Stability Analysis

Of the Linear, One-Dimensional Heat Conduction Equation

MEMS1055, Computer Aided Analysis in Transport Phenomena
Dr. Peyman Givi
Seth Strayer
3/5/19

Contents

1 Introduction **1**

2 Formulation **1**
2.1 FTCS Method 1
2.2 BTCS Method 2
2.3 Steady-State Solution 3

3 Results **4**
3.1 FTCS Method 4
3.2 BTCS Method 6
3.3 Optimization of Grid Parameters 7
3.4 Verification 9

4 Conclusion **10**

5 Appendix **11**
5.1 FTCS Method Code Listings 11
5.2 LU Factorization to Solve a Tridiagonal Matrix 16
5.3 References 18

List of Figures **18**

1 Introduction

In this report we consider the linear, one-dimensional heat conduction equation given by

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \qquad t \geq 0, \quad 0 \leq x \leq 1 \tag{1}$$

and subject to the following boundary conditions:

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

$$u(0, t) = 1 \quad t > 0^+ \tag{3}$$

$$\frac{\partial u}{\partial t}(1, t) = 1 \quad t > 0^+ \tag{4}$$

We would like to solve for the steady-state temperature distribution, $u(x)$, using both Forward Time, Centered Space (FTCS) and Backward Time, Centered Space (BTCS) finite-differencing methods. FTCS is an explicit solving method which has desriable computational efficiency but is only *conditionally* stable. BTCS is an implicit method which is computationally more expensive than FTCS, but is *unconditionally* stable.[1,2] In order to simplify the analysis, a constant diffusivity value of

$$\alpha = 1$$

will be used. To follow is an investigation of the advantages and disadvantages of each of the methods and a discussion of the effects that the time and spatial grid-spacing, Δt and Δx, have on the accuracy of the solutions.

2 Formulation

2.1 FTCS Method

In this subsection, the Forward Time, Central Space (FTCS) approximation is derived. Consider the linear, one-dimensional heat conduction given by Equation 1:

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

Approximating the time-derivative using *forward differencing* yields:

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{O}(\Delta t) \tag{5}$$

Approximating the spacial-derivative using *central differencing* yields:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{6}$$

Substituting Equations 5 and 6 into Equation 1 and collecting truncation error terms yields:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} + \mathcal{O}(\Delta t, \Delta x^2) \tag{7}$$

By dropping the truncation error terms and solving for u_i^{n+1} in Equation 7, we get:

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

Equation 8 represents the FTCS approximation to the heat equation. Finally, by implementing the diffusion coefficient,

$$r = \frac{\alpha \Delta t}{\Delta x^2} \tag{9}$$

and rearranging the terms in Equation 8, we find that:

$$u_i^{n+1} = r u_{i+1}^n + (1 - 2r) u_i^n + r u_{i-1}^n \tag{10}$$

This is known as an *explicit* scheme, meaning that the values of u_i^{n+1} are updated independently of each other. The computational efficiency of this method is desirable for many situations, as solutions are easily obtained by iterating through an outer loop in the time doman ($n = 1 : N$), and an inner loop through the spatial domain ($i = 1 : M$), where N, M are the total number of gridpoints in time and space, respectively. However, the explicit scheme is known as conditionally stable. I.e., stable solutions are only obtained for values of r satisfying

$$r < \frac{1}{2} \tag{11}$$

For values of r that do not lay in this domain, FTCS solutions are known as unstable and have the possibility of oscillating and growing with the number of iterations. Thus, stability (convergence) of the solution is highly dependent upon the diffusivity, α (in this case $\alpha = 1$ so convergence is independent of α), and the grid parameters, Δt and Δx. The relationship between these parameters and the stability of the method will be further investigated throughout this report.

2.2 BTCS Method

In this subsection, the Backward Time, Central Space (BTCS) approximation is derived. Consider the linear, one-dimensional heat conduction given by Equation 1:

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

Approximating the time-derivative using *backward differencing* yields:

$$\frac{\partial u}{\partial t} = \frac{u_i^n - u_i^{n-1}}{\Delta t} + \mathcal{O}(\Delta t) \tag{12}$$

Approximating the spacial-derivative using *central differencing* yields:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{13}$$

Substituting Equations 12 and 13 into Equation 1, collecting truncation error terms, and making the shift $(n - 1) = n$ and $n = (n + 1)$ yields:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2} + \mathcal{O}(\Delta t, \Delta x^2) \tag{14}$$

By dropping the truncation error terms and rearranging Equation 14, we get:

$$-\frac{\alpha}{\Delta x^2} u_{i-1}^{n+1} + \left(\frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2}\right) u_i^{n+1} - \frac{\alpha}{\Delta x^2} u_{i+1}^{n+1} = \frac{1}{\Delta t} u_i^n \tag{15}$$

This is known as an *implicit* scheme, as solving for the term u_i^{n+1} is only possible by solving a system of numerical equations at each time step. This scheme is desirable as it is *unconditionally stable*, meaning that the grid parameters Δt and Δx may take on any values. However, this scheme is clearly much more computationally expensive than the FTCS Method. To go about solving for the values u_i^{n+1}, we can write the system of equations presented by Equation 15 in matrix form as:

$$Au^{n+1} = u^n$$

or

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & 0 & 0 & 0 & a_M & b_M \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ u_{M-1}^{n+1} \\ u_M^{n+1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{M-1} \\ d_M \end{bmatrix} \tag{16}$$

where the coefficients of the interior nodes are given by

$$a_i = -\frac{\alpha}{\Delta x^2} \quad b_i = \frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2}$$
$$c_i = -\frac{\alpha}{\Delta x^2} \quad d_i = \frac{1}{\Delta t} u_i^n$$

for $i = 2, 3, ..., M-1$. The Dirichlet and Neumann boundary conditions presented by Equations 3 and 4 are implemented by

$$b_1 = 1, \quad c_1 = 0, \quad d_1 = 1$$
$$A_M = 0, \quad b_M = 1, \quad d_M = \Delta x + u_{M-1}$$

The system presented by Equation 16 is efficiently solved for using LU Factorization. A more detailed outline of the LU Factorization algorithim is presented in the Appendix, Section 5.2.

2.3 Steady-State Solution

The steady-state solution to Equation 1 is solved for by setting the time-derivative $\frac{\partial u}{\partial t} = 0$:

$$\frac{\partial^2 u}{\partial x^2} = 0 \tag{17}$$

Integrating twice yields:

$$T(x) = C_1 x + C_2$$

Applying the boundary conditions given by Equations 3 and 4 readily yields:

$$T(x) = x + 1 \tag{18}$$

Equation 18 represents the steady-state solution to Equation 1. Evaluating at $x = 0$ and $x = 1$, we have:

$$T(0) = 1 \tag{19}$$

$$T(1) = 2 \tag{20}$$

These values will be validated with the computational solution to ensure stability of both the FTCS and BTCS methods.

