Discretization of the Heat Equation

Phillip Batov, Michael Klamkin

September 4, 2017

Abstract

In this report we compared the efficiency of three discrete methods of discretization on a one-dimensional heat equation. The three methods utilized throughout were MATLABs built-in function labeled ode45, the Forward Euler Method (FEM), and the Backward Euler Method (BEM). Each operation has distinct features that make it a formidable mechanism for solving partial differential equations, including the heat equation discretized in this study. Each method had its own disparate code written in MATLAB, utilizing elements such as for-end loops to generate matrices, process equations, and eventually graph a visual representation of the discretized heat equation. These graphs employed a temperature-color spectrum that coordinated each position and time point (x,t) with a specific temperature based on the behavior of the heat equation. The graphs for the ode45, FEM, and BEM were all similar in nature, demonstrating how they are all competent methods for discretizing PDEs in different ways.

Contents

1 Introduction

The fundamental equation that is attempted to be discretized is the one-dimensional model of the heat equation. The heat equation is a parabolic partial differential equation, utilized to describe how heat is distributed (or variation in temperature) in a given region over time [1]. The equation operates in the following way: a function exists the that describes the temperature at a given location (x, y, z). As heat spreads throughout space, the function will change over time. The heat equation is used to determine this change over time. The rate of change of this function is proportional to the "curvature", meaning sharper corners round off faster. Due to the nature of the partial differential equation, the peaks of the function flatten and the valleys are filled in, over time. An ideal application of the heat equation (to see how it reacts) is observing the heat spreading through a rod, due to the one space variable. In this investigation, using MATLAB as our tool, we attempted to model the heat spreading within a theoretical rod, and illustrate it on a plot. The equation is specified below:

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2}, u(x,0) = f(x) \tag{1}$$

The first method investigated is MATLAB's ode45 function. In order to implement the heat equation, a partial differential equation (PDE), in MATLABs ode45 function, it must be discretized. This is done via the Backwards Euler Method:

$$\gamma u_{j1}^{n+1} + (1+2\gamma)u_j^{n+1}\gamma u_{j+1}^{n+1} = u_j^n, \gamma = \frac{\Delta t}{\Delta x^2} \tag{2}$$

To generate (2), we adopted the finite difference method twice (below), to arrive at the second derivative for the heat equation.

$$\frac{u_j^{n+1}u_j^n}{\Delta t} = \frac{u_{j1}^{n+1}2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} \tag{3}$$

An crucial observation regarding the Forward Euler Method is that it is an explicit method, i.e., yn+1 is given explicitly in terms of known quantities, such as the following [2]

$$y_n, f(y_n, t_n) \tag{4}$$

The Backwards Euler Method is similar to the forward Euler method, but differs in that it is an implicit method- has order one in time. The backward Euler method is an implicit method: the new approximation (5) appears on both sides of the equation, and thus the method needs to solve an algebraic equation for the unknown (5).

$$y_k + 1 \tag{5}$$

2 Problem Formulation

In Section 1 we introduced the three methods for discretizing PDE's: ode45, forward euler, and backward euler. Each method employs unique functions and equations that are processed by MATLAB. However, all three of them illustrate how 1D heat can theoretically behave on a metal rod. Ode45 is a MATLAB integrated tool that solves nonstiff differential equations medium order method. Defined as

$$[t, y] = ode45(odefun, tspan, y_0), tspan = [t_0, t_f], \tag{6}$$

ode45 integrates the system of differential equations from the initial to the final time, with initial conditions y0 (Mathworks, nd).

The second method used is the forward forward euler, or explicit, method. While explicit methods are very easy to implement, drawbacks arise from the limitations on the time step size to ensure numerical stability. The final method used to discretize the system is the backward euler method. It is similar to the forward Euler method, but differs in that it is an implicit method- has order one in time. The backward Euler method is an implicit method, meaning the new approximation appears on both sides of the equation, and thus the method needs to solve an algebraic equation for the unknown.

Out of the three methods specified earlier, we sought out to compare efficiency of each one based on a visual graph. In doing so, we hope to determine which method can be used to not only solve varying partial differential equations, but also to illustrate them graphically.

