Diffusion Simulation

1 Introduction

This is a simulation of heat diffusion. Diffusion is the rate at which something spreads or moves from a high area of concentration to a low area of concentration in accordance with a source or sink. In this case the simulation deals with the diffusion of temperature from a high area of temperature to a lower area of temperature. The system adheres to the following diffusion equation:

$$\frac{\delta u}{\delta t} = D\frac{\delta^2 u}{\delta x^2} \tag{1}$$

In order to simulate the system given, I calculate the initial condition:

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

Then I use the implicit central difference method and rearrange it to create a linear system. I then use the linear system to numerically calculate the next state of the system. For each iteration there is a boundary condition, which is given by the exact solution:

$$u(x, t) = e^{-Dt}cos(x) \tag{3}$$

After assigning the boundary conditions, the rest of the temperatures along the domain are calculated.

2 Algorithms

My simulation indirectly uses the implicit central difference method to solve the equation numerically by using an inverse matrix solution at each time step to update the system. The inverse matrix calculation is based off of a linear system obtained by looking at the implicit central difference method. Here is the implicit central difference scheme of the diffusion equation given in the introduction:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} \tag{4}$$

where Δt is the time step and Δx is the spacial resolution. The superscripts, n, references the time frame and the subscripts, i, reference the spacial frame. So, n+1, denotes the subsequent time interval and n denotes the current one. i represents the grid node we are looking at, i+1 is the node to the right and i-1 is the node to the left. u represents the temperature of the grid

node being referenced. We can rearrange the implicit central difference equation as follows:

$$\begin{aligned} u_i^{n+1} - u_i^n &= \frac{D\Delta t}{\Delta x^2}(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}) \\ u_i^{n+1} - \frac{D\Delta t}{\Delta x^2}(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}) &= u_i^n \\ u_i^{n+1} - \frac{D\Delta t}{\Delta x^2}u_{i+1}^{n+1} + \frac{D\Delta t}{\Delta x^2}2u_i^{n+1} - \frac{D\Delta t}{\Delta x^2}u_{i-1}^{n+1} &= u_i^n \\ u_i^{n+1}(1 + 2\frac{D\Delta t}{\Delta x^2}) - \frac{D\Delta t}{\Delta x^2}u_{i+1}^{n+1} - \frac{D\Delta t}{\Delta x^2}u_{i-1}^{n+1} &= u_i^n \end{aligned} \tag{5}$$

We can the simplify the equation like so:

$$\begin{aligned} C &= 1 + 2\frac{D\Delta t}{\Delta x^2} \\ L &= -\frac{D\Delta t}{\Delta x^2} \\ R &= -\frac{D\Delta t}{\Delta x^2} \\ Cu_i^{n+1} + Ru_{i+1}^{n+1} + Lu_{i-1}^{n+1} &= u_i^n \end{aligned} \tag{6}$$

We can now begin to see the beginning of a linear system in the form of:

$$\begin{aligned} A\vec{u}^{n+1} &= \vec{b} \\ \texttt{In this case:}\quad A\vec{u}^{n+1} &= \vec{u}^n \end{aligned} \tag{7}$$

Expanded the linear system looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ L & C & R & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & L & C & R & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & L & C & R & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & L & C & R & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & L & C & R & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & L & C & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ u_5^{n+1} \\ u_6^{n+1} \\ u_{7->max-1}^{n+1} \\ u_{max}^{n+1} \end{bmatrix} = \begin{bmatrix} BC_0^n \\ u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \\ u_5^n \\ u_6^n \\ u_{7->max-1}^n \\ BC_{max}^n \end{bmatrix}$$

where BC stands for boundary conditions. We can then take the inverse of matrix A and multiply it to the front of both sides to obtain a solution for the next time iteration:

$$\mathbf{u}^{n+1} = A^{-1}\mathbf{u}^n \tag{8}$$

My code repeats the steps of assigning the boundary conditions and multiplying by the inverse of A until $n+1$ corresponds to the end time of the simulation. After every multiplication of A^{-1} the numerical and exact solutions for the given time are plotted.

