Modeling the Ice-Air-Water Interface using the 1D Heat Equation

Comparing and Contrasting Different Solution Methods

JOSEPH FOGARTY AND CHRISTINE PU

May 14, 2019

1 Motivation

The New Arctic is undergoing changes due to the human-caused gradual warming of Earths atmosphere. Some of these effects include the reduction in Arctic sea ice and creation of leads in early spring and fractured sea ice in early Spring and Summer. This causes the arctic terrain to break up into a heterogeneous surface between ice and water, such as in Figure 3.

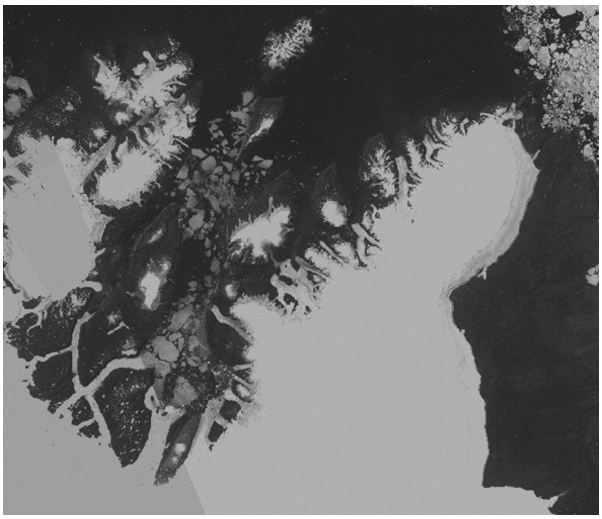

Figure 1: A satellite image of an ice/water map from the NASA Greenland Ice Mapping Project (GIMP)

This changing surface can lead to modifications in the Arctic meteorology, which can spread to lower latitudes. In particular, convective plumes can form when differences in temperature and roughness between the sea ice and the ocean arise. However, in order to study these effects, a model must be made for the interaction between sea

ice, the ocean, and the atmosphere. Part of this system includes modeling the temperature inside a block of ice, which lends itself to solving the heat equation, a special case of the diffusion equation. The 1D heat equation is a highly studied and well-known parabolic partial differential equation (PDE), seen in almost any problem with diffusion of some substance or heat.

$$\frac{\partial \mathbf{u}}{\partial t} = \alpha \nabla^2 \mathbf{u} \tag{1}$$

This equation essentially describes how the heat distribution in a solid changes over time. The second principle of thermodynamics states that heat naturally flows from higher temperature areas to lower temperature areas. The temperature of some heat gradient will increase depending on the mass and specific heat capacity of the material and the temperature difference between the incoming heat and the receptive material. In order to use solve the heat equation as a part of this larger system, we want to explore, compare, and contrast different numerical solution methods.

2 Physical System

2.1 The Heat Equation

We now describe our physical system and problem statement. We consider a block of sea ice floating in the Arctic ocean. If we consider the temperature **u** to be constant in the horizontal, then we are able to solve the following 1D system in the vertical for the temperature of the ice, u:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{2}$$

and

$$\alpha = \frac{k}{\rho c_p}. \tag{3}$$

where α is the thermal diffusivity in $\mathrm{m}^2\,\mathrm{s}^{-1}$, k is the thermal conductivity in $\mathrm{W\,m^{-1}\,K^{-1}}$, ρ is the density in $\mathrm{kg\,m^{-3}}$, and c_p is the specific heat at constant pressure in $\mathrm{J\,kg^{-1}\,K^{-1}}$. For values these values that correspond to ice, we calculate α:

$$\alpha = \frac{k}{\rho c_p} = \frac{2.25\mathrm{W\,K^{-1}\,m^{-1}}}{\left(916.7\mathrm{kg\,m^{-3}}\right)\left(2027\mathrm{J\,kg^{-1}\,K^{-1}}\right)} \approx 1.211 \times 10^{-6}. \tag{4}$$

We will see later on that this small α allows for higher values of dt. Note that even though this is being solved for in the vertical, we will consider the domain of ice as in the x-direction, where $x \in [0, L]$. $x = 0$ will be the surface of the ice (the end exposed to the air), and $x = L$ will be the bottom of the ice (the end that is underwater).

