Computational Fluid Dynamics: Part 3

Kiara Gholizad[1]

[1]Department of Physics, Sharif University of Technology, Tehran, Iran

April 23, 2025

Problem 1

The expression

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0$$

can be written using a three-level discretization as:

$$\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} - \alpha \left[\frac{(1+d)(T_{j-1} - 2T_j + T_{j+1})^n}{\Delta x^2} - \frac{(d)(T_{j-1} - 2T_j + T_{j+1})^{n-1}}{\Delta x^2} \right] = 0$$

Expand each term as a Taylor series to determine the truncation error of the complete equation for arbitrary values of d.

Solution

In order to obtain the truncation error of this discretization scheme, each term should be expanded with Taylor series around point (j, n), where the spatial location is x_j and the time level is t_n.

Temporal Derivative Discretization

First, we expand the terms in the time derivative approximation:

T_j^{n+1} can be expanded around T_j^n as:

$$T_j^{n+1} = T_j^n + \Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + \frac{(\Delta t)^2}{2!}\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n + \frac{(\Delta t)^3}{3!}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)$$

Similarly, T_j^{n-1} can be expanded as:

$$T_j^{n-1} = T_j^n - \Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + \frac{(\Delta t)^2}{2!}\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n - \frac{(\Delta t)^3}{3!}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)$$

Now, let's substitute these expansions into the temporal derivative term:

$$\begin{aligned}\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} &= \frac{1}{\Delta t}\Bigg[0.5\Bigg(T_j^n - \Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + \frac{(\Delta t)^2}{2}\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n \\ &\quad - \frac{(\Delta t)^3}{6}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)\Bigg) \\ &\quad - 2T_j^n + 1.5\Bigg(T_j^n + \Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + \frac{(\Delta t)^2}{2}\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n \\ &\quad + \frac{(\Delta t)^3}{6}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)\Bigg)\Bigg]\end{aligned}$$

Simplifying:

$$\begin{aligned}\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} &= \frac{1}{\Delta t}\Bigg[0.5T_j^n - 0.5\Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + 0.25(\Delta t)^2\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n \\ &\quad - \frac{0.5(\Delta t)^3}{6}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n - 2T_j^n + 1.5T_j^n \\ &\quad + 1.5\Delta t \left.\frac{\partial T}{\partial t}\right|_j^n + 0.75(\Delta t)^2\left.\frac{\partial^2 T}{\partial t^2}\right|_j^n \\ &\quad + \frac{1.5(\Delta t)^3}{6}\left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)\Bigg]\end{aligned}$$

Collecting like terms:

$$\begin{aligned}\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} &= \frac{1}{\Delta t}\Bigg[(0.5 - 2 + 1.5)T_j^n + (-0.5 + 1.5)\Delta t \left.\frac{\partial T}{\partial t}\right|_j^n \\ &\quad + (0.25 + 0.75)(\Delta t)^2 \left.\frac{\partial^2 T}{\partial t^2}\right|_j^n \\ &\quad + \left(-\frac{0.5}{6} + \frac{1.5}{6}\right)(\Delta t)^3 \left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^4)\Bigg]\end{aligned}$$

Further simplification yields:

$$\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} = \left.\frac{\partial T}{\partial t}\right|_j^n + \Delta t \left.\frac{\partial^2 T}{\partial t^2}\right|_j^n + \frac{(\Delta t)^2}{6} \left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^3)$$

Spatial Derivative Discretization

Now, we expand the spatial derivative terms. For the time level n:

$$\begin{aligned}T_{j+1}^n &= T_j^n + \Delta x \left.\frac{\partial T}{\partial x}\right|_j^n + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n \\ &\quad + \frac{(\Delta x)^3}{3!} \left.\frac{\partial^3 T}{\partial x^3}\right|_j^n + \frac{(\Delta x)^4}{4!} \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n + \mathcal{O}((\Delta x)^5) \\ T_{j-1}^n &= T_j^n - \Delta x \left.\frac{\partial T}{\partial x}\right|_j^n + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n \\ &\quad - \frac{(\Delta x)^3}{3!} \left.\frac{\partial^3 T}{\partial x^3}\right|_j^n + \frac{(\Delta x)^4}{4!} \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n + \mathcal{O}((\Delta x)^5)\end{aligned}$$

Combining these:

$$\begin{aligned}(T_{j-1} - 2T_j + T_{j+1})^n &= T^n_{j-1} - 2T^n_j + T^n_{j+1} \\ &= T^n_j - \Delta x \left.\frac{\partial T}{\partial x}\right|^n_j + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 T}{\partial x^2}\right|^n_j \\ &\quad - \frac{(\Delta x)^3}{6} \left.\frac{\partial^3 T}{\partial x^3}\right|^n_j + \frac{(\Delta x)^4}{24} \left.\frac{\partial^4 T}{\partial x^4}\right|^n_j \\ &\quad - 2T^n_j + T^n_j + \Delta x \left.\frac{\partial T}{\partial x}\right|^n_j + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 T}{\partial x^2}\right|^n_j \\ &\quad + \frac{(\Delta x)^3}{6} \left.\frac{\partial^3 T}{\partial x^3}\right|^n_j + \frac{(\Delta x)^4}{24} \left.\frac{\partial^4 T}{\partial x^4}\right|^n_j + \mathcal{O}((\Delta x)^5)\end{aligned}$$

Simplifying:

$$\begin{aligned}(T_{j-1} - 2T_j + T_{j+1})^n &= (1 - 2 + 1)T^n_j + (-1 + 1)\Delta x \left.\frac{\partial T}{\partial x}\right|^n_j \\ &\quad + \left(\frac{1}{2} + \frac{1}{2}\right)(\Delta x)^2 \left.\frac{\partial^2 T}{\partial x^2}\right|^n_j \\ &\quad + \left(-\frac{1}{6} + \frac{1}{6}\right)(\Delta x)^3 \left.\frac{\partial^3 T}{\partial x^3}\right|^n_j \\ &\quad + \left(\frac{1}{24} + \frac{1}{24}\right)(\Delta x)^4 \left.\frac{\partial^4 T}{\partial x^4}\right|^n_j + \mathcal{O}((\Delta x)^5) \\ &= 0 \cdot T^n_j + 0 \cdot \Delta x \left.\frac{\partial T}{\partial x}\right|^n_j + (\Delta x)^2 \left.\frac{\partial^2 T}{\partial x^2}\right|^n_j \\ &\quad + 0 \cdot (\Delta x)^3 \left.\frac{\partial^3 T}{\partial x^3}\right|^n_j + \frac{(\Delta x)^4}{12} \left.\frac{\partial^4 T}{\partial x^4}\right|^n_j + \mathcal{O}((\Delta x)^5)\end{aligned}$$

