UNIVERSITY OF WATERLOO | Department of Mechanical and Mechatronics Engineering

ME 303 - Project 2 - Numerical PDE Solving

Austin W. Milne
Japmeet Brar
Joshua Selvanayagam
Kevin Chu

April 5, 2022

Abstract

Project 2 report for ME 303 in Winter 2022. Using numerical PDE solutions to model heat propagation through homogeneous materials. The source code and report can be found online: https://github.com/Awbmilne/ME303-Numerical-PDE-Solution.

Contents

List of Tables

List of Figures

List of Code Listings

1 Overview

PDEs provide an invaluable tool for modeling systems. While algebra and basic calculus provide insights into physical phenomena and the interaction of forces and energies, PDEs allow the mathematical modeling of the interaction of a single or multidimensional system. This can be used for everything from 1D waves to 3D fluid simulation. One of the most common PDEs used in engineering is the heat equation:

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \tag{1}$$

The heat equation provides insight into how heat flows through a system of most commonly up to 3 dimensions. While not perfect, it provides the groundwork for creating complex simulations of thermal systems. This report focuses on a couple applications of the heat equation and how MATLAB can be used to provide numerical solutions to moderately complex systems.

2 Cooking an Egg with 1D Heat Equation

An interesting health related application of partial differential equations and the 1 dimensional heat equation is modeling heat propagation for cooked foods. For many raw foods, it is important that they are cooked to certain temperatures in order to kill off dangerous bacterial growth. While the food industry is always getting better at preserving foods for longer and longer, it is still important to protect our health with proper cooking. One such common household food is the humble egg. While a cooked egg is lovely for all, a raw egg can carry salmonella and the risk of serious food poisoning. The FDA recommends cooking in-shell eggs to an internal temperature of 68°C for at least 17 seconds [1]. While this may produce a "safe" egg, It would not provide a very delectable breakfast. For the purposes of this report, a boiled egg will be considered to be cooked by having a minimum internal temperature of 80°C for 10 seconds.

2.1 Mathematical Model

In order to mathematically model the heat propagation inside of an egg using the 1D heat equation, some assumptions need to be made. First off, eggs have evolved over tens of millions of years to have very specific shapes, conducive to laying and sitting, providing the strength needed for the babies to survive. Unfortunately, this makes mathematical modeling more difficult. For this report, it is assumed that the eggs are perfectly spherical. The radius of the eggs is determined by the average of the long and short radius. Next, eggs consist of a number of layers. Each of these layers clearly has a different density, thermal conductivity, and specific heat. For the purposes of this report, it is assumed that the entire egg is homogeneous in its thermal and mechanical properties, and that no convection currents occur within the mass.

Through some research, it was found that the 1D heat equation can be represented in spherical coordinates with the below equation [3]:

$$r^2 \frac{\partial T}{\partial t} = \alpha \frac{\partial}{\partial r}\left(r^2 \frac{\partial T}{\partial r}\right) + r^2 \frac{\dot{q}}{\rho\, c_p}, \qquad \text{where} \quad \alpha = \frac{k}{\rho\, c_p} \tag{2}$$

This equation can be more easily utilized in the form:

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial r^2} + \frac{2}{r}\frac{\partial T}{\partial r}\right) + \frac{\dot{q}}{\rho\, c_p}, \qquad \text{where} \quad \alpha = \frac{k}{\rho\, c_p} \tag{3}$$

Equation **??** provides the basic PDE for the system, but the boundary conditions are needed to find a solution. When boiling an egg, the outer surface should always be in contact with 100°C water. This provides an Dirichlet (type 1) boundary condition of $T(r = R, t) = 100$. From there, it is simply a case of providing an initial condition. We can assume that the egg being cooked starts at room temperature (20°C), giving $T(r, t = 0) = 20$. Using these, we can provide a full description for the system:

$$\begin{aligned} \text{PDE:} \quad & T_t = \alpha \left(T_{rr} + \frac{2}{r} T_r\right), \quad r \in (0, R)\,,\; t \in (0, t_{80^\circ\text{C}} + 10) \\ \text{BCs:} \quad & T(r = R, t) = 100 \\ \text{IC:} \quad & T(r, t = 0) = 20 \end{aligned} \tag{4}$$

2.2 Numerical Solution

For the numerical solution, the differential elements in the PDE need to be replaced with relations of the previous elements. In this case T_{rr} can be replaced with $\frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta r^2}$ and T_r can be replaced with $\frac{T_{i+1}^k - T_{i-1}^k}{2\Delta r}$. T_t can then also be replaced with $\frac{T_i^{k+1} - T_i^k}{\Delta t}$. Rearranging the equation then creates an expression for the incremented T_i^{k+1}.

$$\begin{aligned} T_t &= \alpha \left(T_{rr} + \frac{2}{r} T_r\right) \\ \frac{T_i^{k+1} - T_i^k}{\Delta t} &= \alpha \left(\frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta r^2} + \frac{2}{r}\frac{T_{i+1}^k - T_{i-1}^k}{2\Delta r}\right) \\ T_i^{k+1} &= T_i^k + \Delta t\, \alpha \left(\frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta r^2} + \frac{2}{r}\frac{T_{i+1}^k - T_{i-1}^k}{2\Delta r}\right) \end{aligned} \tag{5}$$

This is then implemented in the code as shown in listing 1.

Listing 1: Egg Cooking PDE Incrementing — Q1_egg_cook.m

```
        for i=2:N % Increment over all but the ends
```

```
            r = (i-1) * dr; % Get the radius
            d2T_dr2 = (T(i+1,k)-2*T(i,k)+T(i-1,k))/(dr^2); % Get the instantaneous accel
            dT_dr = (T(i+1,k)-T(i-1,k))/(2*dr); % Get the instaneous slope
            T(i,k+1)=T(i,k) + alpha*dt*(d2T_dr2 + (2/r)*dT_dr); % Increment Temp
        end %
```

The boundary and initial conditions are applied both before the incrementing loop and during the iteration. Listing 2 and 3 show the initial and boundary conditions applied to the system.

Listing 2: Egg Cooking PDE Initial Conditions — Q1_egg_cook.m

```
    % Initial Condition
    T(:,1) = T_start;
    % Boundary conditions
    T(end,1) = T_water; %
```

Listing 3: Egg Cooking PDE Boundary Conditions — Q1_egg_cook.m

```
        T(end,k+1) = T(end,k); % Retain end value
```

The full code can be found in the section 6.2 of the appendix as listing 12, or online.

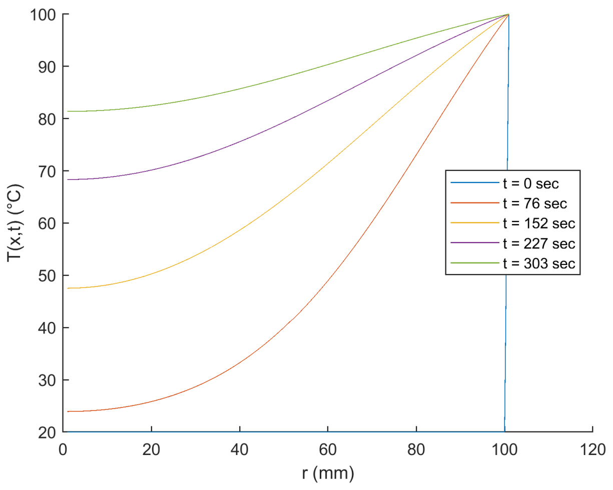
2.3 Results

The numerical solution was run for a selection of egg sizes. While the most obvious choice is a chicken egg, quail and ostrich eggs were also selected to provide variety and insight into the effects of different radii. The chicken egg was also tested from a lower starting temperature. While it is possible to store fresh eggs at room temperature, most eggs found in grocery stores in the North America are washed, as to remove the waxy covering that prevents bacteria entering the egg. As a result, store bought eggs should always remain refrigerated. The chicken egg was also tested at 5°C to see how the cook time differs. Table 1 shows the list of tested parameters and the eventual time to cook.

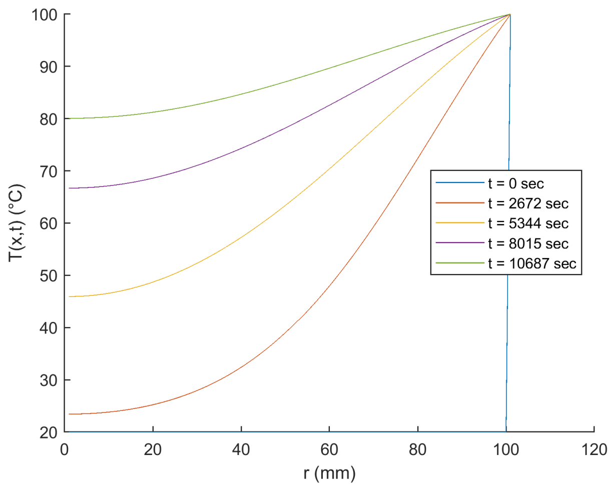
Table 1: Egg Cooking Input Arguments

Egg Type	Radius (mm)	Starting Temp (°C)	Cook Time (sec)	Cook Time
Chicken Egg (fridge)	0.0255	5	992.19	16m32s
Chicken Egg	0.0255	20	916.84	15m17s
Quail Egg	0.0145 [4]	20	303.22	5m3s
Ostrich Egg	0.0875 [2]	20	10687.25	178m7s