3 Results

3.1 FTCS Method

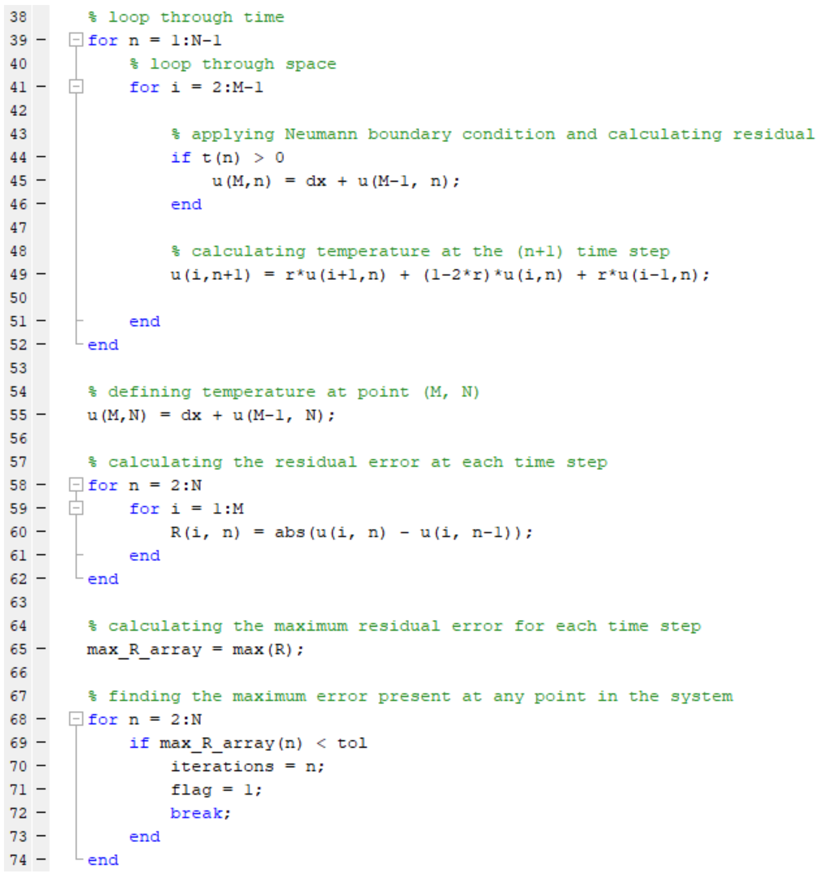
The MATLAB script used to implement the FTCS method is "*Project_2_FTCS.m*". Full code listings for this script can be found in the Appendix, Section 5.1. This script runs for a total of 5 seconds. Afterwards, the number of iterations needed to ensure convergence is calculated. For this study, convergence is acheived when the maximum difference between the values u_i^{n+1} and u_i^n, say, R, is less than some tolerance. This script uses a tolerance of

$$tol = 1 \cdot 10^{-6}$$

such that convergence is achieved when

$$max(R) < 1 \cdot 10^{-6}$$

This tolerance ensures convergence of the computational solution. For the following results, a constant diffusivity of $\alpha = 1$ and a constant spacial grid size of $\Delta x = 0.1$ was used while the time grid size was altered as to change the magnitude of the parameter r. The iterative scheme used to solve for the steady-state solution is shown in Figure 1.



```
% loop through time
for n = 1:N-1
    % loop through space
    for i = 2:M-1

        % applying Neumann boundary condition and calculating residual
        if t(n) > 0
            u(M,n) = dx + u(M-1, n);
        end

        % calculating temperature at the (n+1) time step
        u(i,n+1) = r*u(i+1,n) + (1-2*r)*u(i,n) + r*u(i-1,n);

    end
end

% defining temperature at point (M, N)
u(M,N) = dx + u(M-1, N);

% calculating the residual error at each time step
for n = 2:N
    for i = 1:M
        R(i, n) = abs(u(i, n) - u(i, n-1));
    end
end

% calculating the maximum residual error for each time step
max_R_array = max(R);

% finding the maximum error present at any point in the system
for n = 2:N
    if max_R_array(n) < tol
        iterations = n;
        flag = 1;
        break;
    end
end
```

Figure 1: Iterative Scheme for the FTCS Method

The solution for $r = 0.25 < \frac{1}{2}$ is shown in Figure 2.

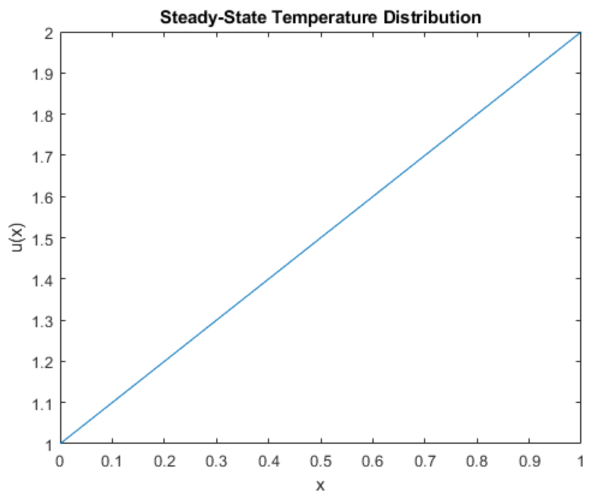


Figure 2: Steady-State Temperature Distribution for $r = 0.25$

Indeed, this solution almost exactly matches that of the analytical solution, given by Equation 18. The solution for $r = 1 > \frac{1}{2}$ is shown in Figure 3.

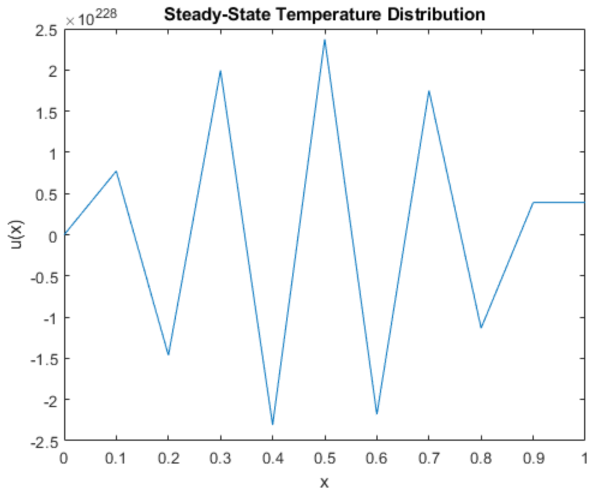
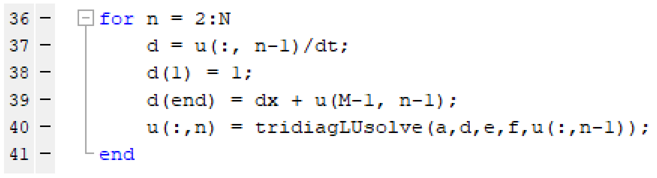


Figure 3: Steady-State Temperature Distribution for $r = 1$

The highly unstable and oscillatory nature of this solution verifies that the FTCS method is conditionally stable for values of $r < \frac{1}{2}$.