3 Solution Methods

The code starts with defining the constants that will be utilized throughout the run of the program. One such constant is n = 100, which represents the discretized number of points on the rod. Another one is c = 0.1, the thermal diffusion constant (TDC). The TDC is based on discrete variables pertaining to the rod, including material among others. We defined further constants as follows

$$\omega = \frac{c}{\Delta x^2} \tag{7}$$

In (7) we define the constant omega. It is a variation of the heat equation, adapted for our specific purposes, that implements the thermal diffusion constant. Then we defined:

$$\Delta x = \frac{1}{n} \tag{8}$$

where the change in position is equal to the inverse of the number of points on the rod. This represents the distance between discretized points.

The ode45 discretization process began with generating a tridiagonal matrix defined as A , using a for loop where

$$A(i, i) = 2 * \omega, A(i, i + 1) = \omega, A(i + 1, i) = \omega \tag{9}$$

The b matrix is a vector with 99 rows in it, defined with the parameters of b(1) being omega multiplied by the initial condition, and b(end) being omega multiplied by the end condition. The line $u_0 = \frac{b}{w}$ overid the previous definition of the u_0 value, by supplementing it with the b matrix divided by omega to label the initial conditions.

The next component was achieved through the use of the ode45 function. The ode45 function is a built-in operation within MATLAB, that allows the user to solve non-stiff differential equations, medium order method. Using ode45, the time and output matrices relationship was able to be defined as $[t, u] = ode45(@dudt, [0, 3], u_0)$, in which dudt is a self-written function that employs the Au + b model. Using the global feature as well, an establishment of the A and b variables as global variables, unique to the entire system rather than a particular function, was made possible.

For the forward euler, constants were kept identical as previous, with exceptions in a few areas. The differential constant was adjusted by a multiplying factor of 1000, while a time step of .001 was implemented here. A new constant was introduced as well:

$$gm = \frac{\Delta t}{c\Delta x^2} \tag{10}$$

This (10) arranged the heat equation into a more applicable form. FEM also introduced:

$$i = 1 : length(x), U(1, i) = 10sin(0.5\pi * x(i)) \tag{11}$$

a for-end loop that allowed us to generate the initial conditions used by the discretized points. A sin curve was used due to its applicability for such one-dimensional situations.

The forward euler method also encompassed the idea of position and time vectors. There is a vector of the steps on the rod, as well as vector of the time steps. Using these vectors, three matrices were generated, row = $length(t)$, col = length(x), and U = $zeros(row, col)$.

final component of the code is a sub-for end loop (used to generate certain results specified previously) that utilizes previously defined constants in it. The loop contains the following equation:

$$U(i + 1, j) = gm(U(i, j + 1) + U(i, j1)) + (1 - 2gm)(U(i, j)) \tag{12}$$

It (12))is a variation of the equation used in the well-established euler method equation:

$$U_{xx}(t_n, x_j) = \frac{u^n_{j-1} - 2u^n_j + u^n_{j+1}}{\Delta x^2} \tag{13}$$

Utilizing this, MATLAB was able to generate a matrix that can potentially be used to view the behavior of 1D heat at discrete points along a metal rod. This data was plotted (in a similar way to ode45) and subsequently analyzed.

For the backward euler method on MATLAB, constants and vectors were kept identical to the FEM, with minor adjustments. In this case, the C value (diffusion constant) was divided by a factor of 500, and the position step was multiplied by 2. Modifying the constants adds an element of variability to the results, allowing a comparison to be drawn from one graph to another.

The next step involved creating a for-end loop (the same one used in FEM) that generated a matrix based on a sin curve

$$U(1,i) = 10sin(0.5\pi * x(i)) \tag{14}$$

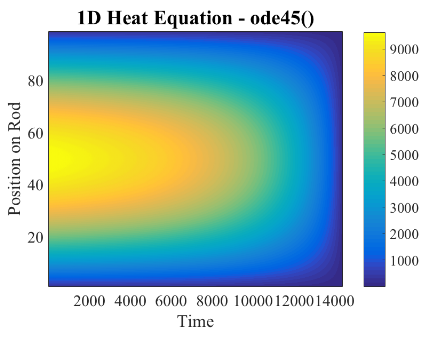
allowing initial conditions to be established.