2.2 Initial and Boundary Conditions

For our boundary conditions, we will set $x = L$ to be a constant value, that is, the temperature of the water. Since this is the interface of melting water, we will set this temperature to be the melting temperature, $u(L) = 273.15$ K. The top temperature is slightly different. If we want to follow the diurnal cycle of temperature, we construct a function that closely mimics the temperature cycle in Barrow, AK in early spring. We can then write $u(0) = T_a(t)$. As a function of time in hours, we write:

$$T_a(t_{hr}) = 7 \sin\left(\frac{\pi}{12}(t_{hr} - 13)\right) + 268 \tag{5}$$

This function has a period of every 24 hours, which makes sense, since we want this diurnal cycle to repeat. We can also start with a some initial profile that connects these two points:

$$u(x, 0) = -14 \sin\left(\pi \frac{x}{L}\right) + \frac{1}{L}\left[T_a(0)(L - x) + T_w x\right] \tag{6}$$

It is easily observed that $u(0, 0) = T_a(0)$ and $u(L, 0) = T_w$.

2.3 Problem Statement

So our full system that we wish to solve numerically is written below:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \alpha \frac{\partial^2 u}{\partial x^2} \\ u(0, t) &= T_a(t) \\ u(L, t) &= T_w \\ u(x, 0) &= -14 \sin\left(\pi \frac{x}{L}\right) + \frac{1}{L}\left[T_a(0)(L - x) + T_w x\right] \end{aligned} \tag{7}$$

3 Algorithms

We will now take a look at three algorithms that are available to us to solve 7, and compare and contrast these methods. We will analyze the forward-time centered-space (FTCS) algorithm (also known as the explicit Euler), the backward-time centered-space (BTCS) algorithm (also known as Euler's implicit method), and the Crank-Nicolson method.

3.1 Forward Time, Centered Space

The classic time-stepping scheme seen in almost any undergraduate differential equations class is the forward time, centered space scheme, also known as the explicit Euler scheme. It is implemented as follows:

$$\frac{u_j^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} \tag{8}$$

We then rewrite as the following, where $r = \frac{\alpha \Delta t}{(\Delta x)^2}$:

$$u_j^{n+1} = u_i^n + r\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) \tag{9}$$

This scheme is both easy to implement and reliable, however it does have a stability condition, which is that $r \leq \frac{1}{2}$.

3.2 Backward Time, Centered Space

The backward time, centered space scheme is also called the Implicit Euler method. Implicit methods find a solution by considering the current and future values of the system. In general, the implicit method follows the following algorithm:

$$\begin{aligned} \frac{u_n^{j+1} - u_n^j}{\Delta t} &= \alpha \frac{\delta_x^2 u_n^{j+1}}{(\Delta x)^2} \\ &= \alpha \left[\frac{u_{n+1}^{j+1} - 2u_n^{j+1} + u_{n-1}^{j+1}}{(\Delta x)^2} \right] \end{aligned} \tag{10}$$

The solution at the next timestep u_n^{j+1} depends not only on the nodes already known at time j but at the next timestep $j+1$. For each timestep, the algorithm is solved for each of the interior points (for $N+1$ nodes, going from 0 to N, there are $N-1$ interior points going from 1 to $N-1$)−. This gives us a system of equations which can be expressed in matrix form. First, we write out the left-hand side (the unknowns) as $A\boldsymbol{u}^{j+1}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1+2r & -r & 0 & \ddots & \ddots & & & & 0 \\ 0 & -r & 1+2r & -r & \ddots & \ddots & & & & \vdots \\ 0 & 0 & -r & 1+2r & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 1+2r & -r & 0 & 0 \\ \vdots & & & & \ddots & \ddots & -r & 1+2r & -r & 0 \\ 0 & & & & \ddots & \ddots & 0 & -r & 1+2r & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0^{j+1} \\ u_1^{j+1} \\ u_2^{j+1} \\ u_3^{j+1} \\ \vdots \\ \vdots \\ u_{N-3}^{j+1} \\ u_{N-2}^{j+1} \\ u_{N-1}^{j+1} \\ u_N^{j+1} \end{bmatrix} \tag{11}$$

This matrix will be inverted to solve for the u values. This is much more computationally expensive than the explicit scheme, but provides the benefit of better stability. Note that we now include the boundary conditions. This current matrix assumes that the boundary conditions do not change, but in the code, they will be set every timestep to some other function $f(t)$ dependent only on time.

