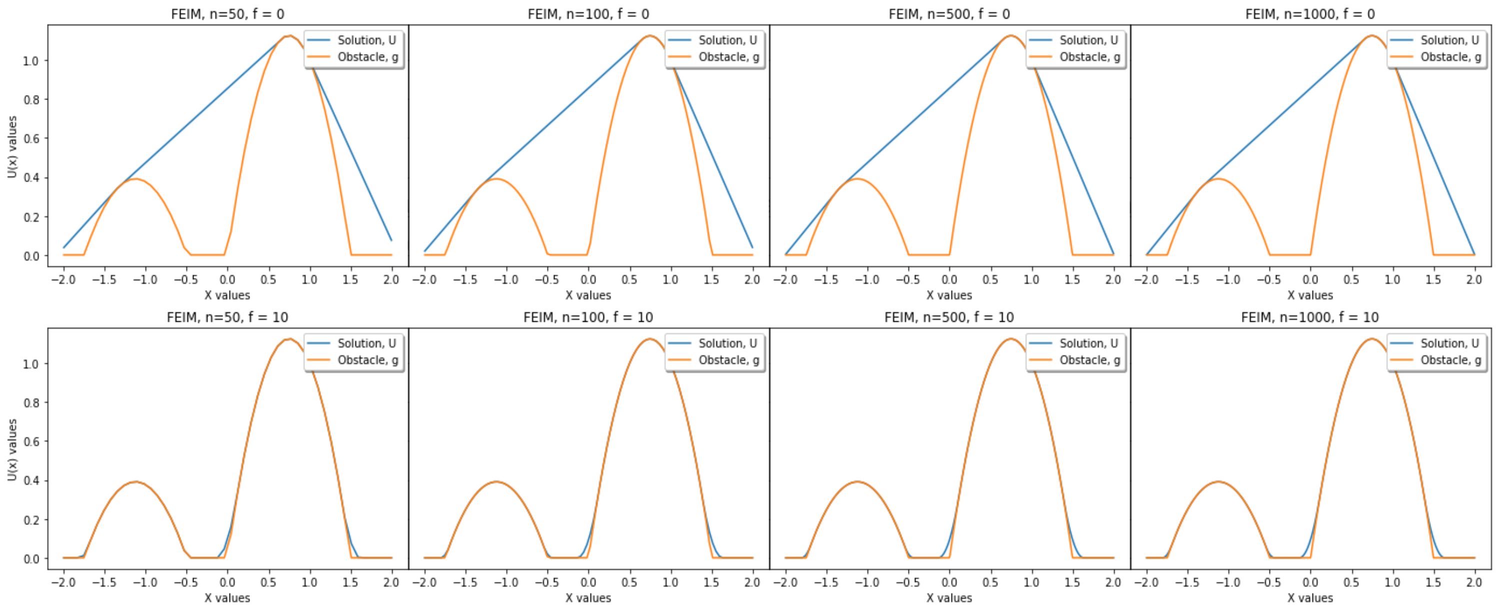


Figure 8 - Numerical solutions for the Obstacle Problem using FEIM

We can see that we can find a good solution for a reduced number interior points using this criteria, so it is not a great advantage to increase a lot the number of interior points here, since it increases exponentially the number of iterations required.

Semi-Smooth Newton Method (SSNM)

We will now proceed to implement another algorithm to solve the PDE, which is the SSNM. Let's first recover the equation:

$$G[u] = \varphi(F[u] - f, u - g) = 0$$

To implement the method, it is necessary to implement the generalized gradient of G, which will, of course, depend on the function g. Several regularization functions $\varphi(a, b)$ can be used; for example, we can use the Fischer Burmeister, φ_{FB}, or the minimum function, φ_{min}. For a function $\varphi(a, b) = \min(a, b)$,

$$\partial\, min(a,b) = \begin{cases} (1,0), & a < b \\ (0,1), & a > b \\ \lambda(1,0) + (1-\lambda)(0,1), & a = b, \qquad \lambda \,\epsilon [0,1] \end{cases}$$

We can see that the function $min(a,b)$ is not differentiable only along the line $a = b$, which is a set that has measure zero.

The next step is to find an algorithm and apply it until we obtain a solution. For this method, we will use the semi smooth Newton's algorithm (which is similar to Newton's method used to solve ODEs).

1. $Counter = 0$
2. Solve for d_k: $H(x_k)d_k = -G(x_k)$, where $H(x_k)$ is the generalized Jacobian $\in \partial\, min(a,b)$
3. Compute $x_{k+1} = x_k + d_k$, $(counter {+}{=}\, 1)$
4. $\|x_{k+1} - x_k\| = \varepsilon$
5. If $\varepsilon \geq\ Stopping\ Criteria$, then repeat. Else, stop

Now we need to re-write the equation using finite differences:

$$G^h[u] = \varphi\big(a(u), b(u)\big) = \min(-D_{xx}u - f, u - g)$$

By the chain rule,

$$\partial G^h[u] = D_a \nabla_u \left(-D_{xx}u - \frac{h^2}{2} f\right) + D_b \nabla_u (u - g) = -D_a(D_{xx}) + D_b I$$

Where I is the identity and the terms D_a and D_b are the following derivatives:

$$\partial\varphi(a,b) = (D_a, D_b) = \left(\frac{\partial}{\partial a}\varphi(a,b), \frac{\partial}{\partial b}\varphi(a,b)\right) = \begin{cases} (1,0), & a < b \\ (0,1), & a > b \end{cases}$$

So the update rule is:

$$u^{k+1} = u^k - \big(\partial G^h[u]\big)^{-1} G^h[u^k]$$

It is advised for the reader to inspect the Python implementation to understand better how the method works. Below, in Table 3, we can find the number of iterations required until the stopping criteria is met, for the different parameters, and with an initial solution equal to the obstacle.