3.2 BTCS Method

The MATLAB script used to implement the BTCS Method is "*Project_2_BTCS.m*". This script runs for a total of 5 seconds. Convergence criteria for the BTCS method is the same of that as the FTCS method. Likewise, for the following results, a constant diffusivity of $\alpha = 1$ and a constant spacial grid size of $\Delta x = 0.1$ was used while the time grid size was altered as to change the magnitude of the parameter r. The iterative scheme used to solve for the steady-state solution is shown in Figure 4.



```
for n = 2:N
    d = u(:, n-1)/dt;
    d(1) = 1;
    d(end) = dx + u(M-1, n-1);
    u(:,n) = tridiagLUsolve(a,d,e,f,u(:,n-1));
end
```

Figure 4: Iterative Scheme for the BTCS Method

This script uses LU Factorization to solve for the temperature distribution at each time step. Code listings used to implement LU factorization are given in Section 5.2 . As mentioned, results for this method are independent of the parameter r. The steady-state solution for various values of r using the BTCS method is shown in Figure 5.

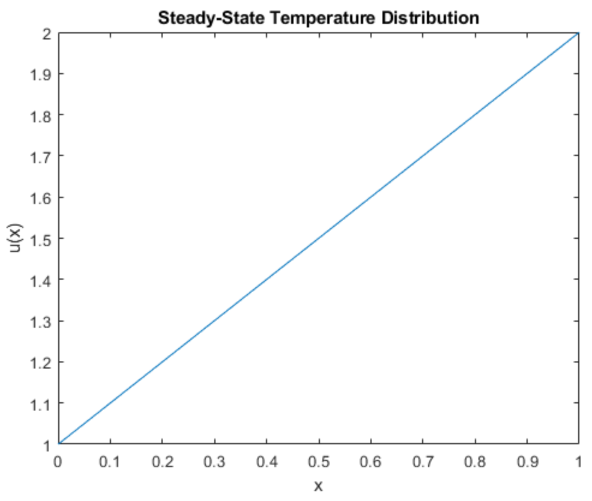


Figure 5: Steady-State Temperature Distribution for any r

3.3 Optimization of Grid Parameters

This section will discuss the optimization of the grid parameters, Δt and Δx for both the FTCS and BTCS methods. The term "optimum" loosely refers to the grid parameter values which minimize the maximum residual error present at any time *not* at steady state, while also minimizing computational expense. The time chosen to analyze the residual error was chosen arbitrarily. The only constraint on this time step value is that the system must not yet be at steady-state. For both methods, it was determined that the $n = 100$ time step had not yet reached steady-state, and thus was sufficient for the analysis.

FTCS Method:

First, Δt will be halfed until the decrease in residual error no longer exceeds the decrease in grid size. Results for $\Delta x = 0.1$ and varying Δt are given in Table 1.

Δt	Δx	r	Residual Error
5.0000e-03	0.1	5.0000e-01	7.289773e-03
2.5000e-03	0.1	2.5000e-01	7.090526e-03
1.2500e-03	0.1	1.2500e-01	4.484370e-03
6.2500e-04	0.1	6.2500e-02	2.818443e-03
3.1250e-04	0.1	3.1250e-02	2.375967e-03
1.5625e-04	0.1	1.5625e-02	2.305904e-03
7.8125e-05	0.1	7.8125e-03	2.241860e-03

Table 1: FTCS Optimization of Δt, $n = 100$

We see that the decrease in residual error plataues at a grid size of $\Delta t = 3.1250e - 04$. Thus, we will hold this value constant as we half the spacial grid sizing. Note that due to the nature of the diffusion coefficient, only spacial grid sizes of $\Delta x = 0.1, 0.05, 0.025$, may be used, as to prevent $r > \frac{1}{2}$. Results for $\Delta t = 3.1250e - 04$ and varying Δx are given in Table 2.

Δx	Δt	r	Residual Error
0.1	3.1250e-04	3.1250e-02	2.375967e-03
0.05	3.1250e-04	1.2500e-01	2.480471e-03
0.025	3.1250e-04	5.0000e-01	4.959318e-03

Table 2: FTCS Optimization of Δx, $n = 100$

We see that decreasing the spatial grid size only increases the residual error. Thus, the optimum grid sizes for the FTCS method which minimize residual error and maximize computational efficiency are:

$$\Delta t = 3.1250e - 04, \quad \Delta x = 0.1 \tag{21}$$

BTCS Method:

Optimization methods for the BTCS method are the same as that of the FTCS method. Results for $\Delta x = 0.1$ and varying Δt are given in Table 3.

Δt	Δx	r	Residual Error
5.0000e-03	0.1	5.0000e-01	7.604029e-03
2.5000e-03	0.1	2.5000e-01	6.951906e-03
1.2500e-03	0.1	1.2500e-01	4.358533e-03
6.2500e-04	0.1	6.2500e-02	2.788937e-03
3.1250e-04	0.1	3.1250e-02	2.371307e-03
1.5625e-04	0.1	1.5625e-02	2.283505e-03
7.8125e-05	0.1	7.8125e-03	2.229286e-03

Table 3: BTCS Optimization of Δt, $n = 100$

We see that the decrease in residual error plataues at a grid size of $\Delta t = 3.1250e - 04$. Thus, we will hold this value constant as we half the spacial grid sizing. Here, any values of Δx may be used, since the BTCS method is unconditionally stable and r may take on any value. Results for $\Delta t = 3.1250e - 04$ and varying Δx are given in Table 4.

Δx	Δt	r	Residual Error
0.1	3.1250e-04	3.1250e-02	2.371307e-03
0.05	3.1250e-04	1.2500e-01	2.467693e-03
0.025	3.1250e-04	5.0000e-01	2.471452e-03
0.0125	3.1250e-04	2.0	2.470966e-03
6.2500e-03	3.1250e-04	8.0	2.469281e-03
3.1250e-03	3.1250e-04	32.0	2.467602e-03

Table 4: BTCS Optimization of Δx, $n = 100$

Here we see that the lowest residual error occurs at $\Delta x = 0.1$. Thus, the optimum grid sizes for the BTCS method which minimize residual error and maximize computational efficiency are:

$$\Delta t = 3.1250e - 04, \quad \Delta x = 0.1 \tag{22}$$

3.4 Verification

This section will provide verification for the accuracy of our model. First, we will plot the residual error vs. the number of iterations for grid space $\Delta x = 0.1$ and $\Delta t = 7.8125e-05$ ($r = 7.8125e-03$). This plot is shown in Figure 6.

