Simulation of Heat Distribution & Evolution in 2-Dimensions Via Numerical Methods in MATLAB

Shea Schmidt

EGR 115 Final Project Report

Abstract

By decomposing the partial differential equation (PDE) describing n-dimensional heat flow, a numerical model can be constructed to simulate the temperature of a square, uniform plate of varying thermal conductivities in MATLAB. My simulation allows the evolution of said system to be animated and rendered in real-time, with the x and y axis representing spatial dimensions and the z axis magnitudes, along with respective color mapping, representing temperatures at each discretized cell. The calculation is performed on a rigid cell mesh, and the animation is interpolated to smooth the final surface to better approximate the real phenomenon.

The calculation of temperature solution states at each time step is handled using first, hard coded initial conditions, and then second, iterated calculation of subsequent cell values. MATLAB's native surface objects do not internally handle discretized data clouds well, thus interpolation between cells is required to smoothly animate and render surface solutions. Accuracy is thus artificially depicted in the final renders that is not mathematically handled by the discretized heat equation function call.

Simulation of Heat Equation in 2-Dimensions Via Numerical Discretization

Using MATLAB's native array calculation handling and surface rendering, a shockingly detailed system can be simulated using as few or as many scales of resolution as desired, with the cost only being on the available hardware.

Statement of Problem

To simulate and render a useful scientific computer simulation of the temperature distribution, evolution, and steady state equilibrium of an idealized, thin, and uniform square sheet of some material. This simulation should also allow for heating and cooling from any one or multiple of the four sides of the square. This simulation assumes time to be of an arbitrary unit and degrees to be so as well; the boundaries of the sheet represent the contact lines to the surrounding medium which is assumed to be of constant temperature (infinite thermal capacity and volume).

Discrete Mathematical Model of PDE

The heat equation is a PDE of n-dimensions whose solution function $u(x_0, x_1, x_2, \dots, x_n, t)$ describes the temperature of the specified point at time t. The PDE as discovered by Fourier for three spatial dimensions and one time dimension is as follows:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right)$$

In which α is a positive constant called the thermal diffusivity of the medium; for our case, the square sheet (Dawkins 1).

Following G. Nervadof's expression for this PDE, the use of the finite-difference method yields the discretized equality:

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} - \alpha\left(\frac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{\Delta x^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{\Delta y^2}\right) = 0$$

Which Nervadof then simplifies by setting $\Delta x = \Delta y$ and by substituting:

$$\gamma = \alpha \frac{\Delta t}{\Delta x^2}$$

Then the final discrete equation for $u(x_0, x_1, x_2, \dots, x_n, t)$ is as follows:

$$u_{i,j}^{k+1} = \gamma(u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - 4u_{i,j}^{k}) + u_{i,j}^{k}$$

This is the final numerical calculation that Nervadof's program and my program uses. By allowing α and Δx to be 2.0 and 1.0 respectively, as selected by Nervadof, in my program as well, stable thermal diffusivity of the medium is accomplished, and the system remains convergent over the time domain such that $t \in \mathbb{R}^+$.

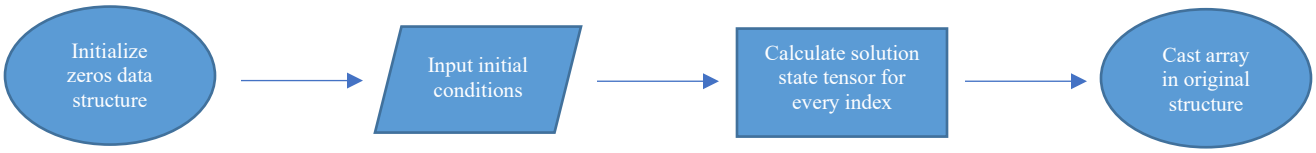
Algorithm and MATLAB Program Development

The solution animated surface is first initialized as a 3-dimensional $n \times n \times t$ tensor array containing zeros for every index in all three dimensions. The rows and columns, *i* and *j* respectively, represent cartesian coordinates *x* and *y* along the square sheet, while the "sheets" along the *z* axis represent time. This means every index among the sheet *k* of the 3-dimensional solution tensor will represent the temperatures at each *x, y* coordinate at time *t = k.* The algorithm takes user defined initial internal temperature conditions and boundary temperature conditions and applies each to respective boundaries of each sheet within the zeros array. Note that internal conditions only are applied to sheet *k = 0*, whereas boundary conditions are held constant on the edges of each sheet for all *k* sheets.