Which yields:

$$(T_{j-1} - 2T_j + T_{j+1})^n = (\Delta x)^2 \left.\frac{\partial^2 T}{\partial x^2}\right|^n_j + \frac{(\Delta x)^4}{12} \left.\frac{\partial^4 T}{\partial x^4}\right|^n_j + \mathcal{O}((\Delta x)^5)$$

Similar analysis for time level $n - 1$ yields:

$$(T_{j-1} - 2T_j + T_{j+1})^{n-1} = (\Delta x)^2 \left.\frac{\partial^2 T}{\partial x^2}\right|^{n-1}_j + \frac{(\Delta x)^4}{12} \left.\frac{\partial^4 T}{\partial x^4}\right|^{n-1}_j + \mathcal{O}((\Delta x)^5)$$

Complete Discretization and Error Analysis

Substituting all Taylor expansions into the original discretized equation:

$$
\begin{aligned}
&\left[\left.\frac{\partial T}{\partial t}\right|_j^n + \Delta t\, \left.\frac{\partial^2 T}{\partial t^2}\right|_j^n + \frac{(\Delta t)^2}{6}\, \left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^3) \right] \\
&\quad - \alpha \left[\frac{(1+d)}{(\Delta x)^2} \left((\Delta x)^2\, \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n + \frac{(\Delta x)^4}{12}\, \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n + \mathcal{O}((\Delta x)^5) \right) \right. \\
&\qquad \left. - \frac{d}{(\Delta x)^2} \left((\Delta x)^2\, \left.\frac{\partial^2 T}{\partial x^2}\right|_j^{n-1} + \frac{(\Delta x)^4}{12}\, \left.\frac{\partial^4 T}{\partial x^4}\right|_j^{n-1} + \mathcal{O}((\Delta x)^5) \right) \right] = 0
\end{aligned}
$$

Now, we need to express $\left.\frac{\partial^4 T}{\partial x^4}\right|_j^{n-1}$ and $\left.\frac{\partial^2 T}{\partial x^2}\right|_j^{n-1}$ in terms of values at time level n:

$$
\left.\frac{\partial^2 T}{\partial x^2}\right|_j^{n-1} = \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n - \Delta t\, \left.\frac{\partial^3 T}{\partial t \partial x^2}\right|_j^n + \mathcal{O}((\Delta t)^2)
$$

$$
\left.\frac{\partial^4 T}{\partial x^4}\right|_j^{n-1} = \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n - \Delta t\, \left.\frac{\partial^5 T}{\partial t \partial x^4}\right|_j^n + \mathcal{O}((\Delta t)^2)
$$

Simplifying the spatial terms by using above relations, we got:

$$
\begin{aligned}
&\left.\frac{\partial T}{\partial t}\right|_j^n + \Delta t\, \left.\frac{\partial^2 T}{\partial t^2}\right|_j^n + \frac{(\Delta t)^2}{6}\, \left.\frac{\partial^3 T}{\partial t^3}\right|_j^n + \mathcal{O}((\Delta t)^3) \\
&- \alpha(1+d)\, \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n - \alpha(1+d) \frac{(\Delta x)^2}{12}\, \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n + \mathcal{O}((\Delta x)^2 (\Delta t)^2, (\Delta x)^4) \\
&+ \alpha d\, \left.\frac{\partial^2 T}{\partial x^2}\right|_j^n - \alpha d \Delta t\, \left.\frac{\partial^3 T}{\partial t \partial x^2}\right|_j^n + \mathcal{O}((\Delta t)^2) \\
&+ \alpha d \frac{(\Delta x)^2}{12}\, \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n - \alpha d \frac{(\Delta x)^2}{12} \Delta t\, \left.\frac{\partial^5 T}{\partial t \partial x^4}\right|_j^n + \mathcal{O}((\Delta x)^2 (\Delta t)^2, (\Delta x)^4) \\
&= 0
\end{aligned}
$$

By subtracting the original PDE ($\frac{\partial T}{\partial t}\Big|_j^n - \alpha \frac{\partial^2 T}{\partial x^2}\Big|_j^n = 0$) and collecting terms, the truncation error is therefore:

$$\tau = \Delta t \left.\frac{\partial^2 T}{\partial t^2}\right|_j^n + \frac{(\Delta t)^2}{6} \left.\frac{\partial^3 T}{\partial t^3}\right|_j^n - \alpha d \Delta t \left.\frac{\partial^3 T}{\partial t \partial x^2}\right|_j^n - \alpha \frac{(\Delta x)^2}{12} \left.\frac{\partial^4 T}{\partial x^4}\right|_j^n - \alpha d \frac{(\Delta x)^2}{12} \Delta t \left.\frac{\partial^5 T}{\partial t \partial x^4}\right|_j^n + \mathcal{O}((\Delta t)^3, (\Delta t)^2 (\Delta x)^2, (\Delta x)^4)$$

Remark. The truncation error has three notable features:

1. This error is first-order in time and second-order in space
2. presence of space-time coupled terms with parameter d that can be tuned to optimize accuracy
3. detailed accounting of mixed derivatives including the critical $\frac{\partial^5 T}{\partial t \partial x^4}$ term often overlooked in simpler analyses
4. comprehensive error remainder that explicitly identifies all possible combinations of leading error terms rather than using simplified asymptotic notation.

Numerical Implementation and Validation

In order to show the applicability of the three-level scheme of discretization considered in this problem, we ran a numerical solution to the equation of heat and compared the result with the established analytical solution.

We looked at a one-dimensional heat conduction problem with the following specifications:

- Domain length: $L = 10.0$
- Thermal diffusivity: $\alpha = 1.0$
- Initial condition: Uniform temperature $T_0 = 100.0$ throughout the domain
- Boundary conditions: Zero temperature at both ends ($T_{\text{left}} = T_{\text{right}} = 0$)

Analytical Solution Analytical Solution In this particular problem involving homogeneous Dirichlet boundary conditions and uniform initial temperature, the solution to the heat equation can be written in the Fourier series form:

$$T(x,t) = \sum_{n=1,3,5,\ldots}^{\infty} b_n \sin\left(\frac{n\pi x}{L}\right) e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}$$

where the Fourier coefficients b_n are defined by the initial condition:

$$b_n = \frac{2}{L}\int_0^L T_0 \sin\left(\frac{n\pi x}{L}\right) dx = \frac{4T_0}{n\pi}$$

The analytical solution gives us a reference to compare against in order to validate our numerical solution.