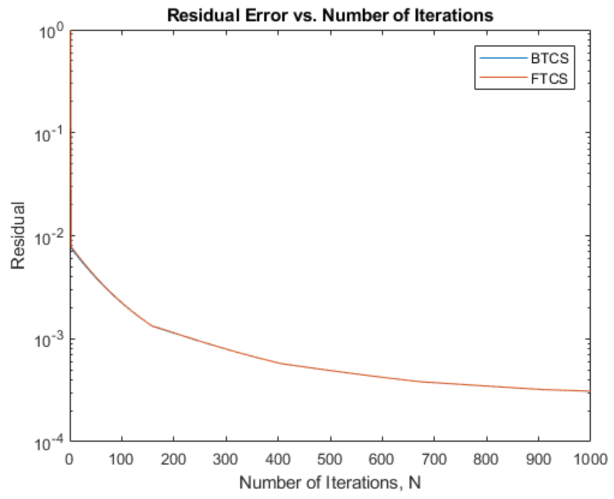


Figure 6: Residual Error vs. Number of Iterations

This plot verifies that as the number of iterations increases, the residual error decreases, which is expected for computational models. Figure 7 shows the residual error vs. mesh grid size factor. For this result, the spacial grid size was kept constant at $\Delta x = 0.1$ while the time grid size was halved from $\Delta t = 5.0000e-03$ to $\Delta t = 7.8125e-05$.

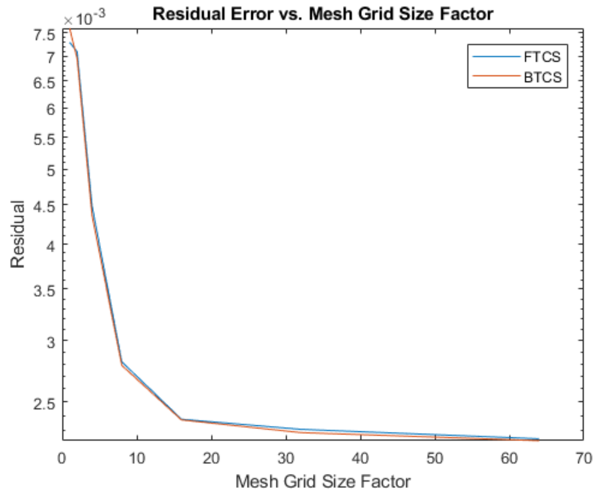


Figure 7: Residual Error vs. Mesh Grid Size Factor

This plot verifies that as the mesh grid size is refined, the residual error decreases, which is also expected for computational models. Both of these results verify stable convergence of our model.

4 Conclusion

In this report, the steady-state temperature distribution, $u(x)$, of the linear, one-dimensional heat conduction equation (1) was solved for using both the FTCS and BTCS approximation methods. The advantages/disadvantages of each of the methods is summarized below.

- FTCS is an explicit scheme which saves on computational expense but is *conditionally* stable. The solution accuracy is highly dependent upon the diffusivity, α, and the grid parameters, Δt and Δx. Thus, careful attention must be given when choosing these parameters to ensure stability. This scheme is useful for scenarios where it is given that the diffusion coefficient $r = \frac{\alpha \Delta t}{\Delta x^2} < \frac{1}{2}$.
- BTCS is an implicit scheme which is *unconditionally* stable but computationally more expensive than the FTCS method. The solution accuracy is not dependent upon any parameters of the system. This scheme is useful for scenarios in which the choice of α, Δt, and Δx causes $r = \frac{\alpha \Delta t}{\Delta x^2} > \frac{1}{2}$. Thus, the BTCS method is desirable for problems involving large values of diffusivity, α, large time steps, or small spacial grid sizing.

Furthermore, from Section 3.3, the optimum values of the grid parameters Δt and Δx for *both* methods was found to be:

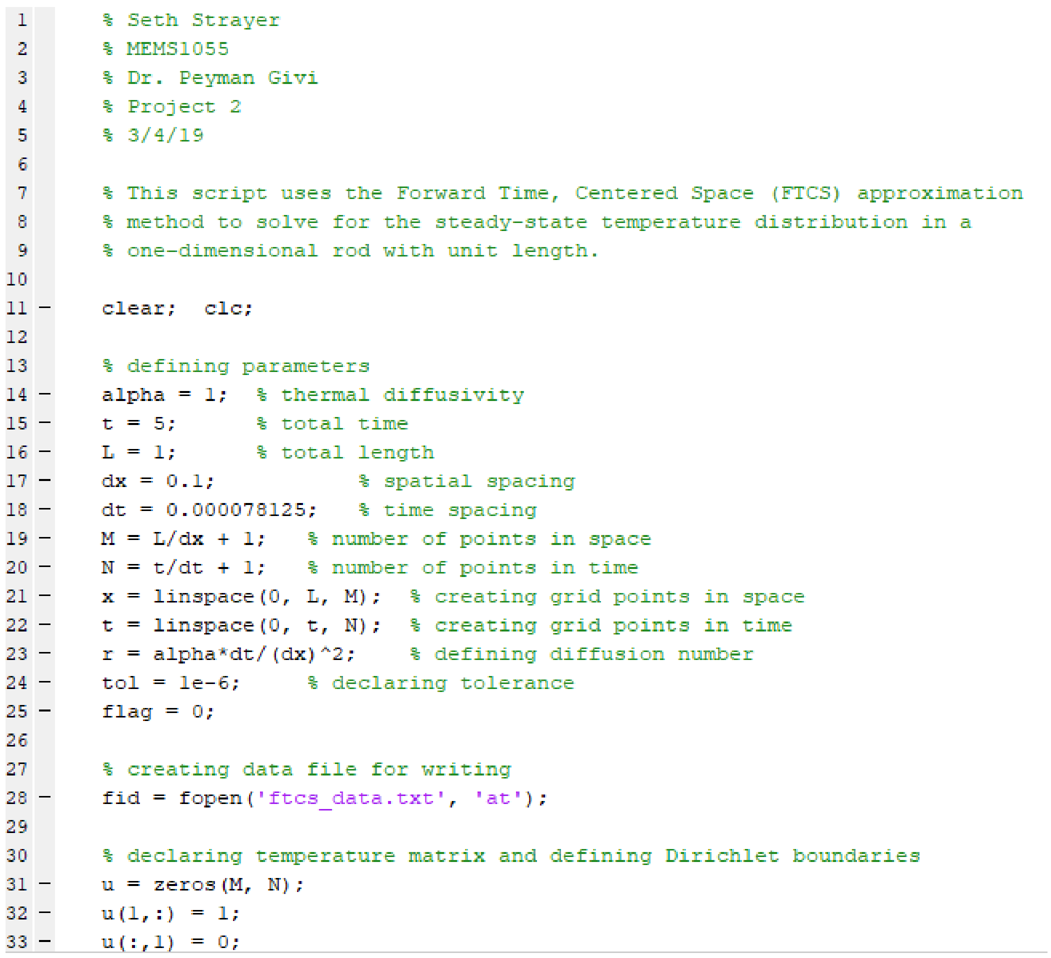
$$\Delta t = 3.1250e-04, \quad \Delta x = 0.1$$

These values represent the optimum grid sizing - i.e., the grid sizing which minimizes the maximum residual error present at any time *not* at steady-state, while also minimizing computational expense. Note that due to the setup of the code, only certain values of Δt and Δx were able to be chosen. These parameters could be more highly optimized by specifying the number of grid points in time and space, rather than specifying the grid spacing.

