Massachusetts Institute of Technology

Project Report

18.0851
Computational Science and Engineering I

Written by
Jered Dominguez-Trujillo

Submitted to:
Prof. Matthew Durey

May 13, 2019

1 Problem Description

Three numerical methods - Explicit Euler, Implicit Euler, and Crank-Nicolson - were developed to solve the one-dimensional heat equation and analyzed with respect to their stabilities and respective errors. As this is an analysis of numerical methods rather than an engineering solution, all values will be presented as dimensionless.

One-Dimensional Heat Equation
To correctly simulate unsteady heat transfer in one-dimension numerically, the one-dimensional Heat Equation (Eq. 1) must be discretized and solved.

$$\frac{\partial u}{\partial t} = K\frac{\partial^2 u}{\partial x^2} + q(x,t) \tag{1}$$

Where u represents the temperature, K the thermal diffusivity, and $q(x,t)$ a heat source function. The variables x and t represent space and time, respectively.

Assumptions
To simplify the numerical methods, the following assumptions were made:
1. Uniform Grid Spacing (Δx)
2. Uniform Time Steps (Δt)

Initial Conditions
The initial condition will be assumed that the temperature field, $u(x, t = 0)$, is zero at all points in the domain, per Eq. 2.

$$u(x, t = 0) = 0 \tag{2}$$

Boundary Conditions
The boundary conditions were defined as "fixed-free" with the fixed (Dirilecht) boundary condition applied at $x = 0$ and the free (Neumann) boundary condiiton applied at $x = L$ per Eq. 3 and Eq. 4, respectively.

Dirilecht Condition at the Left Boundary x = 0:

$$u(x = 0, t) = \begin{cases} C_1 * \frac{t}{t_{ramp}} & t \leq t_{ramp} \\ C_1 & t > t_{ramp} \end{cases} \tag{3}$$

Neumann Condition at the Right Boundary x = L:

$$\frac{\partial u}{\partial x}(x = L, t) = C_2 \tag{4}$$

Heating Source Function
The source function, ($q(x,t)$ in Eq. 1) is defined as sinusoidal in time, with frequency set by ω and amplitude set by Q_{max}, and by a normal distribution in space, with a peak at $\frac{L}{2}$ (analog to the mean in a normal distribution) and standard deviation, σ, per Eq. 5.

$$q(x,t) = -Q_{max} * sin(\omega t) * exp\left(\frac{-(x - \frac{L}{2})^2}{\sigma^2}\right) \tag{5}$$

Parameters
The Heat Equation (Eq. 1) was solved on a domain of length $L = 2*\pi$, with a Thermal Diffusivity constant of $K = 0.1$.

The Dirilecht Boundary Condition on the left ($x = 0$) was set by the constant $C_1 = 1$, and was increased linearly from the initial condition $u(x = 0, t = 0) = 0$ to C_1 over a specified time as $t_{ramp} = 1$. This resulted in the boundary condition described by Eq. 3.

The Nuemann Boundary Condition ($\frac{\partial u}{\partial x}$) on the right ($x = L$) was set by the constant $C_2 = -0.2$. This resulted in the boundary condition described by Eq. 4.

The source function (Eq. 5) is defined to have a maximum amplitude of $Q_{max} = 1$, a frequency of $\omega = q\frac{rad}{s}$ and a distribution (standard deviation) of $sigma = 0.2 * L$.

A table summarizing important parameters used is presented below.

Table 1: Heat Equation Parameters and Boundary Conditions

Length of Domain [L]	2π
Left Boundary Condition [C_1]	1
Right Boundary Condition [C_2]	-0.2
Thermal Diffusivity [K]	0.1
Time Ramp [t_{ramp}]	1
Maximum Heat Source [Q_{max}]	1
Frequency [ω]	$1\frac{rad}{s}$
Distribution of Heat Source [σ]	$0.2 * L$

2 Finite Difference Schemes

For Eq. 6 and Eq. 7, at time t, we define the following $u^{(n)} = \frac{\partial^n u}{\partial x^n}$

First-Order Finite Difference: Central Difference Scheme
The Central Difference Scheme for First-Order Finite Differences is used to apply the Nuemann boundary condition (Eq. 4) using a ghost node located at $L + dx$. The scheme is defined as per Eq. 6

$$\boxed{u^{(1)} = \frac{u(x + \Delta x) - u(x - \Delta x)}{2\Delta x} + O(\Delta x^2)} \tag{6}$$

Second Order Finite Difference
The Second-Order Finite Difference Scheme is used to discretize the spatial term $\frac{\partial^2 u}{\partial x}$ for use in the Explicit Euler, Implicit Euler, and Crank-Nicolson solvers. The scheme is defined as per Eq. 7.

$$\boxed{u^{(2)}(x) = \frac{u(x + \Delta x) - 2u(x) + u(x - \Delta x)}{(\Delta x)^2} + O(\Delta x^2)} \tag{7}$$

3 Explicit Euler

The Explicit Euler scheme is defined as the following, where we let n denote the time step, and i denote the grid point:

$$\frac{u_{n+1} - u_n}{\Delta t} = f(t_n, u_n) \tag{8}$$

Where the temperature across the domain at the next time step is calculated directly from the temperature field across the domain at the current time step as seen in Figure 1.

Expanding Eq. 8, where $f(t_n, u_n)$ is given by the one-dimensional heat equation (Eq. 1) gives:

$$u_{n+1,i} - u_{n,1} = \Delta t\big[K\big(\frac{u_n(x+\Delta x) - 2u_n(x) + u_n(x-\Delta x)}{(\Delta x)^2}\big) + q_n(x)\big]_i \tag{9}$$

Where

$$u_n(x+\Delta x) = u_{n,i+1}, \qquad u_n(x) = u_{n,i}, \qquad u_n(x-\Delta x) = u_{n,i-1} \tag{10}$$

And we define the CFL Number to be:

$$CFL = K\frac{\Delta t}{(\Delta x)^2} \tag{11}$$

This gives the explicit final numerical solution (Eq. 12) for the temperature field at the next $(n+1)$ timestep, given the tempreature field at the current (n) timestep.

$$\boxed{u_{n+1,i} = CFL * u_{n,i+1} + (1-2CFL) * u_{n,i} + CFL * u_{n,i-1} + q_{n,i}\Delta t} \tag{12}$$

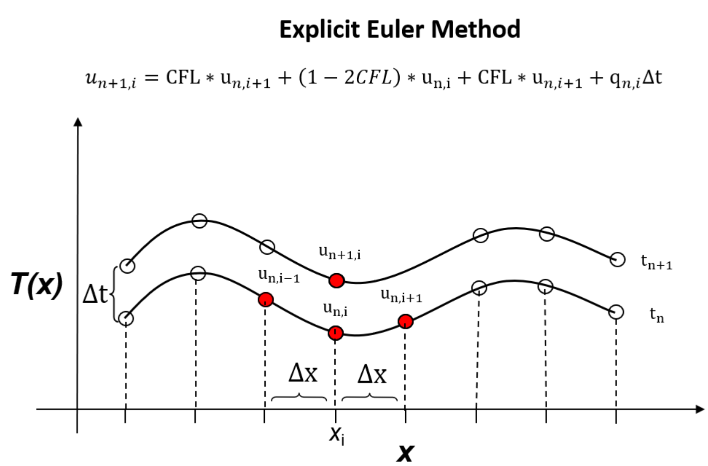


Figure 1: Explicit Euler Scheme Template

Boundary Conditions
1) To satisfy the Dirilecht Boundary Condition, the left node is simply set according to Eq. 3.

$$\boxed{u_{n+1,1} = \begin{cases} C_1 * \frac{t}{t_{ramp}} & t \leq t_{ramp} \\ C_1 & t > t_{ramp} \end{cases}} \tag{13}$$

