Numerical Solution of the Heat Conductance Equation

Vladislav Yotkov

1 Introduction

The Heat Equation (1) is a parabolic partial differential equation (PDE) which describes the conductance of temperature through a particular body over time. In this particular work, we will explore its solution along an aluminium bar over 600 seconds using two numerical methods - Explicit and Implicit (Crank and Nicolson [1947]). Our analysis and results will be conducted (Sections 3 & 4) against the Analytical solution (Eq.2). To model the temperature distribution, we will treat the bar as a two-dimensional spatio-temporal grid as shown on Figure 13 with parameters defined in Table 1.

$$\frac{\partial^2 T(x,t)}{\partial x^2} = \frac{1}{k}\frac{\partial T(x,t)}{\partial t} \tag{1}$$

Bar characteristics	$L = 100$ cm and $k = 0.835$ cm^{2}s^{-1}
Step-sizes	$\Delta x = 20$ cm and $\Delta t = 100$ s
Boundary conditions	$T(0,t) = 0°\text{C}$ $T(L,t) = 0°\text{C}$
Initial conditions	$T(x,0) = 500°\text{C}$

Figure 1: Specification of parameters for heat conductance

2 Analytical Solution of Heat Conductance Equation

As the analytical solution (Eq.2) represents an approximation over a series, we need to tune the hyper-parameter N to achieve high accuracy, but also computational efficiency. We do so by simulating the solution over all possible $N > 0$ until the temperature difference $|T_N - T_{N-1}| \leq \epsilon$, where ϵ is the machine epsilon ($\approx 2.22e^{-16}$) and step sizes are $\Delta x = 20cm,\ \Delta t = 100s$. We find this holds for $N = 195$ as shown on Fig.2[1].

$$T(x,t) = \frac{2000}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}\sin\left(\frac{n\pi x}{L}\right)e^{-\frac{n^2\pi^2 k}{L^2}t} \tag{2}$$

for $n \in 1, 3, 5, \ldots, N$

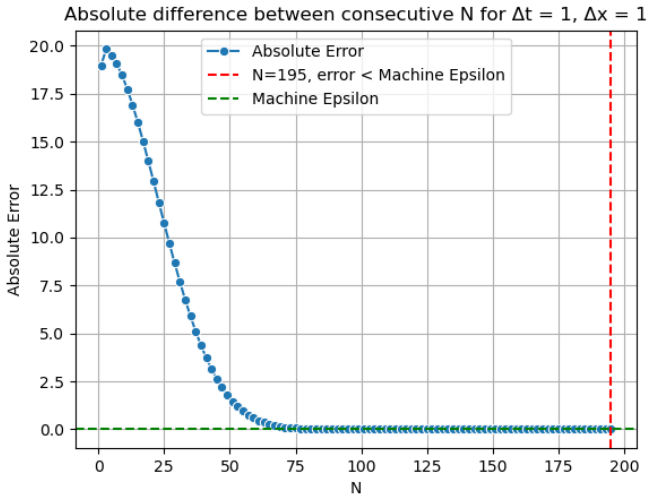


Figure 2: Tuning N parameter of the Analytical solution

3 Explicit Solution of Heat Conductance Equation

3.1 Background

The Explicit methods are used for solving ordinary and partial differential equations - ODEs and PDEs, respectively. These are, for example, Euler's method and Heun's method for ODEs (Süli and Mayers [2003]), and Forward Time Centered Space (FTCS) method for parabolic PDEs(Alebraheem [2017]). As the Heat equation (Eq.1) is parabolic, in this work we will employ the FTCS method.

[1]Log-scale is not used because small values are not representable.

The FTCS is a finite-difference method that estimates the solution to the PDE via forward differences in time (first-order accurate) (Eq.3) and central differences in space (second-order accurate) (Eq.4). Thus, the original Heat equation (Eq.1) can be transformed into a finite-difference one (Eq.5).