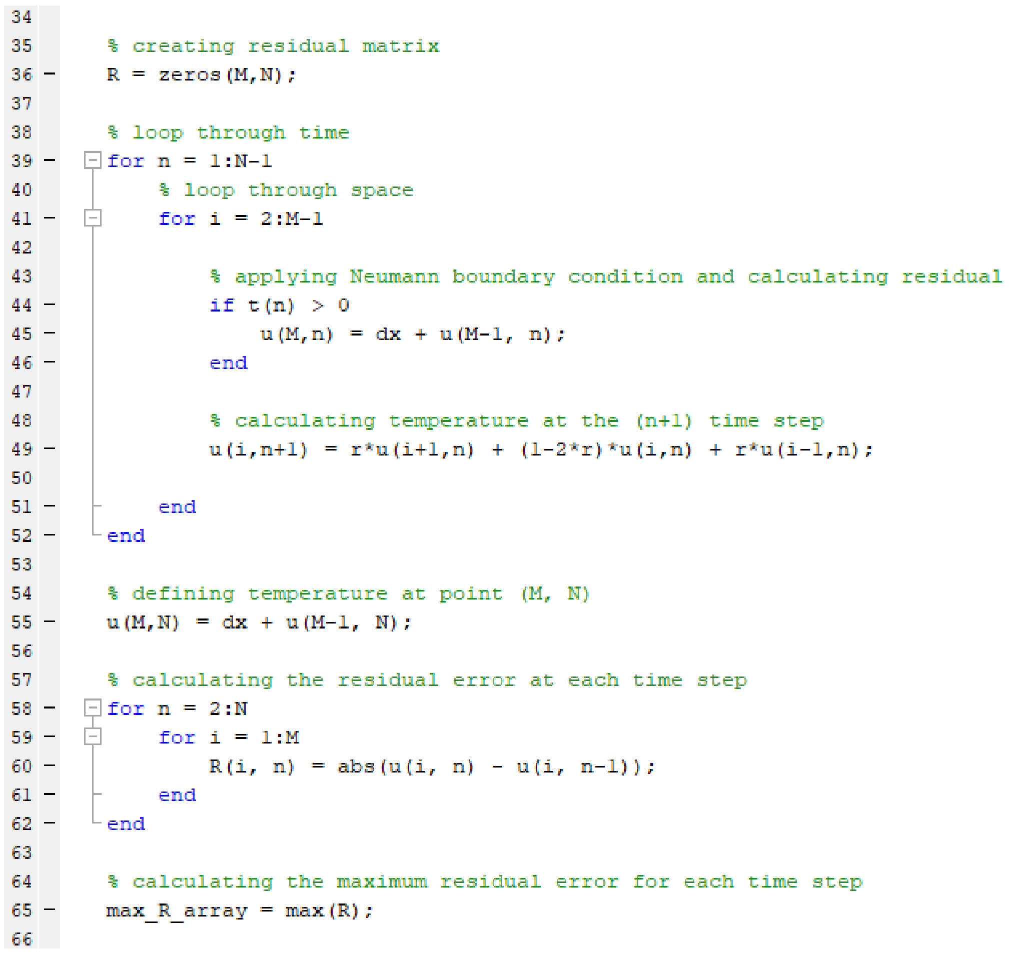
Furthermore, the grid parameters could have taken on values which were not listed in Section 3.3, especially for the stable nature of the BTCS method. This is something that could be further investigated in future reports to see if there is a more trivial relationship between the grid sizing and the maximum residual error.