2) To satisfy the Neumann Boundary Condition, the First-Order Finite Difference Central Difference Scheme (Eq. 6) is applied with a ghost node to obtain Eq. 14, where the ghost node is represented by m, the boundary node by $m-1$ etc.

$$\boxed{u_{n+1,m} = u_{n+1,m-2} + 2\Delta X * C_2} \tag{14}$$

4 Implicit Euler

The Implicit Euler scheme is defined as the following, where we let n denote the time step, and i denote the grid point:

$$u_{n+1} = u_n + \Delta t * f(t_{n+1}, u_{n+1}) \tag{15}$$

Expanding Eq. 15 gives:

$$u_{n+1,i} = u_{n,i} + \Delta t\big[K\big(\frac{u_{n+1}(x+\Delta x) - 2u_{n+1}(x) + u_{n+1}(x-\Delta x)}{\Delta x^2}\big) + q_n(x)\big]_i \tag{16}$$

Note that Eq. 10 still applies and the CFL number is defined as in Eq. 11. After some algebra, we get the following equation (Eq. 17) for the Implicit Euler Scheme.

$$\boxed{-CFL * u_{n+1,i+1} + (2CFL+1) * u_{n+1,i} - CFL * u_{n+1,i-1} = u_{n,i} + q_{n,i}\Delta t} \tag{17}$$

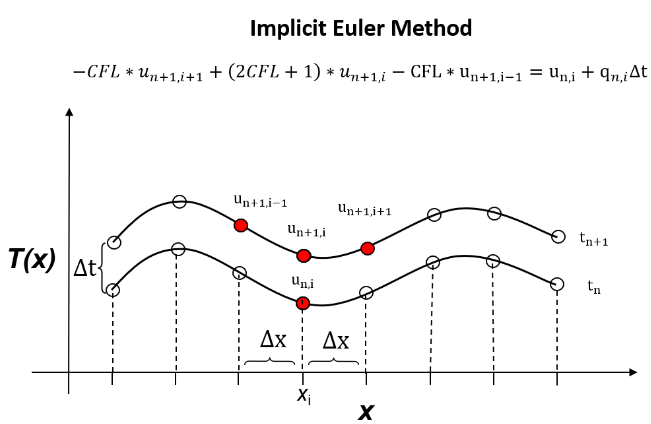


Figure 2: Implicit Euler Scheme Template

Boundary Conditions

1) To satisfy the Dirilecht Boundary Condition, Eq. 3 is inserted into the Implicit Euler scheme (Eq. 15) at the left boundary node to obtain the Eq 18, which is then solved simultaneously with the remaining equations. Note that f_{BC1} represents the value of the left boundary condition as defined by Eq. 3.

$$\boxed{(2CFL+1)*u_{n+1,2} - CFL*u_{n+1,3} = u_{n,2} + q_{n,2}\Delta t + CFL*f_{BC1}} \tag{18}$$

Additionally, the boundary node is defined as in Eq. 13, reproduced below as Eq. 19

$$\boxed{u_{n+1,1} = \begin{cases} C_1 * \frac{t}{t_{ramp}} & t \leq t_{ramp} \\ C_1 & t > t_{ramp} \end{cases}} \tag{19}$$

2) To satisfy the Neumann Condition, the First-Order Finite Difference Central Difference Scheme (Eq. 6) is applied with a ghost node to obtain Eq. 20 by plugging in Eq. 14 into Eq. 17, where the ghost node is represented by m, the boundary node by $m-1$ etc.

$$\boxed{-2CFL*u_{n+1,m-2} + (1+2CFL)*u_{n+1,m-1} = u_{n,m-1} + q_{n,m-1}\Delta t + 2CFL*C_2\Delta x} \tag{20}$$

The ghost node is then set according to Eq. 14, reproduced below as Eq. 21.

$$\boxed{u_{n+1,m} = u_{n+1,m-2} + 2\Delta X * C_2} \tag{21}$$

The final matrix equation which gives the temperatures for all the internal nodes at the next timestep is then fully defined per Eq. 22, and the boundary conditions applied as in Eq. 19 and Eq. 21.

$$\begin{bmatrix} 2*CFL+1 & -CFL & 0 & \dots & \dots & 0 \\ -CFL & 2*CFL+1 & -CFL & \ddots & \ddots & \vdots \\ 0 & -CFL & 2*CFL+1 & -CFL & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -CFL & 2CFL+1 & -CFL \\ 0 & \dots & \dots & 0 & -2*CFL & 2*CFL+1 \end{bmatrix} \begin{bmatrix} u_{n+1,2} \\ u_{n+1,3} \\ u_{n+1,4} \\ \vdots \\ u_{n+1,m-2} \\ u_{n+1,m-1} \end{bmatrix} = \begin{bmatrix} u_{n,2}+q_{n,2}\Delta t+CFL*f_{BC1} \\ u_{n,3}+q_{n,3}\Delta t \\ u_{n,4}+q_{n,4}\Delta t \\ \vdots \\ u_{n,m-2}+q_{n,m-2}\Delta t \\ u_{n,m-1}+q_{n,m-1}\Delta t+2CFL*C_2\Delta x \end{bmatrix} \tag{22}$$

5 Crank-Nicolson

The Crank-Nicolson scheme is defined as the following, where we let n denote the time step, and i denote the grid point

$$u_{n+1,i} = u_{n,i} + \Delta t * \Big[\frac{1}{2} f(t_n, u_n) + \frac{1}{2} f(t_{n+1}, u_{n+1}) + q_n\Big]_i \tag{23}$$

Substituting in the heat equation (Eq. 1),

$$u_{n+1,i} = u_{n,i} + \Delta t * \Big[\frac{1}{2}\big(K\frac{\partial^2 u_n}{\partial x^2}\big) + \frac{1}{2}\big(K\frac{\partial^2 u_{n+1}}{\partial x^2}\big) + q_n\Big]_i \tag{24}$$

And then discretizing,

$$\begin{aligned} u_{n+1,i} = u_{n,i} + \Delta t * \Big[&\frac{1}{2}\big(K\frac{u_n(x+\Delta x) - 2u_n(x) + u_n(x-\Delta x)}{\Delta x^2}\big) + \\ &\frac{1}{2}\big(K\frac{u_{n+1}(x+\Delta x) - 2u_{n+1}(x) + u_{n+1}(x-\Delta x)}{\Delta x^2}\big) + q_n\Big]_i \end{aligned} \tag{25}$$

In addition to Eq. 10, the following (Eq. 26) are applied to Eq. 25.

$$u_{n+1}(x+\Delta x) = u_{n+1,i+1}, \qquad u_{n+1}(x) = u_{n+1,i}, \qquad u_{n+1}(x-\Delta x) = u_{n+1,i-1} \tag{26}$$

We still define the CFL number as in Eq. 11. After some algebra, we get the following equation (Eq. 27) for the Implicit Euler Scheme.

$$\boxed{\begin{aligned} &\frac{-CFL}{2} * u_{n+1,i-1} + (1+CFL) * u_{n+1,i} - \frac{CFL}{2} * u_{n+1,i+1} = \\ &\frac{CFL}{2} * u_{n,i-1} + (1-CFL) * u_{n,i} + \frac{CFL}{2} * u_{n,i+1} + q_{n,1}\Delta t \end{aligned}} \tag{27}$$

