MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project Report

18.0851

COMPUTATIONAL SCIENCE AND ENGINEERING I

WRITTEN BY

JERED DOMINGUEZ-TRUJILLO

SUBMITTED TO:

PROF. MATTHEW DUREY

MAY 10, 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 $[\omega]$	$1\frac{rad}{s}$
Distribution of Heat Source $[\sigma]$	$0.2 * L$

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.

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

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

Where

$$\begin{aligned} u_n(x+\Delta x) &= u_{n,i+1} \\ u_n(x) &= u_{n,i} \\ u_n(x-\Delta x) &= u_{n,i-1} \end{aligned} \tag{10}$$

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.

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

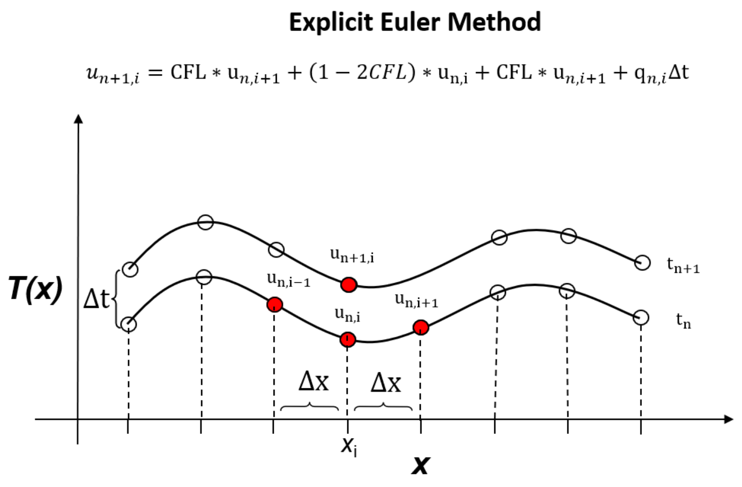


Figure 1: Explicit Euler Scheme Template

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:

$$u_{n+1} = u_n + \Delta t * f(t_{n+1}, u_{n+1}) \tag{13}$$

Unlike the Explicit Euler scheme, the Implicit Euler scheme requires the simultaneous solution of all points in the domain for the next time step.

Expanding Eq. 13 gives:

$$u_{n+1,i} = u_{n,i} + \Delta t\big[K\big(\frac{u_{n+1}(x+\Delta x) - 2u_{n+1}(x) + u_{n+1}(x-\Delta x)}{\Delta x^2}\big) + q_n(x)\big]_i \tag{14}$$

Note that Eq. 10 still applies to Eq. 13 and we still define the CFL number as in Eq. 11. After some algebra, we get the following equation (Eq. 15) for the Implicit Euler Scheme.

$$\boxed{-CFL * u_{n+1,i+1} + (2CFL+1) * u_{n+1,i} - CFL * u_{n+1,i-1} = u_{n,i} + q_{n,i}\Delta t} \tag{15}$$

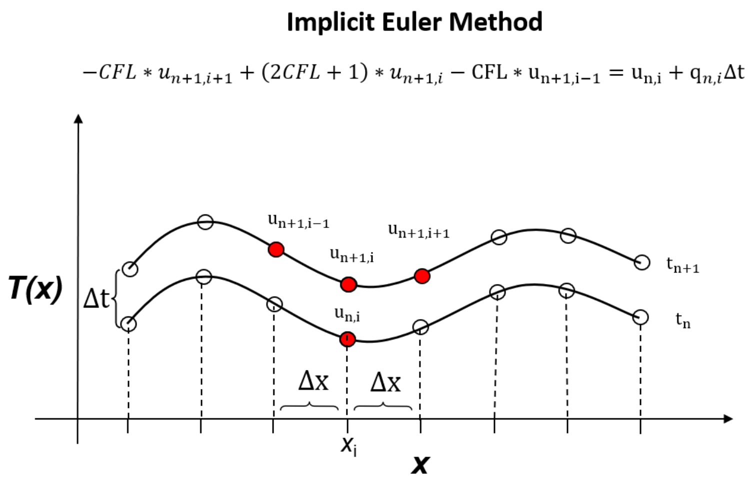

Implicit Euler Method
$-CFL * u_{n+1,i+1} + (2CFL+1) * u_{n+1,i} - \mathrm{CFL} * \mathrm{u}_{n+1,i-1} = \mathrm{u}_{n,i} + q_{n,i}\Delta t$
T(x)
Δt
$u_{n+1,i-1}$
$u_{n+1,i}$
$u_{n+1,i+1}$
t_{n+1}
$u_{n,i}$
t_n
Δx
Δx
x_i
x