For ease of visualisation, we provide the computational molecule for Eq.5 at Fig.3. From the stencil[2], it is clear that the Explicit method estimates a future state $T_{i,j+1}$ based on known states $T_{i-1,j}$, $T_{i,j}$, $T_{i,j+1}$, which are either initially defined or eventually estimated by applying the molecule along the bar and through time.

$$\frac{T_{i,j+1}}{\partial t} = \frac{T_{i,j+1} - T_{i,j}}{\Delta t} \tag{3}$$

$$\frac{T_{i,j+1}}{\partial x^2} = \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} \tag{4}$$

$$\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} = \frac{1}{k}\frac{T_{i,j+1} - T_{i,j}}{\Delta t} \tag{5}$$

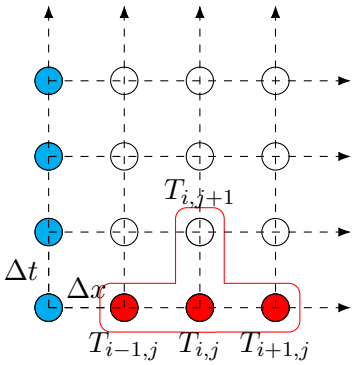


Figure 3: Explicit method's stencil

Furthermore, we can rearrange the expression to solve for $T_{i,j+1}$ and get the following.

$$T_{i,j+1} = \left(\frac{\Delta t \cdot k}{\Delta x^2}\right) \cdot \left(T_{i-1,j} + T_{i,j} \cdot \left(\frac{\Delta x^2}{\Delta t \cdot k} - 2\right) + T_{i+1,j}\right) \tag{6}$$

Throughout this report we will use λ to express the constant spatio-temporal term for conciseness as shown in Eq.7. Hence, the original Explicit equation is transformed into Eq.8:

$$\lambda = \frac{\Delta t \cdot k}{\Delta x^2} \tag{7}$$

$$T_{i,j+1} = \lambda \cdot \left(T_{i-1,j} + T_{i,j} \cdot \left(\frac{1}{\lambda} - 2\right) + T_{i+1,j}\right) \tag{8}$$

While the FTCS is easily-implementable (Snippet 4) due to its formulation (Eq.5) and update scheme (*i.e.*, an unknown state is approximated via known ones) as visible on the computational molecule (Stencil 3), there are a few limitations.

1. FTCS is conditionally stable (Burden and Faires [1989]) for the Heat equation and as such only specific step size configurations produce stable solutions (*i.e.*, result in negligible accumulated rounding errors as defined by Crank [1979])

2. Additionally, to reduce the numerical error the step sizes need to be shrunk, which in turn leads to higher computational costs and makes the solution impractical.

To address the first issue, we could refer to LeVeque [2007] or Burden and Faires [1989] where *von Neumann* analysis is used to derive the stability relationship between Δt, Δx, and the k factor provided below:

$$\lambda = \frac{\Delta t \cdot k}{\Delta x^2} \leq \frac{1}{2} \tag{9}$$

As for the second limitation, which is one of computational intensity, we could implement an Implicit method for a lower numerical error and a more stable PDE solution (Section 3).

3.2 Implementation

While the Python implementation is provided in Fig.4, we emphasize the key moments in constructing the finite-difference solution with FTCS.

[2]We will follow this colouring convention: 1. Red nodes - initial states, 2. Cyan nodes - boundary conditions, 3. White nodes - unknown states

1. Setting the boundary conditions $T(0,t)$, $T(L,t)$ to 0°C, and the initial values along the length of the bar $T(x,0) = 500$°C.

2. Apply the computational molecule (Fig.3) along the grid and through time using pre-defined/estimated states.

```
time_range = range(0, time_limit + step_time, step_time)
space_range = range(0, L + step_space, step_space)
# Initialize the computational matrix with NaN values.
cm = np.array([[np.nan for _ in space_range] for _ in time_range])
# Set the initial boundary conditions for the computational matrix.
cm[0, :] = 500
cm[:, 0] = 0
cm[:, -1] = 0
# Pre-calculate lambda terms
lambda_ = step_time * k / step_space**2
lmd_ctr = -2.0 + 1.0 / lambda_
if 0 < lambda_ <= 0.5:
    print('Explicit solution is STABLE')
else:
    print('Explicit solution is UNSTABLE')
for j, t in enumerate(time_range[:-1]):
    for i, x in enumerate(space_range[:-1]):
        if np.isnan(cm[j+1][i]):
            ftcs_trm = cm[j][i-1] + cm[j][i] * lmd_ctr + cm[j][i+1]
            cm[j+1][i] = lambda_ * ftcs_trm
```