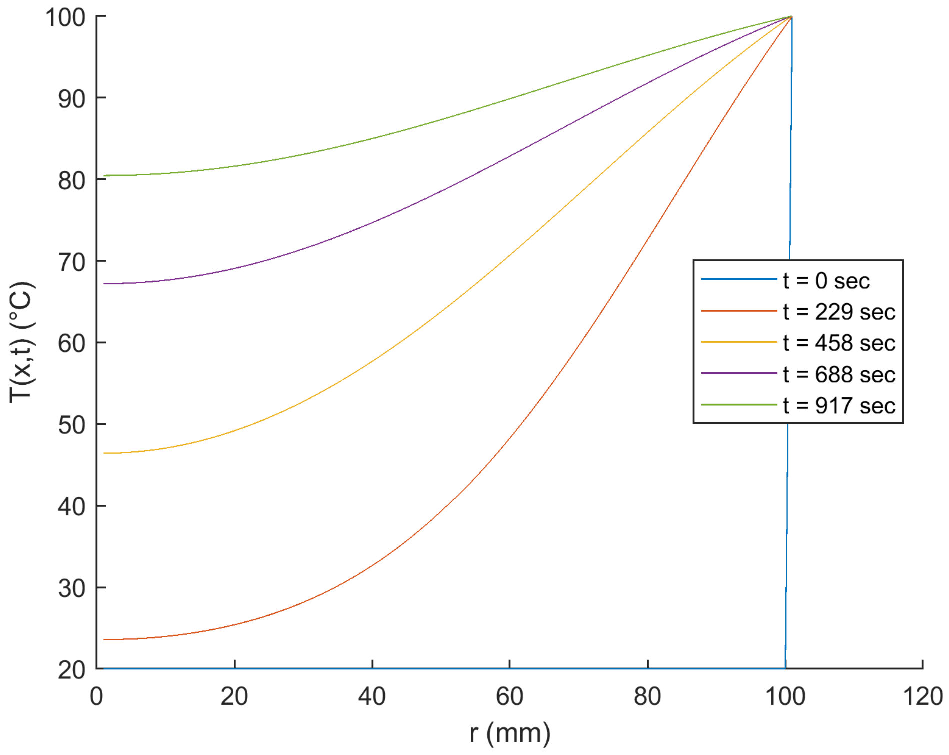
Running the numerical solution for the set of eggs showed very similar behaviour. While the time scale for each egg cooking was very different, the general pattern of heat propagation was very consistent. Figure 1 shows how each of the eggs have almost identical graphs, save for the time and radius scales. Figure 2 provides some more context with a 3D representations of the cooking process.



(a) 2D Quail Egg Cooking Heat

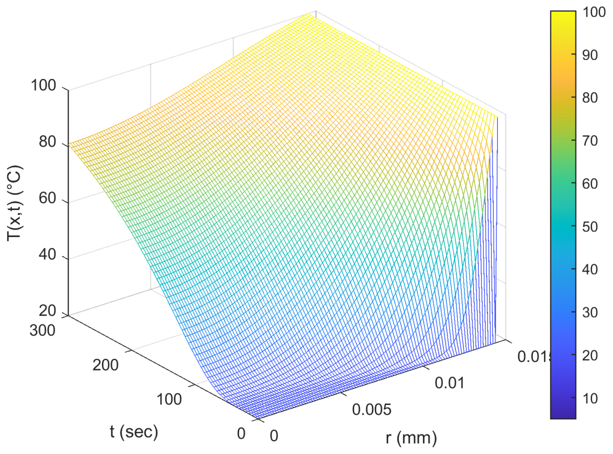


(b) 2D Ostrich Egg Cooking Heat

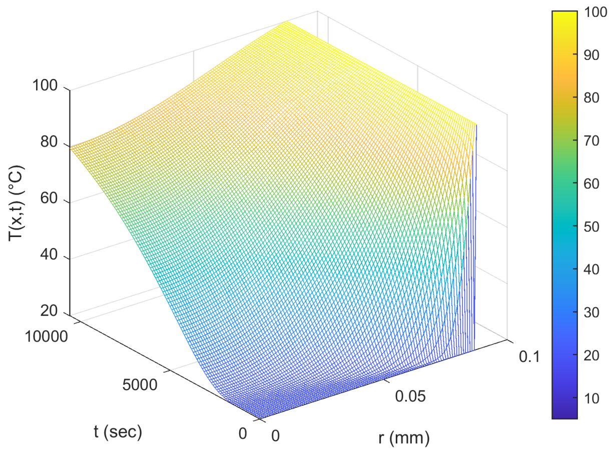


(c) 2D Chicken Egg Cooking Heat

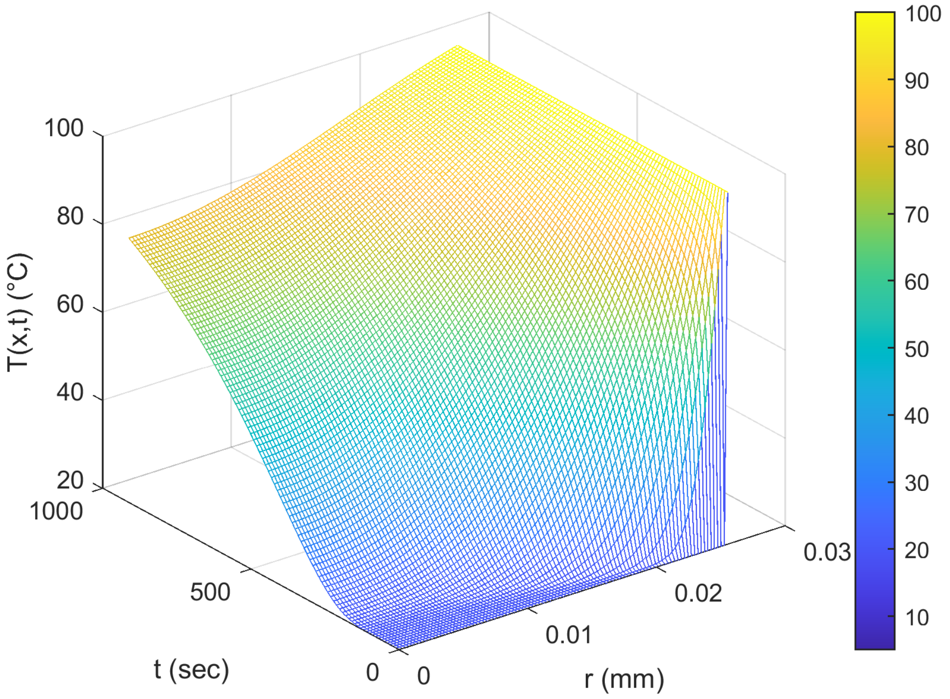
Figure 1: Cooking Heat for Selection of Eggs (2D)



(a) 3D Quail Egg Cooking Heat



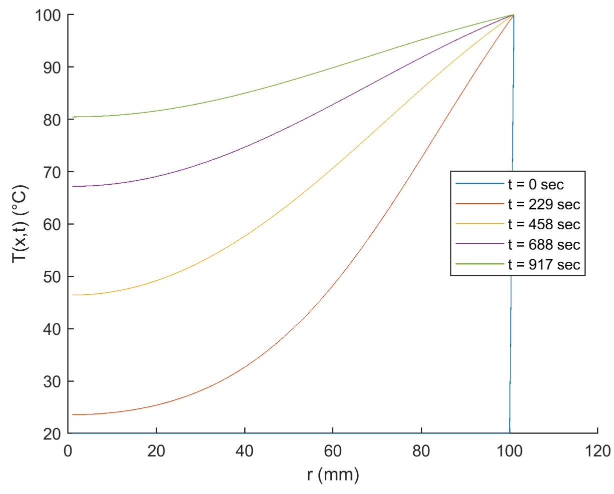
(b) 3D Ostrich Egg Cooking Heat



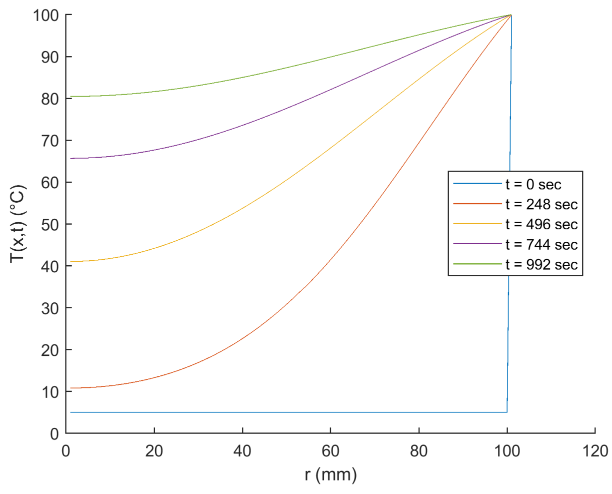
(c) 3D Chicken Egg Cooking Heat

Figure 2: Cooking Heat for Selection of Eggs (3D)

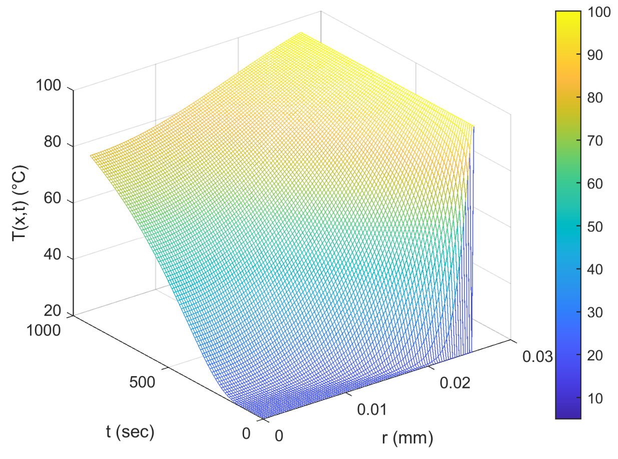
In regards to differences in temperature, there was very little to be seen. While it did require slightly more time to cook the refrigerated egg (about 1 minute), there is little noticeable difference in the graphs of figure 3. If observed closely, the difference in initial temperature between figure 3c and figure 3d can be seen, but the temperatures quickly normalize as the increased temperature difference speeds up the heat transfer into the refrigerated egg.