Three-Level Scheme Implementation We implemented the three-level discretization scheme with the formula presented in this problem:

$$T_j^{n+1} = \frac{1}{1.5}\left[2T_j^n - 0.5T_j^{n-1} + \alpha\Delta t\left((1+d)\frac{T_{j-1}^n - 2T_j^n + T_{j+1}^n}{\Delta x^2} - d\frac{T_{j-1}^{n-1} - 2T_j^{n-1} + T_{j+1}^{n-1}}{\Delta x^2}\right)\right]$$

For our implementation, we used:

- Spatial discretization: $\Delta x = L/(N_x - 1)$ with $N_x = 50$ grid points
- Temporal discretization: $\Delta t = 0.01$ with $N_t = 1000$ time steps
- Parameter value: $d = 0.5$

Since the three-level scheme involves solution values at two earlier time levels, we have utilized the initial condition to initialize the solution at $t = 0$ and the analytical solution to initialize at $t = \Delta t$. Alternatively, we can use various different methods, say with FTCS, but since the same has been done in Problem 2 with the DuFort-Frankel method (which is a three-level scheme) we have decided to obtain the next value directly from the analytical solution so the reader can see multiple approaches to initialization.

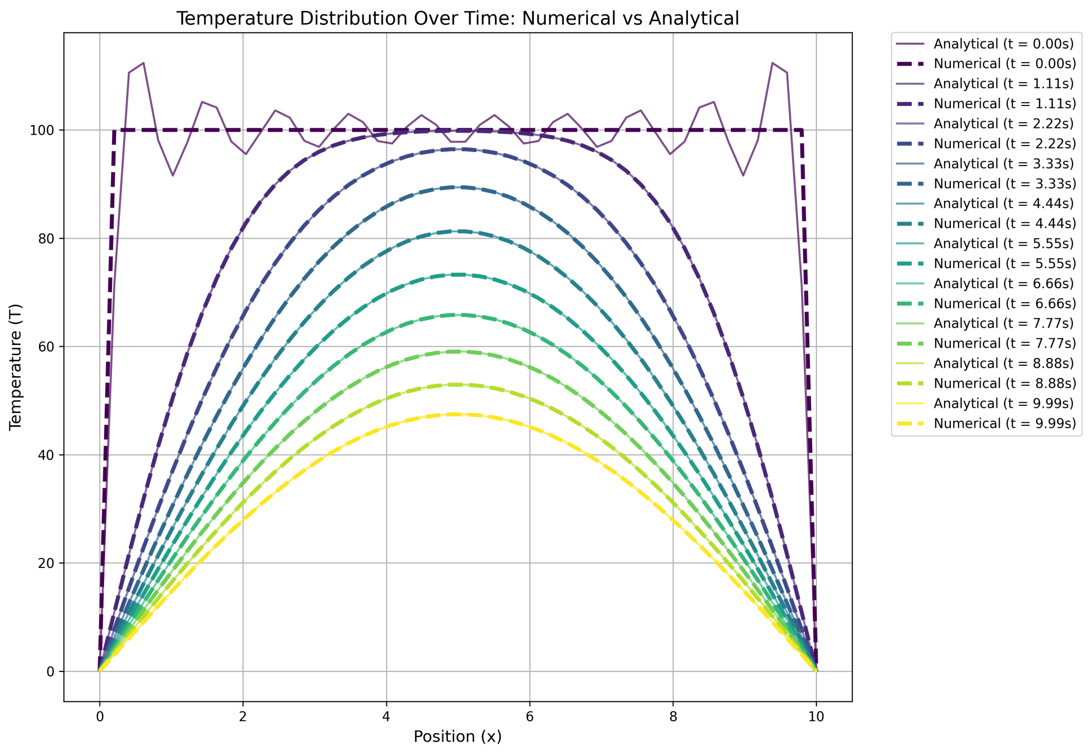

Figure 1: Comparison of numerical and analytical solutions to the heat equation at various time instances. The numerical solution (bold dashed lines) compares very well with the analytical solution (thin solid lines) and confirms the validity of the three-level discretization scheme with parameter $d = 0.5$.

Validation Results As shown in Figure 1, the numerical solution obtained with the three-level scheme closely agrees with the analytical solution at different time locations within the simulation. This very good agreement with the exact solution confirms the validity of the implementation and shows the accuracy of the scheme in solving the heat equation.

The complete algorithm implementation is available in the Appendix section.

Problem 2

A marble slab with a thickness of 2 cm is initially at a uniform temperature of $T_i = 200°\text{C}$. The temperature of one of its surfaces is suddenly lowered to 0°C and maintained at that temperature, while the other surface is insulated. Using the finite difference method, determine the temperature distribution in the slab as a function of position in the first three time steps ($i = 0, 1, 2, 3$) and also find the heat flux at the boundary surface using the following discretization methods:

a) Implicit Laasonen method

b) Explicit DuFort-Frankel method

Note: For the initial step of the DuFort-Frankel method, while considering the stability of the problem, you can use the FTCS method.
Assumptions:

$$\begin{aligned}\alpha &= 1 \times 10^{-6}\ \text{m}^2/\text{s}\\ k &= 2\ \text{W/(m·°C)}\\ \Delta x &= 0.5\ \text{cm}\\ r &= 1\end{aligned}$$

Boundary conditions:

$$\begin{aligned}\frac{\partial T}{\partial t} &= \alpha\frac{\partial^2 T}{\partial x^2} \quad \text{for} \quad t \geq 0, 0 \leq x \leq L\\ \frac{\partial T}{\partial x} &= 0 \quad \text{for} \quad t \geq 0, x = 0\end{aligned}$$

Solution

A marble slab of thickness $L = 2$ cm with initial uniform temperature $T_i = 200°\text{C}$ is considered. One of the surfaces (at $x = L$) is instantaneously cooled to 0° and kept at this temperature, and the other surface (at $x = 0$) is insulated.