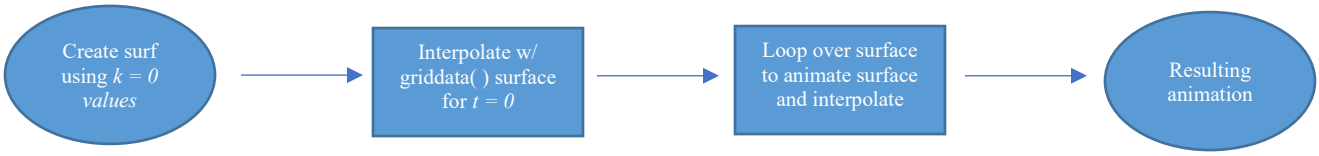
$$for\ k = 0, \quad \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} u_0^{i,j,k} & \cdots & u_0^{i,j+n,k} \\ \vdots & \ddots & \vdots \\ u_0^{i,j+n,k} & \cdots & u_0^{i+n,j+n,k} \end{bmatrix}$$

The discretized solution to the PDE is then expressed as a nested for loop that iterates through every index of the solution tensor to calculate temperature for the *k + 1* sheet index. This

calculation is structured as a function which is called once to perform the operation of taking the loaded initial condition tensor and calculating the solution state for each cell at all defined times $t = k$.



The bulk of the labor in developing this simulation was in then taking this 3-dimensional solution tensor whose indices each contained the numerical temperature for each cell at the given time $t = k$ and then depicting this discretized data set as a smooth, continuous surface. This was accomplished by first creating a surface object "ser" using the data from sheet $k = 0$ and then using the surface interpolation function griddata( ) to interpolate each subsequent surface information subset to the original surface. By looping over the surface and using ser.ZData to update the rendering, animation of the temperature evolution is achieved.



Conclusion

I learned so much about MATLAB rendering and data handling in this project and especially in learning how to read others' algorithm designs and how to implement functions into MATLAB. Nervadof's model was crucial to my final project development, and I could not have completed this project without the mathematical guide he presents. The test case and plots to satisfy this requirement are in the **Figures** section beginning page eleven of this document; these

figures show select frames from first, fixed external heating along one edge with cooling along the remaining three edges at three discrete times (**Figures 1, 2 and 3**) and then secondly, random internal initial temperature distribution with external cooling, also at three discrete times (**Figures 4, 5 and 6).**

This second set of cases suggests my application allows the option for randomizing the internal temperatures of the sheet as well as external constant heating or cooling; this is accurate as watching this system evolution through time is far more entertaining, as the interpolation functions work amazingly well to create resolution in rendering that is not physically or mathematically present in the initial discretization. This document presents only snapshots of the dynamic surface animation that my application provides, and as such this document is incomplete in a sense. Provision of a screen recording, or demonstration of this behavior is available at request if desired.

References

Dawkins, P. (2022). *Section 9.1 : The Heat Equation*. Differential equations - the heat equation. Retrieved December 6, 2022, from https://tutorial.math.lamar.edu/classes/de/theheatequation.aspx

Nervadof, G. (2022, April 14). *Solving 2d Heat Equation Numerically Using Python*. Medium. Retrieved December 6, 2022, from https://levelup.gitconnected.com/solving-2d-heat-equation-numerically-using-python-3334004aa01a