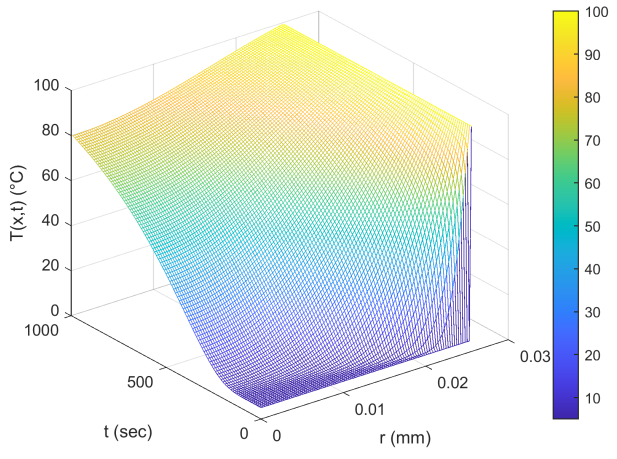
(a) Room Temp Chicken Egg



(b) Refrigerated Chicken Egg



(c) Room Temp Chicken Egg



(d) Refrigerated Chicken Egg

Figure 3: Cooking Heat for Different Initial Egg Temperatures

2.4 Improving Thermal Efficiency

In an industrial or commercial application, cost is almost always the driving factor. One major cost related to food production is energy usage, generally in the form of heat. While it may be fast and convenient to heat an egg using boiling water, it is not terribly efficient. As can be seen in figure 1c, when the center of the egg reaches the required internal temperature, every other part of the egg reaches a temperature significantly above this. All of this excess temperature is a sign of the unnecessary heat in that part of the egg. The easiest solution to reducing this excess heat is to lower the temperature used to heat the egg. If a target temperature of 80°C is desired, a slightly higher temperature of 81°C can be used to slowly force the heat into the egg. In order to determine the energy transferred into the egg, we can use the below equation.

$$Q = \int_0^R (T(r) - T_0)\, \rho\, c_p \left(\frac{4}{3}\pi\, r^3\right)\, dr \tag{6}$$

Discretizing this integral, the existing values from the numerical solution can be used.

$$Q = \sum_{\substack{r=0 \\ \text{step}\,\Delta r}}^{R} (T_r^{k_{final}} - T_0)\, \rho\, c_p \left(\frac{4}{3}\pi\, r^3\right)\, \Delta r \tag{7}$$

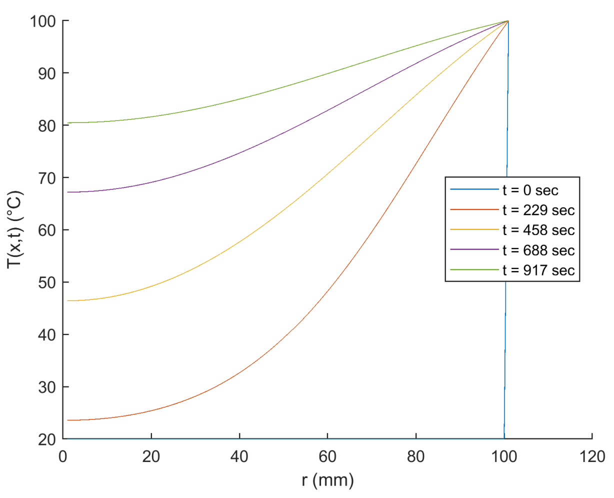
Using equation 7, the data could be generated for table 2 using all of the configurations listed in table 1 and an additional configuration of a chicken egg cooked using 81°C water.

As can be seen, there is an energy savings of almost 20%. While the time required increased more than twofold, the energy savings is significant. In the context of a factory, a large vat could be used to heat the eggs. In which case, the time required would not be of major consequence. Instead, the energy saved would could greatly reduce the cost of production. Not to mention any energy saving thanks to no longer creating large amounts of steam with 100°C water.

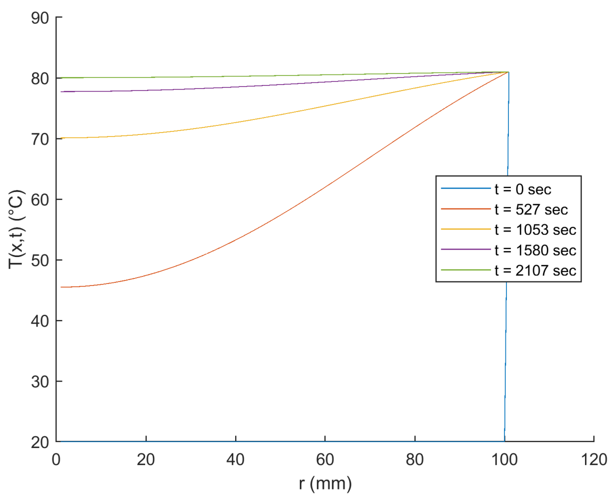
Table 2: Egg Cooking Energy Usage

Egg Type	Starting Temp (°C)	Cooking Temp (°C)	Cook Time	Energy Absorbed (J)
Quail Egg	20	100	5m3s	35.5
Chicken Egg (slow cook)	20	81	35m7s	272.7
Chicken Egg	20	100	15m17s	338.3
Chicken Egg (fridge)	5	100	16m32s	405.6
Ostrich Egg	20	100	178m7s	46833.0

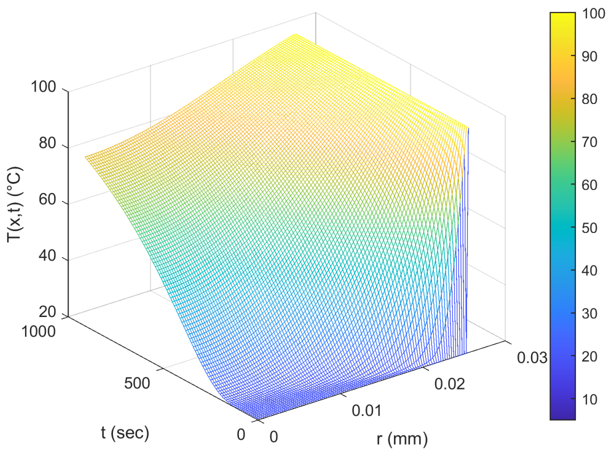
The heat propagation shown in figure 4 highlights the low temperature difference and increased amount of time needed for saturation of 80°C.



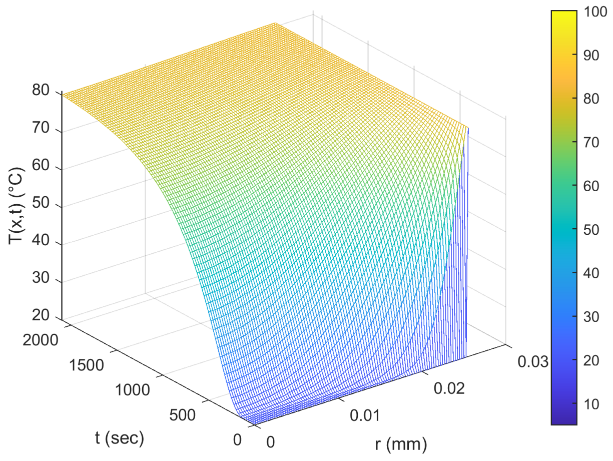
(a) Chicken Egg Cooked at 100°C



(b) Chicken Egg Cooked at 81°C



(c) Chicken Egg Cooked at 100°C



(d) Chicken Egg Cooked at 81°C

Figure 4: Cooking Heat for Low Temp Cooking

3 1D Heat Equation - Analytical and Numerical Solutions

The 1D Heat equation is a fundamental application of Partial Differential Equations. It is used to model how temperature changes along every point on an object, with respect to time. 1D refers to the analysis of temperature along a single dimension or axis of distance. The 2D heat equation accounts for two directions and will be discussed later in the report. Since temperature is dependent on two variables: distance and time, the PDE must accounts for both relationships. The 1D Heat Equation used for this system can be described by a PDE and its respective parameters:

$$\begin{aligned} \text{PDE:} &\quad u_t = 2u_{xx}, \quad x \in (0,1)\,, t \in (0,\infty) \\ \text{BCs:} &\quad u(x=0,t) = 0,\, u(x=1,t) = 2 \\ \text{IC:} &\quad u(x,t=0) = \cos(\pi x) = f(x) \end{aligned} \tag{8}$$

3.1 Analytical Solution

Based on the system described in Equation 8, it is apparent that there are two non-homogeneous Dirichlet boundaries. The boundary at $x = 0$ will be referred to by u_0 and similarly u_L for $x = L = 1$. To analytically solve this PDE, the method of Separation of Variables must be performed. However, this method only works for linear and homogeneous PDEs. Equation 9 denotes the shift equation which accounts for the differing boundaries:

$$\begin{aligned} \phi &= u_0 + \frac{u_L - u_0}{L}x \\ \phi &= 0 + \frac{2-0}{1}x \\ \phi &= 2x \end{aligned} \tag{9}$$

This shift equation permits the use of S.O.V. on a new PDE where the boundaries can be homogeneous at 0 and the shift function will reflect the real boundary conditions in the original PDE. The initial condition for this new PDE of: $v_t = 2v_{xx}$ also incorporates the shift function.

$$\begin{aligned} \text{IC:} &\quad v(x,t=0) = f(x) - \phi \\ &\quad v(x,t=0) = \cos(\pi x) - 2x \end{aligned} \tag{10}$$

Now S.O.V. can be performed on this new PDE with the newly defined B.C. and I.C. The first step shown in equation 11 requires the PDE to be split into two ODEs: one for distance (x) and one for time (t).