Figure 2: Implicit Euler Scheme Template

Boundary Conditions

1) Dirilecht Condition at the Left Boundary x = 0:

$$u(x=0,t) = \begin{cases} C_1 * \frac{t}{t_{ramp}} & t \le t_{ramp} \\ C_1 & t > t_{ramp} \end{cases} \tag{16}$$

$$-CFL * f_{BC1} + (2CFL+1) * u_{n+1,2} - CFL * u_{n+1,3} = u_{n,2} + q_{n,2}\Delta t \tag{17}$$

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

2) Neumann Condition at the Right Boundary x = L:

$$\frac{\partial u}{\partial x}(x = L, t) = C_2 \tag{19}$$

Central Difference

$$\frac{u_{n+1,i+1} - u_{n+1,i-1}}{2\Delta x} = C_2 \tag{20}$$

$$u_{n+1,i+1} = 2C_2\Delta x + u_{n+1,i-1} \tag{21}$$

Where $i + 1 = m$ and m represents the last node (ghost node)

$$u_{n+1,m} = 2C_2\Delta x + u_{n+1,m-2} \tag{22}$$

Plugging in

$$-CFL*u_{n+1,m-2}+(1+2CFL)*u_{n+1,m-1}-CFL*(2C_2\Delta x+u_{n+1,m-2}) = u_{n,m-1}+q_{n,m-1}\Delta t \tag{23}$$

$$\boxed{-2CFL * u_{n+1,m-2} + (1 + 2CFL) * u_{n+1,m-1} = u_{n,m-1} + q_{n,m-1}\Delta t + 2CFL * C_2\Delta x} \tag{24}$$

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

$$u_{n+1,i} = u_{n,i} + \Delta t * \Big[\frac{1}{2}f(t_n, u_n) + \frac{1}{2}f(t_{n+1}, u_{n+1}) + q_n\Big]_i \tag{25}$$

Substituting in the heat equation (Eq. 1),

$$u_{n+1,i} = u_{n,i} + \Delta t * \Big[\frac{1}{2}\big(K\frac{\partial^2 u_n}{\partial x^2}\big) + \frac{1}{2}\big(K\frac{\partial^2 u_{n+1}}{\partial x^2}\big) + q_n\Big]_i \tag{26}$$

And then discretizing,

$$\begin{aligned} u_{n+1,i} = u_{n,i} + \Delta t * \Big[\frac{1}{2}\big(K\frac{u_n(x+\Delta x) - 2u_n(x) + u_n(x-\Delta x)}{\Delta x^2}\big) + \\ \frac{1}{2}\big(K\frac{u_{n+1}(x+\Delta x) - 2u_{n+1}(x) + u_{n+1}(x-\Delta x)}{\Delta x^2}\big) + q_n\Big]_i \end{aligned} \tag{27}$$

In addition to Eq. 10, the following (Eq. 28 are applied to Eq. 27.

$$\begin{aligned} u_{n+1}(x+\Delta x) &= u_{n+1,i+1} \\ u_{n+1}(x) &= u_{n+1,i} \\ u_{n+1}(x-\Delta x) &= u_{n+1,i-1} \end{aligned} \tag{28}$$

We still define the CFL number as in Eq. 11. After some algebra, we get the following equation (Eq. 29) for the Implicit Euler Scheme.