5 Appendix

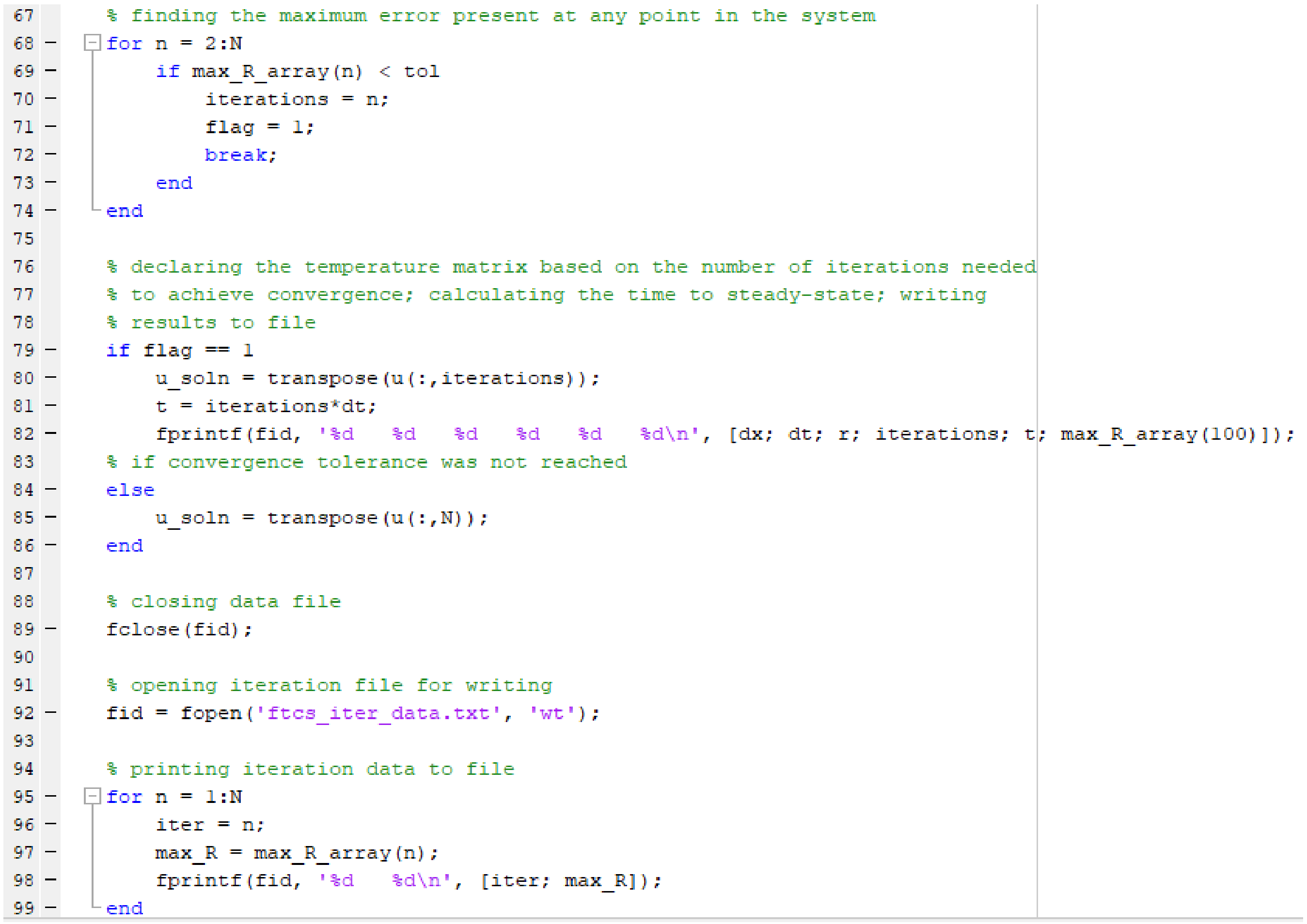
5.1 FTCS Method Code Listings



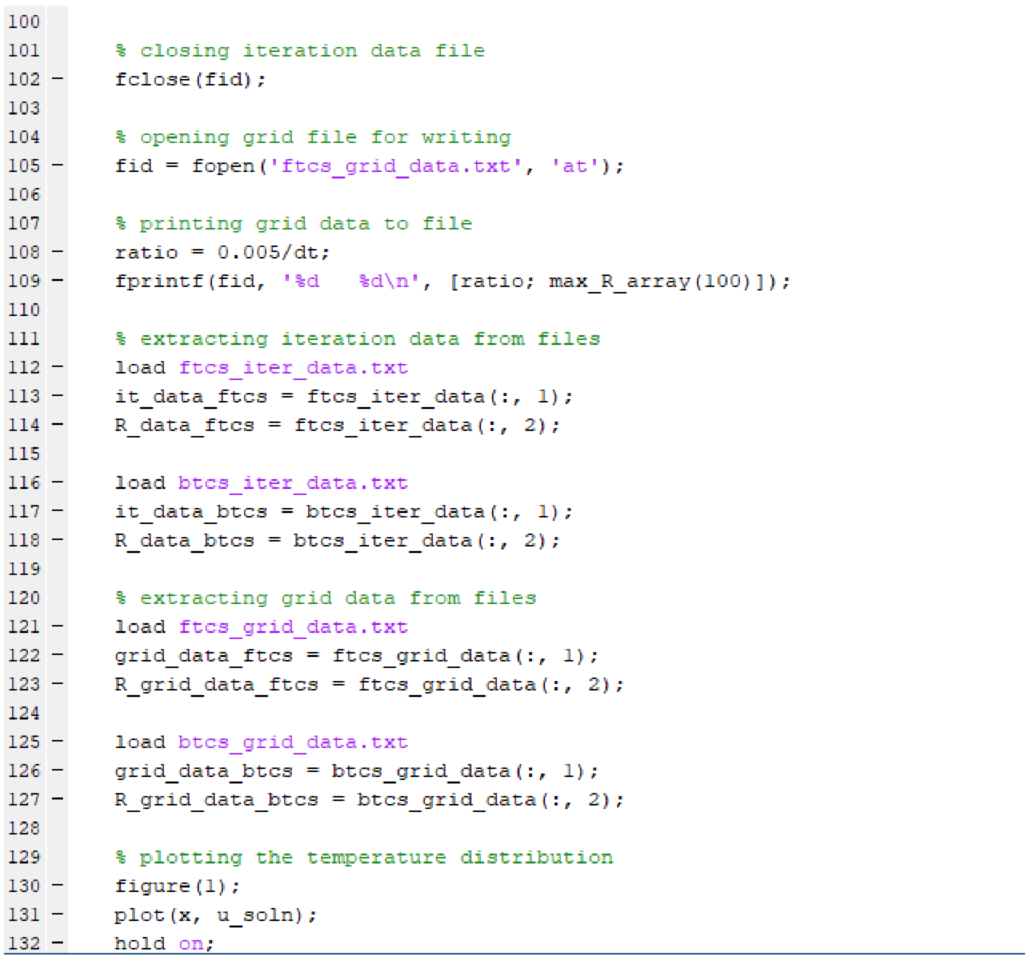
```matlab
% Seth Strayer
% MEMS1055
% Dr. Peyman Givi
% Project 2
% 3/4/19

% This script uses the Forward Time, Centered Space (FTCS) approximation
% method to solve for the steady-state temperature distribution in a
% one-dimensional rod with unit length.

clear;  clc;

% defining parameters
alpha = 1;  % thermal diffusivity
t = 5;      % total time
L = 1;      % total length
dx = 0.1;           % spatial spacing
dt = 0.000078125;   % time spacing
M = L/dx + 1;   % number of points in space
N = t/dt + 1;   % number of points in time
x = linspace(0, L, M);  % creating grid points in space
t = linspace(0, t, N);  % creating grid points in time
r = alpha*dt/(dx)^2;    % defining diffusion number
tol = 1e-6;     % declaring tolerance
flag = 0;

% creating data file for writing
fid = fopen('ftcs_data.txt', 'at');

% declaring temperature matrix and defining Dirichlet boundaries
u = zeros(M, N);
u(1,:) = 1;
u(:,1) = 0;
```



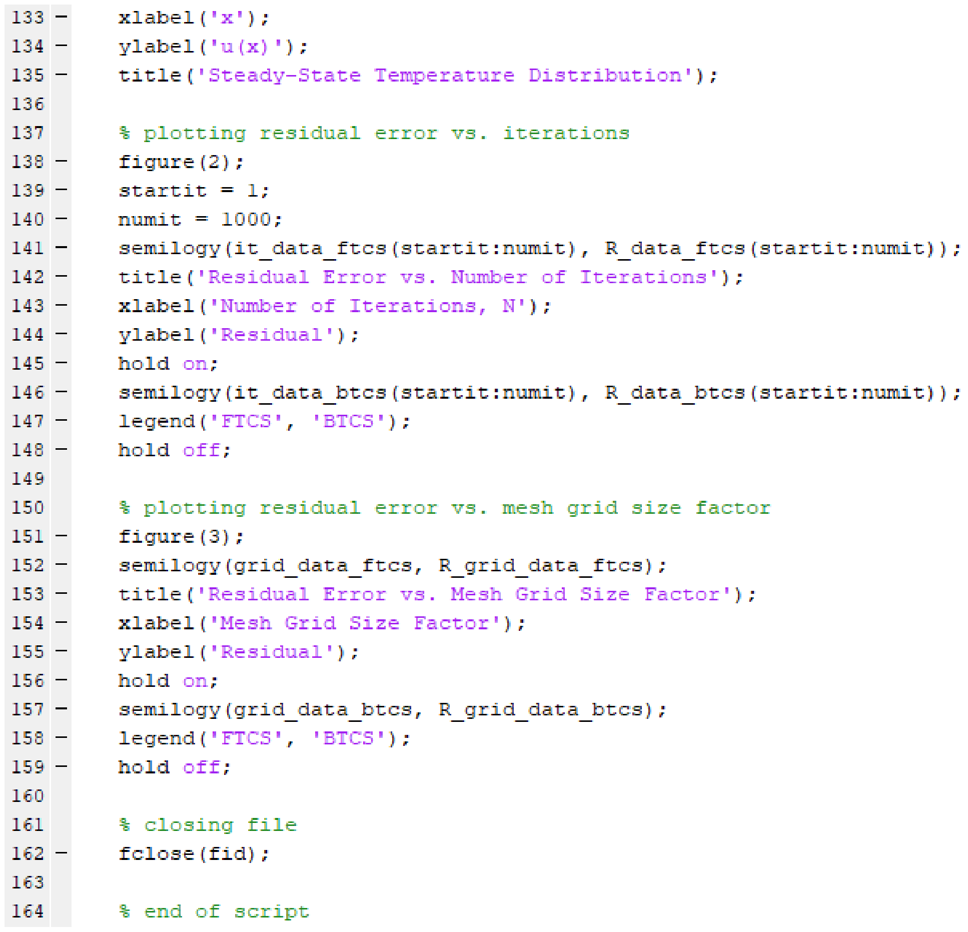
```matlab

% creating residual matrix
R = zeros(M,N);

% loop through time
for n = 1:N-1
    % loop through space
    for i = 2:M-1

        % applying Neumann boundary condition and calculating residual
        if t(n) > 0
            u(M,n) = dx + u(M-1, n);
        end

        % calculating temperature at the (n+1) time step
        u(i,n+1) = r*u(i+1,n) + (1-2*r)*u(i,n) + r*u(i-1,n);

    end
end

% defining temperature at point (M, N)
u(M,N) = dx + u(M-1, N);

% calculating the residual error at each time step
for n = 2:N
    for i = 1:M
        R(i, n) = abs(u(i, n) - u(i, n-1));
    end
end

% calculating the maximum residual error for each time step
max_R_array = max(R);