Figure 3.2 shows the main difference between the implicit and explicit schemes. Although both have equal orders of accuracy, the implicit scheme is unconditionally stable, while the explicit is conditionally stable.

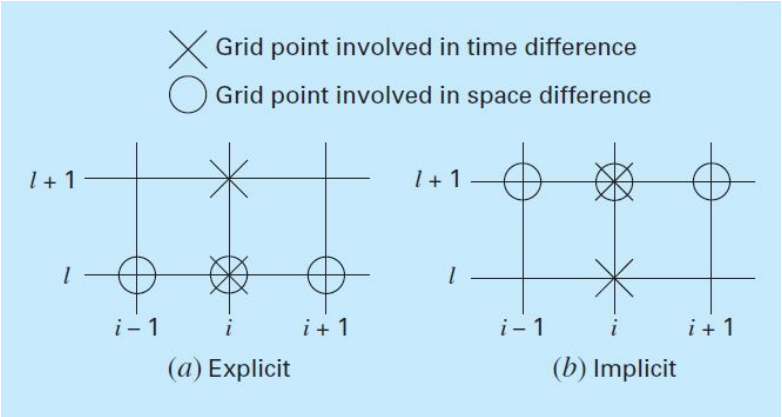

Figure 2: Computational molecules for the a) Explicit and b) Implicit schemes. Taken from Chapra and Canale.

3.3 Crank-Nicolson

The Crank-Nicolson scheme was developed by Crank and Nicolson in 1947 as an implicit scheme to solve the heat equation. Essentially, it can be thought of as an average of the explicit and implicit methods. In general, the scheme is given by

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} = \alpha \frac{\theta \delta_x^2 u_j^{n+1} + (1-\theta)\delta_x^2 u_j^n}{(\Delta x)^2} \tag{12}$$

where θ is a constant. We have already dealt with the cases where $\theta = 0$ and $\theta = 1$ - these are the Explicit Euler and Implicit Euler schemes, respectively. For the Crank-Nicolson case, we set $\theta = 0.5$, which is akin to using the trapezoidal rule in time.

$$\begin{aligned} \frac{u_n^{j+1}-u_n^j}{\Delta t} &= \alpha \frac{\delta_x^2 u_n^j + \delta_x^2 u_n^{j+1}}{2(\Delta x)^2} \\ &= \frac{\alpha}{2}\left[\frac{u_{n+1}^j - 2u_n^j + u_{n-1}^j}{(\Delta x)^2} + \frac{u_{n+1}^{j+1} - 2u_n^{j+1} + u_{n-1}^{j+1}}{(\Delta x)^2}\right] \end{aligned} \tag{13}$$

Again, δ_x^2 is the central difference operator. Setting $r = \frac{\alpha \Delta t}{2(\Delta x)^2}$, we see:

$$u_n^{j+1} - u_n^j = r\left(\left(u_{n+1}^j - 2u_n^j + u_{n-1}^j\right) + \left(u_{n+1}^{j+1} - 2u_n^{j+1} + u_{n-1}^{j+1}\right)\right) \tag{14}$$

Our unknowns are all values at the $j+1$ next timestep. Similar to the implicit method, the solution at the next timestep u_n^{j+1} depends not only on the nodes already known at time j but at the next timestep $j+1$. We can rearrange to get the unknown values on one side and known values on the other:

$$-ru_{n-1}^{j+1}+(1+2r)u_n^{j+1}-ru_{n+1}^{j+1}=ru_{n-1}^{j}+(1-2r)u_n^{j}+ru_{n+1}^{j} \tag{15}$$

For each timestep, this is solved for each of the interior points (for $N+1$ nodes, going from 0 to N, there are $N-1$ interior points going from 1 to $N-1$)– this gives us a system of equations. We write out the first few iterations to get a sense of the system that will need to be solved every time-step:

$$\begin{aligned}
n=1 &\Rightarrow -ru_0^{j+1}+(1+2r)u_1^{j+1}-ru_2^{j+1}=ru_0^{j}+(1-2r)u_1^{j}+ru_2^{j}\\
n=2 &\Rightarrow -ru_1^{j+1}+(1+2r)u_2^{j+1}-ru_3^{j+1}=ru_1^{j}+(1-2r)u_2^{j}+ru_3^{j}\\
n=3 &\Rightarrow -ru_2^{j+1}+(1+2r)u_3^{j+1}-ru_4^{j+1}=ru_2^{j}+(1-2r)u_3^{j}+ru_4^{j}
\end{aligned} \tag{16}$$