Appendix

Full MATLAB code:

```
% Problem Statement: Simulate and plot temperatures of a uniform square plate
% given intial condiditons of interior temperatures and constant heating on
% one or more edges.

clc
clf
clear
close all

% Initialize Time and Sheet Parameters

tileEdge = 25;  % "Physical" or "Real" tile dimensions (animation is
 interpolated for smoothness)
edgeDim = 1:tileEdge;
maxTimeStep = 200;  % Arbitrary units

% corellation coefficients for discretized heat equation in 2-Dim

alpha = 2.0;
deltaX = 1;

deltaT = (deltaX^2)/(4*alpha);
gamma = (alpha * deltaT)/(deltaX^2);

% initial internal conditions

tileInt = 20.0;                                              % for constant
 interior value, uncomment these two lines
tileSpace = zeros(tileEdge,tileEdge,maxTimeStep) + tileInt;   %   <-------

%tileSpace = randi([20 80],[tileEdge tileEdge maxTimeStep] ); % for random
 interior value, uncomment and use this line instead:

% boundary conditions

tileTop = 100.0;    % All boundary values stay constant throughout simulation,
 representing the temperature of contact medium of infinite volume
tileLeft = 0.0;
tileBot = 0.0;
tileRight = 0.0;

for j = 1:tileEdge
        tileSpace(tileEdge,j,:) = tileTop;
end

for i = 1:tileEdge
        tileSpace(i,1,:) = tileLeft;
end

for j = 2:tileEdge-1
        tileSpace(1,j,:) = tileBot;
end
```

```
for i = 1:tileEdge
        tileSpace(i,tileEdge,:) = tileRight;
end

tileSpace = calculate(tileSpace,gamma,deltaX,tileEdge,maxTimeStep);

% Surface and Interpolation Section

X = 1:tileEdge;     % Initialize the surface
Y = 1:tileEdge;
[xq,yq] = meshgrid(1:0.05:tileEdge); %Interpolation mesh resolution param.
Z = tileSpace(X,Y,1);
z1 = griddata(X,Y,Z,xq,yq); % griddata call here interpolates first sheet of
 tileSpace solution
ser = surf(xq,yq,z1);
ser.EdgeColor = "none";
xlabel("x")
ylabel("y")
zlim([0 100])
xlim([0 tileEdge])
ylim([0 tileEdge])
colormap("jet")     % best thermal colormap I found
clim([0 100])
c = colorbar;
c.Label.String = "Temperature Scale";

for ii = 1:maxTimeStep
    Z = tileSpace(X,Y,ii);
    ser.ZData = griddata(X,Y,Z,xq,yq); % griddata call here interpolates
 discrete values accross entire surface (nth sheet)
    title("Temperature Distribution at t = " + sprintf("%.3f",ii*deltaT) + "
 units.")
    pause(0.001)
end

function tileSpace = calculate(tileSpace,gamma,deltaX,tileEdge,maxTimeStep) %
 define timestep state calculation function

    for k = 1:maxTimeStep-1                  %maxTimeStep
        for i = 2:deltaX:tileEdge-1          %deltaX:tileEdge
            for j = 2:deltaX:tileEdge-1      %deltaX:tileEdge

                tileSpace(i, j, k+1) = gamma * (tileSpace(i+1, j, k) +
 tileSpace(i-1, j, k) + tileSpace(i, j+1, k) + tileSpace(i, j-1, k) -
 4*tileSpace(i, j, k)) ...
                + tileSpace(i, j, k);

            end
        end
    end
end
```