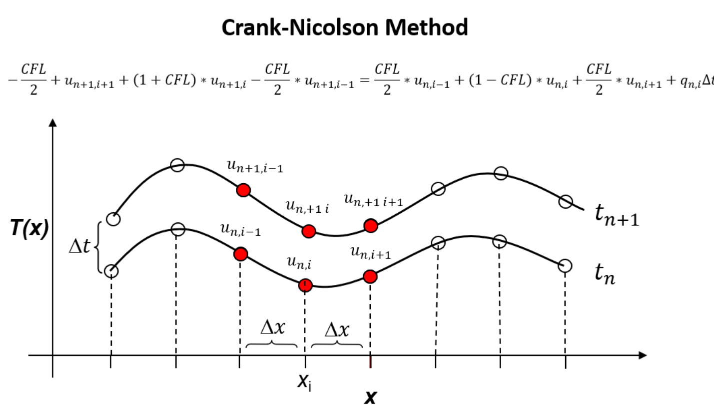


Figure 3: Crank-Nicolson Scheme Template

Boundary Conditions

1) To satisfy the Dirilecht Boundary Condition, Eq. 3 is inserted into the Crank-Nicolson scheme (Eq. 25) at the left boundary node to obtain Eq. 28., which is then solved simultaneously with the remaining equations. Again, note that f_{BC1} represents the value of the left boundary conditions as defined by Eq. 3.

$$\boxed{\begin{aligned}(1+CFL)*u_{n+1,i} - \frac{CFL}{2}*u_{n+1,i+1} = \\ \frac{CFL}{2}*u_{n,i-1} + (1-CFL)*u_{n,i} + \frac{CFL}{2}*u_{n,i+1} + q_{n,1}\Delta t + \frac{CFL}{2}*f_{BC1}\end{aligned}} \tag{28}$$

Additionally, the boundary node is defined as in Eq. 13, reproduced below as Eq. 29

$$\boxed{u_{n+1,1} = \begin{cases} C_1 * \frac{t}{t_{ramp}} & t \le t_{ramp} \\ C_1 & t > t_{ramp} \end{cases}} \tag{29}$$

2) To satisfy the Neumann Condition, the First-Order Finite Difference Central Difference Scheme (Eq. 6) is applied with a ghost node to obtain Eq. 30 by plugging in Eq. 14 into Eq. 25, where the ghost nodes is represented by m, the boundary node by $m-1$ etc.

$$-CFL*u_{n+1,i-1} + (1+CFL)*u_{n+1,i} = \tag{30}$$

$$\frac{CFL}{2}*u_{n,i-1} + (1-CFL)*u_{n,i} + \frac{CFL}{2}*u_{n,i+1} + q_{n,1}\Delta t + \frac{CFL}{2}*\big(2C_2\Delta x\big) \tag{31}$$

The ghost node is then set according to Eq. 14, reproduced below as Eq. 32

$$\boxed{u_{n+1,m} = 2C_2\Delta x + u_{n+1,m-2}} \tag{32}$$

The final matrix equation which gives the temperatures for all the internal nodes at the next timestep if then full defined per Eq. 33, and the boundary conditions applied as in Eq. 29 and Eq. 32.

$$\begin{bmatrix} CFL+1 & -\frac{CFL}{2} & 0 & \dots & \dots & 0 \\ -\frac{CFL}{2} & CFL+1 & -\frac{CFL}{2} & \ddots & \ddots & \vdots \\ 0 & -\frac{CFL}{2} & CFL+1 & -\frac{CFL}{2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -\frac{CFL}{2} & CFL+1 & -\frac{CFL}{2} \\ 0 & \dots & \dots & 0 & -CFL & CFL+1 \end{bmatrix} \begin{bmatrix} u_{n+1,2} \\ u_{n+1,3} \\ u_{n+1,4} \\ \vdots \\ u_{n+1,m-2} \\ u_{n+1,m-1} \end{bmatrix} = \begin{bmatrix} \frac{CFL}{2}*u_{n,1}+(1-CFL)*u_{n,2}+\frac{CFL}{2}*u_{n,3}+q_{n,2}\Delta t+CFL*f_{BC1} \\ \frac{CFL}{2}*u_{n,2}+(1-CFL)*u_{n,3}+\frac{CFL}{2}*u_{n,4}+q_{n,3}\Delta t \\ \frac{CFL}{2}*u_{n,3}+(1-CFL)*u_{n,4}+\frac{CFL}{2}*u_{n,5}+q_{n,4}\Delta t \\ \vdots \\ \frac{CFL}{2}*u_{n,m-3}+(1-CFL)*u_{n,m-2}+\frac{CFL}{2}*u_{n,m-1}+q_{n,m-2}\Delta t \\ \frac{CFL}{2}*u_{n,m-2}+(1-CFL)*u_{n,m-1}+\frac{CFL}{2}*u_{n,m}+q_{n,m-1}\Delta t+CFL*C_2\Delta x \end{bmatrix} \tag{33}$$

6 Stability

A stability analysis indicates that the Explicit Euler Method must satisfy Von Neumann stability, defined below in Eq. 34, making the method conditionally stable. However, the Implicit Euler and Crank-Nicolson methods are unconditionally stable, and do not need to satisfy this condition.

$$CFL = K\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{34}$$

This stability behavior was verified by running each method multiple times with a domain length of $L = 2 * \pi$ over a time period of $T_{max} = 40s$. Each run focused on changing the number of spacial nodes (NX) or the number of time steps (NT) in order to change the spacial resolution (Δx) or the temporal resolution (Δt) systematically in order to slowly increase the Courant number (CFD) defined in Eq. 11. As seen in Table 2, the Explicit Euler method is only stable for Courant numbers (CFL) less than $\frac{1}{2}$, while the Implicit Euler and Crank-Nicolson methods are unconditionally stable at all Courant numbers (CFL).

The fact that the Explicit Euler method is only conditionally stable for small Courant numbers, indicates that to resolve a fine spatial mesh (large NX) with this method would require an even finer timestep (since Courant number scales with $\frac{\Delta t}{\Delta x^2}$), resulting in slow computation. For computations where it is desired to investigate the steady-state behavior and temperature profiles over a longer period of time, this makes the Implicit Euler and Crank-Nicolson methods attractive in this regard as well, in addition to their unconditional stability.

Table 2: Explicit Euler Method Stability Tests

Nodes NX	Time Steps NT	K	Δt	Δx	CFL	Explicit Euler Stability	Implicit Euler Stability	Crank-Nicolson Stability
20	200	0.1	0.20	0.314	0.203	Stable	Stable	Stable
20	100	0.1	0.40	0.314	0.405	Stable	Stable	Stable
20	50	0.1	0.80	0.314	0.811	Unstable	Stable	Stable
50	50	0.1	0.80	0.126	5.066	Unstable	Stable	Stable
100	50	0.1	0.80	0.063	20.26	Unstable	Stable	Stable

7 Error

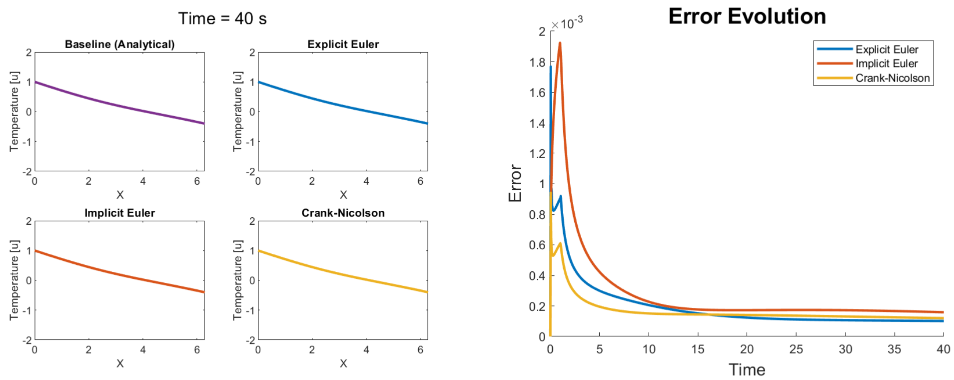
7.1 Error Evolution Over Time

To visualize and understand the accuracy of the three methods and their implementation, each was ran and compared to an "analytical" case (performed using Crank-Nicolson with small timestep and high spacial resolution). This was done for the case with no source function ($q(x,t) = 0$) and for the case with source function ($q(x,t)$) defined by Eq. 5. The solutions were then examined at steady-state (defined at $T = 40s$). Additionally, the L-2 Norm error (compared to the "analytical" solution and defined by Eq. 35) at each time step was calculated and plotted over time to visualize how the error grows for each method as the solution evolves further from initial conditions.