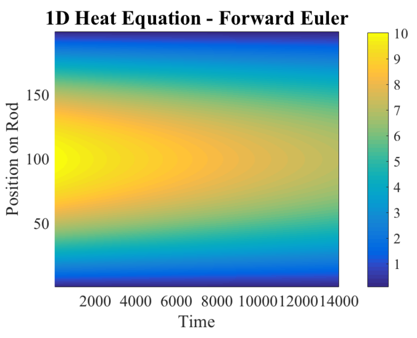
A sparse matrix was also create labeled E: $E = sparse(2 : col - 2, 1 : col - 3, 1, col - 2, col - 2)$. A sparse matrix, i.e. (i, j, v, m, n), specifies the size of S as m-by-n. Two other matrices were created as well- $A_bw = E + E' - 2 * speye(col - 2)$ and $D = speye(col - 2) - (c * row/col^2) * A_bw$. They involved the MATLAB function of speye, which is a sparse identity matrix.

4 Results

To create the figure representing the heat equation, different parameters needed to be set to generate desired results. For example, as denoted by the previous line [t, u] = , the time frame used to see the progression of heat was 3 units, while the position range was 100 distinct points, as discretized earlier.

A colorbar of a temperature spectrum was produced in order to demonstrate the shift of temperature over the given unit of time. At each explicit coordinate of time and position, a temperature value could be assigned due to the conductive nature of the colorbar. To create the shape and coloration of the figure, versatile MATLAB features were used, including pcolor (pseudocolor), which assigns specific colors to the values in the matrix's cells. This way, the plot shows the how the heat equation can be used to show temperature fluctuations on a metal rod.





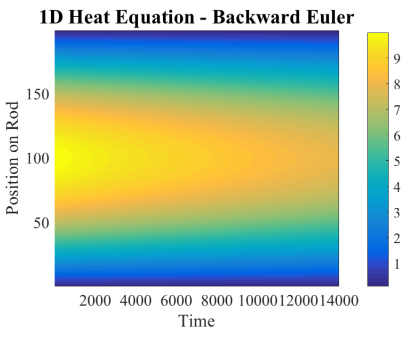
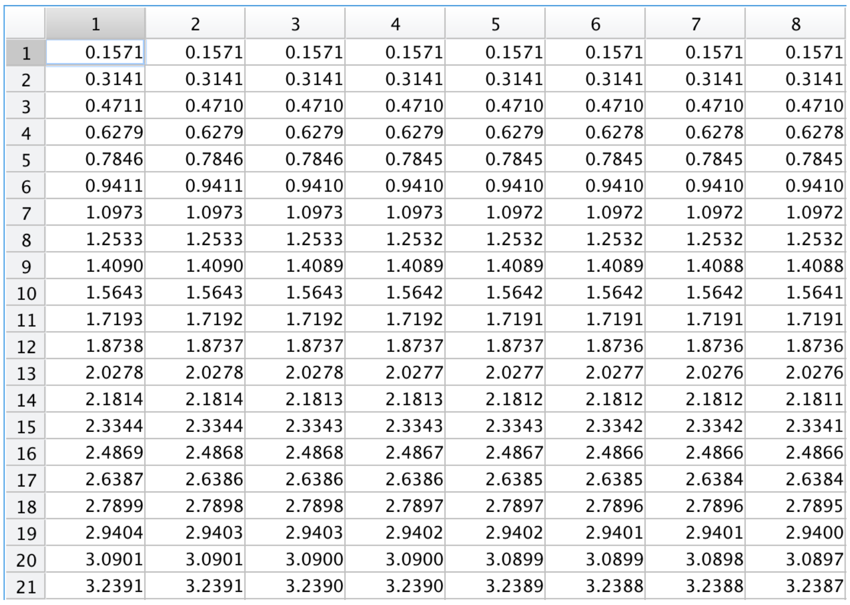


Figure 1: The three separate graphs were generated using three discrete operation of solving PDEs. Their similarity, however, highlights the fact that all three methods are effective in illustrating the discretization of a 1D heat equation model.