Now that we have a sense of the pattern for these linear equations, we can rewrite this in matrix form. First, we write out the left-hand side (the unknowns) as $A\boldsymbol{u}^{j+1}$:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\
0 & 1+2r & -r & 0 & \ddots & \ddots & & & & 0\\
0 & -r & 1+2r & -r & \ddots & \ddots & & & & \vdots\\
0 & 0 & -r & 1+2r & \ddots & \ddots & \ddots & \ddots & & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & & & & \ddots & \ddots & 1+2r & -r & 0 & 0\\
\vdots & & & & \ddots & \ddots & -r & 1+2r & -r & 0\\
0 & & & & \ddots & \ddots & 0 & -r & 1+2r & 0\\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
u_0^{j+1}\\ u_1^{j+1}\\ u_2^{j+1}\\ u_3^{j+1}\\ \vdots\\ \vdots\\ u_{N-3}^{j+1}\\ u_{N-2}^{j+1}\\ u_{N-1}^{j+1}\\ u_N^{j+1}
\end{bmatrix} \tag{17}$$

Note that we now include the boundary conditions. This current matrix assumes that the boundary conditions do not change, but in the code, they will be set every time-step to some other function $f(t)$ dependent only on time. We will also look at the right-hand side, which is fairly similar.

$$\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\
0 & 1-2r & r & 0 & \ddots & \ddots & & & & 0\\
0 & r & 1-2r & r & \ddots & \ddots & & & & \vdots\\
0 & 0 & r & 1-2r & \ddots & \ddots & \ddots & \ddots & & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & & & & \ddots & \ddots & 1-2r & r & 0 & 0\\
\vdots & & & & \ddots & \ddots & r & 1-2r & r & 0\\
0 & & & & \ddots & \ddots & 0 & r & 1-2r & 0\\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
u_0^{j}\\ u_1^{j}\\ u_2^{j}\\ u_3^{j}\\ \vdots\\ \vdots\\ u_{N-3}^{j}\\ u_{N-2}^{j}\\ u_{N-1}^{j}\\ u_N^{j}
\end{bmatrix} \tag{18}$$

We know the solution to 18, since we use the known values from the previous time-step. We can then solve the system $Ax = b$ by setting the solution to 18 equal

to 17, and then solving the system. Figure 3.3 shows how the Crank-Nicolson scheme uses both the implicit and explicit schemes together.

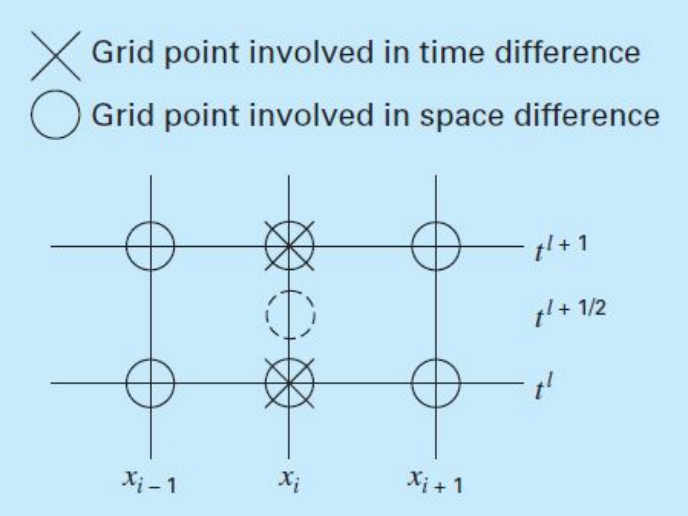

Figure 3: Computational molecule for the Crank-Nicolson scheme. Taken from Chapra and Canale.

4 Results

We now present our results for implementing the three schemes described above. We will consider both a timestep and error analysis.

4.1 Main Results

Our main results are movie files (.mp4 format), which can be found in the GitHub repository. We present here a few select screenshots from each simulation, that includes the beginning of the simulation, the middle of the simulation, and the last timestep. We change the timestep, thus varying the value of r - this relates to not only accuracy, but stability of the solution as well, as we will see in the following figures.

For all of these figures, the x-xis shows the length of the ice, with the top temperature at $x = 0$ and the bottom temperature being $x = 2.0$. Temperature is in the y-axis, in Kelvin.