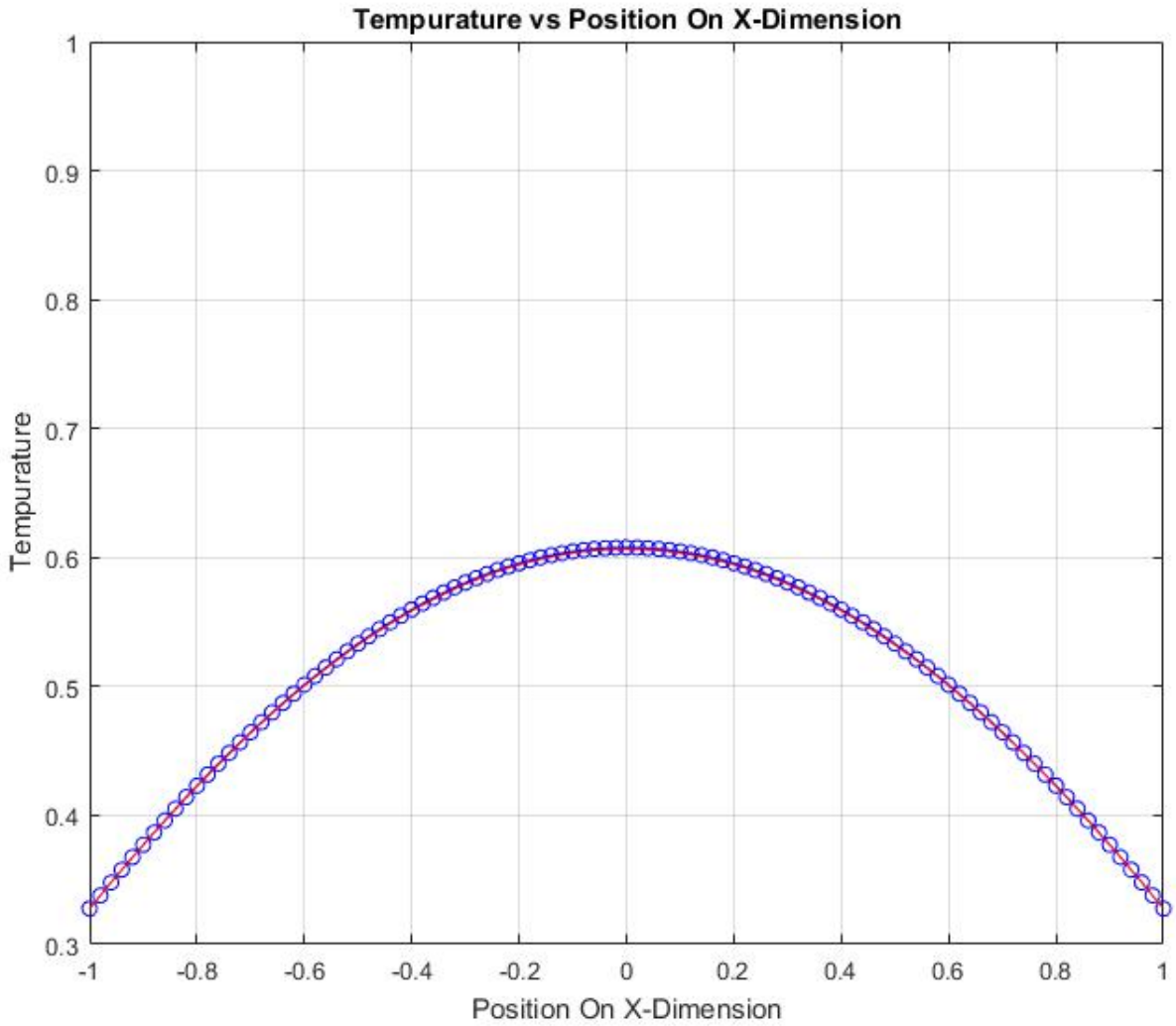
3 Results

In `MatLab`, my simulation code is invoked at the prompt by calling the function