Table 3 – Semi-Smooth Newton Method results

	50	**100**	**500**	**1000**
$\boldsymbol{f = 0}$	11	21	103	204
$\boldsymbol{f = 10}$	3	4	18	35

We can verify that this method is much faster than the previous one, since the number of iterations necessary until the stopping criterion is met is much lower.

The plots of the numerical solutions can be found below, in Fig. 9

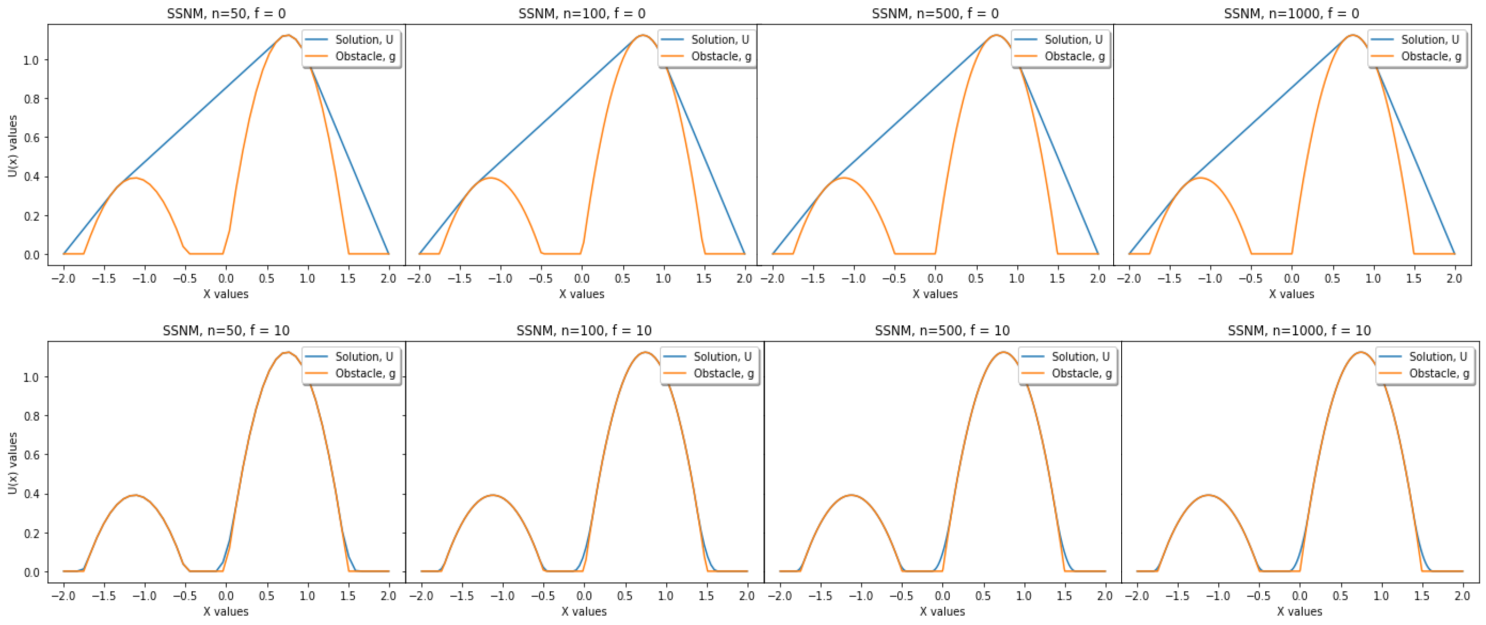


Figure 9- Numerical solutions for the Obstacle Problem using SSNM

Exercise 4

For this particular exercise we will solve the following 2D heat equation, for $t = 1$:

$$\begin{cases} u_t = u_{xx} + u_{tt} + f(x,y,t), \qquad 0 < x < 1, 0 < y < 1, 0 < t \leq 1 \\ u(x,y,0) = u_0(x,y) \\ Dirichlet\ Boundary\ Conditions \end{cases}$$

Knowing that the exact solution is:

$$u(x,y,t) = (1+t)(x^2+y^2)$$

We will first determine the f function, the initial condition, and the Boundary values for every time step (since we have *Dirichlet* Boundary Conditions):

$$f: u_t = u_{xx} + u_{tt} + f(x,y,t) \Leftrightarrow x^2 + y^2 = 2*(1+t) + 2*(1+t) + f(x,y,t) \Leftrightarrow$$

$$\Leftrightarrow f(x,y,t) = x^2 + y^2 - 4*(1+t)$$

$$\text{I.C.: } u(x,y,0) = u_0(x,y) = x^2 + y^2$$

$$\text{B.C.: } \begin{cases} u(0,y,t) = (1+t)(y^2) \\ u(x,0,t) = (1+t)(x^2) \\ u(1,y,t) = (1+t)(1+y^2)' \\ u(x,1,t) = (1+t)(x^2+1) \end{cases}$$

To solve the equation, we will implement the Crank-Nicolson (CN) and ADI schemes. We will also mention some important properties of both schemes when applied to this equation, and make comparisons between the analytical and numerical results we obtained.