4.1.1 $dt = 0.5$, $r \approx 0.0242$

Here, we have the case where $dt = 0.5$s. As can be expected, the two explicit and implicit solutions stay together, while the Crank-Nicolson solution seems to be doing its own thing - however, it is the more accurate solution.

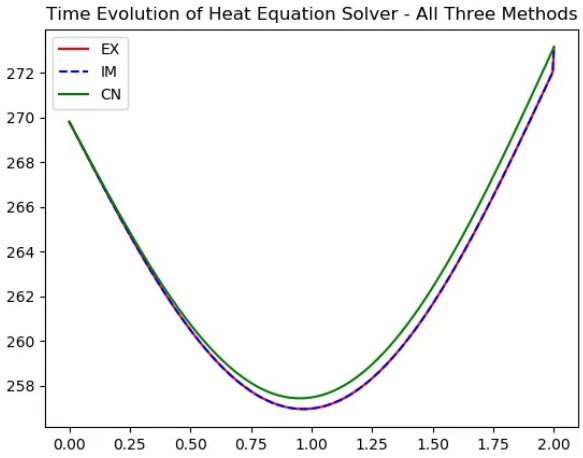

Figure 4: Comparison of Three Methods with $dt = 0.5$ at $t = 0$ days

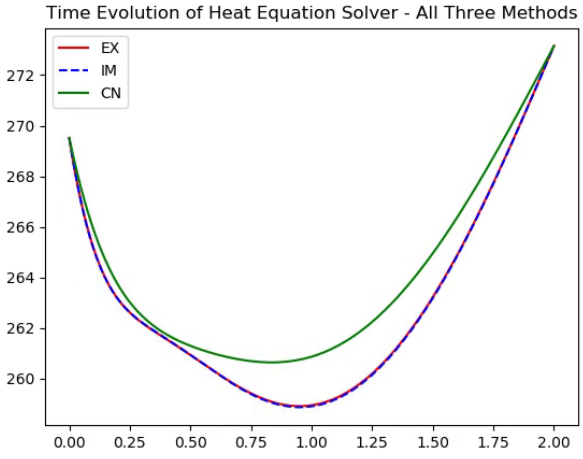

Figure 5: Comparison of Three Methods with $dt = 0.5$ at $t = 2.89$ days

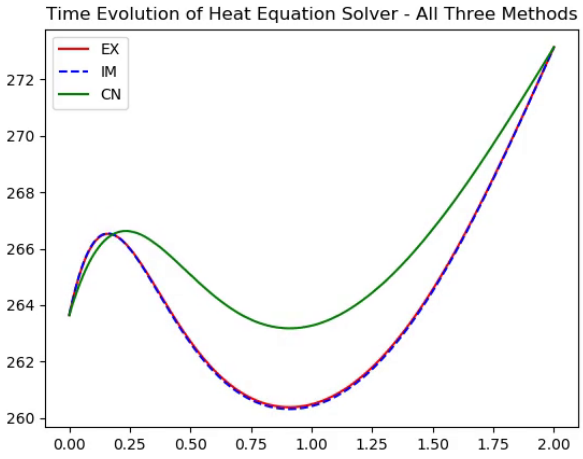

Figure 6: Comparison of Three Methods with $dt = 0.5$ at $t = 5.78$ days

4.1.2 $dt = 5.0$, $r \approx 0.2422$

Now we examine the case where $dt = 5.0$s, thus increasing the value of r by a factor of 10. The difference here is that the explicit solution and implicit solution are not as close together as they were in the first case. We also see that again, the Crank-Nicolson solution seems to be different from both.