Figure 4: Python Snippet: Explicit method for Heat Equation

3.3 Experiments

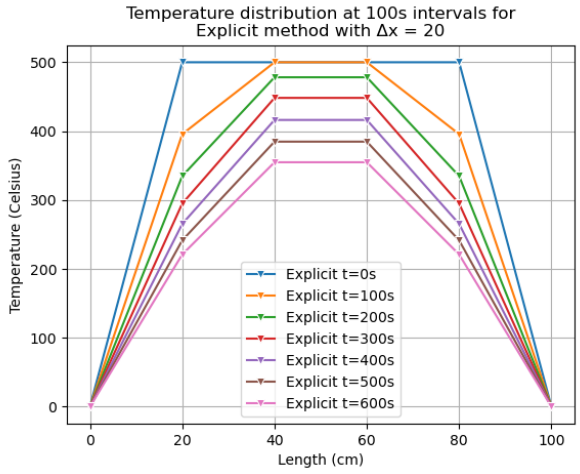
We conducted experiments between the Explicit and the Analytical method to demonstrate the limitations of the FTCS finite-difference approach. As shown on Fig.5a and Fig.5b, the Explicit method for the configuration $\Delta x = 20$, $\Delta t = 100$ is *stable* in terms of temperature distribution along the bar and temperature evolution through time, although it is clear that the solution is not tracking perfectly the analytical one (relative error at $t = 100s$ is around 10%, see Fig.11). Out of the experiments, we achieve the closest approximation to the analytical solution at $\Delta x = 20$, $\Delta t = 50$. This is as expected because 1. we reduce at least one of the step sizes (the temporal one here) and 2. the λ expression (Eq.9) is true . However, once we use the configuration $\Delta x = 10$, $\Delta t = 100$ (Fig.6a, Table 6b), this is no longer the case as $\lambda = 0.84 > \frac{1}{2}$. The results are significant oscillations and instability propagating and accumulating through time (the final temperature approximation here is nonsensical with an absolute error of $1,765$°C!). These experiments reaffirm our theoretical understanding of the FTCS method and once again remind us of the *inherent stability issue*.

4 Implicit Solution of Heat Conductance Equation

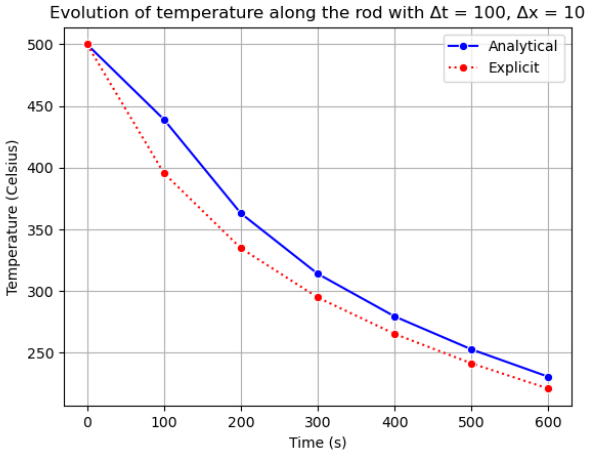
4.1 Background

While the Explicit method is considerably easy to implement, we have outlined and demonstrated that it can be *unstable* and *inaccurate* (even when it is stable). To address these shortcomings, we will explore the implicit Crank-Nicolson (C-N) method (Crank and Nicolson [1947]) and its performance.

Firstly, the C-N is *second-order accurate* for both space and time (unlike for the Explicit one) and it does so by *incorporating the midpoints* (LeVeque [2007]). While the temporal discretisation remains unchanged (Eq.3), the spatial one is redefined from the perspective of the midpoints (Fig.10). Therefore, the Heat problem from Eq.1 can be expressed as Eq.11. Evidently, the method is *centered in both space and time* which is easily observed from the computation molecule (Fig.12).