Figures

Figure 1. Distribution State at *t = 2.500* Under 100˚ Heating Along Top Edge

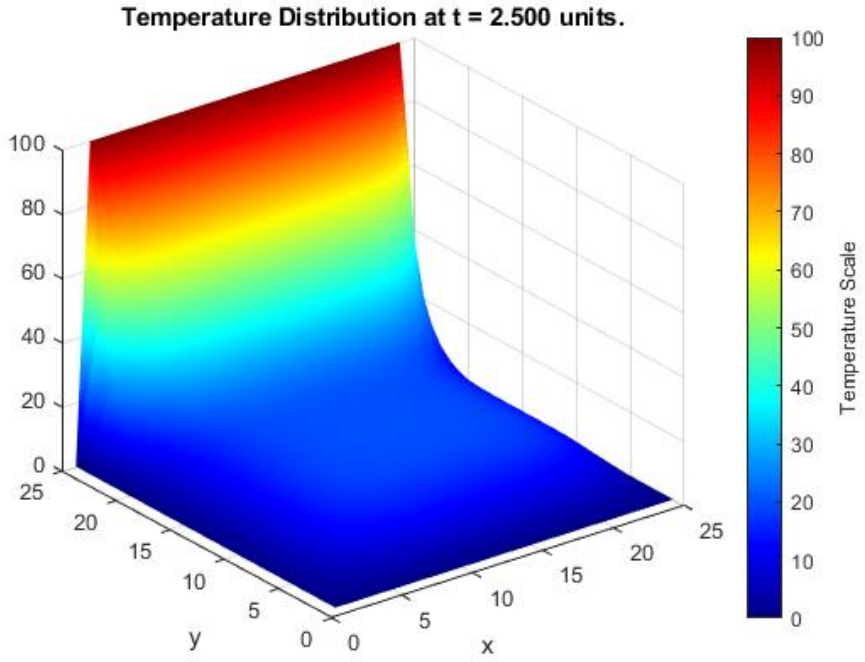


Figure 2. Distribution State at *t = 10.000* Under 100˚ Heating Along Top Edge

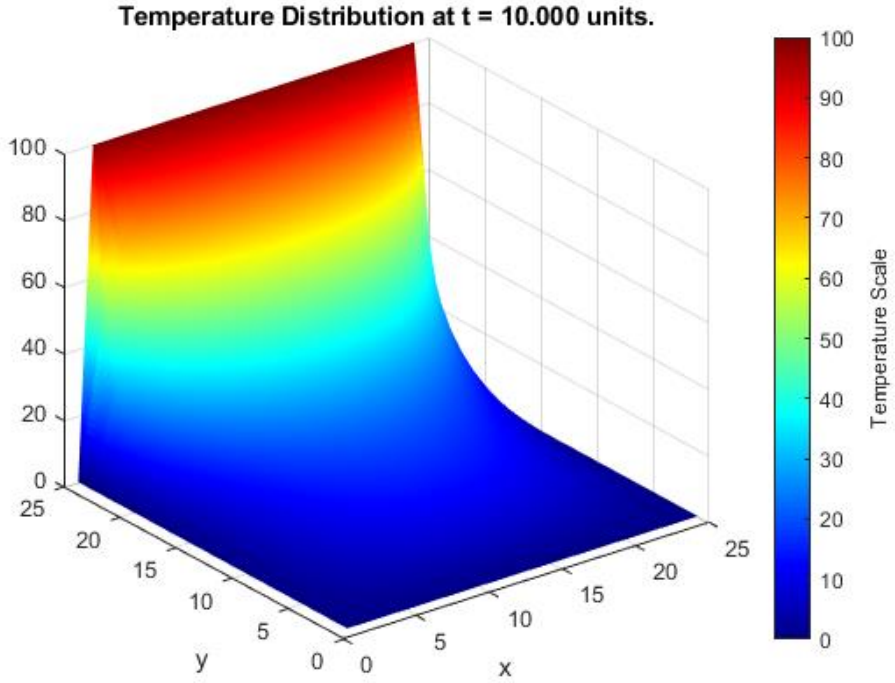


Figure 3. Distribution State at *t = 62.500* Under 100˚ Heating Along Top Edge

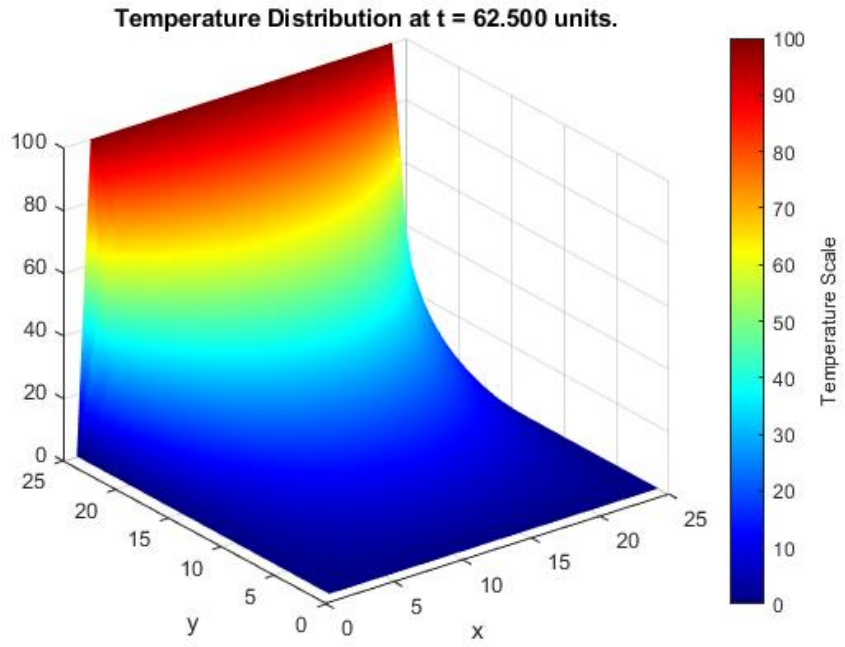


Figure 4. Pseudorandom Normally Distributed Internal Initial Temperature State at *t = 0.250*