$$\boxed{\begin{aligned} \frac{-CFL}{2} * u_{n+1,i-1} + (1 + CFL) * u_{n+1,i} - \frac{CFL}{2} * u_{n+1,i+1} = \quad (29) \\ \frac{CFL}{2} * u_{n,i-1} + (1 - CFL) * u_{n,i} + \frac{CFL}{2} * u_{n,i+1} + q_{n,1}\Delta t \quad (30) \end{aligned}}$$

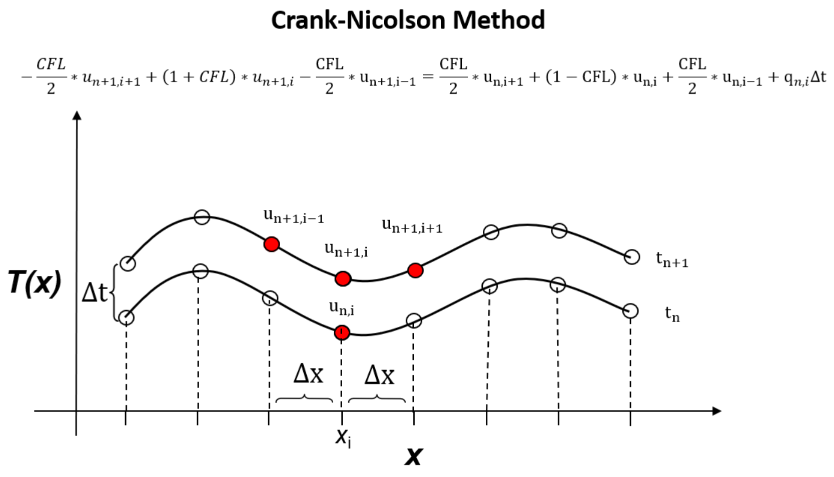

Crank-Nicolson Method
$-\frac{CFL}{2}*u_{n+1,i+1}+(1+CFL)*u_{n+1,i}-\frac{CFL}{2}*u_{n+1,i-1}=\frac{CFL}{2}*u_{n,i+1}+(1-CFL)*u_{n,i}+\frac{CFL}{2}*u_{n,i-1}+q_{n,i}\Delta t$
T(x)
Δt
$u_{n+1,i-1}$
$u_{n+1,i}$
$u_{n+1,i+1}$
$u_{n,i}$
t_{n+1}
t_n
Δx
Δx
x_i
x

Figure 3: Crank-Nicolson Scheme Template

1) Dirilecht Condition at the Left Boundary x = 0:

$$u(x=0,t)=u_{n+1,1}=\begin{cases}C_1*\frac{t}{t_{ramp}} & t\le t_{ramp}\\ C_1 & t>t_{ramp}\end{cases}=f_{BC1} \tag{31}$$

$$\begin{aligned}\frac{-CFL}{2}*f_{BC1}+(1+CFL)*u_{n+1,i}-\frac{-CFL}{2}*u_{n+1,i+1}=\\ \frac{CFL}{2}*u_{n,i-1}+(1-CFL)*u_{n,i}+\frac{CFL}{2}*u_{n,i+1}+q_{n,1}\Delta t\end{aligned} \tag{32}$$

$$\boxed{\begin{aligned}(1+CFL)*u_{n+1,i}-\frac{CFL}{2}*u_{n+1,i+1}= \quad (33)\\ \frac{CFL}{2}*u_{n,i-1}+(1-CFL)*u_{n,i}+\frac{CFL}{2}*u_{n,i+1}+q_{n,1}\Delta t+\frac{CFL}{2}*f_{BC1} \quad (34)\end{aligned}}$$