	1	2	3	4	5	6	7	8
1	0.1571	0.1571	0.1571	0.1571	0.1571	0.1571	0.1571	0.1571
2	0.3141	0.3141	0.3141	0.3141	0.3141	0.3141	0.3141	0.3141
3	0.4711	0.4710	0.4710	0.4710	0.4710	0.4710	0.4710	0.4710
4	0.6279	0.6279	0.6279	0.6279	0.6279	0.6278	0.6278	0.6278
5	0.7846	0.7846	0.7846	0.7845	0.7845	0.7845	0.7845	0.7845
6	0.9411	0.9411	0.9410	0.9410	0.9410	0.9410	0.9410	0.9410
7	1.0973	1.0973	1.0973	1.0973	1.0972	1.0972	1.0972	1.0972
8	1.2533	1.2533	1.2533	1.2532	1.2532	1.2532	1.2532	1.2532
9	1.4090	1.4090	1.4089	1.4089	1.4089	1.4089	1.4088	1.4088
10	1.5643	1.5643	1.5643	1.5642	1.5642	1.5642	1.5642	1.5641
11	1.7193	1.7192	1.7192	1.7192	1.7191	1.7191	1.7191	1.7191
12	1.8738	1.8737	1.8737	1.8737	1.8737	1.8736	1.8736	1.8736
13	2.0278	2.0278	2.0278	2.0277	2.0277	2.0277	2.0276	2.0276
14	2.1814	2.1814	2.1813	2.1813	2.1812	2.1812	2.1812	2.1811
15	2.3344	2.3344	2.3343	2.3343	2.3343	2.3342	2.3342	2.3341
16	2.4869	2.4868	2.4868	2.4867	2.4867	2.4866	2.4866	2.4866
17	2.6387	2.6386	2.6386	2.6386	2.6385	2.6385	2.6384	2.6384
18	2.7899	2.7898	2.7898	2.7897	2.7897	2.7896	2.7896	2.7895
19	2.9404	2.9403	2.9403	2.9402	2.9402	2.9401	2.9401	2.9400
20	3.0901	3.0901	3.0900	3.0900	3.0899	3.0899	3.0898	3.0897
21	3.2391	3.2391	3.2390	3.2390	3.2389	3.2388	3.2388	3.2387

Figure 2: The table represents position-time coordinates generated by the ode45 function. The function relates each value to a specific color, and plots it on a graph.

5 Discussion and data analysis

Our experimented demonstrated the effectiveness of all three methods when it comes to solving partial differential equations. The graphs for all three methods show an effective color curve that pairs each position and time value with a temperature. Essentially, it converts a three-dimensional problem (time, position, temperature) into a two-dimensional physical representation. This highlights the viability of ode45, FEM, and BEM as possible operation for solving a one-dimensional heat equation problem.

However, there are some unique elements that belong to each individual method based on generated graphs. The Forward Euler and Backward Euler Method are evidently more similar to each other than they are to the residual graph. This can be explained by the fact that both of methods are based on the more fundamental euler method. The ode45 graph has differing code written along with it, which can suggest why the heat has a more u-shaped cooling period, rather that the gradual linear colling shape that the FEM and BEM display. It can also be said that the ode45 has a more abrupt cooling phase in the alloted time, but that can be suggested by the differing thermal diffusion constants.

Some aspects of the study that can be improved or elaborated on include setting up varying PDE's to be solved utilizing these methods. While the heat equation shows their efficiencies, testing on more PDE's would either solidify the results, or disprove them. While changing the PDE is a possible approach to see if variability exists, we can also possibly change an internal aspect of the code, such as the thermal diffusion constant/ number of points and see the subsequent effect it has.

6 Stability Analysis

Although the above methods are inherently similar, their stabilities are not. Below is the stability analysis for both the FEM and BEM.

6.1 Forward Euler

Due to the nature of the heat equation: it becomes flatter and flatter over time, never forming new local maximums, the Maximum Principle of the Heat Equation may be used.

In the heat equation, the Courant-Friedrichs-Lewy condition (CFL condition) may be used to calculate the stability of the system. It is derived as follows:

$$(1-2\gamma) \leq 0, \qquad \gamma \geq \frac{1}{2}, \;\; \Rightarrow \;\; \Delta t \leq \frac{1}{2}\Delta x^2 \tag{15}$$

This is necessary to satisfy the Discrete Maximum Principle which is defined as (16) below, for all n.

$$\max_j |u_j^{n+1}| \leq \max_j |u_j^n| \tag{16}$$

6.2 Backward Euler

$$(1+2\gamma)u_j^{n+1} = u_j^n + \gamma u_{j-1}^{n+1} + \gamma u_{j+1}^{n+1} \tag{17}$$