$$\begin{array}{c} v_t = 2v_{xx} \\ \Downarrow \\ \text{S.O.V.} \end{array} \qquad (11a)$$

$$\begin{aligned} \text{IC:}\, & v(x,\, t=0) = f(x) - \left[u_0 + \frac{u_L - u_0}{L}x\right] \\ \text{BC:}\, & v(0,\, t) = 0 \\ & v(1,\, t) = 0 \end{aligned} \qquad (11b)$$

$$v = XT \quad \rightarrow \quad \frac{X''}{X} = \frac{T'}{\alpha^2 T} = k \qquad (11c)$$

$$\begin{aligned} X'' - kX &= 0 \\ k &= -\mu^2 < 0 \\ X &= A\cos(\mu\, x) + B\sin(\mu\, x) \\ u = X(0)\,T = 0 \quad &\rightarrow \quad X(0) = 0 \\ u = X(1)\,T = 0 \quad &\rightarrow \quad X(1) = 0 \\ X(0) = A(1) + B(0) = 0 \quad &\rightarrow \quad A = 0 \\ X(L) = 0 + B\,\sin(\mu\, L) &= 0 \\ B \neq 0\,, \quad \mu L &= n\,\pi \\ X_n &= B_n\,\sin\left(\underbrace{\frac{n\,\pi}{L}}_{L=1}\,x\right) \\ & \boxed{X_n = B_n\,\sin(n\,\pi\, x)} \end{aligned} \qquad (11d)$$

$$\begin{aligned} T' - k\alpha^2 T &= 0 \\ T' + \left(\mu\alpha^2\right) T &= 0 \\ T = C\cdot e^{-(\mu\alpha)^2 t} \quad &\rightarrow \alpha = \sqrt{2} \\ & \boxed{T_n = C_n\, e^{-2(n\pi)^2 t}} \end{aligned} \qquad (11e)$$

Next, refer to the equation $v = XT$, shown in equation 11c. X_n and B_n can be plugged back into this equation to find V_n. To find the solution to the PDE, V_n can be written as the sum of infinite combinations of X_n and V_n to give us $v(x, t)$. This is shown in equation 12.

$$\begin{aligned} V_n &= X_n T_n \\ v(x,t) &= \sum_{n=1}^{\infty} C_n B_n e^{-2(n\pi)^2 t}\, \sin(n\pi x) \\ v(x,t) &= \sum_{n=1}^{\infty} D_n e^{-2(n\pi)^2 t}\, \sin(n\pi x) \end{aligned} \qquad (12)$$

Equation 12 provides the solution for the shifted PDE, but the value of coefficient D_n is still required. To solve for D_n, the use of the shifted I.C. described in equation 10 presents itself. By equating the I.C. and the PDE when $v(x, t = 0)$, D_n can be found.

$$v(x,t=0)=\sum_{n=1}^{\infty} D_n e^{-2(n\pi)^2(0)}\,\sin(n\pi x)$$
$$v(x,t=0)=\sum_{n=1}^{\infty} D_n\,\sin(n\pi x) \tag{13a}$$
$$cos(\pi x)-2x=\sum_{n=1}^{\infty} D_n\,\sin(n\pi x)$$

Use Euler's Coefficient Formula for a Fourier sine series to solve for D_n

$$D_n=\frac{2}{L}\int_0^L [f(x)]\,\sin(n\pi x)\,dx$$
$$D_n=2\int_0^1 [cos(\pi x)-2x]\,\sin(n\pi x)\,dx \tag{13b}$$

Due to the complexity of the integral, it was solved using WolframAlpha

$$D_n=\frac{2\left[\frac{\pi n^3(\cos(\pi n)+1)}{n^2-1}-2\sin(\pi n)+2\pi n\,\cos(\pi n)\right]}{\pi^2 n^2} \tag{13c}$$

Now that D_n has been solved, the PDE of $v_t=2v_{xx}$ has been solved. Yet, there is one step left: $v(x,t)$ must be plugged back into the solution for the original 1D Heat Equation PDE. This is shown below:

$$u(x,t)=v(x,t)+\phi$$
$$u(x,t)=2x+\sum_{n=1}^{\infty} D_n e^{-2(n\pi)^2 t}\,\sin(n\pi x) \tag{14}$$

Therefore the final analytical solution to the 1D Heat Equation from equation 8 is found.

3.2 Numerical Solution

The same 1D Heat Equation PDE from equation 8 can also be solved numerically. Similarly to the process used in Section 2.2, the differential elements of the PDE need to be replaced with relations of their previous elements. This is performed using an explicit time-advancement scheme. For this case, u_t can be replaced with $\frac{T_i^{k+1}-T_i^k}{\Delta t}$ and u_{xx} can be replaced with $\frac{T_{i+1}^k-2T_i^k+T_{i-1}^k}{\Delta x^2}$. The theory behind this solution method consists of using the temperature distributions from the previous time-step to determine the temperature distribution at the next time-step. Since the I.C. dictates the the temperature along every point of the material at $t=0$, the temperature of a specific point after Δt has passed, is governed by the temperature of its surrounding points just before Δt. To demonstrate this concept, the heat equation PDE is rearranged to solve for T_i^{k+1}.

$$
\begin{aligned}
T_t &= 2T_{xx} \\
\frac{T_i^{k+1} - T_i^k}{\Delta t} &= 2\left(\frac{T_{i+1}^k - 2T_i^k + T_{i-1}^k}{\Delta x^2}\right) \\
T_i^{k+1} &= T_i^k + \frac{2\Delta t}{\Delta x^2}\left(T_{i+1}^k - 2T_i^k + T_{i-1}^k\right) \\
F &= \frac{2\Delta t}{\Delta x^2}
\end{aligned} \tag{15}
$$

This is then implemented in the code as shown in listing 4.

Listing 4: Explicit Time Advancement Incrementing — Q2_1D_heat_eqn.m

```matlab
% Space: discretized using second order derivatives
% Time: explicit time advancement,
% i.e, k+1 values are determined by k values
% BCs: Dirichlet (constant temperature)
for k=1:M-1 % time
    for i=2:N % space
        T(i,k+1)=T(i,k)+alpha*(T(i+1,k)-2*T(i,k)+T(i-1,k));
    end
end
```

Note: alpha in the code has been already initialized and calculated according to F shown in equation 15.

The boundary and initial conditions are applied both before the incrementing loop and during the iteration. Listing 5 and 6 show the initial and boundary conditions applied to the system.

Listing 5: 1D Heat Equation Initial Conditions — Q2_1D_heat_eqn.m

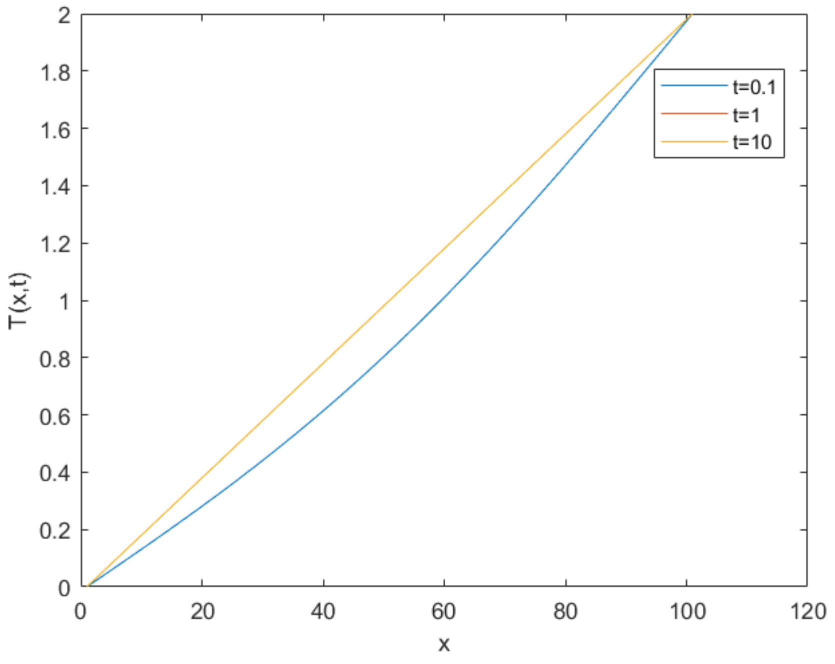
```matlab
% Initial Condition
T(:,1) = cos(pi*x);
```

Listing 6: 1D Heat Equation Boundary Conditions — Q2_1D_heat_eqn.m

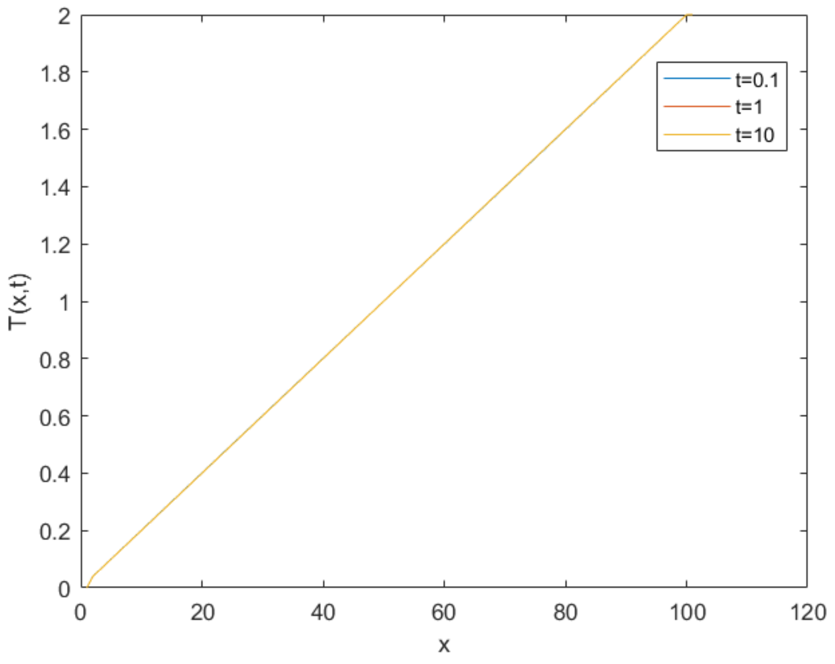
```matlab
% Dirichlet Boundary conditions at either end
T(1,:) = 0;
T(end,:) = 2;
```

The full code can be found in the section 6.2 of the appendix as listing 13.