2) Neumann Condition at the Right Boundary x = L:

$$\frac{\partial u}{\partial x}(x=L,t)=C_2 \tag{35}$$

Central Difference

$$\frac{u_{n+1,i+1}-u_{n+1,i-1}}{2\Delta x}=C_2 \tag{36}$$

$$u_{n+1,i+1}=2C_2\Delta x+u_{n+1,i-1} \tag{37}$$

Where $i+1=m$ and m represents the last node (ghost node)

$$u_{n+1,m}=2C_2\Delta x+u_{n+1,m-2} \tag{38}$$

Plugging in

$$\begin{aligned} \frac{-CFL}{2} * u_{n+1,i-1} + (1 + CFL) * u_{n+1,i} - \frac{CFL}{2} * \big(2C_2\Delta x + u_{n+1,i-1}\big) = \\ \frac{CFL}{2} * u_{n,i-1} + (1 - CFL) * u_{n,i} + \frac{CFL}{2} * u_{n,i+1} + q_{n,1}\Delta t \end{aligned} \tag{39}$$

$$\boxed{\begin{aligned} -CFL * u_{n+1,i-1} + (1 + CFL) * u_{n+1,i} = \quad (40) \\ \frac{CFL}{2} * u_{n,i-1} + (1 - CFL) * u_{n,i} + \frac{CFL}{2} * u_{n,i+1} + q_{n,1}\Delta t + \frac{CFL}{2} * \big(2C_2\Delta x\big) \quad (41) \end{aligned}}$$

6 Stability

A stability analysis indicates that the Explicit Euler Method must satisfy Von Neumann stability, which is defined below in Eq. 42, making the method conditionally stable. However, the Implicit Euler Method and the Crank-Nicolson methods are unconditionally stable, and do not need to satisfy the condition in Eq. 42.

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 the Courant number (CFD) defined in Eq. 11. As seen in Table **??**, 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).

$$CFL = K \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{42}$$

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	500	0.1	0.08	0.314	0.081	Stable	Stable	Stable
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

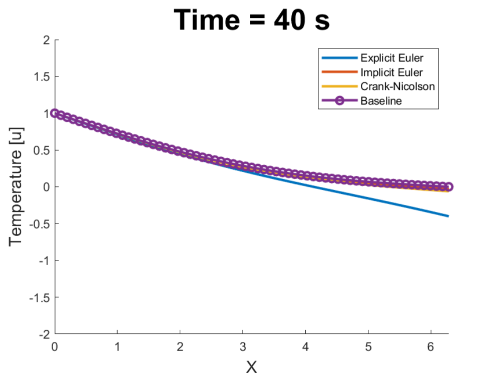
7 Error

7.1 Error Evolution Over Time

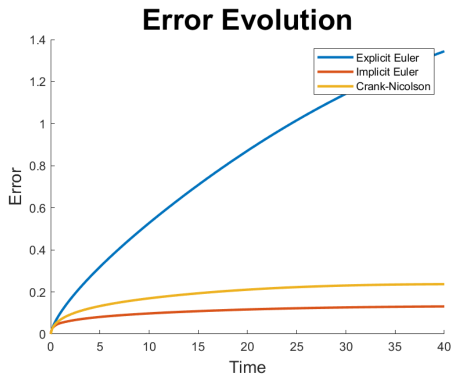
To visualize and understand the accuracy of the three methods, each was ran and compared to a baseline "analytical" case (performed using Crank-Nicolson with small timestep and high spacial resolution). This was done for the case with no source function ($q(x,t) = 0$) and for the case with the source function ($q(x,t)$) defined by Eq. 5. The solutions were then examined at the final time of $T = 40s$ and plotted against one another. Additionally, the L-2 Norm error (compared to the baseline "analytical" solution and defined by Eq. 43) 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.

$$Error = \sqrt{\sum_{j=1}^{n} R_j^2} = \sqrt{\sum_{j=1}^{n} (U_j - U_{j,analytical})^2} \tag{43}$$

When no source term is present, it can clearly be seen in Figure 4 that the Explicit Euler method results in the largest error while the Implicit Euler and Crank-Nicolson methods perform much better. After letting the solution run for $T = 40s$ (Figure 4a), this can most clearly be seen at the right boundary, where the Neumann boundary condition (See Eq. 4) has been enforced. Additionally, as seen in Figure 4b, as the solution evolves further from its initial condition, the Explicit Euler method increases in error proportional to $\frac{1}{T}$ while the Implicit and Crank-Nicolson methods increase in error proportional to $\frac{1}{T^2}$.