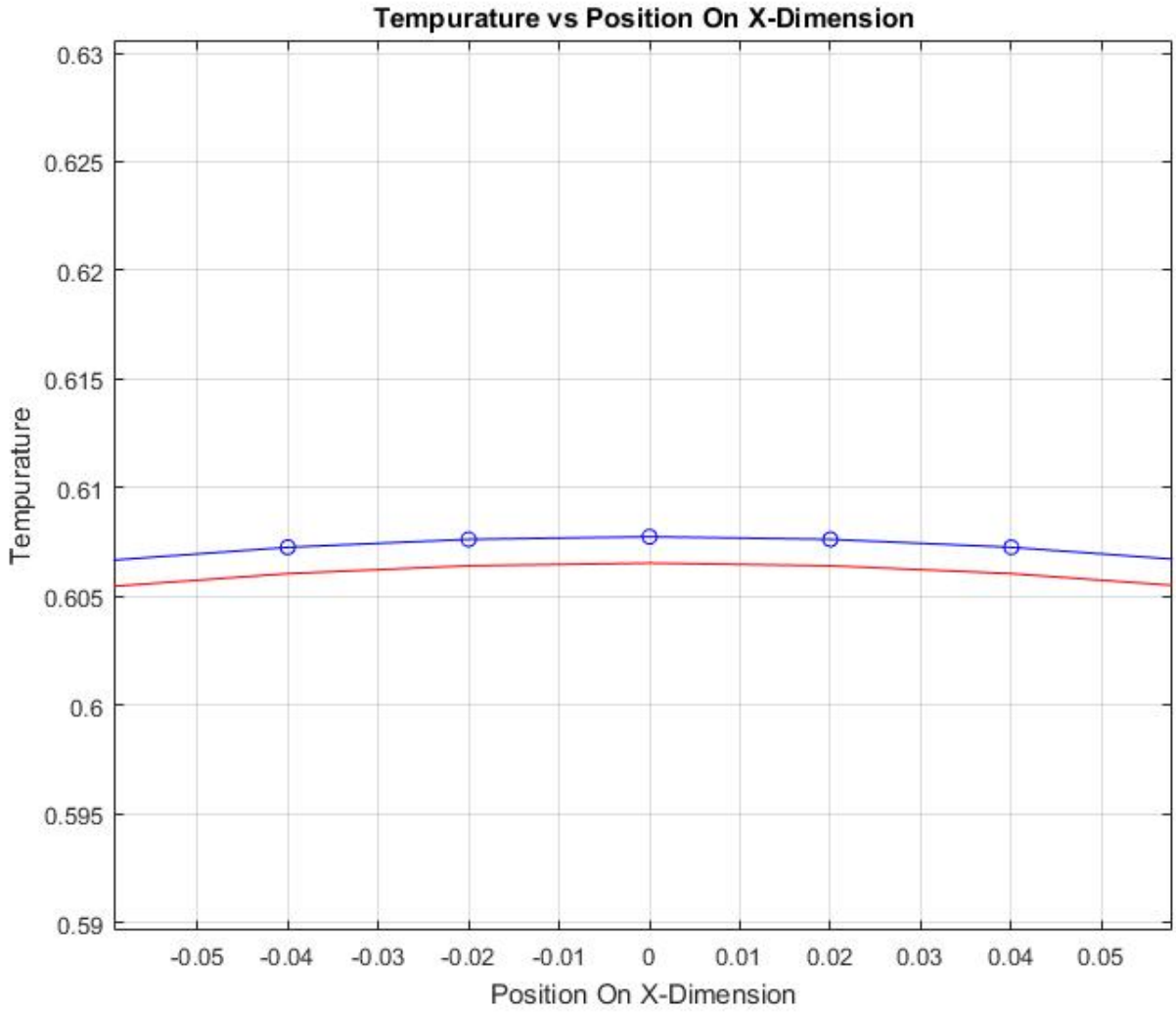
`HeatDiffusionPlot(timeStep, spacialResolution, diffusionConstant, domain, simLength)`