```



```matlab
% finding the maximum error present at any point in the system
for n = 2:N
    if max_R_array(n) < tol
        iterations = n;
        flag = 1;
        break;
    end
end

% declaring the temperature matrix based on the number of iterations needed
% to achieve convergence; calculating the time to steady-state; writing
% results to file
if flag == 1
    u_soln = transpose(u(:,iterations));
    t = iterations*dt;
    fprintf(fid, '%d   %d   %d   %d   %d   %d\n', [dx; dt; r; iterations; t; max_R_array(100)]);
% if convergence tolerance was not reached
else
    u_soln = transpose(u(:,N));
end

% closing data file
fclose(fid);

% opening iteration file for writing
fid = fopen('ftcs_iter_data.txt', 'wt');

% printing iteration data to file
for n = 1:N
    iter = n;
    max_R = max_R_array(n);
    fprintf(fid, '%d   %d\n', [iter; max_R]);
end
```



```matlab

    % closing iteration data file
    fclose(fid);

    % opening grid file for writing
    fid = fopen('ftcs_grid_data.txt', 'at');

    % printing grid data to file
    ratio = 0.005/dt;
    fprintf(fid, '%d   %d\n', [ratio; max_R_array(100)]);

    % extracting iteration data from files
    load ftcs_iter_data.txt
    it_data_ftcs = ftcs_iter_data(:, 1);
    R_data_ftcs = ftcs_iter_data(:, 2);

    load btcs_iter_data.txt
    it_data_btcs = btcs_iter_data(:, 1);
    R_data_btcs = btcs_iter_data(:, 2);

    % extracting grid data from files
    load ftcs_grid_data.txt
    grid_data_ftcs = ftcs_grid_data(:, 1);
    R_grid_data_ftcs = ftcs_grid_data(:, 2);

    load btcs_grid_data.txt
    grid_data_btcs = btcs_grid_data(:, 1);
    R_grid_data_btcs = btcs_grid_data(:, 2);

    % plotting the temperature distribution
    figure(1);
    plot(x, u_soln);
    hold on;
```



```
xlabel('x');
ylabel('u(x)');
title('Steady-State Temperature Distribution');

% plotting residual error vs. iterations
figure(2);
startit = 1;
numit = 1000;
semilogy(it_data_ftcs(startit:numit), R_data_ftcs(startit:numit));
title('Residual Error vs. Number of Iterations');
xlabel('Number of Iterations, N');
ylabel('Residual');
hold on;
semilogy(it_data_btcs(startit:numit), R_data_btcs(startit:numit));
legend('FTCS', 'BTCS');
hold off;

% plotting residual error vs. mesh grid size factor
figure(3);
semilogy(grid_data_ftcs, R_grid_data_ftcs);
title('Residual Error vs. Mesh Grid Size Factor');
xlabel('Mesh Grid Size Factor');
ylabel('Residual');
hold on;
semilogy(grid_data_btcs, R_grid_data_btcs);
legend('FTCS', 'BTCS');
hold off;

% closing file
fclose(fid);

% end of script
```

5.2 LU Factorization to Solve a Tridiagonal Matrix

The system of equations presented by Equation 16 can be efficiently solved for by using LU Factorization. It is desirable for us to us LU factorization in this case since the coefficient matrix is known to be positive definite, symmetric, and tridiagonal.[2] To use LU factorization, we wish to find L and U such that $A = LU$. After some manipulation, it is found that the L and U matrices corresponding to the tridiagonal coefficient matrix have the form:

$$L = \begin{bmatrix} e_1 & & & & & \\ a_2 & e_2 & & & & \\ & a_3 & e_3 & & & \\ & & \ddots & \ddots & & \\ & & & a_{M-1} & e_{M-1} & \\ & & & & a_M & e_M \end{bmatrix} \qquad U = \begin{bmatrix} 1 & f_1 & & & & \\ & 1 & f_2 & & & \\ & & 1 & f_3 & & \\ & & & \ddots & \ddots & \\ & & & & 1 & f_{M-1} \\ & & & & & 1 \end{bmatrix}$$

Thus, evaluating $A = LU$ term by term yields the corresponding system of equations:

$$\begin{gathered} e_1 = b_1 \\ e_1 f_1 = c_1 \\ \\ a_2 = a_2 \\ a_2 f_1 + e_2 = b_2 \\ e_2 f_2 = c_2 \\ \vdots \\ a_i = a_i \\ a_i f_{i-1} + e_i = b_i \\ e_i f_i = c_i \\ \vdots \\ a_M = A_M \\ a_M f_{M-1} + e_M = b_M \\ e_M f_M = c_M \end{gathered}$$

Solving for the unknowns at each step yields:

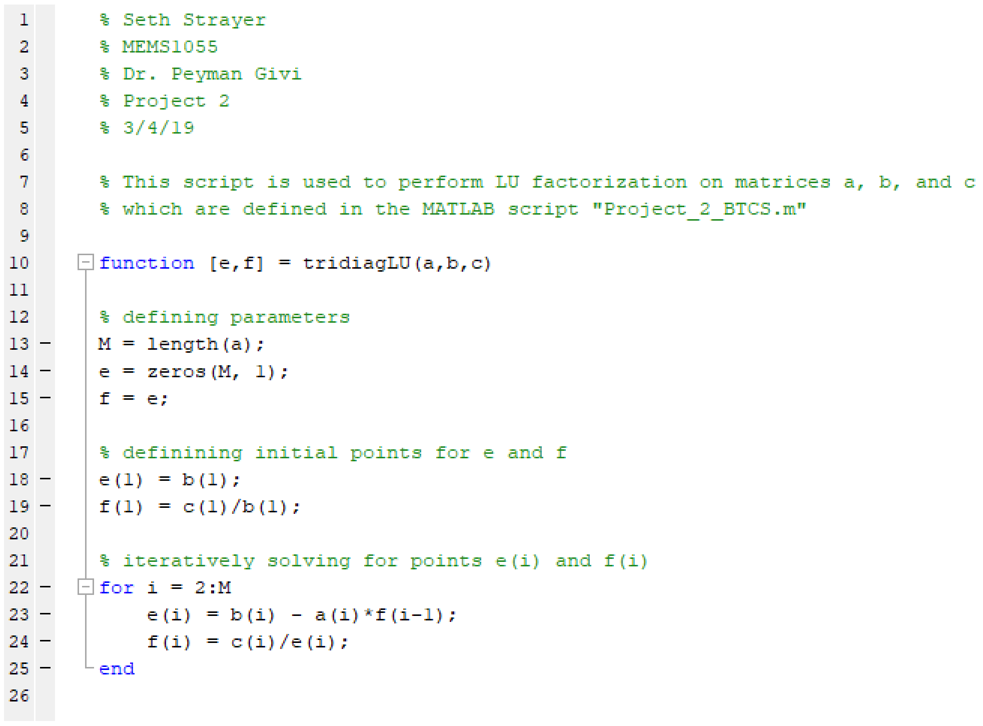
$$\begin{gathered} f_1 = c_1/e_1 = c_1/b_1 \\ \\ e_2 = b_2 - a_2 f_1 \\ f_2 = c_2/e_2 \\ \vdots \\ e_i = b_i - a_i f_{i-1} \\ f_i = c_i/e_i \end{gathered}$$

$$\vdots$$

$$e_M = b_M - a_M f_{M-1}$$

$$f_M = c_M / e_M$$

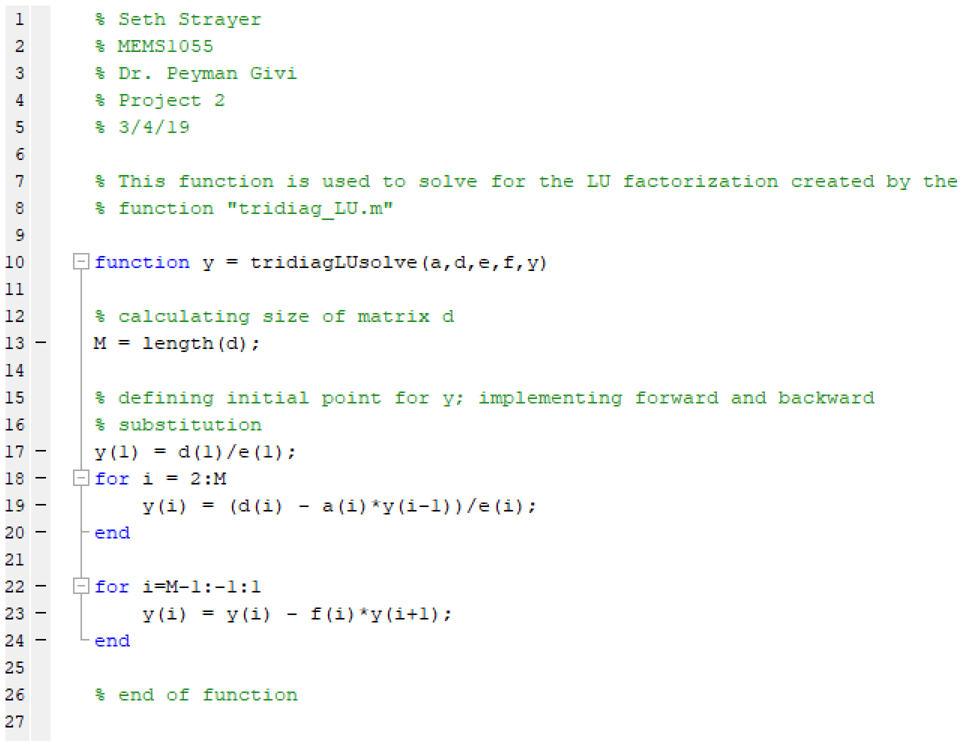
The corresponding MATLAB code used to solve for these unknowns is then easily given by Figure 8.



```
% Seth Strayer
% MEMS1055
% Dr. Peyman Givi
% Project 2
% 3/4/19

% This script is used to perform LU factorization on matrices a, b, and c
% which are defined in the MATLAB script "Project_2_BTCS.m"

function [e,f] = tridiagLU(a,b,c)

% defining parameters
M = length(a);
e = zeros(M, 1);
f = e;

% definining initial points for e and f
e(1) = b(1);
f(1) = c(1)/b(1);

% iteratively solving for points e(i) and f(i)
for i = 2:M
    e(i) = b(i) - a(i)*f(i-1);
    f(i) = c(i)/e(i);
end

```

Figure 8: LU Factorization Code

Finally, we use forward substitution to solve the system $Ly = b$ for y and backward substitution to solve $Ux = y$ for x. The corresponding code is given in Figure 9 on the following page. In general, these methods have been widely studied and are very effective in solving the BTCS iterative scheme, given the nature of the coefficient matrix A.[2]



```
% Seth Strayer
% MEMS1055
% Dr. Peyman Givi
% Project 2
% 3/4/19

% This function is used to solve for the LU factorization created by the
% function "tridiag_LU.m"

function y = tridiagLUsolve(a,d,e,f,y)

% calculating size of matrix d
M = length(d);

% defining initial point for y; implementing forward and backward
% substitution
y(1) = d(1)/e(1);
for i = 2:M
    y(i) = (d(i) - a(i)*y(i-1))/e(i);
end

for i=M-1:-1:1
    y(i) = y(i) - f(i)*y(i+1);
end

% end of function
```

Figure 9: LU Factorization Solver Code

5.3 References

[1] ”Implicit Versus Explicit Methods.” Flow-3D. Accessed February 20, 2019.
https://mat.iitm.ac.in/home/sjroy/Mechanical/Implicit versus Explicit Methods.htm.

[2] Recktenwald, Gerald W. Finite-Difference Approximations to the Heat Equation. Report. KTH Royal Institute of Technology. March 6, 2011. Accessed February 20, 2019.
http://www.nada.kth.se/ jjalap/numme/FDheat.pdf.

List of Figures

1 Iterative Scheme for the FTCS Method 4
2 Steady-State Temperature Distribution for $r = 0.25$ 5
3 Steady-State Temperature Distribution for $r = 1$ 5
4 Iterative Scheme for the BTCS Method 6
5 Steady-State Temperature Distribution for any r 6
6 Residual Error vs. Number of Iterations 9
7 Residual Error vs. Mesh Grid Size Factor 9
8 LU Factorization Code 17
9 LU Factorization Solver Code 18