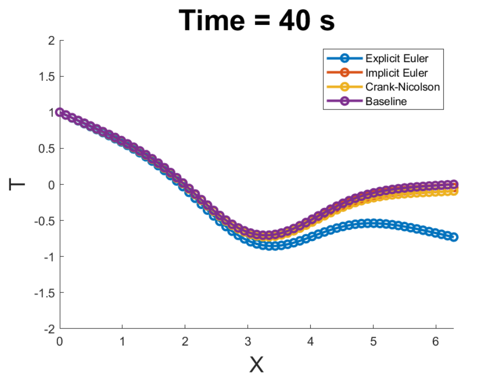
(a) Comparison of Temperature Profiles at T = 40s For Each Method



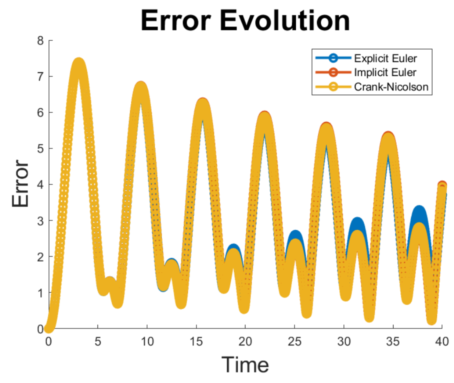
(b) Error Evolution Over Time ($T = 0s$ to $T = 40s$)

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

When a source term is present, as defined by Eq.5, a similar pattern is observed (Figure 5). At $Time = 40s$ (Figure 5a), The Explicit Euler method has the largest error, while the Implicit Euler and Crank-Nicolson methods have the smallest error. However, examining Figure 5b indicates the total error oscillates at the same frequency of the source function