In terms of discretization, although we have a 2D space grid, we will consider $h_x = h_y = h$, so the refinement on the x direction will be equal to the refinement on the y direction.

Please consult the notebook *Exercise4.ipynb* for details about the code.

Crank-Nicolson (CN) Scheme

Describing the previous equation using the CN Scheme, we obtain:

$$u_t = u_{xx} + u_{tt} + f(x,y,t) = 0 \rightarrow$$

$$\frac{U^{n+1}_{(x,y)} - U^{n}_{(x,y)}}{\Delta t} = \frac{1}{2}\Big(\frac{U^{n}_{(x+1,y)} - 2U^{n}_{(x,y)} + U^{n}_{(x-1,y)} + U^{n+1}_{(x+1,y)} - 2U^{n+1}_{(x,y)} + U^{n+1}_{(x-1,y)}}{\mathrm{h}^2} +$$

$$+ \frac{U^{n}_{(x,y+1)} - 2U^{n}_{(x,y)} + U^{n}_{(x,y-1)} + U^{n+1}_{(x,y+1)} - 2U^{n+1}_{(x,y)} + U^{n+1}_{(x,y-1)}}{\mathrm{h}^2} + f^{n}_{(x,y)} + f^{n+1}_{(x,y)}\Big)$$

Which is an implicit (in time) scheme.

Taking a closer look at the previous equation, we can see that it's possible to re-write it as:

$$U^{n+1}_{(x,y)}\left(\frac{1}{\Delta t} + \frac{2}{h^2}\right) + U^{n+1}_{(x+1,y)}\left(-\frac{1}{h^2}\right) + U^{n+1}_{(x-1,y)}\left(-\frac{1}{h^2}\right) + U^{n+1}_{(x,y+1)}\left(-\frac{1}{h^2}\right) + U^{n+1}_{(x,y-1)}\left(-\frac{1}{h^2}\right) =$$

$$= U^{n}_{(x,y)}\left(\frac{1}{\Delta t} - \frac{2}{h^2}\right) + U^{n}_{(x+1,y)}\left(\frac{1}{h^2}\right) + U^{n}_{(x-1,y)}\left(\frac{1}{h^2}\right) + U^{n}_{(x,y+1)}\left(\frac{1}{h^2}\right) + U^{n}_{(x,y-1)}\left(\frac{1}{h^2}\right) + \frac{f^{n}_{(x,y)} + f^{n+1}_{(x,y)}}{2} \Leftrightarrow$$

$$\Leftrightarrow U^{n+1}A = U^{n}B + F$$

Once again, we only need to solve a linear system of equations. Like in exercise 2, instead of inverting the matrix A, we will solve the system using LU decomposition.

However, it's important to notice that for some points in the grid, the previous equation is slightly different. To better understand this, let's consider the grid from Fig. 10. As we can see, there are nine different cases. The equation from applies to the points within the light blue shape. As for the points next to the boundary, they have particular equations because of the *Dirichlet* boundary conditions imposed. We will only exemplify one case since for the others the idea is similar.

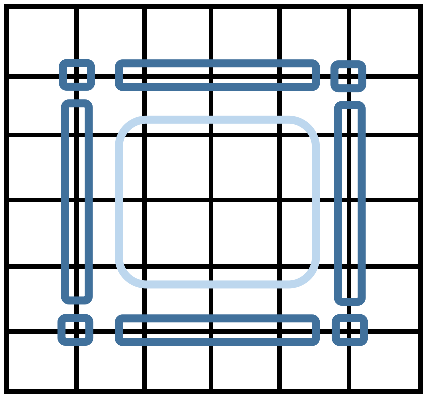


Figure 10 – Exemplifying grid

For the upper left point, when calculating U^{n+1} the values of U^{n} and U^{n+1} at the points above and left are known from the boundary conditions, so the equation is:

$$U^{n+1}_{(x,y)}\left(\frac{1}{\Delta t} + \frac{2}{h^2}\right) + U^{n+1}_{(x+1,y)}\left(-\frac{1}{h^2}\right) + U^{n+1}_{(x,y-1)}\left(-\frac{1}{h^2}\right) =$$

$$= U^n_{(x,y)}\left(\frac{1}{\Delta t}-\frac{2}{h^2}\right)+U^n_{(x+1,y)}\left(\frac{1}{h^2}\right)+U^n_{(x-1,y)}\left(\frac{1}{h^2}\right)+U^n_{(x,y+1)}\left(\frac{1}{h^2}\right)+U^n_{(x,y-1)}\left(\frac{1}{h^2}\right)+\frac{f^n_{(x,y)}+f^{n+1}_{(x,y)}}{2}$$
$$+U^{n+1}_{(x-1,y)}\left(\frac{1}{h^2}\right)+U^{n+1}_{(x,y+1)}\left(\frac{1}{h^2}\right)$$

The two points where the solution is known are the green ones in Fig. 11. In the code implemented in Python, these known terms were inserted in an auxiliary vector on the right-hand side of the equation.