$$Error = \sqrt{\sum_{j=1}^{n} \frac{R_j^2}{N_X}} = \sqrt{\sum_{j=1}^{n} \frac{(U_j - U_{j,analytical})^2}{N_X}} \tag{35}$$

Note: NX represents the number of spatial nodes and was used to normalize the residual

When no source term is present, the steady-state solution is simply a line with y-intercept $= C_1 = 1$ and slope $C_2 = -0.2$. As seen in Figure 4a, all three methods converge to this steady-state solution in agreement with the pseudo-analytical solution obtained with fine spatial and temporal resolutions. All three methods have an initial spike in error due to the left boundary condition increasing from 0 to C_1 over the time $t = 0 \rightarrow t = t_{ramp}$ as defined in Eq. 3, and then the error decreases as the solution approaches a steady-state (Figure 4b).

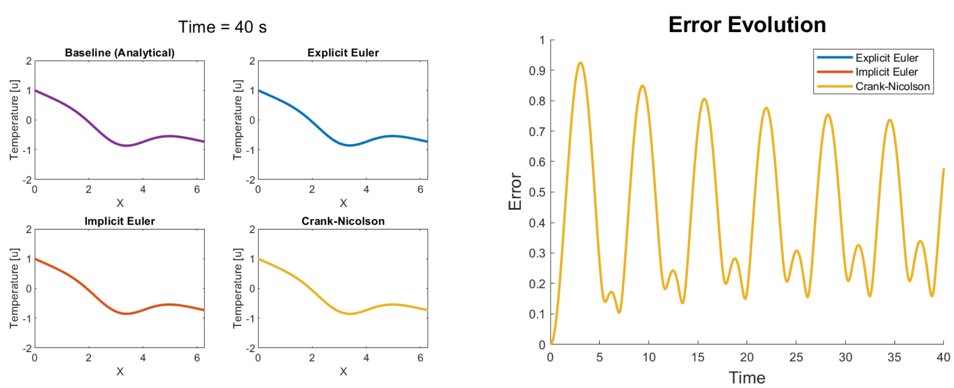


(a) Comparison of Temperature Profiles at T = 40s For Each Method

(b) Error Evolution Over Time ($T = 0s$ to $T = 40s$)

Figure 4: Error Comparison and Evolution for No Source Function Case

When a source term is present ($q(x,t)$ defined by Eq.5), the steady-state solution (again defined as $T = 40s$) is the downward sloping temperature profile seen in Figure 4a now superimposed by the temporal oscillation due to the source term. As seen in Figure 5a, all three methods converge closely to the pseudo-analytical solution at $T = 40s$, and satisfy the imposed boundary conditions, verifying the accuracy of the numerical model with the addition of an arbitrary source function. Interestingly, the error of all three methods oscillates with the angular frequency of the source function ($1\frac{rad}{s}$) and similar errors over time are observed for all three methods (Figure 5b). However, the error is observed to gradually decrease as the solution approaches steady-state.



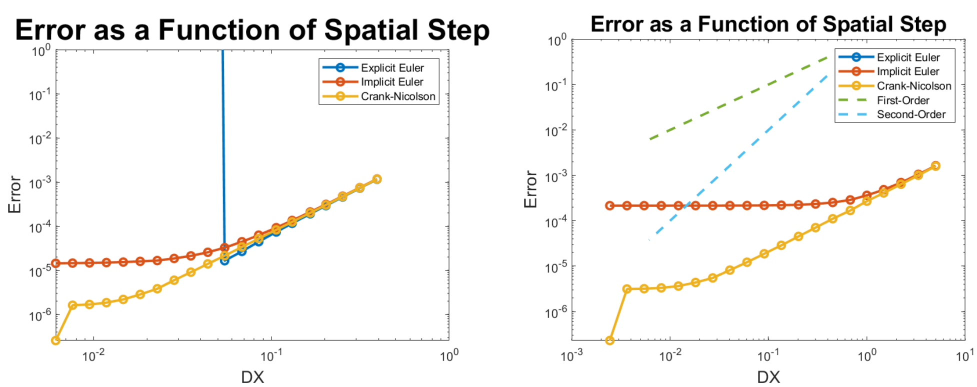
(a) Comparison of Temperature Profiles at T = 40s For Each Method

(b) Error Evolution Over Time ($T = 0s$ to $T = 40s$)

Figure 5: Error Comparison and Evolution for Heat Source Function ($q(x,t)$) Case

7.2 Spatial Resolution

The error (as defined by the L-2 norm in Eq. 35) of the three methods as a function of changing spatial resolution was investigated by again calculating a pseudo-analytical solution with extremely fine spatial and temporal resolution using the Crank-Nicolson method, and comparing the results of each method with varying number of spatial nodes. The comparison was performed at the steady-state ($T = 40s$) and was performed for both the case without a source term (Figure 6a) and the case with the source term defined as in Eq. 5 (Figure 6b). Each case was run at 20 different spatial node counts, varying from 8 nodes to 256 nodes ($2^3 - 2^8\ nodes\ \rightarrow \Delta x = 7.9 * 10^{-1}\ - 2.5 * 10^{-2}$), and a constant number of timesteps ($2^{12} = 4096\ timesteps \rightarrow \Delta t = 9.77 * 10^{-3} s$) so that the error could be investigated as solely a function of changing spatial step size. The results are presented on logarithmic axes below in Figure 6. For comparison, the first-order and second-order curves with respect to spatial resolution (Δx) were additionally plotted.



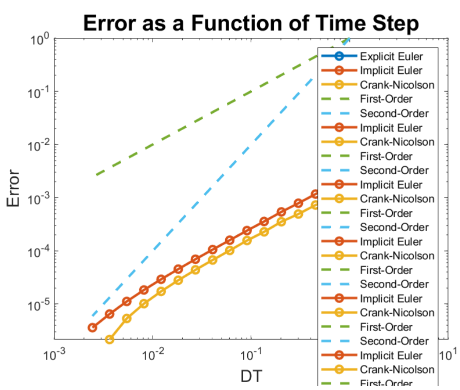
(a) Error with No Source Term **(b)** Error with Source Term

Figure 6: Error as a Function of Spatial Step

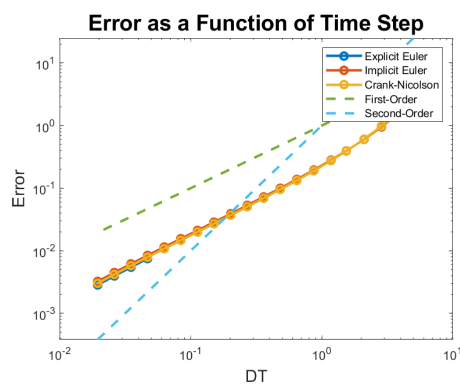
Since the Heat Equation (Eq. 1) was discretized spatially using the Second-Order Central Difference Scheme (Eq. 7) for all three methods, analysis of the truncation error from the Taylor expansion indicates that the error should be of order O(Δx^2). This would translate to a slope of 2 on a log-log plot, and the matching result can be seen above for both the no source and source case in Figure 6 for all three methods, confirming that all three methods have error of order O(Δx^2).