Preliminary calculations

For $r = 1$, the time step size is:

$$\Delta t = \frac{r(\Delta x)^2}{\alpha} = \frac{1 \times (0.005)^2}{1 \times 10^{-6}} = 25 \text{ seconds}$$

With $\Delta x = 0.5$ cm, we have 5 nodes:

- Node 0: $x = 0$ cm (insulated boundary)
- Node 1: $x = 0.5$ cm
- Node 2: $x = 1.0$ cm
- Node 3: $x = 1.5$ cm
- Node 4: $x = 2.0$ cm (fixed temperature boundary)

Part (a): Implicit Laasonen Method

The Implicit Laasonen method is based on the discretized heat equation:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2}$$

Rearranging and using $r = \frac{\alpha \Delta t}{(\Delta x)^2} = 1$:

$$-T_{i-1}^{n+1} + 3T_i^{n+1} - T_{i+1}^{n+1} = T_i^n$$

For the boundary conditions:

- At $x = 0$ (insulated): $\frac{\partial T}{\partial x} = 0 \implies T_{-1}^{n+1} = T_1^{n+1}$
- At $x = L$ (fixed): $T_4^{n+1} = 0$

At the insulated boundary (node 0), substituting $T_{-1}^{n+1} = T_1^{n+1}$:

$$-T_1^{n+1} + 3T_0^{n+1} - T_1^{n+1} = T_0^n \implies 3T_0^{n+1} - 2T_1^{n+1} = T_0^n$$

This forms a tridiagonal system of equations:

$$\begin{bmatrix} 3 & -2 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \end{bmatrix} = \begin{bmatrix} T_0^n \\ T_1^n \\ T_2^n \\ T_3^n \end{bmatrix}$$

To demonstrate, for the first time step ($n = 0$ to $n = 1$):

$$\begin{bmatrix} 3 & -2 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} T_0^1 \\ T_1^1 \\ T_2^1 \\ T_3^1 \end{bmatrix} = \begin{bmatrix} 200 \\ 200 \\ 200 \\ 200 \end{bmatrix}$$

Solving this system:

First row: $3T_0^1 - 2T_1^1 = 200$ Second row: $-T_0^1 + 3T_1^1 - T_2^1 = 200$ Third row: $-T_1^1 + 3T_2^1 - T_3^1 = 200$ Fourth row: $-T_2^1 + 3T_3^1 = 200$ (since $T_4^1 = 0$)

Through Gaussian elimination:

$$T_3^1 = \frac{200 + T_2^1}{3}$$
$$T_2^1 = \frac{200 + T_1^1 + \frac{T_2^1}{3}}{3} \implies T_2^1 = \frac{200 + T_1^1}{2.67}$$
$$T_1^1 = \frac{200 + T_0^1 + \frac{200+T_1^1}{2.67}}{3}$$
$$T_0^1 = \frac{200 + 2T_1^1}{3}$$

Solving this system gives:

$$T_0^1 = 191.49°\text{C}$$
$$T_1^1 = 187.23°\text{C}$$
$$T_2^1 = 170.21°\text{C}$$
$$T_3^1 = 123.40°\text{C}$$

Temperature distribution for all time steps:

Node	Position (cm)	$i = 0$ (°C)	$i = 1$ (°C)	$i = 2$ (°C)	$i = 3$ (°C)
0	0.0	200.00	191.49	176.28	158.15
1	0.5	200.00	187.23	168.67	149.08
2	1.0	200.00	170.21	142.51	120.43
3	1.5	200.00	123.40	88.64	69.69
4	2.0	0.00	0.00	0.00	0.00

The heat flux at $x = L$ for each time step is calculated using:

$$q'' = -k\frac{\partial T}{\partial x}\bigg|_{x=L} \approx -k\frac{T_4 - T_3}{\Delta x}$$

- For $i = 1$: $q_1'' = -2\frac{0-123.40}{0.005} = 49,361.70$ W/m^2
- For $i = 2$: $q_2'' = 35,454.96$ W/m^2
- For $i = 3$: $q_3'' = 27,875.13$ W/m^2

Part (b): Explicit DuFort-Frankel Method

The DuFort-Frankel scheme is an explicit method that approximates the diffusion term with a central difference both in time and space:

$$\frac{T_i^{n+1} - T_i^{n-1}}{2\Delta t} = \alpha \frac{T_{i+1}^n - (T_i^{n+1} + T_i^{n-1}) + T_{i-1}^n}{(\Delta x)^2}$$

Rearranging for T_i^{n+1} with $r = 1$:

$$3T_i^{n+1} = -T_i^{n-1} + 2(T_{i+1}^n + T_{i-1}^n)$$

In the first step ($n = 0$), since T_i^{-1} values are not known, we apply the FTCS with $r = 0.5$ to ensure stability:

$$T_i^1 = T_i^0 + 0.5(T_{i+1}^0 - 2T_i^0 + T_{i-1}^0)$$

Applying to each node for the first time step:
Node 0 (insulated boundary with $T_{-1}^0 = T_1^0 = 200$):

$$T_0^1 = 200 + 0.5(200 - 2 \cdot 200 + 200) = 200$$

Node 1:

$$T_1^1 = 200 + 0.5(200 - 2 \cdot 200 + 200) = 200$$

Node 2:

$$T_2^1 = 200 + 0.5(200 - 2 \cdot 200 + 200) = 200$$

Node 3:

$$T_3^1 = 200 + 0.5(0 - 2 \cdot 200 + 200) = 100$$

Node 4 (fixed boundary): $T_4^1 = 0$
For the second time step ($n = 1$) using DuFort-Frankel with $r = 1$:
Node 0 (insulated boundary):

$$3T_0^2 = -T_0^0 + 2(T_1^1 + T_1^1) = -200 + 2(200 + 200) = 600 \implies T_0^2 = 200$$

Node 1:

$$3T_1^2 = -T_1^0 + 2(T_2^1 + T_0^1) = -200 + 2(200 + 200) = 600 \implies T_1^2 = 200$$

Node 2:

$$3T_2^2 = -T_2^0 + 2(T_3^1 + T_1^1) = -200 + 2(100 + 200) = 400 \implies T_2^2 = 133.33$$

Node 3:

$$3T_3^2 = -T_3^0 + 2(T_4^1 + T_2^1) = -200 + 2(0 + 200) = 200 \implies T_3^2 = 66.67$$

Node 4 (fixed boundary): $T_4^2 = 0$
Complete temperature distribution:

Node	Position (cm)	$i = 0$ (°C)	$i = 1$ (°C)	$i = 2$ (°C)	$i = 3$ (°C)
0	0.0	200.00	200.00	200.00	200.00
1	0.5	200.00	200.00	200.00	155.56
2	1.0	200.00	200.00	133.33	111.11
3	1.5	200.00	100.00	66.67	55.56
4	2.0	0.00	0.00	0.00	0.00

The heat flux at the boundary:

- For $i = 1$: $q_1'' = 40,000.00$ W/m^2
- For $i = 2$: $q_2'' = 26,666.67$ W/m^2
- For $i = 3$: $q_3'' = 22,222.22$ W/m^2

Comparison of Methods

The Implicit Laasonen method gives a smooth, physically acceptable temperature field with monotonic cooling over the slab, accurately capturing the diffusion of heat and preserving the insulated boundary condition. Its unconditional stability guarantees accurate results, although at a cost of having to solve a tridiagonal system at every time step. In contrast, the Explicit DuFort-Frankel method, although less demanding to calculate, oscillates, especially at the earlier time steps, with temperatures at the insulated boundary staying constant or assuming unrealistic values (e.g., $T_0^2 = 200$°C). This is exacerbated by the use of $r = 1$, which amplifies numerical artifacts. Both algorithms demonstrate decreasing heat flux at the fixed boundary, but the Laasonen approach gives better and more intuitive results at the cost of increased computing time.