(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

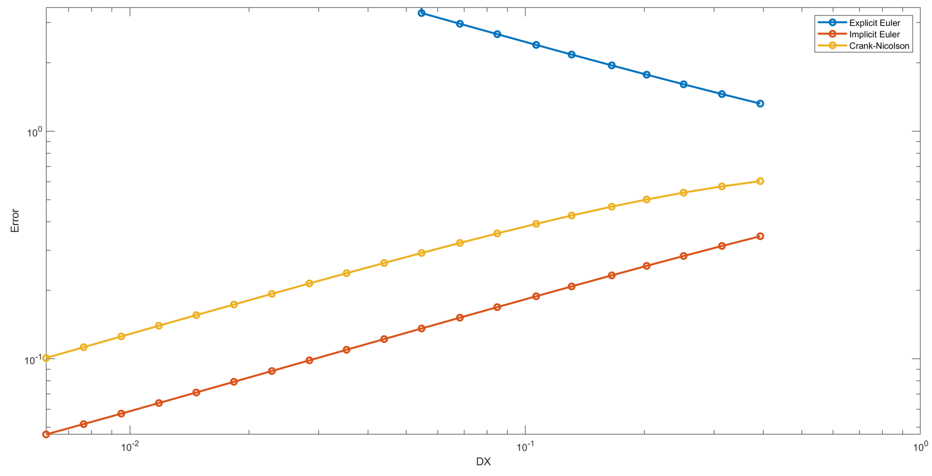


Figure 6: Error as a Function of Spacial Step

7.3 Temporal Resolution

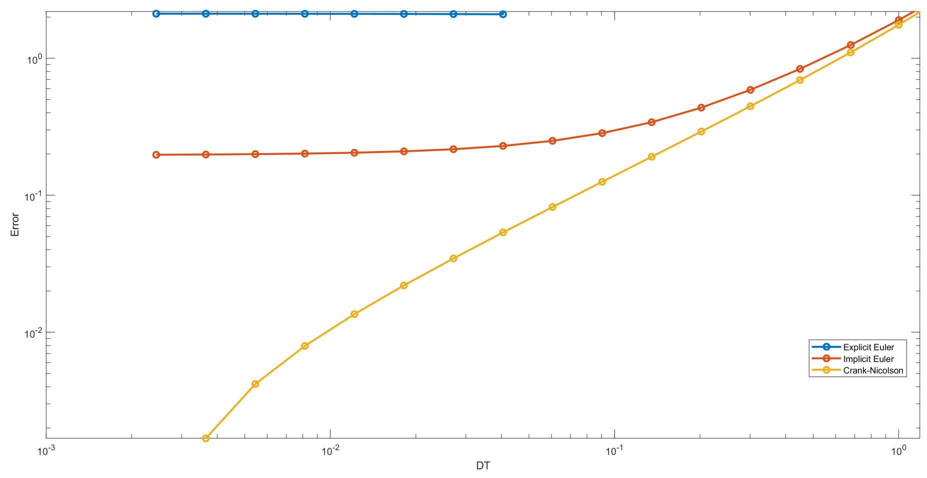


Figure 7: Error as a Function of Time Step

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 = TMAX ./ DT + 1;          % 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
            L(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
    L(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;

%% 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'));
ResB = BASELINE - BB(:, 1:end-1); ErrorB = sqrt(sum(ResB' .* ResB'));
ResC = BASELINE - CC(:, 1:end-1); ErrorC = sqrt(sum(ResC' .* ResC'));

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', 24); legend('show');

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

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

plot(XX, AA(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
plot(XX, BB(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
plot(XX, CC(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Crank-Nicolson');
plot(XX(1:end-1), BASELINE(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Baseline');
```

```
xlabel('X', 'FontSize', 14); ylabel('Temperature [u]', 'FontSize', 14);
title('Time = 40 s', 'FontSize', 24); legend('show'); axis([0 L -2 2]);

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

clear all;

%% 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'));
ResB = BASELINE - BB(:, 1:end-1); ErrorB = sqrt(sum(ResB' .* ResB'));
ResC = BASELINE - CC(:, 1:end-1); ErrorC = sqrt(sum(ResC' .* ResC'));

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', 24); 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;
```

```
plot(XX, AA(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Explicit Euler');
plot(XX, BB(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Implicit Euler');
plot(XX, CC(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Crank-Nicolson');
plot(XX(1:end-1), BASELINE(end, :), '-', 'LineWidth', 2, 'DisplayName', 'Baseline');

xlabel('X', 'FontSize', 14); ylabel('Temperature [u]', 'FontSize', 14);
title('Time = 40 s', 'FontSize', 24); legend('show'); axis([0 L -2 2]);

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

clear all;

%% 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 = 2^14; 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));

        hold off;
        for jj = 0:SCHEME
            loglog(DT(1:ii), RVX(jj + 1, 1:ii), '-o', 'LineWidth', 2, 'DisplayName', sch{jj+1});
            hold on;
        end

        xlabel('DT', 'Fontsize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Time Step', 'FontSize', 24);
        ylim([0, 1]);
        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 = 2^10; MINNX = 2^4;

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));

        hold off;
        for jj = 0:SCHEME
            loglog(DX(1:ii), RVT(jj + 1, 1:ii), '-o', 'LineWidth', 2, 'DisplayName', sch{jj+1});
            hold on;
        end

        xlabel('DX', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Spatial Step', 'FontSize', 24);
        ylim([0, 1]);
        legend('show');
    end
end

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

clear all;

%% 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 = 2^14; 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));

        hold off;
        for jj = 0:SCHEME
            loglog(DT(1:ii), RVX(jj + 1, 1:ii), '-o', 'LineWidth', 2, 'DisplayName', sch{jj+1});
            hold on;
        end

        xlabel('DT', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Time Step', 'FontSize', 24);
        ylim([0, 1]);
        legend('show');
    end
end

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

%% 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 = 2^10; MINNX = 2^4;

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));

        hold off;
        for jj = 0:SCHEME
            loglog(DX(1:ii), RVT(jj + 1, 1:ii), '-o', 'LineWidth', 2, 'DisplayName', sch{jj+1});
            hold on;
        end

        xlabel('DX', 'FontSize', 14); ylabel('Error', 'FontSize', 14);
        title('Error as a Function of Spatial Step', 'FontSize', 24);
        ylim([0, 1]);
        legend('show');
    end
end

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

clear all;
```