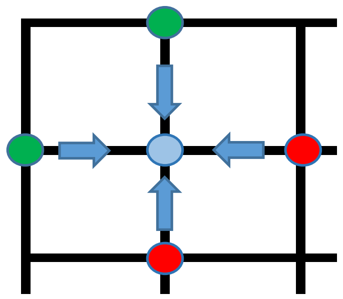


Figure 11- Dirichlet boundary condition example

During the implementation, it was observed that using different discretization, the error in infinity norm between the solution vector and the analytical vector is always zero. This leads us to investigate the truncation error of the scheme. In addition to this, the truncation error depends of this PDE is:

$$\tau = C_1 O(\Delta t^{p_1}) + C_2 O({h_x}^{p_2}) + C_3 O({h_y}^{p_3})$$

Generally speaking, the constants C_1, $C_{2,}$ and C_3 depend on high order derivatives of the function $u(x,y,t)$. But $u(x,y,t)$ is a polynomial of a low order – degree 1 in t and degree 2 in x and y. If we write u_t, u_{xx}, and u_{yy} using finite differences and Taylor's expansion, we will obtain:

$$\tau = -\frac{\Delta t^2}{24}u_{ttt}(x_i,y_j,\xi_1)+\frac{{h_x}^2}{12}u_{xxxx}(\xi_2,y_j,t_n)+\frac{{h_y}^2}{12}u_{yyyy}(x_i,\xi_3,t_n)=0$$

For this function $u(x,y,t)$ the truncation error is, in fact 0. Literature that proves this results was included. Tables 4.1 and 4.2 exhibit the results for different discretization (in space and time, respectively).

Table 4.1 – Error and CPU time of CN scheme for time discretization

$n_x = n_y$	n_t	$\Vert u-\widehat{u_i}\Vert_\infty$	CPU time (ms)
20	5	0	68.1
20	10	0	128
20	20	0	268
20	40	0	419
20	80	0	843

Table 5.2 - Error and CPU time of CN scheme for space discretization

$n_x = n_y$	n_t	$\Vert u-\widehat{u_i}\Vert_\infty$	CPU time (ms)
12	30	0	72.7
24	30	0	716
48	30	0	20 300
96	30	0	1 054 000

As expected, the error is 0 for all refinements. Regarding CPU times, we will analyze them at the same time as the ones from ADI. Below we present the numerical plots for the case with the greatest refinement, where we can see that the numerical solution perfectly converged to the analytical one.

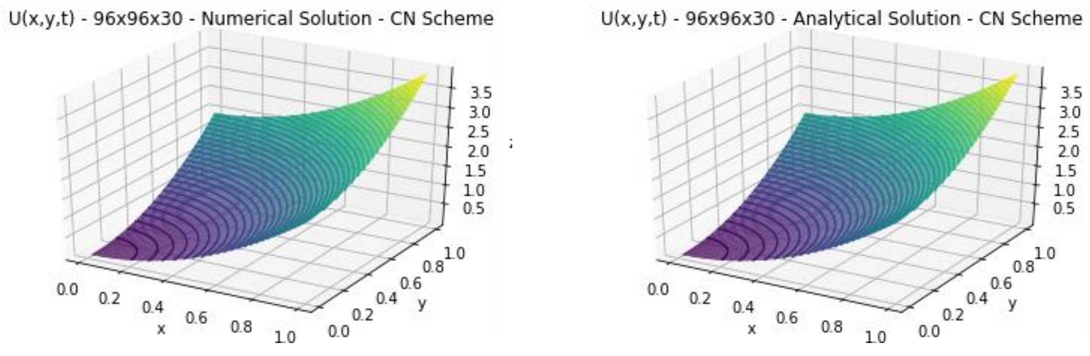


Figure 12 - Numerical results obtained for the CN scheme (greatest refinement)

ADI Scheme

The ADI scheme works differently from the CN scheme. For each time step, the method computes the solution of two systems of linear equations. The first system (also called the predictor) computes the derivatives in one direction (we started with x-direction), from time k to time $k+\frac{1}{2}$, and the second system (also called the corrector) computes the derivatives in the other direction, y, from time $k+\frac{1}{2}$ to time $k+1$. In terms of structure of the matrices, the difference with respect to the CN scheme, is that now there will be zeroes in the positions related to the direction where the derivatives are not being calculated. Since these linear systems are very similar to the one in the CN scheme, we won't formalize them, and they can be found in the Literature provided.

It can be proved that the ADI's truncation error on the Heat equation is $O(\Delta t^2)$, which is something that can be found in the literature provided as well. We are expecting that when doubling the time refinement, our error decays by four. In addition to this, we should expect the same error for different space discretization, since the truncation error does not depend on h.

We can find the numerical results obtained, in tables 5.1 and 5.2

Table 5.1 – Time refinement for the ADI scheme

$n_x = n_y$	n_t	$\|u-\widehat{u_i}\|_\infty$	**CPU time (ms)**
20	5	1.2898e-2	97.9
20	10	3.7978e-3	161
20	20	1.0434e-3	301
20	40	2.7373e-4	568
20	80	7.0134e-5	1140

$$\Rightarrow \boxed{\frac{\|u-\widehat{u_i}\|_\infty}{\|u-\widehat{u_{i+1}}\|_\infty} \approx 4}$$

Table 5.2 – Space refinement for the ADI scheme

$n_x = n_y$	n_t	$\|u-\widehat{u_i}\|_\infty$	**CPU time (ms)**
12	30	4.6899e-4	197
24	30	4.8548e-4	979
48	30	5.0257e-4	40 200
96	30	5.1136e-4	2 100 000

$$\Rightarrow \boxed{\frac{\|u-\widehat{u_i}\|_\infty}{\|u-\widehat{u_{i+1}}\|_\infty} \approx 1}$$

As the theoretical results showed, in fact, having a more refined grid in time helps reducing the error, and when the number of points in time doubles, the error drops by four. At the same time, if we keep the time refinement the same and just double the number of points in space, the error is approximately the same, which goes on agreement with the truncation error not depending on the space. Also, the CPU time increases a lot with a more refined grid in space, whereas in time the effect is smaller. Comparing them with the CPU time of the CN scheme, ADI takes more time to compute. There is no doubt that the CN scheme is better than the ADI, for this particular function. It takes less CPU time and the error is always 0.