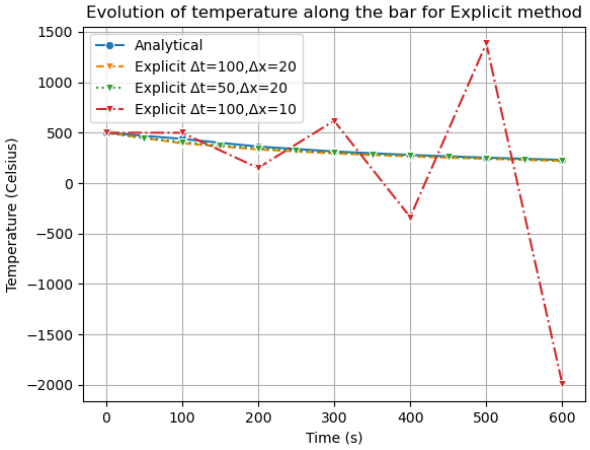


(a) Temperature distribution along the bar for Explicit method



(b) Analytical vs. Explicit method: temperature evolution

Figure 5: FTCS: Temperature distribution along the bar and temperature evolution through time



(a) Analytical vs. Explicit method: temperature evolution

Method	Δt	Δx	Temperature (Celsius)	λ
Analytical	100	20	230.57688	-
Explicit	100	20	220.962066	0.21
Explicit	50	20	225.046963	0.10
Explicit	100	10	-1995.656788	0.84

(b) Comparison of temperature per method configuration

Figure 6: Temperature evolution and comparison for Analytical & Explicit methods

$$\frac{T_{i,j+1}}{\partial x^2} = \frac{1}{2}\left(\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i-1,j+1} - 2T_{i,j+1} + T_{i+1,j+1}}{\Delta x^2}\right) \tag{10}$$

$$\frac{1}{2}\left(\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i-1,j+1} - 2T_{i,j+1} + T_{i+1,j+1}}{\Delta x^2}\right) = \frac{1}{k}\frac{T_{i,j+1} - T_{i,j}}{\Delta t} \tag{11}$$

To facilitate readability we will substitute Eq.7 inside Eq.11 to arrive at:

$$-\lambda T_{i-1,j+1} + (2 + 2\lambda)T_{i,j+1} - \lambda T_{i+1,j+1} = \lambda T_{i-1,j} + (2 - 2\lambda)T_{i,j} + \lambda T_{i+1,j} \tag{12}$$

What makes this implicit method challenging, though, is that $T_{i-1,j+1}$, $T_{i,j+1}$, $T_{i+1,j+1}$ are unknown states which are part of *a system of algebraic equations*. To build this system we apply the stencil (Fig.12) along the bar length and across the time, which can be conveniently expressed as the matrix-vector product $Ax = b$ (Eq.12), where a is a constant matrix, and b is calculated per Δt from initial or already estimated states.

$$
(13)\qquad \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \cdots & 0 \\ 0 & -\lambda & 1+2\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} T_{1,j+1} \\ T_{2,j+1} \\ T_{3,j+1} \\ \vdots \\ T_{n-1,j+1} \\ T_{n,j+1} \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \lambda & 1-2\lambda & \lambda & 0 & \cdots & 0 \\ 0 & \lambda & 1-2\lambda & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda & 1-2\lambda & \lambda \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_{1,j} \\ T_{2,j} \\ T_{3,j} \\ \vdots \\ T_{n-1,j} \\ T_{n,j} \end{bmatrix}}_{b}
$$

However, we must note that *second-order accuracy* is not the only benefit to the C-N method as it has been proven to be *unconditionally stable* (LeVeque [2007], Østerby [2003]) for any spatio-temporal configuration. Therefore, unlike the explicit FTCS method, larger steps in space and time do not significantly degrade the accuracy, and while extremely small steps are thus unnecessary, the method is still computationally intensive due to solving *the system of linear equations* (Eq.13) (Lax [2007]).