7.3 Temporal Resolution

The error (again defined by Eq. 35) of the three methods was next calculated as a function of changing temporal resolution using the psuedo-analytical solution calculated before. The comparison was performed again at steady-state ($T = 40s$) and was performed for both the case without a source term (Figure 7a) and with a source term as defined in Eq. 5 (Figure 7b). Each case was again run 20 times, this time at various timestep counts, which varied from 8 timesteps to 2048 timesteps ($2^3\ timesteps - 2^{11}\ timesteps\ \rightarrow \Delta t = 5s - 1.9 * 10^{-1}s$) and a constant number of spatial nodes ($2^6\ nodes\ \rightarrow \Delta x = 9.8 * 10^{-1}$). A finer spatial resolution was not chosen so that the Von Neumann stability criteria (Eq. 34) would be able to be satisfied for the Explicit method runs (a very fine mesh would require an unrealistically small timestep for the Explicit method to remain stable for any of the 20 runs). However, this spatial resolution was maintained as constant for all runs so that the error as a function of solely temporal resolution could be investigated. The results are presented on logarithmic axes below in Figure **??**. For comparison, the first-order and second-order curves with respect to temporal resolution (Δt) were additionally plotted.



(a) Error as a Function of Temporal Step with No Source Term



(b) Error as a Function of Temporal Step with Source Term

Since the Heat Equation (Eq. 1) was discretized temporally using a simple First-Order Forward-Difference Scheme