Below (Fig. 13) we present the numerical and analytical solution for the greatest refinement found in the table above, where it may be seen that the numerical solution perfectly converged to the analytical one.

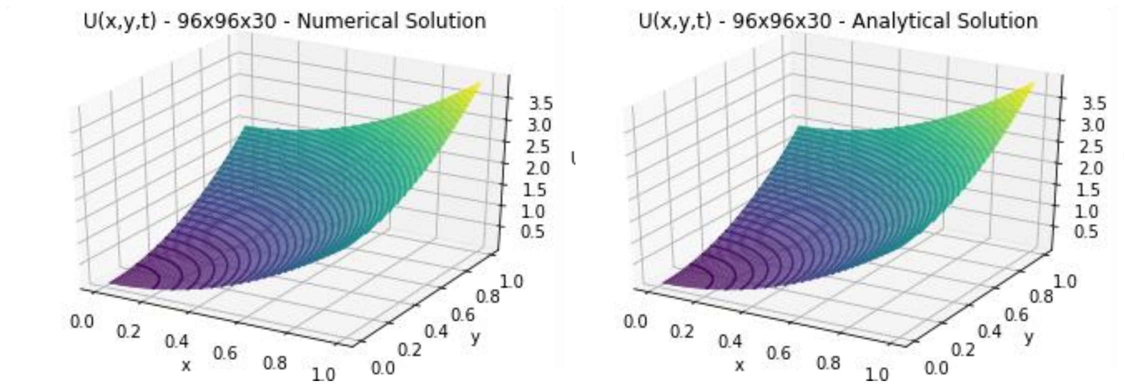


Figure 13 – Numerical results obtained for the ADI scheme (greatest refinement)

Exercise 5

European Vs. American Options

European and American Options are both two very similar financial products acquired by traders and investors, for several purposes. They are characterized as a contract between two identities, where they agree to buy or sell an asset at a predetermined (strike) price in the future. These can be a call or put option, depending on the value the purchaser forecasts; call options allow the holder to buy the asset (bullish sentiment), while put options permit to sell (bearish sentiment).

Nevertheless, these two products have a particular difference. While the European options can only be exercised on the expiration date, the American options can be exercised at any time before the expiration date. Naturally, since they can be exercised when the price moves in favor of the trader, this comes with a premium that they have to pay. So to make a profit with American options, traders and investors need the asset to move far enough from the strike price to make a profit.

There are also other differences. For example, while most stocks and ETFs have American options, equity Indices have European options. Other differences reside in the settlement price and the day the option stops trading, but these are less relevant. Appropriate literature will be referenced regarding these differences, but the most important one is the first one mentioned above.

Analytical Solution for European Call Options

Let's now consider a call option with the following parameters (see Table 6).

Table 6 – Parameters of the Call Option

Strike Price, $\boldsymbol{K}$	Risk-Free rate, $\boldsymbol{r}$	Volatility, $\boldsymbol{\sigma}$	Expiration Time, $\boldsymbol{T}$
100	5%	20%	1

The solution to the famous Black-Scholes equation gives us the formula to calculate the value of a call or put options. However, deriving the solution is not a trivial task. For the sake of simplicity (and since the proof is well-known and can be found in different places) we won't present the deduction, but we will leave useful Literature in case the reader wants to see the proof. We now present Black-Scholes equation and the boundary conditions for a call option, and the expression to evaluate it:

$$B.S.eq.: \frac{\partial V(S,t)}{\partial t} + rS\frac{\partial V(S,t)}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V(S,t)}{\partial S^2} - rV(S,t) = 0$$

$$B.C.: \begin{cases} V(S=0,t) = 0 \\ V(S,t) = S - Ke^{-r(T-t)},\ \ if\ S \to \infty \\ V(S,t=T) = \max(S-K,0) \end{cases}$$

$$Solution: V(S,t) = S * N(d_1) - K * e^{-r(T-t)} * N(\mathrm{d}_2)$$

Where $d_1 = \frac{\log\left(\frac{S}{K}\right)+\mathrm{T}\left(\mathrm{r}+\frac{\sigma^2}{2}\right)}{\sigma\sqrt{T}}, d_2 = \frac{\log\left(\frac{S}{K}\right)+\mathrm{T}\left(\mathrm{r}-\frac{\sigma^2}{2}\right)}{\sigma\sqrt{T}}$ and $N(x)$is the cumulative normal distribution

Plugging in the known parameters, we can obtain the value (see table 7.1) for different values $\boldsymbol{S}$. Let's consider $S\ \epsilon\{\mathbf{80}, \mathbf{85}, \dots, \mathbf{140}\}$.