4.2 Implementation

While the Python implementation is provided in Fig.7, we emphasize the key moments in constructing the finite-difference solution with Crank-Nicolson's method.

1. Introduce constant matrices A, B with λ-terms based on the application of the stencil sliding operation (Fig.12) and the expression rearrangement (Eq.12).

2. Build up the current state-vector from T_j and perform matrix-vector product $b = B \cdot T_j$. T_j must be computed at every iteration because it requires the temperatures from the previous time step. At start, we compute $T_{1 \le x \le n,2}$ via the initial ($T_{1<x<n,1} = 500°\text{C}$) and the boundary conditions ($T_{1,1} = 0°\text{C}$, $T_{n,1} = 0°\text{C}$).

3. Use an existing linear solver (based on Lax [2007]) to solve system of equations (Fig.13) for T_{j+1}. Numpy's version[3] calls the standard library for linear algebra LAPACK[4][5] which implements the following procedure. The algorithm amounts to obtaining the LUP -decomposition of A, *i.e.*, $PA = LU$, where P, L, U are the permutation, the lower triangular, and the upper triangular matrices, respectively. Thus, the problem (Fig.13) becomes $LU \cdot x = P \cdot b$. Furthermore, we apply the permutation matrix P to b, to arrive at $P \cdot b = c$ and then we solve two consecutive systems: (a) lower triangular system $L \cdot y = c$ for y and (b) upper triangular system $U \cdot x = y$ for x.

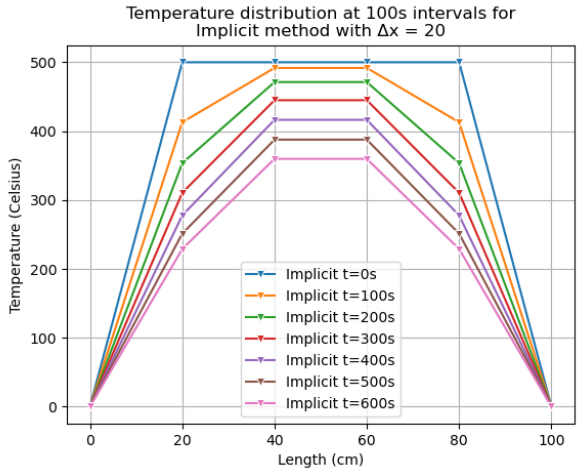
```
for j, t in enumerate(time_range[:-1]):
    A = np.zeros(shape=(len(space_range), len(space_range)))
    B = np.zeros(shape=(len(space_range), len(space_range)))
    for i, x in enumerate(space_range):
        if i == 0 or i == len(space_range) - 1:
            A[i, i] = 1
            B[i, i] = 1
        else:
            A[i, i - 1] = -lambda_
            A[i, i] = 2 + 2 * lambda_
            A[i, i + 1] = -lambda_
            B[i, i - 1] = lambda_
            B[i, i] = 2 - 2 * lambda_
            B[i, i + 1] = lambda_
    # Solve the system of linear equations: A * U_{j+1} = B * U_j
    comp_matrix[j + 1] = np.linalg.solve(A, B @ comp_matrix[j])
```

Figure 7: Python Snippet: Implicit method for Heat Equation

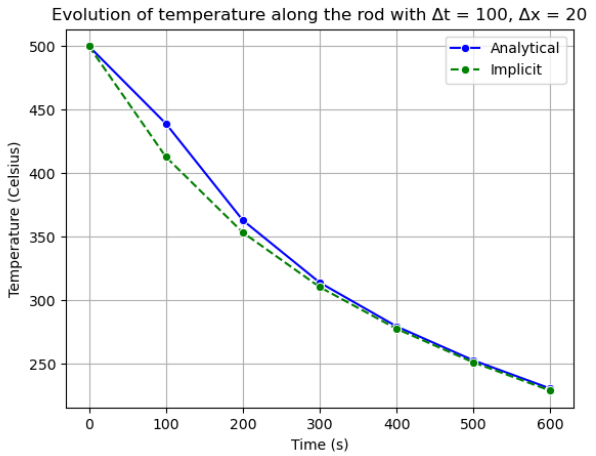
[3] https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html
[4] https://github.com/Reference-LAPACK/lapack
[5] https://en.wikipedia.org/wiki/LAPACK