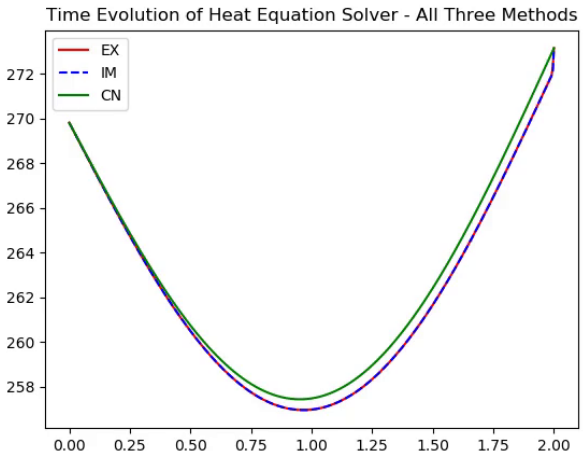

Figure 7: Comparison of Three Methods with $dt = 5$ at $t = 0$ days

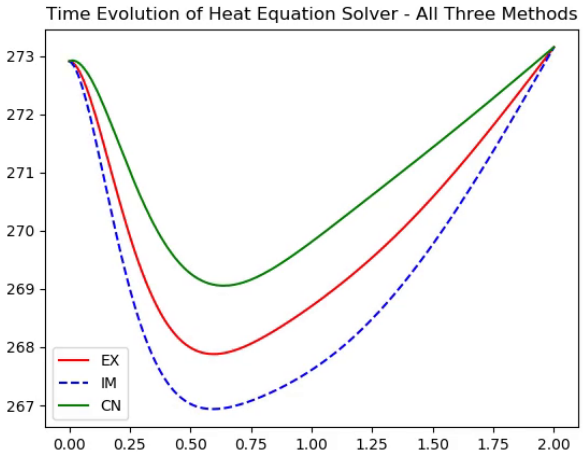

Figure 8: Comparison of Three Methods with $dt = 5$ at $t = 5.78$ days

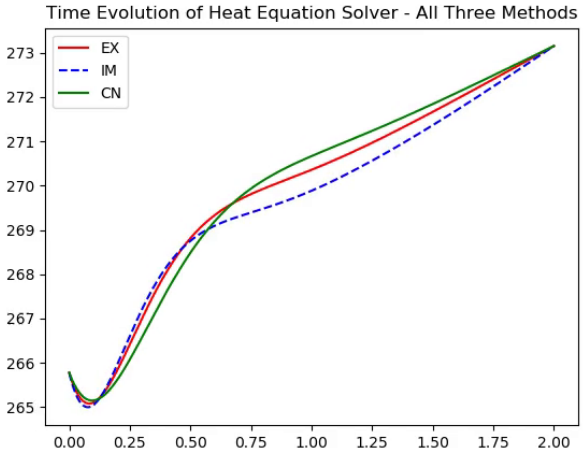

Figure 9: Comparison of Three Methods with $dt = 5$ at $t = 11.65$ days

4.1.3 $dt = 50.0$, $r \approx 2.4218$

We will finally analyze an extreme case where $dt = 50.0$s, again increasing the value of r by a factor of 10. Now that $r > 0.5$, the explicit solution "bugs out" extremely early on in the simulation - in the first timestep to be exaact. It can be seen in Figure 4.1.3 the magnitude that the explicit solution goes to - so much that the Implicit and Crank-Nicolson curves are indistinguishable.

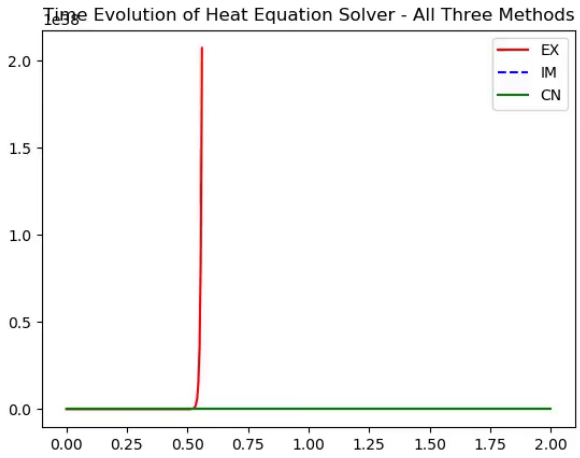

Figure 10: Comparison of Three Methods with $dt = 50$ at $t = 0$ days

It can be seen that higher values of r correspond to much more sharper gradients in the middle of the ice block. This likely has to do with the fact that since the timestep increases, the inside of the ice (due to the low value of α) does not have enough time to "catch up"

4.2 Timestep Analysis

We now present some numerical data that shows how some arbitrary point in these functions behave with differeing timesteps. As expected, and can be seen in the following table, the Explicit scheme is unstable for values of r over 0.5.