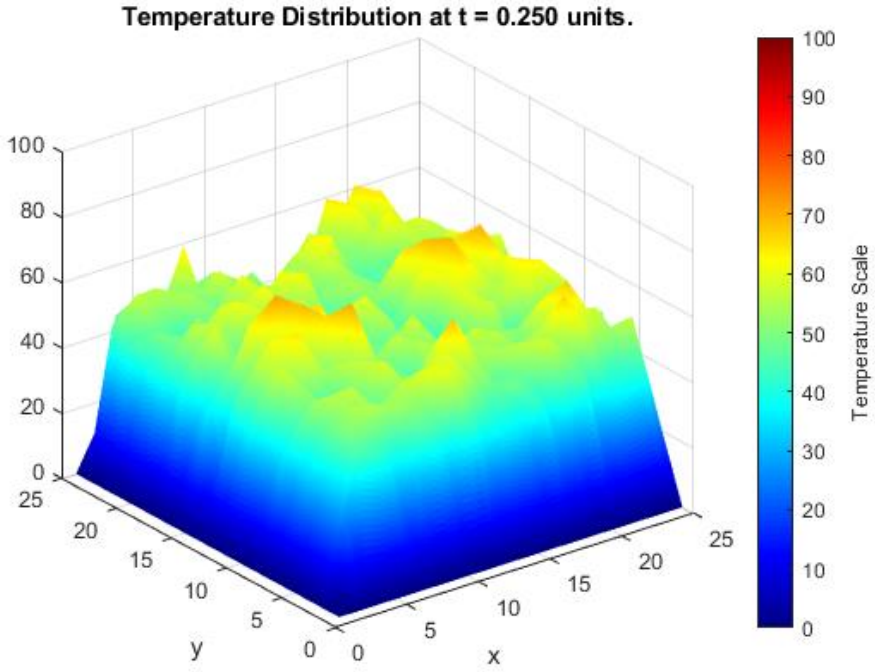


Figure 5. . Pseudorandom Normally Distributed Internal Initial Temperature State at *t = 2.500*

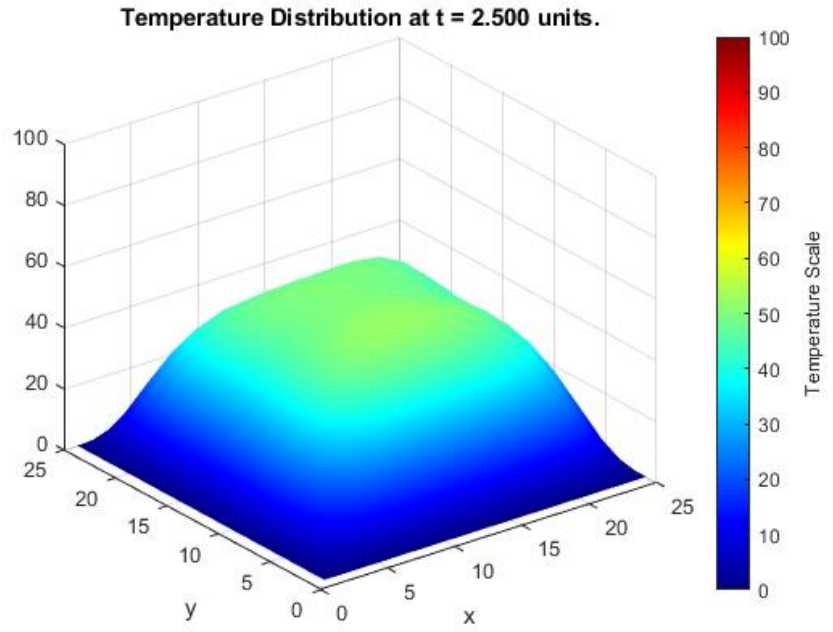


Figure 6. Pseudorandom Normally Distributed Internal Initial Temperature State at *t = 10.000*