8 Conclusions

9 Appendix

MATLAB Code for Numerical Solution of Heat Equation

Listing 1: Numerical Heat Equation

```
%% 18.0851 Project
% Author     : Jered Dominguez-Trujillo
% Date       : May 2, 2019
% Description : Numerical Solution to Heat Equation

% SCHEME = 0 -> EXPLICIT
% SCHEME = 1 -> IMPLICIT
% SCHEME = 2 -> CRANK_NICOLSON

function U = NumHT(SCHEME, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG, W, SIGMA, QMAX)
    clc;

    % Boundary Conditions
    LENGTH = L;                         % Length of Domain
    TIME_RAMP = TR;                     % Time Ramp at X = 0 for BC to Go from 0 to C1
    C1 = BC1;                           % Dirilecht Condition U(x) = C1 at X = 0
    C2 = BC2;                           % Neumann Condition at dU/dX = C2 at X = L
    K = KT;                             % Thermal Diffusivity

    % Default Arguments
    if nargin <= 12
        QMAX = 1;                       % Default Maximum Value of Heat Source Function

        if nargin <= 11
            SIGMA = 0.2 .* LENGTH;       % Default Standard Deviation of Heat Source Function
        end

        if nargin <= 10
            W = 1;                      % Default Frequency of Heat Source Function (rad/s)
        end
    end

    % Spatial Domain
    NODES = NX;                         % Nodes
    DX = LENGTH ./ NODES;                % DX Calculation
    X = linspace(0, LENGTH + DX, NODES + 2);% X Vector with Ghost Node

    % Time Domain
    TMAX = TM;                          % End Time of Simulation
    DT = TMAX ./ NT;                     % DT Calculation
    TIMESTEPS = round(TMAX ./ DT + 1, 0);         % Number of Time Steps
    T = linspace(0, TMAX, TIMESTEPS);   % Time Vector

    % Calculate CFL Number
    CFL = (DT .* K) ./ (DX .* DX);      % Multiplication Factor K (DT / DX^2)

    % Print Out Simulation Info
    if SCHEME == 0
        fprintf('Explicit Method:\n');
    elseif SCHEME == 1
        fprintf('Implicit Method:\n');
    elseif SCHEME == 2
        fprintf('Crank-Nicolson Method:\n');
    end

    fprintf('Thermal Diffusivity [K]: %.3f\n', K);
```

```
fprintf('BCs:\n(1) U(x) = %.2f At X = 0\n(2) dU/dX = %.2f at X = L = %.2f\n\n', C1, C2, L);
fprintf('Length: %.2f\t\tDX: %.5f\t\tNodes: %.0f\n', LENGTH, DX, NODES);
fprintf('Max Time: %.2f\t\tDT: %.5f\t\t\n\n', TMAX, DT);
if SOURCE_FLAG == 1
    fprintf('Q(x) = - QMAX * SIN(OMEGA * T) * EXP(-(X - L/2)^2 / SIGMA ^ 2)\n');
    fprintf('QMAX: %.2f\t\tOMEGA: %.2f rad/s\t\tSIGMA: %.2f\t\t L: %.2f\n\n', QMAX, W, SIGMA,
        LENGTH);
end
fprintf('CFL Number: %.5f\n', CFL);

% Initialize Matrices
U = zeros(TIMESTEPS, NODES + 2);
Q = zeros(TIMESTEPS, NODES + 2);

% Time Tolerance
eps = 10^-7;

% If Heat Source Flag is True
if SOURCE_FLAG == 1
    for ii = 1:TIMESTEPS
        Q(ii, :) = - QMAX .* (sin((W .* T(ii)))) .* exp(-((X - (LENGTH ./ 2)) .^ 2) ./ (SIGMA .^
            2));
    end
end

% Initialize
CURRENT_T = 0;
TIMESTEP = 0;

% Plot Initial Conditions
fTemperature = figure('Name', 'Temperature History', 'NumberTitle', 'off');
figure(fTemperature);

% Temperature Profile
subplot(211);
plot(X, U(1, :), 'o-');
xlabel('X', 'FontSize', 18); ylabel('Temperature [u]', 'Fontsize', 14);
axis([0 LENGTH -2 2]);

% Heat Source Profile
subplot(212);
plot(X, Q(1, :), 'o-');
xlabel('X', 'FontSize', 18); ylabel('Heat Source [Q]', 'Fontsize', 14);
axis([0 LENGTH -QMAX QMAX]);

suptitle(['Time = ', num2str(CURRENT_T), ' s']);
% pause();

% Explicit Scheme
if SCHEME == 0

    % Iterate Until TMAX
    while CURRENT_T < TMAX - eps
        CURRENT_T = CURRENT_T + DT;
        TIMESTEP = TIMESTEP + 1;

        n = TIMESTEP;

        % Explicit Euler Method
        for ii = 2:NODES + 1
            U(n + 1, ii) = CFL .* U(n, ii + 1) + (1 - 2 .* CFL) .* U(n, ii) + CFL .* U(n, ii -
                1) + Q(n, ii) .* DT;
```

```
        end

        % Boundary Conditions
        U(n + 1, 1) = C1 .* (CURRENT_T / TIME_RAMP) .* (CURRENT_T <= TIME_RAMP) + C1 .*
             (CURRENT_T > TIME_RAMP);
        U(n + 1, NODES + 2) = U(n + 1, NODES) + 2 .* DX .* C2;

        % Plot New Temperature Profile
        figure(fTemperature);

        % Temperature Profile
        subplot(211);
        plot(X, U(n + 1, :), 'o-');
        xlabel('X', 'FontSize', 18); ylabel('Temperature [u]', 'Fontsize', 14);
        axis([0 L -2 2]);

        % Heat Source Profile
        subplot(212);
        plot(X, Q(n + 1, :), 'o-');
        xlabel('X', 'FontSize', 18); ylabel('Heat Source [Q]', 'Fontsize', 14);
        axis([0 LENGTH -QMAX QMAX]);

        suptitle(['Time = ', num2str(CURRENT_T), ' s']);

        pause(0.01);
    end

% Implicit Scheme
elseif SCHEME == 1
    LOWER = zeros(1, NODES);
    DIAG  = zeros(1, NODES + 1);
    UPPER = zeros(1, NODES);

    LOWERI = 2:NODES + 1; LOWERJ = 1:NODES;
    DIAGI = 1:NODES + 1; DIAGJ = 1:NODES + 1;
    UPPERI = 1:NODES; UPPERJ = 2:NODES + 1;

    DIAG(1) = 1; UPPER(1) = 0;

    for ii = 2:NODES
        LOWER(ii) = -CFL;
        DIAG(ii) = 2 .* CFL + 1;
        UPPER(ii) = -CFL;
    end

    DIAG(NODES + 1) = 2 .* CFL + 1;

    % Iterate Until TMAX
    while CURRENT_T < TMAX - eps
        CURRENT_T = CURRENT_T + DT;
        TIMESTEP = TIMESTEP + 1;

        n = TIMESTEP;

        RHS = zeros(NODES + 1, 1);

        % Implicit Euler Method
        for ii = 2:NODES + 1
            RHS(ii) = U(n, ii) + Q(n, ii) .* DT;
        end

        % Boundary Conditions
```

```
        fBC1 = C1 .* (CURRENT_T / TIME_RAMP) .* (CURRENT_T <= TIME_RAMP) + C1 .* (CURRENT_T >
            TIME_RAMP);
        LOWER(1) = 0;
        RHS(1) = fBC1;
        RHS(2) = RHS(2) + CFL .* fBC1;

        % Forward Difference Scheme
        % DIAG(NODES + 1) = CFL + 1;
        % RHS(NODES + 1) = RHS(NODES + 1) + C2 .* CFL .* DX;

        % Central Difference Scheme
        LOWER(NODES) = -2 .* CFL;
        RHS(NODES + 1) = RHS(NODES + 1) + 2 .* C2 .* CFL .* DX;

        MA = sparse([LOWERI, DIAGI, UPPERI], [LOWERJ, DIAGJ, UPPERJ], [LOWER, DIAG, UPPER],
            NODES + 1, NODES + 1);

        U(n + 1, 1:end-1) = MA \ RHS;

        % Forward Difference Scheme
        % U(n + 1, end) = U(n + 1, end - 1) + C2 .* DX;

        % Central Difference Scheme
        U(n + 1, end) = U(n + 1, end - 2) + 2 .* C2 .* DX;

        % Plot New Temperature Profile
        figure(fTemperature);

        % Temperature Profile
        subplot(211);
        plot(X, U(n + 1, :), 'o-');
        xlabel('X', 'FontSize', 18); ylabel('Temperature [u]', 'Fontsize', 14);
        axis([0 LENGTH -2 2]);

        % Heat Source Profile
        subplot(212);
        plot(X, Q(n + 1, :), 'o-');
        xlabel('X', 'FontSize', 18); ylabel('Heat Source [Q]', 'Fontsize', 14);
        axis([0 LENGTH -QMAX QMAX]);

        suptitle(['Time = ', num2str(CURRENT_T), ' s']);

        pause(0.01);
    end

% Crank-Nicolson Scheme
elseif SCHEME == 2
    LOWER = zeros(1, NODES);
    DIAG  = zeros(1, NODES + 1);
    UPPER = zeros(1, NODES);

    LOWERI = 2:NODES + 1; LOWERJ = 1:NODES;
    DIAGI = 1:NODES + 1; DIAGJ = 1:NODES + 1;
    UPPERI = 1:NODES; UPPERJ = 2:NODES + 1;

    DIAG(1) = 1; UPPER(1) = 0;

    for ii = 2:NODES
        LOWER(ii) = -CFL ./ 2;
        DIAG(ii) = CFL + 1;
        UPPER(ii) = -CFL ./ 2;
    end
```

```
DIAG(NODES + 1) = CFL + 1;

% Iterate Until TMAX
while CURRENT_T < TMAX - eps
    CURRENT_T = CURRENT_T + DT;
    TIMESTEP = TIMESTEP + 1;

    n = TIMESTEP;

    RHS = zeros(NODES + 1, 1);

    % Crank-Nicolson Method
    for ii = 2:NODES + 1
        RHS(ii) = CFL .* U(n, ii - 1) ./ 2 + (1 - CFL) .* U(n, ii) + CFL .* U(n, ii + 1) ./
             2 + Q(n, ii) .* DT;
    end

    % Boundary Conditions
    fBC1 = C1 .* (CURRENT_T / TIME_RAMP) .* (CURRENT_T <= TIME_RAMP) + C1 .* (CURRENT_T >
         TIME_RAMP);
    LOWER(1) = 0;
    RHS(1) = fBC1;
    RHS(2) = RHS(2) + CFL .* fBC1 ./ 2;

    % Forward Difference Scheme
    % DIAG(NODES + 1) = CFL ./ 2 + 1;
    % RHS(NODES + 1) = RHS(NODES + 1) + C2 .* CFL .* DX ./ 2;

    % Central Difference Scheme
    LOWER(NODES) = -CFL;
    RHS(NODES + 1) = RHS(NODES + 1) + C2 .* CFL .* DX;

    MA = sparse([LOWERI, DIAGI, UPPERI], [LOWERJ, DIAGJ, UPPERJ], [LOWER, DIAG, UPPER],
         NODES + 1, NODES + 1);

    U(n + 1, 1:end-1) = MA \ RHS;

    % Forward Difference Scheme
    % U(n + 1, end) = U(n + 1, end - 1) + C2 .* DX;

    % Central Difference Scheme
    U(n + 1, end) = U(n + 1, end - 2) + 2 .* C2 .* DX;

    % Plot New Temperature Profile
    figure(fTemperature);

    % Temperature Profile
    subplot(211);
    plot(X, U(n + 1, :), 'o-');
    xlabel('X', 'FontSize', 14); ylabel('Temperature [u]', 'Fontsize', 14);
    axis([0 LENGTH -2 2]);

    % Heat Source Profile
    subplot(212);
    plot(X, Q(n + 1, :), 'o-');
    xlabel('X', 'FontSize', 14); ylabel('Heat Source [Q]', 'Fontsize', 14);
    axis([0 LENGTH -QMAX QMAX]);

    suptitle(['Time = ', num2str(CURRENT_T), ' s']);

    pause(0.01);
```

```
        end
    end

    close(fTemperature);
end
```