where *timeStep* is the time interval between each calculation, *spacialResolution* is the amount of space between each grid node, *diffusionConstant* is the diffusion constant, *domain* is a 1x2 vector containing the min boundary value and max boundary value, and *simLength* is the real world time in seconds the simulation covers. For this lab the time step is equal to the spacial resolution, the spacial resolution is something that 2 is divisible by, the diffusion constant is .5, the domain is [-1, 1], and the simulation length is one second.

`HeatDiffusionPlot( .02, .02, .5, [-1,1], 1 )` will run a simulation with 100 grid nodes along the x-dimension and 50 time steps. The result is shown here in blue, compared to the exact solution in red:



It is very hard to differentiate between the two because they are very close. the average percent error across all the nodes is only 0.15 percent. here is a more zoomed in image that shows the center points, where there is the highest percent of error:



In all the simulations that I did, no matter the time step or the spacial resolution, the points towards the center of the domain always had the biggest error percentage.

To analyze the accuracy of my implementation, I used a script called `Analysis` that produces an Excel spreadsheet with multiple sets of data and comparisons. The script uses a modified version of `HeatDiffusionPlot`, `HeatDiffusion(timeStep, spacialResolution, diffusionConstant, domain, simLength)` that does the same exact thing but simply does not create a plot and only gives the return value.

The return value for both functions is a matrix with three columns that contains numbered grid nodes, absolute error, and percent error in that order.

4 Conclusion and Error Analysis

We can find the order of accuracy of the implicit central difference scheme and subsequently my implementation by looking at the percent errors of the grid nodes using different time intervals and spacial resolutions. For this lab, when the number of grid nodes goes up by a factor of 2, so does the amount of iterations. A numerical solution is said to be nth order accurate if the error, E, is proportional to the step-size h to the nth power:

$$E(h) = Ch^n \tag{9}$$

In this case, h is the time step or spacial resolution. The corresponding big-O notation:

$$O(h^n) \tag{10}$$

Lets take a look at the error obtained when the number of grid nodes is increased by a factor of 2, and the time step remains the same. I will display all the error data in a provided Excel file because it rather large. For now I will supply a smaller table that shows the percent error when the number of grid nodes is increased from 20 to 40, but the number of iterations is a constant 10:

Table 1: Percent Error of Simulation at Final Time

Grid Node	20 Grid Nodes 10 Iterations	40 Grid Nodes 10 Iterations	Error of 20 Over 40
0	0	0	0
2	0.498166458	0.486292481	1.024417356
4	0.758919579	0.74087877	1.024350554
6	0.901471421	0.880084138	1.024301407
8	0.973701869	0.950628736	1.024271444
10	0.995871684	0.972282754	1.024261389
12	0.973701869	0.950628736	1.024271444
14	0.901471421	0.880084138	1.024301407
16	0.758919579	0.74087877	1.024350554
18	0.498166458	0.486292481	1.024417356
20	0	0	0

We can use these errors to determine the order of accuracy with respect to spacial resolution:

$$\begin{aligned} E(.1) &= 1 \cdot E(.05) \\ Ch_1^n &= 1 \cdot Ch_2^n \\ h_1 &= 2h_2 \\ C2^n h_2^n &= 1Ch_2^n \\ 2^n &= 1 \\ n &= 0 \end{aligned} \tag{11}$$

From this it is visible that the accuracy of the numerical solution has no dependence on the spacial resolution. So although the accuracy of my implementation is still unknown, we have established that the spacial resolution has no effect on the accuracy of my solution. This pattern repeats itself for 40 vs 80 nodes, 80 vs 160 nodes, and 160 vs 320 nodes. It also holds for 20 vs 320 nodes.