As seen in the above equation (17), all of the coefficients are positive as is γ. Thus, it may be rewritten with absolute value for the purpose of stability analysis using the triangle inequality theorem. Hence:

$$(1+2\gamma)|u_j^{n+1}| \leq |u_j^n| + \gamma|u_{j-1}^{n+1}| + \gamma|u_{j+1}^{n+1}| \tag{18}$$

Since all of the coefficients γ and $(1+2\gamma)$ are positive, they stay outside of the absolute values. Then, we take the maximum of each term:

$$(1+2\gamma)|u_j^{n+1}| \leq \max_i |u_i^n| + \gamma \max_i |u_i^{n+1}| + \gamma \max_i |u_i^{n+1}| \tag{19}$$

Since the maximum of each term is always greater than or equal to any single term, the inequality is preserved. Through like-terms simplification:

$$(1+2\gamma)|u_j^{n+1}| = \max_i |u_i^n| + 2\gamma \max_i |u_i^{n+1}| \tag{20}$$

Thus, this holds for any j, even when it reaches the max. We are left with a simple inequality showing the discrete maximum principle:

$$\max_j |u_j^{n+1}| \leq \max_i |u_j^n| \tag{21}$$

Through this proof, it is evident that there is no constraint on γ besides $\gamma \geq 0$ which is inherently true since Δx and Δt are always positive. Thus, the BEM is unconditionally stable.

7 Conclusions

In this report we present experimental results of applying three distinct discretization methods to the one-dimensional heat equation, and analyzing the visual graph for each.

The heat equation is defined by [1] as a parabolic partial differential equation, utilized to describe how heat is distributed (or variation in temperature) in a given region over time. Its relation between position and time defines it as a partial differential equation. This means that it is able to be discretized by varying methods, including the ones addressed in this report: ode45, forward euler method, and backward euler method.

We writing code in the program MATLAB to discretized the PDE using the three methods. This process involved defining constants, generating matrices, and graphing the final results on a position-time graph.

The final graphs showed that while it cannot be concluded for other PDEs, the three methods proved effective it providing a similar visual representation of the heat equation in two-dimensional graphs.

References

[1] P Dawkins. Pauls online math notes: Differential equations, 2016.

[2] Michael Zeltkevic. Forward and backward euler methods, Apr 1998.