3.3 Comparison of Numerical and Analytical Solutions



(a) Numerical Solution at Varying Times



(b) Analytical Solution at Varying Times

Figure 5: 1D Heat Equation Solutions at $t = 0.1, 1, 10$ seconds

For the numerical solution to the 1D heat equation, figure 5a shows a slight variation in the temperature distribution after $t = 0.1$ seconds have elapsed. However, it is apparent that at $t = 1$ second and $t = 10$ seconds, the function has already reached a steady-state - illustrated by the linear increase in temperature from boundary condition $T_0 = 0$ to $T_L = 2$ (units of temperature). In fact, the plots of temperature at $t = 1$ and $t = 10$ are identical and on-top of one another. We can conclude that the numerical solution approximates that the temperature distribution will reach a steady-state with respect to time, after only 1 second. Furthermore, the analytical solution plots shown in figure 5b demonstrate a nearly rapid progression towards steady-state as all 3 plots are identical, falling on the same line. The major difference between the numerical and analytical solution is the temperature distribution at $t = 0.1$ seconds. Although the numerical solution only suggests a small deviation from steady-state after 0.1 seconds elapsed, we can conclude from both figures 5a and 5b, that the material being modelled by equation 8 rapidly reaches a steady-state of temperature across its length.

3.4 Solution Accuracy and Parameter Choice

The problem defined a material with length (L) of 1 unit and required an analysis of the material at three distinct time-points, the largest being 10 seconds. This served as the total time that the time-advancement scheme would increment to. Additionally, the PDE defined in equation 8 denotes that $\alpha = 2$. With these parameters defined, an appropriate grid-spacing was required for both time (Δt) and distance (Δx). Listing 7 below shows the exact grid spacing chosen for the numerical solution analysis.

Listing 7: Numerical Solution Grid Spacing Parameters — Q2_1D_heat_eqn.m

```
Length = 1; % x in (0,L)
TotalTime = 10; % t in (0,t) defined by project outline
c = 2; % conductivity
N = 100; % number of grid points
M = 400000; % number of time points
dx=Length/N; dt=TotalTime/M; % grid spacing
alpha = c*dt/dx^2;
```

100 grid points were evaluated along the material's length from $x \in (0, L = 1)$. 400,000 time points were evaluated from $t \in (0, 10)$ seconds. This equated to a $\Delta x = 0.01$ and a $\Delta t = 2.5 \times 10^{-5}$. When using the explicit time-advancement scheme to numerically solve a PDE, it is important to consider stability and its effect on accuracy. Stability can be defined by equation 16.

$$\begin{gathered} F = \frac{2\Delta t}{\Delta x^2} \\ F = \frac{2 \times 2.5 \times 10^{-5}}{0.01^2} \\ F = 0.5 \\ (1 - 2F) > 0 \\ (1 - 2 \times 0.5) > 0 \checkmark \end{gathered} \tag{16}$$

Since, the chosen Δx and Δt satisfied equation 16, the explicit method would remain stable. This stability coincides with the accuracy of the numerical solutions shown in figure 5a, as there are an extremely large amount of time points captured and evaluated along a sufficiently small amount of grid points.

Similarly, in order to derive an analytical solution to the 1D Heat Equation PDE, several steps and simplifications are made. To simplify the Fourier coefficient, a complex integral is required to be solved due to the non-homogeneous Dirichlet boundaries and the trigonometric initial condition. Errors made during the simplification can lead to inaccuracy within the sum of the Fourier series used to approximate the non-steady component of the temperature distribution. However, as illustrated in figure 5b, over time the solution approaches a steady state governed by the shift function $\phi = 2x$ to reach the boundary conditions dictated at each respective endpoint.

4 2D Heat Equation - Cooling Flatbread

An interesting application of 2D Heat Equations is analyzing and modelling the process of cooling down something like a piece of flatbread on a large square plane. This question analyzes and numerically solves the 2D Heat Equation for this problem for two different types of setups. The first setup is using exclusively Dirichlet boundary conditions to provide beginning values for the solution. The second setup that was analyzed used mixed boundary conditions. It used 2 boundary conditions which were Dirichlet boundary conditions similar to the previous setup, and then it used 2 boundary conditions which were Neumann boundary conditions.

4.1 Exclusively Dirichlet Boundary Conditions

The first problem in this question focused on solving a PDE for cooling a piece of flatbread using exclusively Dirichlet boundary conditions. This situation is modelled by the following PDE:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T, \qquad \text{where} \quad \alpha = 1 \tag{17}$$

This PDE can be rewritten explicitly using x and y instead of the Laplacian operator as the following:

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \tag{18}$$

In order to solve this PDE the following initial conditions were provided:

$$T(x, y, t = 0) = \sin(\pi x)\ \sin(4\pi y) \tag{19}$$

Another thing that is required in order to obtain the numerical solution to an ODE is boundary conditions. For this question, the Dirichlet boundary conditions were provided. Dirichlet boundary conditions are boundary conditions that provide the value of the function at each of the boundary's. For this problem, the following boundary conditions were provided:

$$\begin{aligned} &T(x = 0, t) = 0, \\ &T(x = 1, t) = 0, \\ &T(y = 0, t) = \sin(\pi x), \\ &T(y = 1, t) = \cos(2\pi x) - 1 \end{aligned} \tag{20}$$

In order to numerically solve this PDE, the first step is to discretize the problem.

$$\frac{T_{i,j}^{k+1} - T_{i,j}^{k}}{\Delta t} = \frac{T_{i+1,j}^{k} - 2T_{i,j}^{k} + T_{i-1,j}^{k}}{\Delta x^2} + \frac{T_{i,j+1}^{k} - 2T_{i,j}^{k} + T_{i,j-1}^{k}}{\Delta y^2} \tag{21}$$

This equation can then be rearranged to isolate for the $T_{i,j}^{k+1}$ term, which will be used in MATLAB to set up the numerical solution to this PDE:

$$T_{i,j}^{k+1} = T_{i,j}^{k} + \Delta t \left(\frac{T_{i+1,j}^{k} - 2T_{i,j}^{k} + T_{i-1,j}^{k}}{\Delta x^2} + \frac{T_{i,j+1}^{k} - 2T_{i,j}^{k} + T_{i,j-1}^{k}}{\Delta y^2} \right) \tag{22}$$

4.2 Mixed Boundary Conditions

The second problem in this question focused on solving the exact same PDE, just using slightly different initial conditions and using different types of boundary conditions. This problem used a combination of 2 Dirichlet boundary conditions as well as 2 Neumann boundary conditions.

The initial conditions that were provided for this problem are the following:

$$T(x, y, t = 0) = \sin(4\pi x)\,\cos(4\pi y) \tag{23}$$

The major difference between this problem and the first problem in this question is the provided boundary conditions. In this problem, 2 Neumann boundary conditions were given, which instead of providing the value of the function at the boundary, provide the value of the gradient of the function at the boundaries. This gradient can then be solved and used to determine the values at the boundary. 2 Dirichlet boundary conditions were also provided, similar to the first problem. These were the provided boundary conditions:

$$\begin{aligned} &\nabla T(x = 0, t) \cdot \mathbf{n} = 0, \\ &\nabla T(x = 1, t) \cdot \mathbf{n} = 2, \\ &T(y = 0, t) = 1, \\ &T(y = 1, t) = -1 \end{aligned} \tag{24}$$

Despite the different initial and boundary conditions, the setup for discretizing and solving the PDE numerically is quite similar as before and it arrives at the same equation as Equation 22.

4.3 Interpretation

An extremely important aspect of dealing with using PDEs to model scenarios as well as solving PDEs is the ability to understand and interpret the physical meaning of the setups and initial/boundary conditions.

The physical meaning of the PDE that is being analyzed in this case is that it is modelling the cooling of a piece of Flatbread in a 1x1 grid in the x-y plane. The physical meaning of the initial condition (different for both problems) is that it is the starting temperature of the flatbread.

The boundary conditions also provide an insight on the physical system. The first 2 Dirichlet BCs that are provided can be interpreted physically as implying that the temperature along both x planes (when x = 0 or x = 1) for any time and any y-value, is 0 degrees. The 2 other Dirichlet BCs indicate what the temperatures are along the y planes and provide the functions to model the temperature depending on the x value.

The mixed boundary conditions that are provided in the second problem can also be interpreted physically and they are slightly different. The 2 Dirichlet BCs that are provided in this problem have the same physical interpretation as the Dirichlet BCs from the other problem, however it is the Neumann BCs that are different. The first Neumann boundary condition that is provided is $\nabla T(x=0,t)\cdot\mathbf{n}=0$. This means that along the x=0 plane, there is no energy transfer at the boundary. This could be the case if the boundary is insulated. The second Neumann boundary condition that is provided is $\nabla T(x=1,t)\cdot\mathbf{n}=2$. This tells us that along the x=1 plane, there is a constant temperature gradient of 2. This would be the case if there is a constant heat drain on one side of the system. This could be a heat exchanger or a heat sink of some sort.

4.4 Solving Numerically and Visualizing Solutions

In order to solve these PDEs numerically, the discretized equation 22 was used as well as the Initial and boundary conditions. These where set up in MATLAB using the demo code provided.