Numerical Implementation and Results

The numerical methods were applied and carried out with the below parameters to the initial problem:

- Domain length: $L = 0.02$ m (2 cm)

- Thermal diffusivity: $\alpha = 1 \times 10^{-6}$ m^2/s
- Thermal conductivity: $k = 2$ W/(m $\cdot$ °C)
- Spatial step size: $\Delta x = 0.005$ m (0.5 cm)
- Stability parameter: $r = 1.0$
- Initial temperature: $T_{\text{initial}} = 200$C
- Number of time steps: 3
- Left boundary condition: Insulated ($\partial T/\partial x = 0$ at $x = 0$)
- Right boundary condition: Fixed temperature ($T = 0$C at $x = L$)

Figure 2 illustrates the temperature profiles of both methods at each time step ($i = 0, 1, 2, 3$). The Implicit Laasonen method reveals smooth, monotonically decreasing temperatures throughout the domain, with a smooth cooling trend preserving the insulated boundary condition at $x = 0$. The Explicit DuFort-Frankel method reveals limited cooling, especially at the insulated boundary (node 0, $x = 0\,$cm), where the temperature is kept at a constant value of 200°C throughout all time steps. The interior nodes reveal a lagging cooling response compared to the Laasonen method, reflecting possible numerical inaccuracies due to the use of $r = 1$.

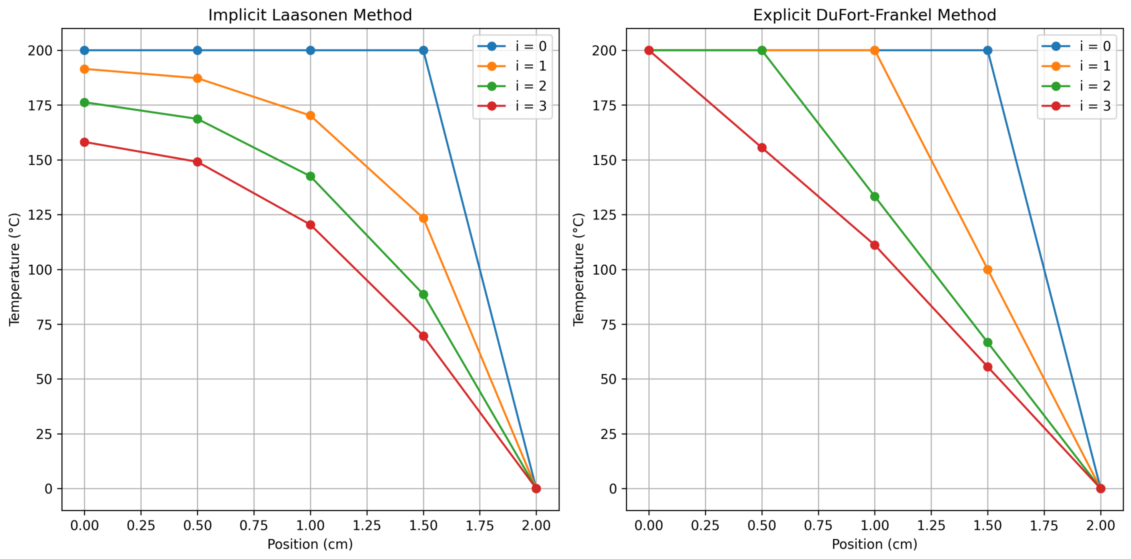

Figure 2: Temperature profiles at various time steps for Implicit Laasonen method (left) and Explicit DuFort-Frankel method (right) with a coarse mesh ($\Delta x = 0.5$ cm) at the first three time steps. The x-axis is the position of the slab in cm, and the y-axis indicates temperature in °C.

For a more comprehensive analysis, a longer simulation with additional spatial and time resolution was performed:

- Spatial step size: $\Delta x = 0.0001$ m (0.01 cm)
- Number of time steps: 1000
- Resulting time step: $\Delta t = 0.01$ seconds

Figure 3 illustrates temperature profiles at a set of time steps (i = 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000) to demonstrate steady-state development. Both the methods with the finer grid converge to the same temperature distributions over time.

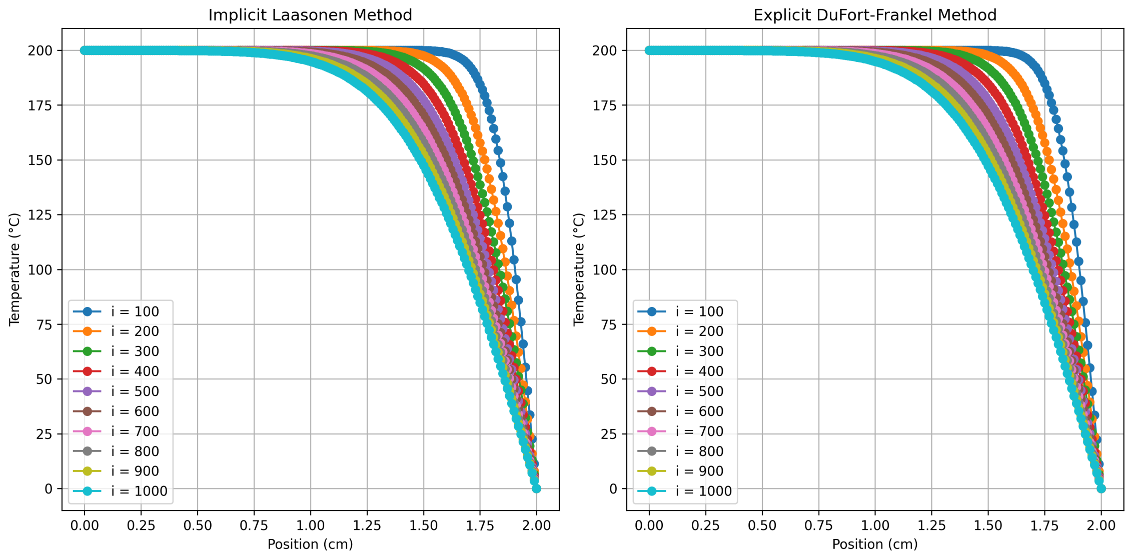

Figure 3: Temperature profiles at different time steps with a more dense spatial grid ($\Delta x = 0.0001$ m) and an extended simulation time (1000 time steps). The Implicit Laasonen (left) and Explicit DuFort-Frankel (right) schemes exhibit convergence towards the same steady-state distribution with the progression in simulation. Every line is a different time step as labeled in the legend.