Now we will analyze the effect of doubling the number of iterations:

Table 2: Percent Error of Simulation at Final Time

Node	20N 10I	40N 20I	80N 40I	160N 80I	E10I/E20I	E10I/E40I	E10I/E80
0	0	0	0	0	0	0	0
8	0.291825744	0.145693912	0.072792814	0.036383023	2.00300575	4.008991104	8.0209317
16	0.498166458	0.248901259	0.124404908	0.062191126	2.001462187	4.004395545	8.0102498
24	0.648164697	0.324072296	0.16203205	0.081014964	2.000062041	4.000225242	8.0005552
32	0.758919579	0.379683816	0.189894937	0.09496052	1.998819931	3.996523505	7.9919484
40	0.84106311	0.421005511	0.210617079	0.105336752	1.99774846	3.993328146	7.9845172
48	0.901471421	0.451444903	0.225894694	0.112989899	1.996858121	3.99067108	7.9783363
56	0.944686297	0.473252434	0.236847883	0.118478741	1.996157291	3.988578179	7.9734667
64	0.973701869	0.487911578	0.244214935	0.122171561	1.995652314	3.987069302	7.9699552
72	0.99041343	0.49636135	0.248463137	0.124301447	1.9953476	3.986158442	7.9678351
80	0.995871684	0.49912232	0.249851527	0.124997604	1.995245744	3.985853906	7.9671262
88	0.99041343	0.49636135	0.248463137	0.124301447	1.9953476	3.986158442	7.9678351
96	0.973701869	0.487911578	0.244214935	0.122171561	1.995652314	3.987069302	7.9699552
104	0.944686297	0.473252434	0.236847883	0.118478741	1.996157291	3.988578179	7.9734667
112	0.901471421	0.451444903	0.225894694	0.112989899	1.996858121	3.99067108	7.9783363
120	0.84106311	0.421005511	0.210617079	0.105336752	1.99774846	3.993328146	7.9845172
128	0.758919579	0.379683816	0.189894937	0.09496052	1.998819931	3.996523505	7.9919484
136	0.648164697	0.324072296	0.16203205	0.081014964	2.000062041	4.000225242	8.0005552
144	0.498166458	0.248901259	0.124404908	0.062191126	2.001462187	4.004395545	8.0102498
152	0.291825744	0.145693912	0.072792814	0.036383023	2.00300575	4.008991104	8.0209317
160	0	0	0	0	0	0	0

In this table N stands for the number of nodes and I for the number of Iterations and E stands for error. The time steps equal to 1/I. So time steps of .1, .05, .025,and .0125 are represented here. Column one shows the node in context with relation to a spacial resolution of .0125 (160 nodes). Columns 2 - 5 show the percent error of the final solution in comparison to the exact solution. Columns 6 - 8 represent comparisons of errors given by different time steps and spacial resolutions. Because we know spacial resolution has no impact on error, we

can see:

$$\begin{aligned} E(.1) = 2 \cdot E(.05) = 4 \cdot E(.025) = 8 \cdot E(.0125) \\ Ch_1^n = 2 \cdot Ch_2^n \\ h_1 = 2h_2 \\ C2^n h_2^n = 2Ch_2^n \\ 2^n = 2 \\ n = 1 \end{aligned} \tag{12}$$

From this we can see that the time step, unlike the spacial resolution, does effect the error of the numerical solution. The above equations show that my implementation of the implicit central difference method has an order of accuracy of 1, dependent of the time step only. So the error for my implementation is given by:

$$O(\Delta t) \tag{13}$$

A Implementation in MatLab

The MatLab implementation:

HeatDiffusionPlot.m

```
function error = HeatDiffusionPlot( timeStep, spacialResolution, ...
                                                diffusionConstant, domain, simLength)
%%%%%%%%% Creat transformation matrix A for the system and inverse it %%%%%%%%
    left = -diffusionConstant * timeStep / spacialResolution^2;
    center = 1 + 2*(diffusionConstant * timeStep / spacialResolution^2);
    right = -diffusionConstant * timeStep / spacialResolution^2;

    i = 1;
    while i <= ((domain(2)-domain(1)) / spacialResolution) + 1
        transformation(i, i) = center;
        if i > 1 && i < ((domain(2)-domain(1)) / spacialResolution) + 1
            transformation(i, i - 1) = left;
        end
        if i < ((domain(2)-domain(1)) / spacialResolution) + 1 && i > 1
            transformation(i, i + 1) = right;
        end
        if i == 1 || i == ((domain(2)-domain(1)) / spacialResolution) + 1
            transformation(i, i) = 1;
        end
        i = i + 1;
    end

    transformation = inv(transformation);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Create the grid nodes
    Xaxis = [];
    for x = domain(1): spacialResolution: domain(2)
        Xaxis = [Xaxis; x];
    end

    %Set up  initial conditions
    exactTemp = [];
    for x = domain(1): spacialResolution: domain(2)
        exactTemp = [exactTemp; cos(x)];
    end

    finalTempurature = exactTemp;
```

```
%%%%%%%%%%%%ITERATE THROUGH UNTIL SIMLENGTH IS REACHED%%%%%%%%%%%%%%%%%%%%%%%%%
    for t = timeStep: timeStep: simLength
        %Assign boundary values
        finalTempurature(1,1) = exp(-diffusionConstant*t)*cos(domain(1));
        finalTempurature(((domain(2)-domain(1)) / spacialResolution) + 1) ...
                        = exp(-diffusionConstant*t)*cos(domain(2));
        %Multily by inverse matrix to get next step
        finalTempurature = transformation * finalTempurature;

        %Update exact solution
        exactTemp = [];
        for x = domain(1): spacialResolution: domain(2)
            exactTemp = [exactTemp; exp(-diffusionConstant*t) * cos(x)];
        end

        %Plot both the exact solution (red) and numerical solution (blue) at
        %time t
        clf;
        plot(Xaxis,finalTempurature,'Marker','o','Color','blue');hold on;
        plot(Xaxis,exactTemp,'Color','red');
        axis([-1 1 .3 1]);
        axis('on');
        grid on;
        title('Tempurature vs Position On X-Dimension');
        xlabel('Position On X-Dimension')
        ylabel('Tempurature');
        pause(.04);
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Calculate error
    error = [];
    i = 1;
    while i <= ((domain(2)-domain(1)) / spacialResolution) + 1
        error = [error; i - 1];
        i = i + 1;
    end
    error = [error, (finalTempurature - exactTemp)];
    error = [error, ((finalTempurature - exactTemp) ./ exactTemp) * 100];
    %RETURNS a matrix with three columns, numbered grid nodes, absolute error,
    %and percent error in that order.
end
```