Table 7 – Values of the call option for different S values

Price, S	Value, $\boldsymbol{V(S,0)}$
80	1.85942
85	3.21360
90	5.09122
95	7.51087
100	10.45058
105	13.85791

Price, S	Value, $\boldsymbol{V(S,0)}$
110	17.66295
115	21.79051
120	26.16904
125	30.73604
130	35.44027
135	40.24176
140	45.11061

Explicit Scheme for European Call Options

We will proceed to approximate the Black-Scholes equation by finite differences, using an explicit scheme:

$$\frac{V_{i,j} - V_{i-1,j}}{\Delta t} + r(j\Delta s)\frac{V_{i,j+1} - V_{i,j-1}}{2\Delta s} + \frac{1}{2}\sigma^2(j\Delta s)^2\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta s^2} - rV_{i,j} = 0$$

Now we can re-write the equation as,

$$V_{i-1,j} = \frac{1}{2}\Delta t(\sigma^2 j^2 - rj)V_{i,j-1} + [1 - \Delta t(\sigma^2 j^2 + r)]V_{i,J} + \left[\frac{\Delta t}{2}(\sigma^2 j^2 + rj)\right]V_{i,j+1}$$

Starting at $T = 1$, we can solve a system of equations backward (for the previous time-step), which will allow us to calculate the value V at all time steps, but in particular for $t = 0$. As exemplified in Fig. 14, since we know all the points identified with a green circle, using the previous point it is easy to find all the interior points of the grid.

We will now present the results for a $\Delta t = 0.0001$ and $\Delta s = 1$, and compare them with the analytical solution we presented above (see Table 8).

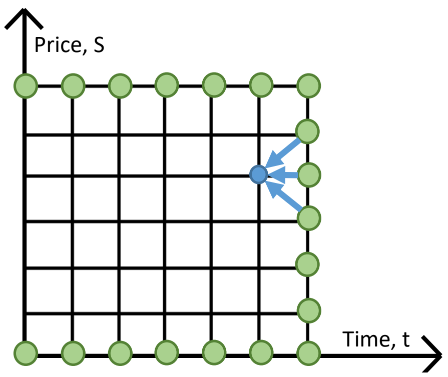


Figure 14 - Explicit Scheme representation

Table 8 - Explicit Scheme results for Call Options

Price	Analytical Solution	Numerical Solution	Accuracy (%)
80	1.85942	1.85849	99.95
85	3.21360	3.21201	99.95
90	5.09122	5.08911	99.96
95	7.51087	7.50850	99.97
100	10.45058	10.44822	99.98
105	13.85791	13.85576	99.98
110	17.66295	17.66116	99.99
115	21.79051	21.78912	99.99
120	26.16904	26.16803	100.00
125	30.73604	30.73535	100.00
130	35.44027	35.43982	100.00
135	40.24176	40.24150	100.00
140	45.11061	45.11047	100.00

The accuracies are almost perfect for the explicit scheme. Below, in Fig. 15, we present the solution for three specific times, and we can see that at time $t = 1$ the option value is equal to its payoff, like it should. In Fig. 16 we can visualize the global solution on a 3D plot.

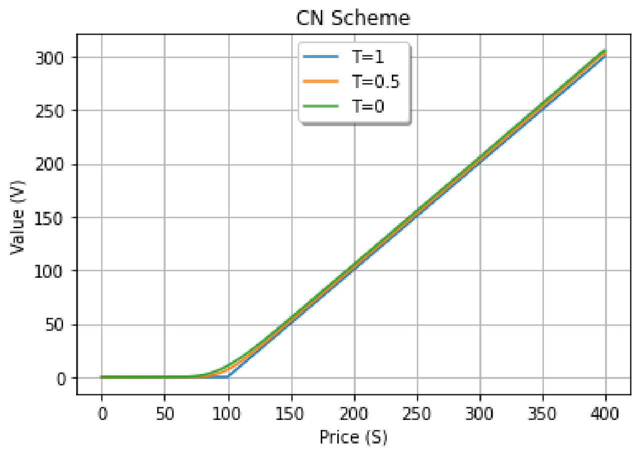


Figure 15 – Slice plot for t=0, t=0.5 and t=1 for Explicit scheme

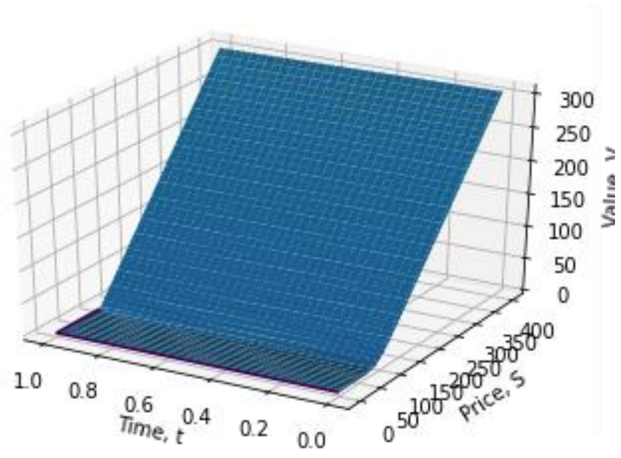


Figure 16 - Global Solution for Explicit scheme

Crank Nicolson Scheme for European Call Options

In case we use the Crank Nicolson, then the finite difference scheme we will obtain is,

$$\left[-\frac{\Delta t}{4}(\sigma^2 j^2 - rj)\right]V_{i-1,j-1} + \left[1-\left(-\frac{\Delta t}{2}(\sigma^2 j^2 + r)\right)\right]V_{i-1,j} - \left[\frac{\Delta t}{4}(\sigma^2 j^2 + rj)\right]V_{i-1,j+1} =$$

$$= \left[\frac{\Delta t}{4}(\sigma^2 j^2 - rj)\right]V_{i,j-1} + \left[1+\left(-\frac{\Delta t}{2}(\sigma^2 j^2 + r)\right)\right]V_{i,j+}\left[\frac{\Delta t}{4}(\sigma^2 j^2 + rj)\right]V_{i,j+1} \Leftrightarrow$$

$$\Leftrightarrow a_j V_{i-1,j-1} + (1-b_j)V_{i-1,j} - c_j V_{i-1,j+1} = a_j V_{i,j-1} + (1+b_j)V_{i,j} + c_j V_{i,j+1}$$