Additionally, a comparison was made between the Implicit Laasonen method and a Three-Level scheme with a different problem setup (Problem 1):

- Domain length: $L = 10$ m
- Thermal diffusivity: $\alpha = 1$ m^2/s
- Thermal conductivity: $k = 1$ W/(m $\cdot$ °C)
- Initial temperature: $T_{\text{initial}} = 100$C
- Boundary conditions: Fixed temperature ($T = 0$C) at both ends
- Spatial nodes: 50
- Time steps: 1000
- Time step size: $\Delta t = 0.01$ seconds (adjusted based on stability criteria)

Figure 4 indicates a comparison between the two schemes at 10 evenly distributed time points. The two schemes have good agreement with each other, with the Three-Level scheme (plotted with bold dashed lines) closely tracing the Implicit Laasonen solution (plotted with thin solid lines).

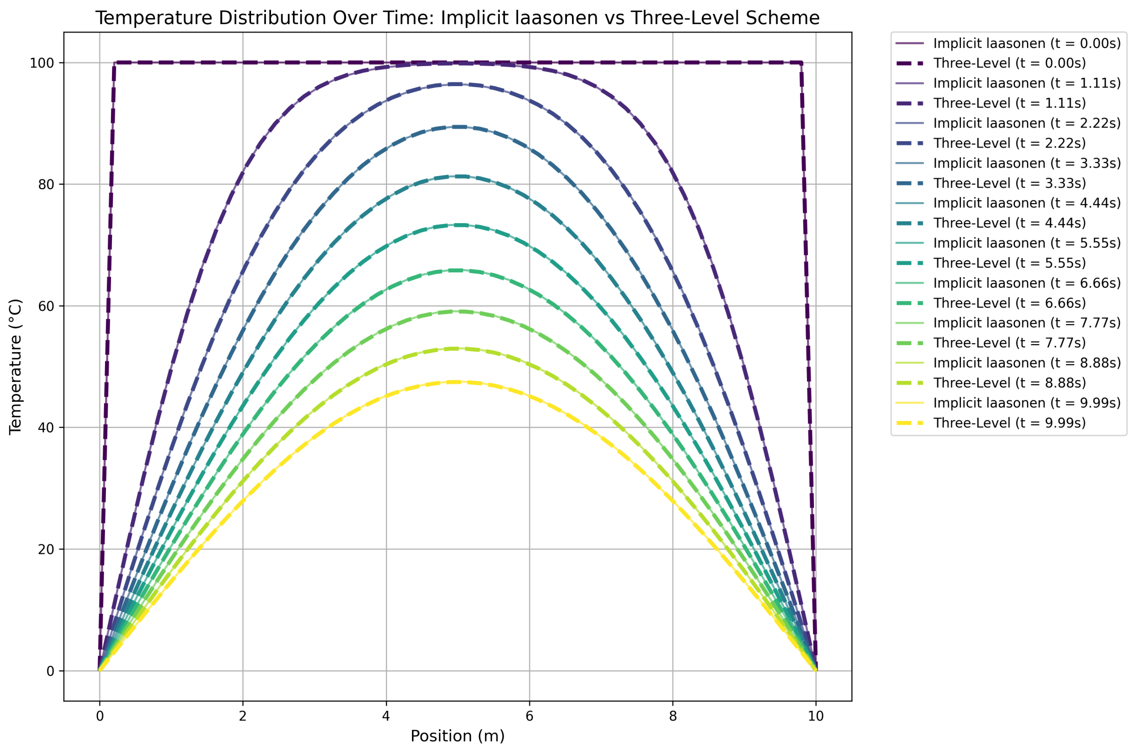

Figure 4: Comparison between Implicit Laasonen method (thin solid lines) and Three-Level scheme (bold dashed lines) for Problem 1 with fixed temperature boundary conditions at both ends. The temperature profiles are shown at 10 evenly spaced time points throughout the simulation. The color gradient corresponds to progression in time, with earlier times in darker colors and later times in lighter colors.

The implementation illustrates that both approaches accurately solve the heat equation but with varying numerical features. The Implicit Laasonen method returns more physically reasonable results with smooth temperature profiles, whereas the DuFort-Frankel method, although faster to compute per time step, oscillates and thus demands appropriate interpretation. As simulation moves to steady-state and finer discretization, the methods' differences reduce, and their convergence to the physics-based solution is affirmed.

The algorithms of both the Implicit Laasonen and Explicit DuFort-Frankel schemes are presented in the Appendix.

Appendix: Numerical Methods and Algorithms

This appendix gives explicit algorithms, implementation remarks, and usage details of the numerical schemes that were used to solve heat equation problems in this work.

Comprehensive Heat Equation Solver

The numerical solution framework established here can accommodate a range of boundary conditions and discretization methods of the one-dimensional heat equation. The implementation is made flexible, modular, and with a user-friendly interface that is easily adoptable to different physical applications.

Supported Features

- **Boundary Conditions:** Fixed temperature (Dirichlet), insulated (Neumann), and specified heat flux
- **Numerical Schemes:** Three-level scheme, Implicit Laasonen method, Explicit DuFort-Frankel method
- **Analysis Capabilities:** Comparison with analytical solutions, heat flux calculation, stability analysis
- **Visualization:** Automated creation of temperature profiles and error analysis plots

Three-Level Scheme Implementation

The three-level scheme discussed in Problem 1 is realized in the form of a general-purpose solver function that accommodates different boundary conditions and model parameters and supports comparison with an analytical solution.