4.3 Experiments

As in Ch.3.3, we conduct experiments to demonstrate the *accuracy* and the *stability* of the implicit Crank-Nicolson method. Similarly to the explicit method for $\Delta x = 20$ (Fig.5a,8a), the implicit temperature distribution along the bar seems just as stable. Nevertheless, regarding the temperature evolution, the implicit method (Fig.9a) tracks the analytical one more closely than FTCS does (Fig.6a). In fact, the relative error at the last time step falls from around 4% to less than 1%, simply by utilising Crank-Nicolson (Fig.11). This *error reduction* occurs because of the nature of C-N (*i.e.*, *second-order accurate in both space and time*) and hence the *boost in accuracy*. To further demonstrate the *unconditional stability* aspect, we reduce both step sizes individually as done in Ch.3.3. From the graph (Fig.9a) and the table (Table 9b, it is clear that by doing so, the implicit solution only gets closer to the analytical without accumulating any oscillations across time (unlike the explicit approach). Also, what stands out is that halving the spatial step ($\Delta x = 10,\ \Delta t = 100$) results in a more accurate solution than halving the time step. Our explanation is that this phenomenon is due to the further fragmentation of the spatial grid (*i.e.*, making it *denser*), which once we apply the computational molecule, produces a smoother approximation over the state-region.

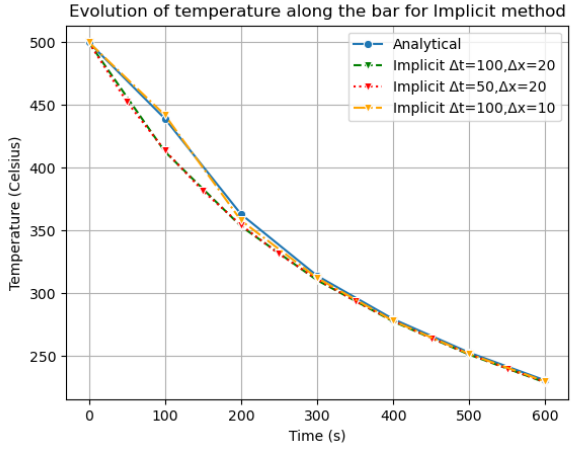


(a) Temperature distribution along the bar for Implicit method



(b) Analytical vs. Implicit method: temperature evolution

Figure 8: Crank-Nicolson: Temperature distribution along the bar and temperature evolution through time



(a) Analytical vs. Implicit method: temperature evolution

Method	Δt	Δx	Temperature (Celsius)
Analytical	100	20	230.576880
Implicit	100	20	228.955176
Implicit	50	20	229.317966
Implicit	100	10	229.712404

(b) Comparison of temperature per method configuration

Figure 9: Temperature evolution and comparison for Analytical & Implicit methods

5 Modelling the Initial Temperature Change

Throughout the previous experiments between the explicit & implicit methods, the analytical solution was beneficial to make the comparisons. However, in this part of the task we will be assuming that the time range

of interest is $0s < t \leq 60s$ and the initial temperature distribution is non-uniform (Eq.14, Fig.14), for which the analytical method is undefined.

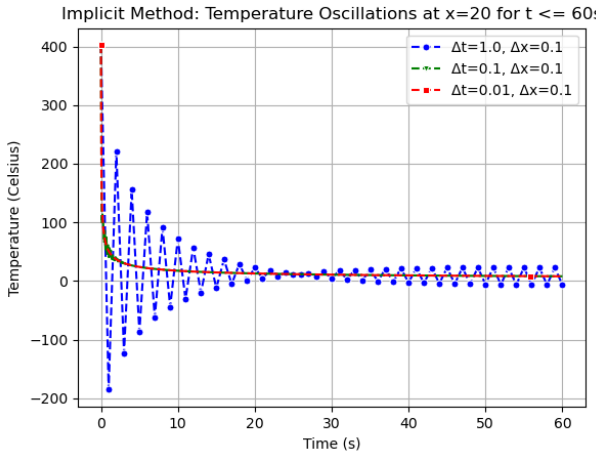
$$T(x,0) = -xg(x-L) + c, \quad g = 0.1,\ c = 400 \tag{14}$$