This will lead to a linear system of equations,

$$AV^{i-1} = BV^i$$

Where,

$$A = \begin{bmatrix} 1-b_1 & -c_1 & 0 & \cdots & 0 \\ -a_2 & 1-b_2 & -c_2 & \dots & 0 \\ 0 & -a_3 & 1-b_3 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -c_{n-2} \\ 0 & 0 & \cdots & -a_{n-1} & 1-b_{n-1} \end{bmatrix} \qquad B = \begin{bmatrix} 1+b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & 1+b_2 & c_2 & \dots & 0 \\ 0 & a_3 & 1+b_3 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & c_{n-2} \\ 0 & 0 & \cdots & a_{n-1} & 1+b_{n-1} \end{bmatrix}$$

And V^{i-1} and $V^i$are, respectively, the vectors with the values of the options' values of the current time-step we are determining and the previous time-step already determined. The rationale is the same as the previous scheme but now we have two additional terms – the right and left values from the current time-step, marked as yellow. The image in Fig. 17 illustrates this fact. We now present the results for a $\Delta t = 0.01$ and $\Delta s = 0.5$, and compare them with the analytical solution we presented above (Table 9).

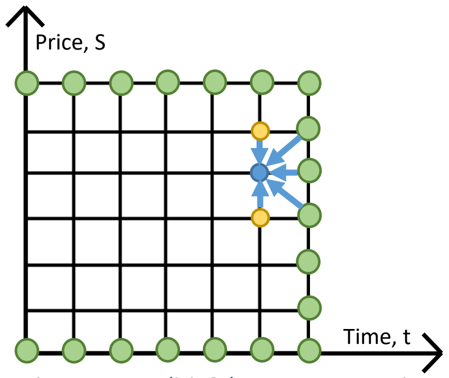


Figure 15 - Explicit Scheme representation

Table 9 - CN results for Call Options

Price	Analytical Solution	Numerical Solution	Accuracy (%)
80	1.85942	1.85918	99.99
85	3.21360	3.21319	99.99
90	5.09122	5.09069	99.99
95	7.51087	7.51028	99.99
100	10.45058	10.44999	99.99
105	13.85791	13.85737	100.00
110	17.66295	17.66250	100.00
115	21.79051	21.79015	100.00
120	26.16904	26.16878	100.00
125	30.73604	30.73586	100.00
130	35.44027	35.44015	100.00
135	40.24176	40.24169	100.00
140	45.11061	45.11057	100.00

In general, the numerical solutions for the CN scheme are better because the accuracies are a little higher than the ones obtained for the explicit scheme, even with a smaller time refinement. However, the increase in accuracy is negligible, so in terms of accuracy they are equal. In order to decide which was the best method, we compare CPU times, and for the Explicit scheme the CPU time was 10.9 seconds, while for the CN it was 4.63 seconds. So it seems that the CN is preferable because it is faster and allows to use a larger time increment. The equivalent slice plot and global solution plot for the CN scheme can be found below, in Fig. 16 and 17.

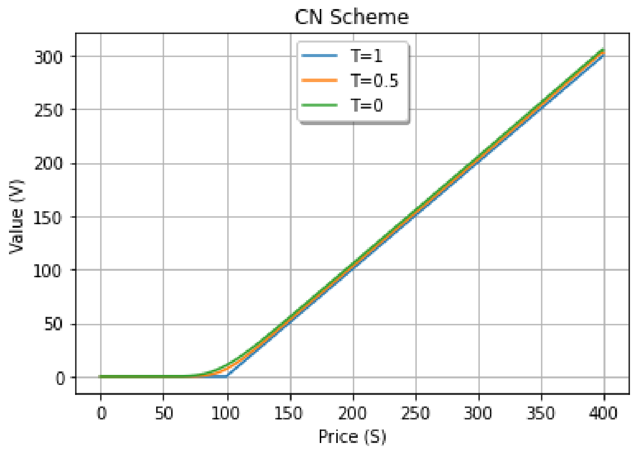


Figure 16 - Slice plot for t=0, t=0.5 and t=1 for CN scheme

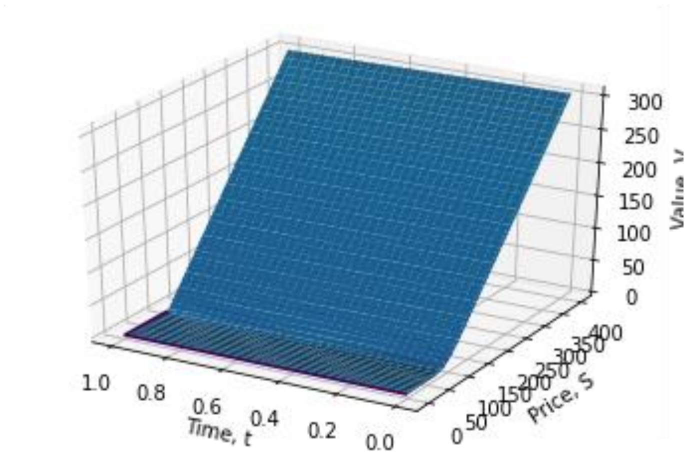


Figure 17 – Global solution for CN scheme

American Call Options

For the American call options, we have already seen that they are different from the European call options. If the call option value is lower than the payoff then we have arbitrage, and since that can't happen, at that specific price and time, the call option value is given by the payoff at expiration.