Algorithm 1 Three-Level Scheme for Heat Equation

```
Require: $\alpha, k, L, T_0, N_x, N_t, dt, d$, boundary conditions
Ensure: Temperature distribution $T(x,t)$ and boundary heat fluxes
 1: Calculate $dx = L/(N_x - 1)$ and verify stability
 2: Initialize temperature field $T[:,0] = T_0$ and flux arrays
 3: Apply initial boundary conditions at $t = 0$
 4: Initialize second time level using analytical solution
 5: Apply boundary conditions at $t = dt$ and calculate initial fluxes
 6: for $n = 1$ to $N_t - 2$ do
 7:     for $j = 1$ to $N_x - 2$ do
 8:         $D_j^n = (T_{j-1}^n - 2T_j^n + T_{j+1}^n)/dx^2$
 9:         $D_j^{n-1} = (T_{j-1}^{n-1} - 2T_j^{n-1} + T_{j+1}^{n-1})/dx^2$
10:         $T_j^{n+1} = \frac{1}{1.5}(2T_j^n - 0.5T_j^{n-1} + \alpha \cdot dt \cdot ((1+d)D_j^n - d \cdot D_j^{n-1}))$
11:     end for
12:     Apply boundary conditions based on type:
13:     if left_bc_type = 'fixed' then $T_0^{n+1}$ = left_bc_value
14:     else if left_bc_type = 'insulated' then $T_0^{n+1} = T_1^{n+1}$
15:     end if
16:     Similar logic for right boundary
17:     Calculate boundary heat fluxes
18: end for
19: return $T$, $q_{left}$, $q_{right}$, $x$
```

Algorithm 2 Analytical Solution for Heat Equation

```
Require: $x, t, T_0$, boundary conditions, $\alpha, L, num\_terms$
Ensure: Analytical temperature distribution $T_{analytical}$
 1: Calculate steady-state solution based on boundary types:
 2: if left_bc_type = 'fixed' and right_bc_type = 'fixed' then
 3:     $T_{steady}$ = left_bc_value + (right_bc_value − left_bc_value) $\cdot x/L$
 4: else if left_bc_type = 'insulated' and right_bc_type = 'insulated' then
 5:     $T_{steady} = T_0$ for all $x$
 6: else if left_bc_type = 'insulated' then
 7:     $T_{steady}$ = right_bc_value for all $x$
 8: else
 9:     $T_{steady}$ = left_bc_value for all $x$
10: end if
11: $T_{initial\_adjusted} = T_0 - T_{steady}$ for all $x$
12: Initialize $T_{transient} = 0$ for all $x$
13: for $n = 1, 3, 5, \ldots, 2 \cdot num\_terms - 1$ do
14:     if left_bc_type = 'insulated' then
15:         $b_n = \frac{2}{L} \int_0^L T_{initial\_adjusted} \cdot \cos(n\pi x/L) dx$
16:         $T_{transient} += b_n \cdot \cos(n\pi x/L) \cdot e^{-\alpha(n\pi/L)^2 t}$
17:     else
18:         $b_n = \frac{2}{L} \int_0^L T_{initial\_adjusted} \cdot \sin(n\pi x/L) dx$
19:         $T_{transient} += b_n \cdot \sin(n\pi x/L) \cdot e^{-\alpha(n\pi/L)^2 t}$
20:     end if
21: end for
22: return $T_{steady} + T_{transient}$
```

Function Usage

The three-level scheme is realized as the `heat_equation_three_level_solver` function with the following signature:

```
heat_equation_three_level_solver(alpha=1.0, k=1.0, L=10.0, T0=100.0,
Nx=50, Nt=1000, dt=0.01, d=0.5,
left_bc_type='fixed', right_bc_type='fixed',
left_bc_value=0.0, right_bc_value=0.0,
num_terms=10)
```

A key feature of this implementation is automatic creation of analytical solutions via Fourier series expansion tailored to meet the required boundary conditions.

This allows direct comparison against the numerical results and visualization of both sets of solutions. The accuracy of the analytical solution can be precisely controlled through the `num_terms` parameter, which determines how many terms of the Fourier series are included in the calculation. By increasing this value, users can obtain arbitrary levels of accuracy in the analytical solution, especially at or near discontinuities or early in time steps where temperature gradients are sharpest.

Implicit Laasonen Method

The Implicit Laasonen method is realized as a separate solver function that builds and solves the tridiagonal system of equations at every time step.

Algorithm 3 Implicit Laasonen Method for Heat Equation

```
Require: $L$, $\alpha$, $k$, $dx$, $r$, $T_{initial}$, $time\_steps$, boundary conditions
Ensure: Temperature distribution $T(x,t)$ and heat fluxes
 1: Calculate time step: $dt \leftarrow r \cdot dx^2/\alpha$
 2: Create spatial grid with $nodes = L/dx + 1$ points
 3: Initialize temperature array: $T[:,0] \leftarrow T_{initial}$
 4: Apply initial boundary conditions
 5: Initialize heat flux arrays
 6: Set up tridiagonal matrix $A$:
 7:    Interior: $A[i,i] \leftarrow 1 + 2r$, $A[i,i-1] \leftarrow -r$, $A[i,i+1] \leftarrow -r$
 8: if left BC insulated then
 9:     Ghost node: $T[-1,n+1] \leftarrow T[1,n+1]$
10:     $A[0,0] \leftarrow 1 + 2r$, $A[0,1] \leftarrow -2r$, $A[0,2:] \leftarrow 0$
11: else
12:     Set $A[0,:]$ for fixed/flux BC
13: end if
14: if right BC insulated then
15:     Ghost node: $T[nodes,n+1] \leftarrow T[nodes-2,n+1]$
16:     $A[nodes-1,nodes-1] \leftarrow 1+2r$, $A[nodes-1,nodes-2] \leftarrow -2r$, $A[nodes-1,0:nodes-2] \leftarrow 0$
17: else
18:     Set $A[nodes-1,:]$ for fixed/flux BC
19: end if
20: for $n = 1$ to $time\_steps$ do
21:     Set up right-hand side: $b \leftarrow T[:,n-1]$
22:     Modify $b$ for fixed/flux BCs
23:     Solve system: $A \cdot T[:,n] = b$
24:     Calculate heat fluxes at boundaries
25: end for
26: return $T$, boundary heat fluxes, spatial grid $x$, time step $dt$
```

Function Usage

The Implicit Laasonen method is implemented as the `laasonen_solver` function:

```
laasonen_solver(L, alpha, k, dx, r, T_initial, time_steps,
left_bc_type, right_bc_type,
left_bc_value=None, right_bc_value=None)
```

Explicit DuFort-Frankel Method

Explicit DuFort-Frankel method implementation involves an initialization stage via the FTCS method to maintain stability.