8 Appendix

MATLAB Code:

```
%% Solving the Heat Equation in 1 Dimension via Finite ...
    Difference Method
% Dependent on the function dudt in dudt.m
% By Michael Klamkin 2017
clear all; close all;
global A b
ode45tic = tic; % timer for entire script
```

```
%% Constants
n = 100; % number of points on the rod
c = 0.05; % thermal diffusion constant
dx = 1/n; % Δ x
omega = c/dx.^2; % just to not type out k/dx.^2 5 times
tstop = 3; %length of simulation
%% Initialization

A = zeros(n-1);

for i = 1:n-2
    A(i,i) = -2*omega; %central diag
    A(i, i+1) = omega; %lower diag
    A(i+1, i) = omega; %upper diag
end

A(n-1,n-1) = -2 * omega;

for i = 1:n-1 %init cond
    b(i,1) = omega*(10*sin(pi*i*0.01));
end

u_0 = b / omega; %initial conditions
%% ode45 and plotting via imagesc

[t, u] = ode45( @dudt, [0,tstop], u_0); % see help/doc for more info

figure(1);
     subplot(2, 2, 1);
        fig1 = pcolor(fliplr(u')); shading interp; %colormap jet ...
            %set(gca,'Ydir','reverse'); %pcolor allows for ...
            interp shading style
        cbar = colorbar;
        cbarpos = get(get(cbar,'YLabel'),'Position');
        set(get(cbar,'YLabel'),'Position',[cbarpos(1) + 3, ...
            cbarpos(2) + 40, cbarpos(3)],'Rotation',-90, ...
            'FontSize', 15);
        title('1D Heat Equation - ode45()'); % figure title
        set(gca, 'FontName', 'Times New Roman'); set(gca, ...
            'TitleFontSizeMultiplier', 1.25); set(gca, ...
            'FontSize', 15); % figure styling
        ylabel('Position on Rod'); % y axis label
        xlabel('Time'); % x axis label
display(toc(ode45tic), 'ode45()'); % final timer
%% Forward Euler For-Loop
% Heat Equation Solution using Forward Euler FEM
% By Michael Klamkin 2017
fwdtic = tic; % timer for entire script

%% Constants
c = c * 1000; % diff. const. adjusted
dt = 0.0001; % time step
gm = dt/((c)*(dx^2)); % a const for use in fwd euler

%% Vectors
x = 0:dx:2; % vect of steps of pos on rod
t = 0:dt:14; % vect of steps of time

%% Initial Conditions
row = length(t); % num time samples
col = length(x); % num pos samples
U = zeros(row,col); % build U
```

```
for i = 1:length(x) %init cond
    U(1,i) = 10*sin(0.5*pi*x(i));
end

%% for Loop + Plot
for i = 1:row-1
    for j = 2:col-1
        U(i+1,j) = gm*(U(i,j+1) + U(i,j-1)) + (1-2*gm)*(U(i,j));
    end
end

U(:,1) = []; U(:, end) = []; %shave 0-set rows/column

figure(1);
     subplot(2, 2, 2);
     fig2 = pcolor(U'); shading interp; %colormap jet ...
         %set(gca,'Ydir','reverse'); %pcolor allows for interp ...
         shading style
        cbar2 = colorbar;
        cbarpos2 = get(get(cbar2,'YLabel'),'Position');
        set(get(cbar2,'YLabel'),'Position',[cbarpos2(1) + 3, ...
            cbarpos2(2) + 40, cbarpos2(3)],'Rotation',-90, ...
            'FontSize', 15);
        title('1D Heat Equation - Forward Euler'); % figure title
        set(gca, 'FontName', 'Times New Roman'); set(gca, ...
            'TitleFontSizeMultiplier', 1.25); set(gca, ...
            'FontSize', 15); % figure styling
        ylabel('Position on Rod'); % y axis label
        xlabel('Time'); % x axis label
display(toc(fwdtic), 'Forward Euler');

%% Backward Euler For-Loop
% Heat Equation Solution using Forward Euler FEM
% By Michael Klamkin 2017
clear U;
clear u;
backtic = tic;
%% Constants
dt = 0.0001; % time step
c = c*dt;
%% Vectors
x = 0:dx:2; % vect of steps of pos on rod
t = 0:dt:14; % vect of steps of time

%% Initial Conditions
row = length(t); % num time samples
col = length(x); % num pos samples

bc=zeros(col-2,1);
bc(1)=0;%c*row*1/dx^2;
bc(col-2)=0;c*row*1/dx^2;  %Dirichlet B.Cs

for i = 1:length(x) %init cond
    u(1, i) = 10*sin(0.5*pi*x(i));
end

u = u';

E=sparse(2:col-2,1:col-3,1, col -2,col-2);
A_bw=E+E'-2*speye(col-2);        %Dirichlet B.Cs
D=speye(col-2)-(c*row/col^2)*A_bw;

```

```
%% for Loop + Plot
for i=0:row

    U=u;U(1)=[];U(end)=[];
    U=U-bc;
    U=(D\U);
    u = [1;U;1];
    u_plot(:,i+1)=[1;U;1];                    %Dirichlet
end
u_plot(1, :) = []; u_plot(end, :) = [];

figure(1);
     subplot(2, 2, 4);
     fig3 = pcolor(u_plot); shading interp; %colormap jet ...
         %set(gca,'Ydir','reverse'); %pcolor allows for interp ...
         shading style
        cbar3 = colorbar;
        cbarpos3 = get(get(cbar3,'YLabel'),'Position');
        set(get(cbar3,'YLabel'),'Position',[cbarpos3(1) + 3, ...
            cbarpos3(2) + 40, cbarpos3(3)],'Rotation',-90, ...
            'FontSize', 15);
        title('1D Heat Equation - Backward Euler'); % figure title
        set(gca, 'FontName', 'Times New Roman'); set(gca, ...
            'TitleFontSizeMultiplier', 1.25); set(gca, ...
            'FontSize', 15); % figure styling
        ylabel('Position on Rod'); % y axis label
        xlabel('Time'); % x axis label

display(toc(backtic), 'Backward Euler');
```