HeatDiffusion.m

```
function error = HeatDiffusion( timeStep, spacialResolution, ...
                                          diffusionConstant, domain, simLength)

    left = -diffusionConstant * timeStep / spacialResolution^2;
    center = 1 + 2*(diffusionConstant * timeStep / spacialResolution^2);
    right = -diffusionConstant * timeStep / spacialResolution^2;

    i = 1;
    while i <= ((domain(2)-domain(1)) / spacialResolution) + 1
        transformation(i, i) = center;
        if i > 1 && i < ((domain(2)-domain(1)) / spacialResolution) + 1
            transformation(i, i - 1) = left;
        end
        if i < ((domain(2)-domain(1)) / spacialResolution) + 1 && i > 1
            transformation(i, i + 1) = right;
        end
        if i == 1 || i == ((domain(2)-domain(1)) / spacialResolution) + 1
            transformation(i, i) = 1;
        end
        i = i + 1;
    end

    transformation = inv(transformation);

    Xaxis = [];
    for x = domain(1): spacialResolution: domain(2)
        Xaxis = [Xaxis; x];
    end

    exactTemp = [];
    for x = domain(1): spacialResolution: domain(2)
        exactTemp = [exactTemp; cos(x)];
    end

    finalTempurature = exactTemp;

    for t = timeStep: timeStep: simLength
        finalTempurature(1,1) = exp(-diffusionConstant*t)*cos(domain(1));
        finalTempurature(((domain(2)-domain(1)) / spacialResolution) + 1) ...
                        = exp(-diffusionConstant*t)*cos(domain(2));
        finalTempurature = transformation * finalTempurature;
```

```
        exactTemp = [];
        for x = domain(1): spacialResolution: domain(2)
            exactTemp = [exactTemp; exp(-diffusionConstant*t) * cos(x)];
        end

    end

    error = [];
    i = 1;
    while i <= ((domain(2)-domain(1)) / spacialResolution) + 1
        error = [error; i - 1];
        i = i + 1;
    end
    error = [error, (finalTempurature - exactTemp)];
    error = [error, ((finalTempurature - exactTemp) ./ exactTemp) * 100];

end
```