To model the temperature without the analytical solution we will make use of the implicit method: Crank-Nicolson due to its advantages over FTCS (namely, *accuracy* and *stability*). Therefore, we need to find a step-size configuration $\Delta x,\ \Delta t$, such that it lowers a hypothetical error ξ (the smoothness of the solution), but is still computationally efficient (measured in seconds).

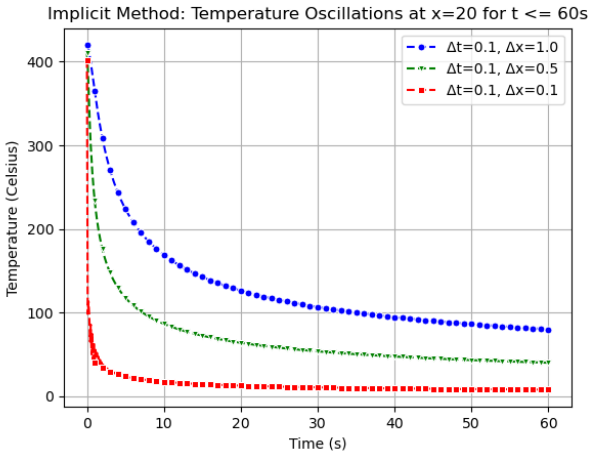
To achieve this we could start exploring the smoothness of different configurations, and later on analyse with respect to the computational resources.

1. If we fix Δx and vary Δt, we can observe that lowering the temporal step results in a smoother solution. However, we report that there are some oscillations, observed for $\Delta x = 1,\ \Delta t = 0.1$. Although, the implicit Crank-Nicolson method is *unconditionally stable*, it can still suffer from *high-frequency oscillations* as discussed in LeVeque [2007].

2. Furthermore, if we fix Δt instead and vary Δx, we arrive at similar conclusions, i.e., reducing the spatial step provides for a smoother solution (however, some visible oscillations once again for $\Delta x = 0.1,\ \Delta t = 0.1$)



(a) Temperature oscillations with fixed x and varied t



(b) Temperature oscillations with fixed t and varied x

Figure 10: Crank-Nicolson: Early temperature oscillations

We also find that when $\Delta t \leq 0.01,\ \Delta x \leq 0.1$ the solution becomes impractical. Hence, we provide the following table that summarises the computational efficiency (Fig.1).

Method	Δt	Δx	Computational Time (s)
Implicit	0.01	0.1	74
Implicit	0.01	0.5	4
Implicit	0.01	0.9	2

Table 1: Computational Efficiency Comparison

Therefore, based on these experiments we conclude that for the configuration $\Delta t = 0.01,\ \Delta x = 0.1$ we arrive at smooth and efficient solution to the Heat equation.