For the Explicit scheme, that's exactly what we are going to do. After we compute the values for all the prices in a particular time, we will then replace the value by the value at payoff if and only if these are greater. This arbitrage case is never a question in calls that don't pay dividends (American call options with no dividend yield have the same value as European call options), so we will consider the case of a call on a dividend-paying stock.

The possibility of exercising American call options early makes options pricing a very hard task. In particular, there is no closed-form solution for American call options like the one we have seen for European call options.

Let's recall the Black-Scholes equation for call options, but this time with a dividend yield, d.

$$\frac{\partial V(S,t)}{\partial t} + (r-d)S\frac{\partial V(S,t)}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V(S,t)}{\partial S^2} - rV(S,t) = 0$$

The explicit scheme for this PDE is:

$$V_{i-1,j} = \frac{1}{2}\Delta t(\sigma^2 j^2 - (r-d)j)V_{i,j-1} + [1 - \Delta t(\sigma^2 j^2 + r)]V_{i,J} + \left[\frac{\Delta t}{2}(\sigma^2 j^2 + (r-d)j)\right]V_{i,j+1}$$

The results for a European and American call option with dividends $d = 0.05$ can be found below in Figs. 18 and 19. The same parameters as above were used, except Δs which was increased to 20.

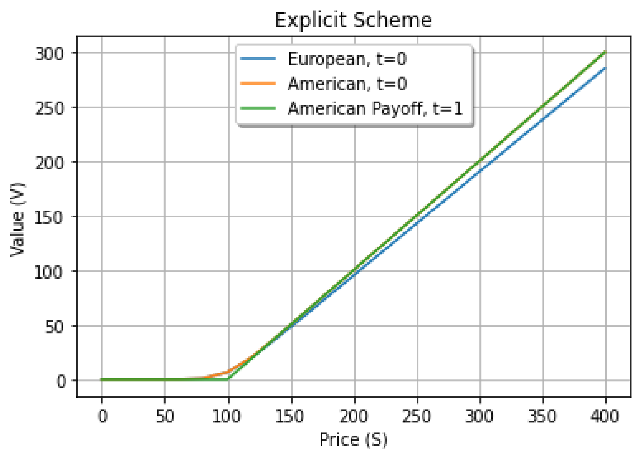


Figure 17 - Slice plot: t=0 for European , t=0 for American and t=1 for American (Explicit scheme)

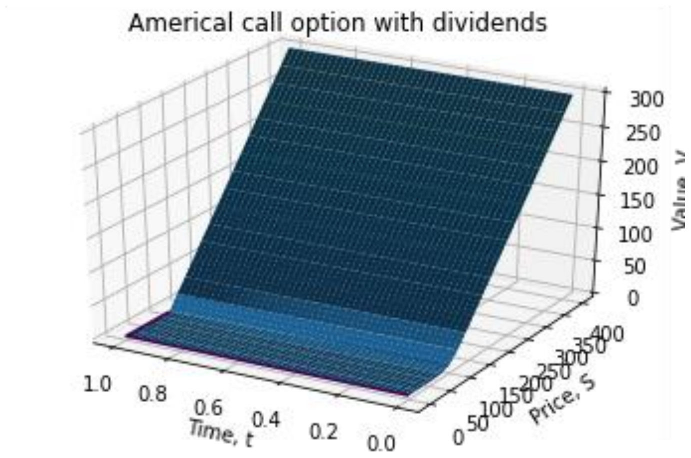


Figure 18 – Global Solution for American call option with dividends

Final Remarks

Making this project was a great opportunity to develop knowledge in finite difference schemes, and their application in famous PDEs. It was really interesting to study the theory behind the various schemes and, their advantages or disadvantages in specific PDEs. Putting these schemes into practice was a way to verify these theoretical results and how they reveal themselves in practical examples. One of the most interesting facts was that small changes in the way we define the schemes may help, for example, in having a truncation error of higher order which ultimately leads to a faster convergence and better results.

Bibliography

- Strikwerda, J., 2004. Finite difference schemes and partial differential equations. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics.

- Li, Z., Tang, T. and Qiao, Z., 2018. Numerical Solution of Differential Equations. 1st ed. Cambridge University Press.

- Reti, D., 2021. Option pricing using the Black-Scholes model, without the formula. [online] Medium. Available at: <https://towardsdatascience.com/option-pricing-using-the-black-scholes-model-without-the-formula-e5c002771e2f> [Accessed 5 May 2022].

- Carrington, R., 2017. Speed Comparison of Solution Methods for the Obstacle Problem. Master of Science Thesis. McGill University, Montreal.

- Fernandes, M., 2009. Finite Differences Schemes for Pricing of European and American Options. [online] Lisbon. Available at: <https://docplayer.net/15563956-Finite-differences-schemes-for-pricing-of-european-and-american-options.html> [Accessed 5 May 2022].

- Uddin, M., Ahmed, M. and Bhowmilk, S., 2014. A Note on Numerical Solution of a Linear Black-Scholes Model. GANIT: Journal of Bangladesh Mathematical Society, 33, pp.103-115.

- Prasad, S., 2017. Truncation Error Crank Nicolson Method. [video] Available at: <https://www.youtube.com/watch?v=3PnWx0RCgDo> [Accessed 6 May 2022].