Algorithm 4 DuFort-Frankel for Heat Equation

```
Require: $L$, $\alpha$, $k$, $dx$, $r$, $T_0$, $N_t$, BCs
Ensure: $T(x,t)$, heat fluxes
 1: $dt \leftarrow r \cdot dx^2/\alpha$, $r_f \leftarrow 0.5$
 2: Grid: $nodes \leftarrow L/dx + 1$, $x \leftarrow [0, L]$
 3: $T[:,0] \leftarrow T_0$, apply BCs, init. flux arrays
 4: FTCS step ($n = 1$, $r_f = 0.5$):
 5: for $i = 1$ to $nodes - 2$ do
 6:     $T[i,1] \leftarrow T[i,0] + r_f(T[i+1,0] - 2T[i,0] + T[i-1,0])$
 7: end for
 8: if left BC insulated then
 9:     Ghost node: $T[-1,0] \leftarrow T[1,0]$
10:     $T[0,1] \leftarrow T[0,0] + r_f(T[1,0] - 2T[0,0] + T[-1,0])$
11: else
12:     Apply left BC (fixed/flux)
13: end if
14: Apply right BC, compute fluxes
15: DuFort-Frankel steps:
16: for $n = 2$ to $N_t$ do
17:     for $i = 1$ to $nodes - 2$ do
18:         $T[i,n] \leftarrow \frac{(1-2r)T[i,n-2]+2r(T[i+1,n-1]+T[i-1,n-1])}{1+2r}$
19:     end for
20:     if left BC insulated then
21:         Ghost node: $T[-1,n-1] \leftarrow T[1,n-1]$
22:         $T[0,n] \leftarrow \frac{(1-2r)T[0,n-2]+2r(T[1,n-1]+T[-1,n-1])}{1+2r}$
23:     else
24:         Apply left BC
25:     end if
26:     Apply right BC, compute fluxes
27: end for
28: return $T$, fluxes, $x$, $dt$
```

Function Usage

The Explicit DuFort-Frankel method is implemented as the `dufort_frankel_solver` function:

```
dufort_frankel_solver(L, alpha, k, dx, r, T_initial, time_steps,
left_bc_type, right_bc_type,
left_bc_value=None, right_bc_value=None)
```

Boundary Condition Implementation

In Problem 1, the boundary conditions are implemented without ghost nodes:

- **Fixed (Dirichlet) boundaries:** Temperature values are directly assigned to boundary nodes
- **Insulated (Neumann) boundaries:** For $\frac{\partial T}{\partial x} = 0$, we set $T_0 = T_1$ (first-order approximation)
- **Flux boundaries:** For prescribed flux q'' where $-k\frac{\partial T}{\partial x} = q''$, we set $T_0 = T_1 + \frac{q'' \cdot \Delta x}{k}$

Boundary values are updated after each interior node calculation. This direct method is simple and yet guarantees stability of the solution and physical validity.

In problem 2, both the Laasonen and DuFort-Frankel schemes apply ghost nodes to the boundary conditions:

- **Insulated boundary at $x = 0$:** Using a fictitious node at $x = -\Delta x$ with $T_{-1}^n = T_1^n$
  - Laasonen method: $(1 + 2r)T_0^{n+1} - 2rT_1^{n+1} = T_0^n$
  - DuFort-Frankel method: $T_0^n = \frac{(1-2r)T_0^{n-2} + 4rT_1^{n-1}}{1+2r}$
- **Flux boundary at $x = 0$:** First-order approximation
  - Laasonen method: Modifies matrix with $A[0, 0] = 1 + r$, $A[0, 1] = -r$
  - DuFort-Frankel method: $T_0^n = T_1^n + \frac{q_{\text{left}} \cdot \Delta x}{k}$

These approaches guarantee numerical accuracy and proper temperature distribution within the physical field.

User Guide

In order to use the numerical solvers within this package, do the following:

1. **Choose a Numerical Method:**
   - `heat_equation_three_level_solver`: Three-level scheme with analytical solution comparison
   - `laasonen_solver`: Implicit Laasonen method with tridiagonal matrix solution
   - `dufort_frankel_solver`: Explicit DuFort-Frankel method with FTCS initialization
2. **Define Physical Parameters:**
   - Domain length L (m)
   - Thermal diffusivity α (m^2/s)
   - Thermal conductivity k ($\mathrm{W}/(\mathrm{m}\cdot{}^{\circ}\mathrm{C})$)
   - Initial temperature distribution $T_{initial}$ (°C) - can be uniform value or array
3. **Set Discretization Parameters:**
   - Spatial step size Δx (m) or number of grid points N_x
   - Time step size Δt (s) or stability parameter $r = \alpha \Delta t / \Delta x^2$
   - Number of time steps N_t
   - For three-level scheme: parameter d (default 0.5) controlling weighting of current vs. previous time steps
   - For analytical solution: parameter `num_terms` (default 10) controlling Fourier series precision
   - For DuFort-Frankel method: FTCS initialization parameter r_{FTCS} (default 0.5) can be modified inside the function to adjust first time step stability
4. **Specify Boundary Conditions:**
   - Left boundary type: 'fixed' (Dirichlet), 'insulated' (Neumann), or 'flux'
   - Right boundary type: 'fixed', 'insulated', or 'flux'

- Left boundary value: temperature for 'fixed', heat flux for 'insulated'/'flux'
- Right boundary value: temperature for 'fixed', heat flux for 'insulated'/'flux'

5. **Execute the Solver Function:**

```
T, q_left, q_right, x, dt = chosen_solver(parameters)
```

   The returned values are the temperature matrix distribution, heat fluxes at boundaries, spatial grid, and time step modified (if stability requires adjustment).

6. **Analyze and Visualize Results:**
   - Use `print_table` to display tabulated temperature and flux results
   - Create plots of temperature profiles using matplotlib
   - For the three-level solver, compare numerical results with the built-in analytical solution
   - Calculate error metrics between numerical and analytical solutions if needed

The solvers also test numerically for stability and correct time steps where necessary. Initialization for the DuFort-Frankel method employs the Forward-Time Central-Space (FTCS) scheme with a standard stability coefficient of $r_{FTCS} = 0.5$. This parameter can be changed directly in the function to enhance the stability of the initialization process in response to particular classes of problems. The three-level solver also includes integrated visualization of numerical vs. exact results, suitable for both verification and validation of the implementation.

The complete implementation, including all numerical methods and visualization scripts used in this analysis, is available on GitHub at `https://github.com/KiaraGholizad/Computational_Fluid_Dynamics`.