Listing 2: Wrapper Used to Iterate and Calculate Errors

```
%% 18.0851 Project
% Author     : Jered Dominguez-Trujillo
% Date       : May 9, 2019
% Description : Wrapper for NumHT.m

% SCHEME = 0 -> EXPLICIT
% SCHEME = 1 -> IMPLICIT
% SCHEME = 2 -> CRANK_NICOLSON

clear all; close all;

COLORS = get(gca,'colororder');

%% Plot 3 Methods at t = tfinal against each other without Source
% Baseline
SCHEME = 2;              % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
MNX = 2^18; TM = 40; MNT = 2^12; TR = 1; SOURCE_FLAG = 0;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, MNX, TM, MNT, TR, SOURCE_FLAG);

% 3 Method Runs
BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
NX = 2^6; TM = 40; NT = 2^10; TR = 1; SOURCE_FLAG = 0;

DXX = L ./ NX;
XX = linspace(0, L + DXX, NX + 2);

DT = TM ./ NT;
TIMESTEPS = TM ./ DT + 1;
TT = linspace(0, TM, TIMESTEPS);

AA = NumHT(0, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
BB = NumHT(1, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
CC = NumHT(2, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);

BASELINE = BASELINE(1:MNT/NT:end, 1:MNX/NX:end);
ResA = BASELINE - AA(:, 1:end-1); ErrorA = sqrt(sum(ResA' .* ResA') ./ NX);
ResB = BASELINE - BB(:, 1:end-1); ErrorB = sqrt(sum(ResB' .* ResB') ./ NX);
ResC = BASELINE - CC(:, 1:end-1); ErrorC = sqrt(sum(ResC' .* ResC') ./ NX);

fErrorTime1 = figure('Name', 'Error Evolution with Time', 'NumberTitle', 'off');
figure(fErrorTime1); hold on;

plot(TT, ErrorA, '-', 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
plot(TT, ErrorB, '-', 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
plot(TT, ErrorC, '-', 'Linewidth', 2, 'DisplayName', 'Crank-Nicolson');

xlabel('Time', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
title('Error Evolution', 'FontSize', 18); legend('show');

saveas(fErrorTime1, 'Figures/MATLAB/NoSourceErrorTime.png');
saveas(fErrorTime1, 'Figures/MATLAB/Figs/NoSourceErrorTime.fig');

fCompare1 = figure('Name', 'Solution Comparison: T = 40', 'NumberTitle', 'off');
figure(fCompare1); hold on;

subplot(2, 2, 1);
plot(XX(1:end-1), BASELINE(end, :), '-', 'Color', COLORS(4, :), 'LineWidth', 2, 'DisplayName',
    'Baseline');
```

```
xlabel('X'); ylabel('Temperature [u]');
title('Baseline (Analytical)'); axis([0 L -2 2]);

subplot(2, 2, 2);
plot(XX, AA(end, :), '-', 'Color', COLORS(1, :), 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
xlabel('X'); ylabel('Temperature [u]');
title('Explicit Euler'); axis([0 L -2 2]);

subplot(2, 2, 3);
plot(XX, BB(end, :), '-', 'Color', COLORS(2, :), 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
xlabel('X'); ylabel('Temperature [u]');
title('Implicit Euler'); axis([0 L -2 2]);

subplot(2, 2, 4);
plot(XX, CC(end, :), '-', 'Color', COLORS(3, :), 'LineWidth', 2, 'DisplayName', 'Crank-Nicolson');
xlabel('X'); ylabel('Temperature [u]');
title('Crank-Nicolson'); axis([0 L -2 2]);

suptitle('Time = 40 s');

saveas(fCompare1, 'Figures/MATLAB/NoSourceCompare.png');
saveas(fCompare1, 'Figures/MATLAB/Figs/NoSourceCompare.fig');

%% Plot 3 Methods at t = tfinal against each other with Source
% Baseline
SCHEME = 2;              % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
MNX = 2^18; TM = 40; MNT = 2^12; TR = 1; SOURCE_FLAG = 1;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, MNX, TM, MNT, TR, SOURCE_FLAG);

% 3 Method Runs
BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
NX = 2^6; TM = 40; NT = 2^10; TR = 1; SOURCE_FLAG = 0;

DXX = L ./ NX;
XX = linspace(0, L + DXX, NX + 2);

DT = TM ./ NT;
TIMESTEPS = TM ./ DT + 1;
TT = linspace(0, TM, TIMESTEPS);

AA = NumHT(0, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
BB = NumHT(1, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
CC = NumHT(2, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);

BASELINE = BASELINE(1:MNT/NT:end, 1:MNX/NX:end);
ResA = BASELINE - AA(:, 1:end-1); ErrorA = sqrt(sum(ResA' .* ResA') ./ NX);
ResB = BASELINE - BB(:, 1:end-1); ErrorB = sqrt(sum(ResB' .* ResB') ./ NX);
ResC = BASELINE - CC(:, 1:end-1); ErrorC = sqrt(sum(ResC' .* ResC') ./ NX);

fErrorTime2 = figure('Name', 'Error Evolution with Time', 'NumberTitle', 'off');
figure(fErrorTime2); hold on;

plot(TT, ErrorA, '-', 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
plot(TT, ErrorB, '-', 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
plot(TT, ErrorC, '-', 'Linewidth', 2, 'DisplayName', 'Crank-Nicolson');

xlabel('Time', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
title('Error Evolution', 'FontSize', 18); legend('show');
```

```
saveas(fErrorTime2, 'Figures/MATLAB/SourceErrorTime.png');
saveas(fErrorTime2, 'Figures/MATLAB/Figs/SourceErrorTime.fig');

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
NX = 2^6; TM = 40; NT = 2^10; TR = 1; SOURCE_FLAG = 1;

DXX = L ./ NX;
XX = linspace(0, L + DXX, NX + 2);

AA = NumHT(0, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
BB = NumHT(1, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
CC = NumHT(2, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);

fCompare2 = figure('Name', 'Solution Comparison: T = 40', 'NumberTitle', 'off');
figure(fCompare2); hold on;

subplot(2, 2, 1);
plot(XX(1:end-1), BASELINE(end, :), '-', 'Color', COLORS(4, :), 'LineWidth', 2, 'DisplayName',
     'Baseline');
xlabel('X'); ylabel('Temperature [u]');
title('Baseline (Analytical)'); axis([0 L -2 2]);

subplot(2, 2, 2);
plot(XX, AA(end, :), '-', 'Color', COLORS(1, :), 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
xlabel('X'); ylabel('Temperature [u]');
title('Explicit Euler'); axis([0 L -2 2]);

subplot(2, 2, 3);
plot(XX, BB(end, :), '-', 'Color', COLORS(2, :), 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
xlabel('X'); ylabel('Temperature [u]');
title('Implicit Euler'); axis([0 L -2 2]);

subplot(2, 2, 4);
plot(XX, CC(end, :), '-', 'Color', COLORS(3, :), 'LineWidth', 2, 'DisplayName', 'Crank-Nicolson');
xlabel('X'); ylabel('Temperature [u]');
title('Crank-Nicolson'); axis([0 L -2 2]);

suptitle('Time = 40 s');

saveas(fCompare2, 'Figures/MATLAB/SourceCompare.png');
saveas(fCompare2, 'Figures/MATLAB/Figs/SourceCompare.fig');

%% Hold DX Constant with Source
% Baseline
SCHEME = 2;             % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
NX = 2^6; TM = 40; NT = 2^14; TR = 1; SOURCE_FLAG = 1;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
BLAST = BASELINE(end, :);

sch = {'Explicit Euler', 'Implicit Euler', 'Crank-Nicolson'};

% Run Through DT
MAXNT = 2^11; MINNT = 2^3;

NT = round(logspace(log(MINNT)/log(10), log(MAXNT)/log(10), 20), 0);
DT = TM ./ NT;

RVX = zeros(3, length(NX));
```

```
fErrorDT1 = figure('Name', 'Numerical Error - Constant DX', 'NumberTitle', 'off');
figure(fErrorDT1);

for SCHEME = 0:2
    for ii = 1:length(NT)
        U = NumHT(SCHEME, BC1, BC2, KT, L, NX, TM, NT(ii), TR, SOURCE_FLAG);