Analysis.m

```
error00 = HeatDiffusion( .02, .02, .5, [-1,1], 1 );

error10 = HeatDiffusion( .1, .1, .5, [-1,1], 1 );
error11 = HeatDiffusion( .05, .1, .5, [-1,1], 1 );
error12 = HeatDiffusion( .1, .05, .5, [-1,1], 1 );

error20 = HeatDiffusion( .05, .05, .5, [-1,1], 1 );
error21 = HeatDiffusion( .025, .05, .5, [-1,1], 1 );
error22 = HeatDiffusion( .05, .025, .5, [-1,1], 1 );

error30 = HeatDiffusion( .025, .025, .5, [-1,1], 1 );
error31 = HeatDiffusion( .0125, .025, .5, [-1,1], 1 );
error32 = HeatDiffusion( .025, .0125, .5, [-1,1], 1 );

error40 = HeatDiffusion( .0125, .0125, .5, [-1,1], 1 );
error41 = HeatDiffusion( .00625, .0125, .5, [-1,1], 1 );
error42 = HeatDiffusion( .0125, .00625, .5, [-1,1], 1 );

T1 = table;
T1.Grid_Node = error00(1:101,1);
T1.Absolute_Error = error00(1:101,2);
T1.Percent_Error = error00(1:101,3);
writetable(T1, 'Grid Point Error Analysis.xlsx','Sheet',1,'Range','A2');

T2 = table;
T2.Grid_Node = error10(1:21,1);
T2.Absolute_Error = error10(1:21,2);
T2.Percent_Error = error10(1:21,3);
writetable(T2, 'Grid Point Error Analysis.xlsx','Sheet',1,'Range','D2');

T3 = table;
T3.Grid_Node = error20(1:41,1);
T3.Absolute_Error = error20(1:41,2);
T3.Percent_Error = error20(1:41,3);
writetable(T3, 'Grid Point Error Analysis.xlsx','Sheet',1,'Range','G2');

T4 = table;
T4.Grid_Node = error30(1:81,1);
T4.Absolute_Error = error30(1:81,2);
T4.Percent_Error = error30(1:81,3);
writetable(T4, 'Grid Point Error Analysis.xlsx','Sheet',1,'Range','J2');
```

```
T5 = table;
T5.Grid_Node = error40(1:161,1);
T5.Absolute_Error = error40(1:161,2);
T5.Percent_Error = error40(1:161,3);
writetable(T5, 'Grid Point Error Analysis.xlsx','Sheet',1,'Range','M2');

T6 = table;
T6.Grid_Node = error40(1:8:161,1);
T6.Percent_Error_20_Grid_Nodes = error10(1:21,3);
T6.Percent_Error_40_Grid_Nodes = error20(1:2:41,3);
T6.Percent_Error_80_Grid_Nodes = error30(1:4:81,3);
T6.Percent_Error_160_Grid_Nodes = error40(1:8:161,3);
T6.Error_Of_20_Over_Error_Of_40 = error10(1:21,3) ./ error20(1:2:41,3);
T6.Error_Of_20_Over_Error_Of_80 = error10(1:21,3) ./ error30(1:4:81,3);
T6.Error_Of_20_Over_Error_Of_160 = error10(1:21,3) ./ error40(1:8:161,3);
writetable(T6, 'Grid Point Error Analysis.xlsx','Sheet',2,'Range','A2');

T7 = table;
T7.Grid_Node = error10(1:21,1);
T7.Percent_Error_20_Grid_Nodes_10_Iterations = error10(1:21,3);
T7.Percent_Error_20_Grid_Nodes_20_Iterations = error11(1:21,3);
T7.Percent_Error_40_Grid_Nodes_10_Iterations = error12(1:2:41,3);
T7.Error_Of_10_Iterations_Over_Error_Of_20_Iterations = error10(1:21,3) ./ error11(1:21,
T7.Error_Of_20_Grid_Nodes_Over_Error_Of_40_Grid_Nodes = error10(1:21,3) ./ error12(1:2:4
writetable(T7, 'Grid Point Error Analysis.xlsx','Sheet',3,'Range','A2');

T8 = table;
T8.Grid_Node = error20(1:41,1);
T8.Percent_Error_40_Grid_Nodes_20_Iterations = error20(1:41,3);
T8.Percent_Error_40_Grid_Nodes_40_Iterations = error21(1:41,3);
T8.Percent_Error_80_Grid_Nodes_20_Iterations = error22(1:2:81,3);
T8.Error_Of_20_Iterations_Over_Error_Of_40_Iterations = error20(1:41,3) ./ error21(1:41,
T8.Error_Of_40_Grid_Nodes_Over_Error_Of_80_Grid_Nodes = error20(1:41,3) ./ error22(1:2:8
writetable(T8, 'Grid Point Error Analysis.xlsx','Sheet',4,'Range','A2');

T9 = table;
T9.Grid_Node = error30(1:81,1);
T9.Percent_Error_80_Grid_Nodes_40_Iterations = error30(1:81,3);
T9.Percent_Error_80_Grid_Nodes_80_Iterations = error31(1:81,3);
T9.Percent_Error_160_Grid_Nodes_40_Iterations = error32(1:2:161,3);
T9.Error_Of_40_Iterations_Over_Error_Of_80_Iterations = error30(1:81,3) ./ error31(1:81,
```

```
T9.Error_Of_80_Grid_Nodes_Over_Error_Of_160_Grid_Nodes = error30(1:81,3) ./ error32(1:2:
writetable(T9, 'Grid Point Error Analysis.xlsx','Sheet',5,'Range','A2');

T10 = table;
T10.Grid_Node = error40(1:161,1);
T10.Percent_Error_160_Grid_Nodes_80_Iterations = error40(1:161,3);
T10.Percent_Error_160_Grid_Nodes_160_Iterations = error41(1:161,3);
T10.Percent_Error_320_Grid_Nodes_80_Iterations = error42(1:2:321,3);
T10.Error_Of_80_Iterations_Over_Error_Of_160_Iterations = error40(1:161,3) ./ error41(1:
T10.Error_Of_160_Grid_Nodes_Over_Error_Of_320_Grid_Nodes = error40(1:161,3) ./ error42(1
writetable(T10, 'Grid Point Error Analysis.xlsx','Sheet',6,'Range','A2');
```