The initial and boundary conditions for the Dirichlet BC were setup in the MATLAB code as shown in listing 8.

Listing 8: Dirichlet Boundary and Initial Conditions — Q3_Dirichlet_BC.m

```
% Initial Conditions
T(:,:,1) = sin(pi*X).*sin(4*pi*Y); % at (x,y,t=0)

% Dirichlet boundary conditions (these could also be applied in the loop)
T(1:end,1,:) = 0;                               % at (x=0,y,t)
T(1,:,:) = repmat(sin(pi.*x)',1,1,M);           % at (x,y=0,t)
T(end,:,:) = repmat((cos(2*pi.*x)-1)',1,1,M);   % at (x,y=1,t)
T(:,end,:) = 0;                                 % at (x=1,y,t)  %
```

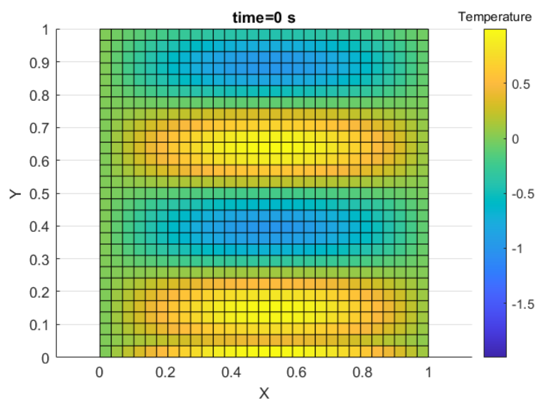
Equation 22 was then implemented into MATLAB using nested for loops and used to numerically calculate each successive value in order to solve the PDE. This is shown below in listing 9:

Listing 9: Dirichlet Numerical Solution — Q3_Dirichlet_BC.m

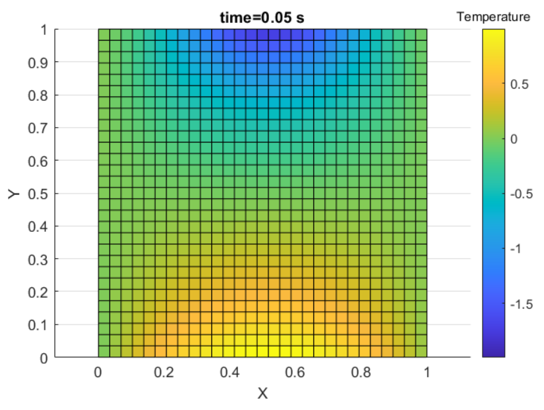
```
% PDE Solution -------------------------------------------------------------
% Space: second order derivatives
% Time: explicit time marching, i.e, k+1 values are determined by k values
```

```
for k=1:M-1  % time step up to specified final time
    for i=2:length(y)-1 % x node step
        for j=2:length(x)-1 % y node step
            % Solving for temperature at next time step
            T(i,j,k+1) = ((((T(i,j+1,k)-2*T(i,j,k)+T(i,j-1,k))/dx^2)....
            +((T(i+1,j,k)-2*T(i,j,k)+T(i-1,j,k))/dy^2))*dt)+T(i,j,k);
        end
    end
end %
```

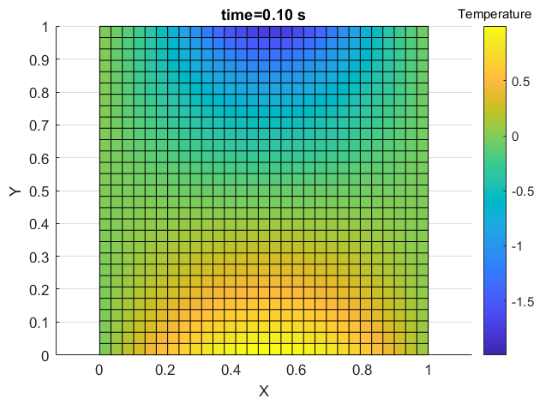
Now that the PDE had been numerically solved, MATLAB's plotting function was used to visualize the solution to the PDE and see the change of Temperatures through different time steps. This plotting function was used to create plots of the temperature on the x-y plane at 4 different time steps; 0s, 0.05s, 0.10s, and 0.15s. These plots are shown below in figure 6.



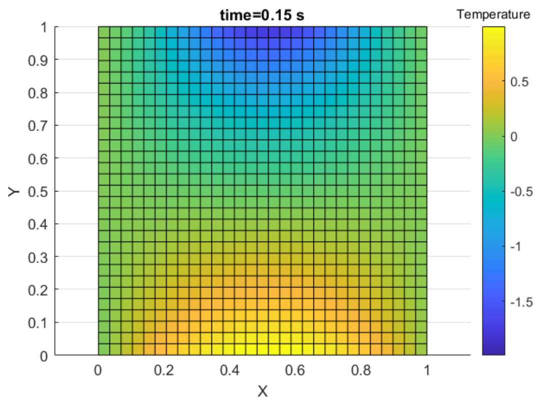
(a) Temperature at t = 0s



(b) Temperature at t = 0.05s



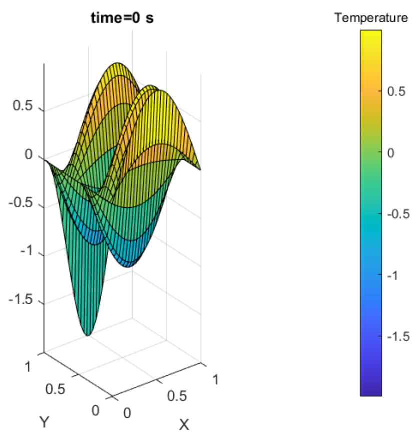
(c) Temperature at t = 0.10s



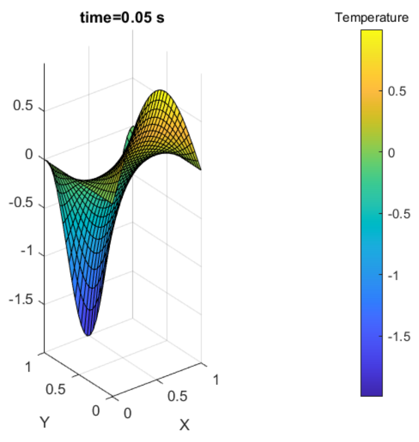
(d) Temperature at t = 0.15s

Figure 6: Temperature along x-y plane at different time

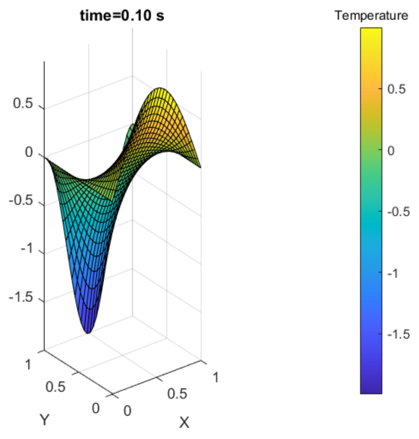
3D versions of the same plots were also included in order to visualize the solution to the PDE. These are shown below in figure 7.



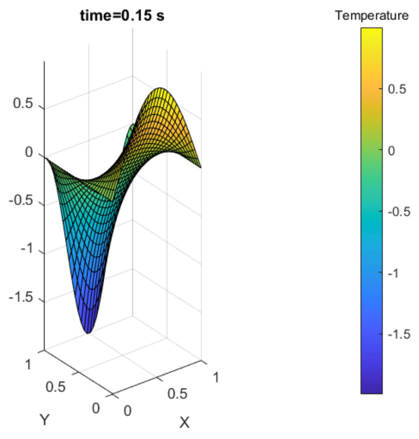
(a) Temperature at t = 0s



(b) Temperature at t = 0.05s



(c) Temperature at t = 0.10s



(d) Temperature at t = 0.15s

Figure 7: Temperature function at different time

The numerical solution for the second PDE using different Initial and boundary conditions was quite similar in MATLAB to the solution to the first one, however there were some important differences. The biggest difference was of course in the initialization of the Initial and boundary conditions. Listing 10 shows the code that sets up the initial condition as well as the 2 Dirichlet boundary conditions.