        U = U(:, 1:end-1);
        ULAST = U(end, :);

        B = BLAST(1:end-1);

        RES = ULAST - B;
        RVX(SCHEME + 1, ii) = sqrt(sum(RES .* RES) ./ NX);

        clf;
        hold off;
        for jj = 0:SCHEME
            if jj == 0
                inx = find(RVX(jj + 1, :) < 10^2, 1);
                loglog(DT(inx:ii), RVX(jj + 1, inx:ii), '-o', 'Color', COLORS(jj + 1, :),
                     'LineWidth', 2, 'DisplayName', sch{jj+1});
            else
                loglog(DT(1:ii), RVX(jj + 1, 1:ii), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            end
            hold on;
        end

        loglog(DT, DT, '--', 'Color', COLORS(5, :), 'LineWidth', 2, 'DisplayName', 'First-Order');
        loglog(DT, DT .^ 2, '--', 'Color', COLORS(6, :), 'LineWidth', 2, 'DisplayName',
             'Second-Order');

        xlabel('DT', 'Fontsize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Time Step', 'FontSize', 18);
        ylim([0, max(DT) .^ 2]);
        legend('show');
    end
end

saveas(fErrorDT1, 'Figures/MATLAB/ConstantDXSource.png');
saveas(fErrorDT1, 'Figures/MATLAB/Figs/ConstantDXSource.fig');

%% Hold DT Constant with Source
% Baseline
SCHEME = 2;                 % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
MNX = 2^18; TM = 40; NT = 2^12; TR = 1; SOURCE_FLAG = 1;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, MNX, TM, NT, TR, SOURCE_FLAG);
BLAST = BASELINE(end, :);

sch = {'Explicit Euler', 'Implicit Euler', 'Crank-Nicolson'};

% Run through DX
MAXNX = 2^8; MINNX = 2^3;

NX = round(logspace(log(MINNX)/log(10), log(MAXNX)/log(10), 20), 0);
DX = L ./ NX;

RVT = zeros(3, length(NX));
fErrorDX1 = figure('Name', 'Numerical Error - Constant DT', 'NumberTitle', 'off');
```

```
figure(fErrorDX1);

for SCHEME = 0:2
    for ii = 1:length(NX)
        U = NumHT(SCHEME, BC1, BC2, KT, L, NX(ii), TM, NT, TR, SOURCE_FLAG);
        U = U(:, 1:end-1);
        ULAST = U(end, :);

        B = BLAST(1:MNX/NX(ii):end-1);

        RES = ULAST - B;
        RVT(SCHEME + 1, ii) = sqrt(sum(RES .* RES) ./ NX(ii));

        clf;
        hold off;
        for jj = 0:SCHEME
            if jj == 0
                inx = find(RVT(jj + 1, :) < 10^2, 1, 'last');
                loglog(DX(1:inx), RVT(jj + 1, 1:inx), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            else
                loglog(DX(1:ii), RVT(jj + 1, 1:ii), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            end
            hold on;
        end

        loglog(DX, DX, '--', 'Color', COLORS(5, :), 'LineWidth', 2, 'DisplayName', 'First-Order');
        loglog(DX, DX .^ 2, '--', 'Color', COLORS(6, :), 'LineWidth', 2, 'DisplayName',
             'Second-Order');

        xlabel('DX', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Spatial Step', 'FontSize', 18);
        ylim([0, max(DX) .^ 2]);
        legend('show');
    end
end

saveas(fErrorDX1, 'Figures/MATLAB/ConstantDTSource.png');
saveas(fErrorDX1, 'Figures/MATLAB/Figs/ConstantDTSource.fig');

%% Hold DX Constant without Source
% Baseline
SCHEME = 2;                % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
NX = 2^6; TM = 40; NT = 2^14; TR = 1; SOURCE_FLAG = 0;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, NX, TM, NT, TR, SOURCE_FLAG);
BLAST = BASELINE(end, :);

sch = {'Explicit Euler', 'Implicit Euler', 'Crank-Nicolson'};

% Run Through DT
MAXNT = 2^11; MINNT = 2^3;

NT = round(logspace(log(MINNT)/log(10), log(MAXNT)/log(10), 20), 0);
DT = TM ./ NT;

RVX = zeros(3, length(NX));

fErrorDT2 = figure('Name', 'Numerical Error - Constant DX', 'NumberTitle', 'off');
```

```
figure(fErrorDT2);

for SCHEME = 0:2
    for ii = 1:length(NT)
        U = NumHT(SCHEME, BC1, BC2, KT, L, NX, TM, NT(ii), TR, SOURCE_FLAG);
        U = U(:, 1:end-1);
        ULAST = U(end, :);

        B = BLAST(1:end-1);

        RES = ULAST - B;
        RVX(SCHEME + 1, ii) = sqrt(sum(RES .* RES) ./ NX);

        clf;
        hold off;
        for jj = 0:SCHEME
            if jj == 0
                inx = find(RVX(jj + 1, :) < 10^2, 1);
                loglog(DT(inx:ii), RVX(jj + 1, inx:ii), '-o', 'Color', COLORS(jj + 1, :),
                     'LineWidth', 2, 'DisplayName', sch{jj+1});
            else
                loglog(DT(1:ii), RVX(jj + 1, 1:ii), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            end
            hold on;
        end

        loglog(DT, DT, '--', 'Color', COLORS(5, :), 'LineWidth', 2, 'DisplayName', 'First-Order');
        loglog(DT, DT .^ 2, '--', 'Color', COLORS(6, :), 'LineWidth', 2, 'DisplayName',
             'Second-Order');

        xlabel('DT', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Time Step', 'FontSize', 18);
        ylim([0, max(DT) .^ 2]);
        legend('show');
    end
end

saveas(fErrorDT2, 'Figures/MATLAB/ConstantDXNoSource.png');
saveas(fErrorDT2, 'Figures/MATLAB/Figs/ConstantDXNoSource.fig');

%% Hold DT Constant without Source
% Baseline
SCHEME = 2;                % Crank-Nicolson

BC1 = 1; BC2 = -0.2; KT = 0.1; L = 2*pi;
MNX = 2^18; TM = 40; NT = 2^12; TR = 1; SOURCE_FLAG = 0;

BASELINE = NumHT(SCHEME, BC1, BC2, KT, L, MNX, TM, NT, TR, SOURCE_FLAG);
BLAST = BASELINE(end, :);

sch = {'Explicit Euler', 'Implicit Euler', 'Crank-Nicolson'};

% Run through DX
MAXNX = 2^8; MINNX = 2^3;

NX = round(logspace(log(MINNX)/log(10), log(MAXNX)/log(10), 20), 0);
DX = L ./ NX;

RVT = zeros(3, length(NX));
fErrorDX2 = figure('Name', 'Numerical Error - Constant DT', 'NumberTitle', 'off');
figure(fErrorDX2);
```

```
for SCHEME = 0:2
    for ii = 1:length(NX)
        U = NumHT(SCHEME, BC1, BC2, KT, L, NX(ii), TM, NT, TR, SOURCE_FLAG);
        U = U(:, 1:end-1);
        ULAST = U(end, :);

        B = BLAST(1:MNX/NX(ii):end-1);

        RES = ULAST - B;
        RVT(SCHEME + 1, ii) = sqrt(sum(RES .* RES) ./ NX(ii));

        clf;
        hold off;
        for jj = 0:SCHEME
            if jj == 0
                inx = find(RVT(jj + 1, :) < 10^2, 1, 'last');
                loglog(DT(1:inx), RVX(jj + 1, 1:inx), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            else
                loglog(DT(1:ii), RVT(jj + 1, 1:ii), '-o', 'Color', COLORS(jj + 1, :), 'LineWidth',
                     2, 'DisplayName', sch{jj+1});
            end
            hold on;
        end

        loglog(DX, DX, '--', 'Color', COLORS(5, :), 'LineWidth', 2, 'DisplayName', 'First-Order');
        loglog(DX, DX .^ 2, '--', 'Color', COLORS(6, :), 'LineWidth', 2, 'DisplayName',
             'Second-Order');

        xlabel('DX', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Spatial Step', 'FontSize', 18);
        ylim([0, max(DX) .^ 2]);
        legend('show');
    end
end

saveas(fErrorDX2, 'Figures/MATLAB/ConstantDTNoSource.png');
saveas(fErrorDX2, 'Figures/MATLAB/Figs/ConstantDTNoSource.fig');
```