Δ t	r	Explicit	Implicit	Crank-Nicolson
0.5	0.024217	265.744951	265.744438	265.760300
1	0.048435	265.745478	265.744438	265.760315
5	0.242176	265.750063	265.744436	265.760440
10	0.484352	265.754792	265.744433	265.760595
50	2.421762	-3.256684e+61	265.744418	265.761787
100	4.843524	7.965404e+62	265.744411	265.763156

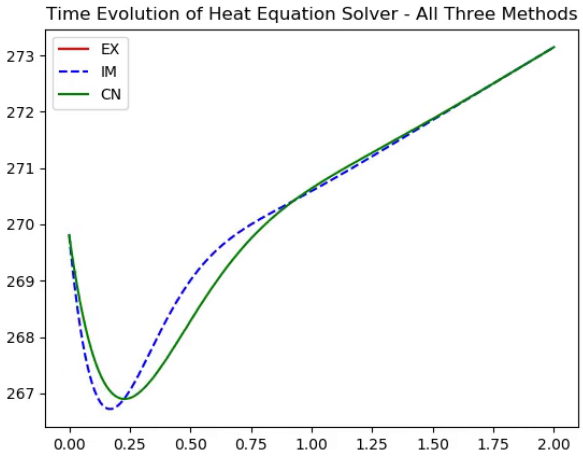

Figure 11: Comparison of Three Methods with $dt = 50$ at $t = 57$ days

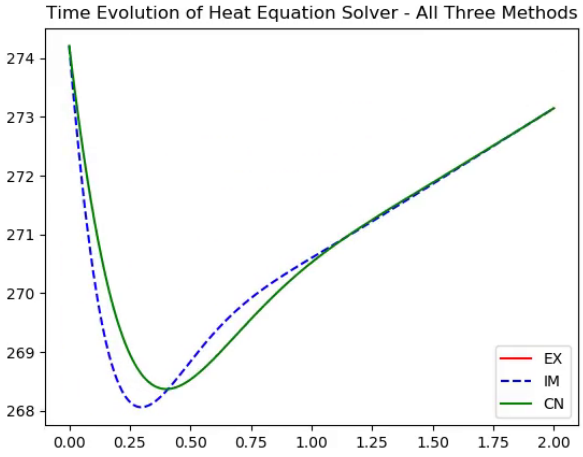

Figure 12: Comparison of Three Methods with $dt = 50$ at $t = 114$ days

Note that in this table, certain time-steps had to be really high in order to show some kind of error in the explicit function. In section 2, we saw that the value of α was really low, because of the properties of ice. This low α allowed for a larger time-step. Recall that for the explicit scheme, as long as the time-step was below $\frac{1}{2}$, (Carnahan (1969)), then the solution is stable and it converges.

5 Roadblocks

For a system with changing boundary conditions, it is tough to verify these different schemes against the analytical solution. These solutions exist, but they require computing Fourier coefficients and eigenvalues for a system - something that we did not have the time to do in this analysis. Coding up the three schemes, especially the Crank-Nicolson scheme, took up a lot of time, and we were only able to do the analysis that is presented here (as well as compiling the .mp4 files to see these methods evolve in time).

We also had to deal with semi-sparse matrices for the Crank-Nicolson and implicit schemes (we say semi-sparse because these matrices are padded with all zeroes, and only 1's at the corners). This is not an issue, but sparse matrices in `scipy.linalg.sparse` are not easily mutable - they must be created as a row-based linked list sparse matrix using `.tolil()`, and then converted to a compressed sparse column using `.tocsc()`. This is not an issue in itself, but figuring it out took longer than expected.

6 Conclusions

If we were to solely base this analysis on our results, it seems that the best time-stepping scheme is the implicit method. However, if we are to base this solely on the algorithms, then the Crank-Nicolson easily wins out. Computationally, using the Crank-Nicolson scheme can be much faster since we are dealing with sparse tridiagonal matrices.

In general, the Crank-Nicolson case is the best for solving 1D linear parabolic PDEs out of the three methods tested here. It's accuracy is better than either the implicit or explicit (both of these are first-order accurate in time and second-order accurate in space, while the Crank-Nicolson is second-order accurate with respect to both). This makes sense, since the difference approximations are evaluated at the midpoint of the time increment, rather than the current time-step (implicit) or previous time-step (explicit).

References

Carnahan, B. (1969). Applied numerical methods.

Chapra, S. C. and Canale, R. P. (2015). *Numerical methods for engineers*. McGraw-Hill Education, New York, NY, seventh edition edition.

Crank, J. and Nicolson, E. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type.

Lapidus, L. and Pinder, G. F. (1983). *Numerical solution of partial differential equations in science and engineering*.