Listing 10: Neumann Initial Conditions — Q3_Neumann_BC.m

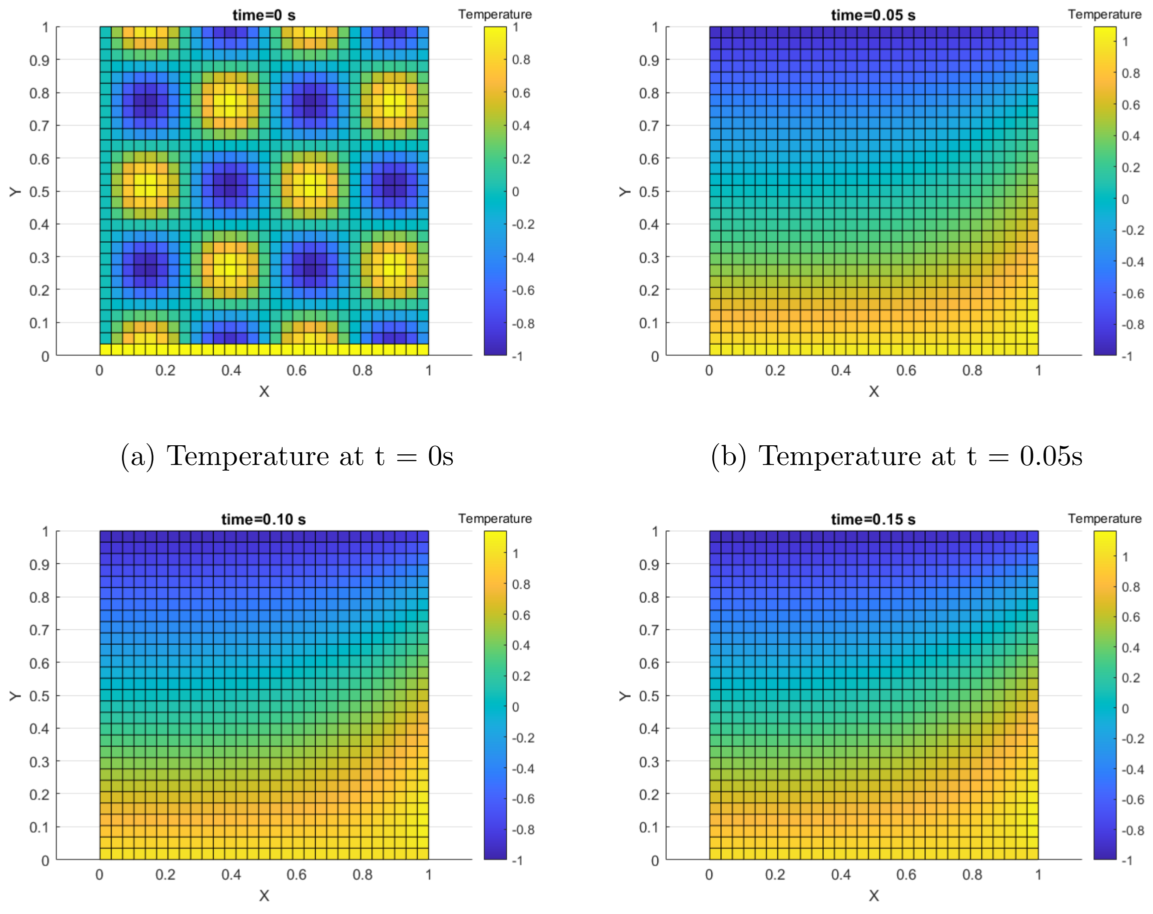
```
% Initial Conditions
T(:,:,1) = sin(4*pi*X).*cos(4*pi*Y); % at (x,y,t=0)

% Mixed boundary conditions (these could also be applied in the loop)
T(1,:,:) = 1;                                   % at (x,y=0,t)
T(end,:,:) = -1;                                % at (x,y=1,t)
```

It is important to note that only 2 of the 4 boundary conditions have been set up so far in the code and the Neumann boundary conditions are still missing. Those were implemented in the loop which is used to iteratively solve the PDE as shown in listing 11

Listing 11: Neumann Numerical Solution — Q3_Neumann_BC.m

```matlab
% PDE Solution -------------------------------------------------------------
% Space: second order derivatives
% Time: explicit time marching, i.e, k+1 values are determined by k values
for k=1:M-1  % time step up to specified final time
    for i=2:length(y)-1 % x node step
        for j=2:length(x)-1 % y node step
            % Solving for temperature at next time step
            T(i,j,k+1) = ((((T(i,j+1,k)-2*T(i,j,k)+T(i,j-1,k))/dx^2)....
            +((T(i+1,j,k)-2*T(i,j,k)+T(i-1,j,k))/dy^2))*dt)+T(i,j,k);
        end
    end
    T(1:end,1,k+1) = T(1:end,2,k);
    T(1:end,length(y),k+1) = T(1:end,length(y)-1,k+1) + 2*dx;
end %
```

Since this PDE has now also been solved numerically, The same steps as before were followed MATLAB plotting was used to visualize it at 4 different time steps in the same way that was done for the Dirichlet BC problem. The 2D plots are shown below in figure 8.



(a) Temperature at t = 0s

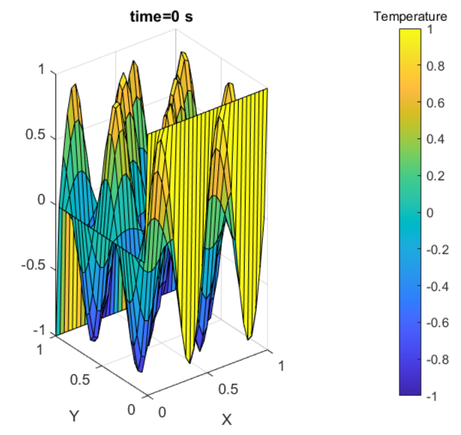
(b) Temperature at t = 0.05s

(c) Temperature at t = 0.10s

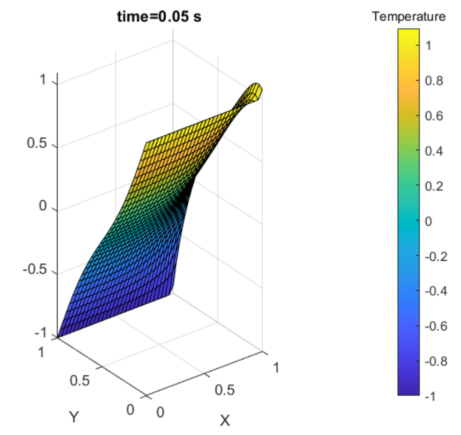
(d) Temperature at t = 0.15s

Figure 8: Temperature along x-y plane at different time

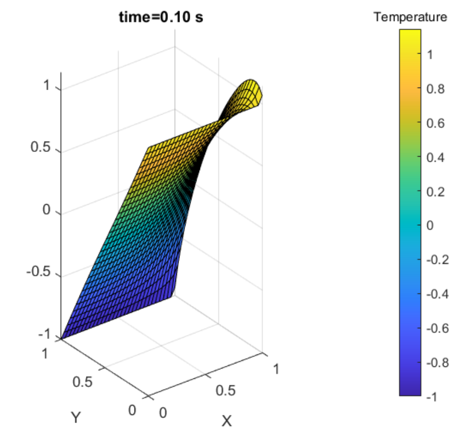
Similar to before, 3D versions of these plots were included in order to visualize the solution to the PDE. These are shown below in figure 9.



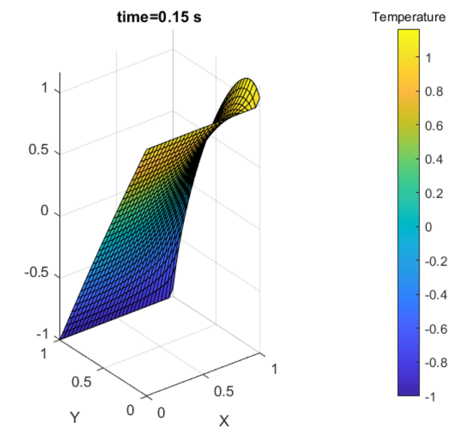
(a) Temperature at t = 0s



(b) Temperature at t = 0.05s



(c) Temperature at t = 0.10s



(d) Temperature at t = 0.15s

Figure 9: Temperature along x-y plane at different time

5 References

[1] Center for Food Safety and Applied Nutrition. Safety of eggs and menu and deli items made from raw shell eggs, Mar 2022.

[2] Martin Freesmeyer, Christian Kuehnel, Thomas Opfermann, Tobias Niksch, Steffen Wiegand, Ronny Stolz, Ralph Huonker, Otto W. Witte, and Thomas Winkens. The use of ostrich eggs for in ovo research: Making preclinical imaging research affordable and available. *Journal of Nuclear Medicine*, 59(12):1901–1906, 2018.

[3] Adam Powell. Transport phenomena in materials engineering - heat transfer - 1-d thermal diffusion equation and solution.

[4] Anna Wilkanowska and Dariusz Kokoszyński. Layer age and quality of pharaoh quail eggs. *Journal of Central European Agriculture*, 13:10–21, 03 2012.

6 Appendix

6.1 Group Member Contribution

Table 3: Group Member Contributions

Member	Contributions
Japmeet Brar	Question 3, Including Math, code, graphs and report.
Kevin Chu	Question 2, Including Math, code, graphs, and report.
Austin W. Milne	Question 1, Including Math, code, graphs, and report. Overall report formatting.
Joshua Selvanayagam	Question 2, Including Math, code, graphs, and report.