6 Appendix

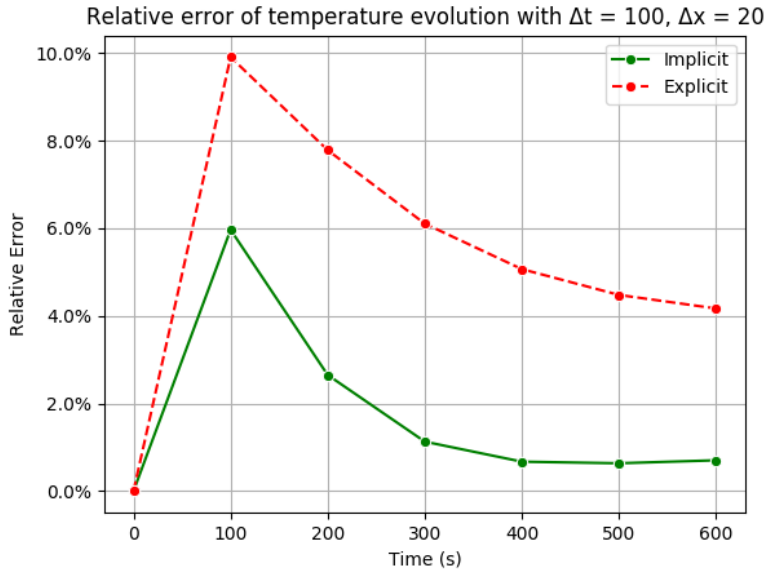


Figure 11: Temperature evolution error against Analytical solution

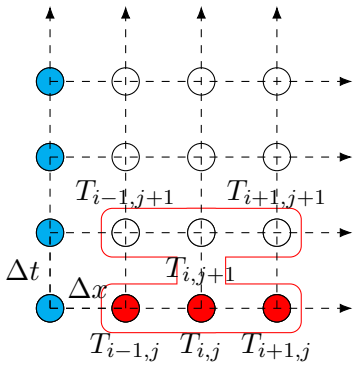


Figure 12: Implicit method's stencil

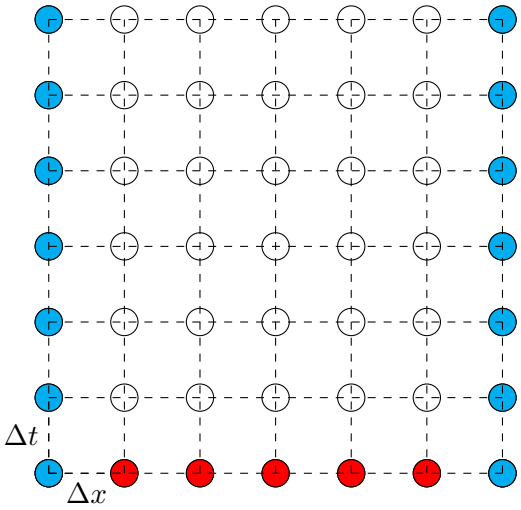


Figure 13: Temperature Distribution along the aluminium bar in 2D representation

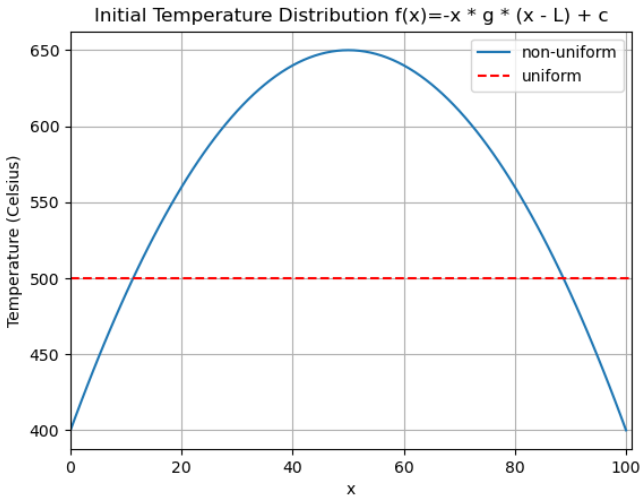


Figure 14: Initial Temperature Distributions

7 Bibliography

References

Alebraheem, J. (2017). Forward time centered space scheme for the solution of transport equation. *International Annals of Science*, 2:1–5.

Burden, R. L. and Faires, J. D. (1989). *Numerical Analysis*. The Prindle, Weber and Schmidt Series in Mathematics. PWS-Kent Publishing Company, Boston, fourth edition.

Crank, J. (1979). *The Mathematics of Diffusion*. Oxford science publications. Clarendon Press.

Crank, J. and Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67.

Lax, P. D. (2007). *Linear algebra and its applications*, volume 78. John Wiley & Sons.

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics.

Süli, E. and Mayers, D. (2003). *An Introduction to Numerical Analysis*. An Introduction to Numerical Analysis. Cambridge University Press.

Østerby, O. (2003). Five ways of reducing the crank–nicolson oscillations. *BIT Numerical Mathematics*, 43:811–822.