6.2 Full Code Listings

Listing 12: Numerical 1D Heat Equation in Spherical— Q1_egg_cook.m

```
% This code serves to calculate the time needed to cook a selection of eggs
% by means of boiling and hot water. The time is caculated by the numerical
% solution of the 1D heat equation in spherical coordinates.

arg_set = ... % Code Name, Text Name, Radius (mm), initial temp, Outer temp, 3D graph div
 {"reg_egg",     "20 deg Chicken Egg",               0.0255, 20, 100,  1000; % 45mm x 57mm
  "fridge_egg",  "5 deg Chicken Egg",                0.0255, 5,  100,  1000;
  "quail_egg",   "20 deg Quail Egg",                 0.0145, 20, 100,   500; % 32mm x 25mm
  "ostrich_egg", "20 deg Ostrich Egg",               0.0875, 20, 100, 10000; % 20cm x 15cm
  "slow_egg",    "20 deg Chicken Egg in warm water", 0.0255, 20, 81,   2500; }; %

size_ = size(arg_set);
len = size_(1);
for j = 1:len
    args = arg_set(j,:);

    % Parameter definitions ---------------------------------------------------
    safe_name = args{1};
    log = fopen(sprintf('out/q1/%s/output.log',safe_name),'w');
    human_name = args{2};
    R = args{3}; % Radius of egg [M]
    T_start = args{4}; % Room temperature in celsius
    T_water = args{5}; % Temperature of the cooking water
    graph_div = args{6};
    T_min = 80; % Minimum cooking temp throughout
    t_hold = 10; % Hold for 10 seconds
    k = .500; % conductivity of egg [W K^-1 m^-1]
    rho = 1035; % Density of egg [kg m^-3]
    c_p = 3200; % Specific heat of Egg [J kg^-1 K^-1]
    N = 100; % number of grid points
    dt=0.01; % Size of time step
    dr=R/N;  % grid spacing

    alpha = k / (rho*c_p); % Increment coefficient
    coeff = 1-2*alpha*dt/(dr^2);
    fprintf(log, "alpha = %2.10f, coeff = %2.10f \n", alpha, coeff);

    % Initialization ----------------------------------------------------------
    % solution grid
    x = linspace(0,R,N+1);
    T = ones(N+1,1);
    % Initial Condition
    T(:,1) = T_start;
    % Boundary conditions
    T(end,1) = T_water; %

    % PDE Solution -----------------------------------------------------------
    k = 1;
    at_temp_time = 0;
    while (at_temp_time < t_hold); % While holding for temp
        T(1,k+1) = max(T(2,k) - (T(3,k) - T(2,k)), T_start); % Set center to 1 'slope' lower
        T(end,k+1) = T(end,k); % Retain end value
        for i=2:N % Increment over all but the ends
            r = (i-1) * dr; % Get the radius
            d2T_dr2 = (T(i+1,k)-2*T(i,k)+T(i-1,k))/(dr^2); % Get the instantaneous accel
            dT_dr = (T(i+1,k)-T(i-1,k))/(2*dr); % Get the instaneous slope
            T(i,k+1)=T(i,k) + alpha*dt*(d2T_dr2 + (2/r)*dT_dr); % Increment Temp
        end %
        % At every N seconds, Output some temp values.
        time = k * dt;
        if not(mod(time, 10))
            fprintf(log, "Time: %5.6f - Temp: %3.14f, %3.14f, %3.14f, %3.14f, %3.14f, %3.14f
                , %3.14f \n", ...
            time, T(1, k), T(N/100, k), T(N/10, k), T(N*5/10,k), T(N*9/10,k), T(N*99/100,k),
                T(N-1,k));
        end
        k = k + 1; % Increment the counter
        % Increment the time if we are at temp
        if all(T(:,k) > T_min)
            at_temp_time = at_temp_time + dt;
        end
    end %


    % Final Tally -------------------------------------------------------------

    size_t = length(T(1,:));
    size_r = length(T(:,1));
    t_set = linspace(0, time, size_t);
    r_set = linspace(0, R, size_r);

    % Cook Time
    time = size_t * dt;
    min = (time - mod(time, 60)) / 60;
    sec = round(time - min*60);
    fprintf(log, "Time to cook %s: %f sec - %im%is\n", human_name, time, min, sec);

    % Calculate heat energy absorbed
    vol = 4 * pi .* r_set.^3 .* dr;
    energy = (T(:,end) - T_start) .* vol' .* c_p .* rho;
    fprintf(log, "Total energy absorbed for %s: %f Joules \n", human_name, sum(energy));

    % Visualization -----------------------------------------------------------
    f = figure('visible','off');
    % line plot
    indexes = [1, round(size_t/4), round(size_t/2), round(size_t*3/4), round(size_t)];
    times = indexes * dt;
    hold on;
    plot(T(:,indexes(1)));
    plot(T(:,indexes(2)));
    plot(T(:,indexes(3)));
    plot(T(:,indexes(4)));
    plot(T(:,indexes(5)));
    xlabel('r (mm)'); ylabel('T(x,t) (C)');
    legend(split(sprintf('t = %i sec,', round(times(:))), ","), 'Location', "east");
    exportgraphics(f, sprintf("out/q1/%s/2D_Plot.png",safe_name), 'Resolution', 300);
    clf(f);

    % surface plot
    [X,Y] = meshgrid(r_set,t_set(1:graph_div:end));
    mesh(X,Y,T(:,1:graph_div:end)');
    colormap('parula');
    xlabel('r (mm)'); ylabel('t (sec)'); zlabel('T(x,t) (C)');
    colorbar;
    caxis([5 100]);
    exportgraphics(f, sprintf("out/q1/%s/3D_Plot.png",safe_name), 'Resolution', 300);
    clf(f);
    fclose(log);
end
exit
```

Listing 13: 1D Heat Equation Numerical and Analytical Solutions— Q2_1D_heat_eqn.m

```matlab
% ---------------------------------------------------------------------------
% This is a script that plots the Numerical and Analytical Solutions
% for the 1D Heat Equation.

% Author: Joshua Selvanayagam
%
% Modified: 2022-04-04
% ---------------------------------------------------------------------------

clear all; clc; close all;

% Parameter definitions ------------------------------------------------------
Length = 1; % x in (0,L)
TotalTime = 10; % t in (0,t) defined by project outline
c = 2; % conductivity
N = 100; % number of grid points
M = 400000; % number of time points
dx=Length/N; dt=TotalTime/M; % grid spacing
alpha = c*dt/dx^2;

% Initialization -------------------------------------------------------------
% solution grid
x = linspace(0,Length,N+1);
T = ones(N+1,M);

% Initial Condition
T(:,1) = cos(pi*x);

% Dirichlet Boundary conditions at either end
T(1,:) = 0;
T(end,:) = 2;

% Matrix Initialization for Analytical Solution
V = ones(N+1,M);
V(:,1) = cos(pi*x);
V(1,:) = 0;
V(end,:) = 2;

% Numerical Solution ---------------------------------------------------------
% Space: discretized using second order derivatives
% Time: explicit time advancement,
% i.e, k+1 values are determined by k values
% BCs: Dirichlet (constant temperature)
for k=1:M-1 % time
    for i=2:N % space
        T(i,k+1)=T(i,k)+alpha*(T(i+1,k)-2*T(i,k)+T(i-1,k));
    end
end

% Analytical Solution --------------------------------------------------------
v = 0;
individualSum = 0;
time = 0;
space = 0;
D_const = 0;

for j=1:M-1 %time points
   time = TotalTime * j/M; %scale time based on 0 to 10 seconds
   for grid_space=2:N %space points
       space = grid_space./100; %scale grid based on 0 to 1 unit length
       v = 0;
        for n_sum=2:100 %summation of Fourier Series
            %Calculate D_n constant at each n
            D_const = (2.*((pi.*(n_sum.^3).*(cos(pi.*n_sum)+1)...
            ./((n_sum.^2)-1))-(2.*sin(pi.*n_sum))...
            +(2.*pi.*n_sum.*cos(pi.*n_sum)))./((pi.^2).*(n_sum.^2));

            %Calculate the partial sum at each n
            individualSum = D_const.*(exp((-2.*((n_sum.*pi).^2)).*time)...
            .*sin(n_sum.*pi.*space));

            %Add the partial sum to the total after each iteration of n
            v = v + individualSum;
        end
        V(grid_space, j+1) = 2.*space + v;
    end
end

% Visuaization ------------------------------------------------------------
% line plot
figure(1)
plot(T(:,4000)); hold on
plot(T(:,40000))
plot(T(:,400000))
xlabel('x'); ylabel('T(x,t)');
legend('t=0.1','t=1','t=10')

figure(2)
plot(V(:,4000)); hold on
plot(V(:,40000))
plot(V(:,400000))
xlabel('x'); ylabel('T(x,t)');
legend('t=0.1','t=1','t=10')
```

Listing 14: Numerical 2D Heat Equation With Dirichlet BC— Q3_Dirichlet_BC.m

```matlab
% -------------------------------------------------------------------------
% This script calculates heat diffusion on a square domain using Dirichlet Boundary
    Conditions.
% -------------------------------------------------------------------------

clear all; clc; close all;

% Parameter definitions ---------------------------------------------------
N = 30; % numbers of nodes along both x and y directions
L = 1; % Length of square domain
alpha = 1;  % thermal diffusivity
dt = 0.0002; % time step
dx = L/N; % grid spacing in x direction
dy = dx; % grid spacing in y direction (same as x direction)
finalTime = 0.15; % final time given in the problem
M = int16(finalTime/dt); % number of time steps required

% Initialization ----------------------------------------------------------
% Solution Grid
x=linspace(0,L,N);
y=linspace(0,L,N);
[X,Y]=meshgrid(x,y);
T = ones([size(X) M]); % all time steps are stored for demo purposes

% Initial Conditions
T(:,:,1) = sin(pi*X).*sin(4*pi*Y); % at (x,y,t=0)

% Dirichlet boundary conditions (these could also be applied in the loop)
T(1:end,1,:) = 0;                             % at (x=0,y,t)
T(1,:,:) = repmat(sin(pi.*x)',1,1,M);         % at (x,y=0,t)
T(end,:,:) = repmat((cos(2*pi.*x)-1)',1,1,M); % at (x,y=1,t)
T(:,end,:) = 0;                               % at (x=1,y,t)  %

% PDE Solution ------------------------------------------------------------
% Space: second order derivatives
% Time: explicit time marching, i.e, k+1 values are determined by k values
for k=1:M-1  % time step up to specified final time
    for i=2:length(y)-1 % x node step
        for j=2:length(x)-1 % y node step
            % Solving for temperature at next time step
            T(i,j,k+1) = ((((T(i,j+1,k)-2*T(i,j,k)+T(i,j-1,k))/dx^2)....
            +((T(i+1,j,k)-2*T(i,j,k)+T(i-1,j,k))/dy^2))*dt)+T(i,j,k);
        end
    end
end %

% Visualization -----------------------------------------------------------
% Temperature distribution at t = 0
figure()
surf(x,y,T(:,:,1))
view(2)
axis('equal')
title('time=0 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')

% Temperature distribution at t = 0.05
figure()
surf(x,y,T(:,:,250))
view(2)
axis('equal')
title('time=0.05 s')
h = colorbar();
title(h,'Temperature')
```

```
xlabel('X')
ylabel('Y')


% Temperature distribution at t = 0.1
figure()
surf(x,y,T(:,:,500))
view(2)
axis('equal')
title('time=0.10 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')

% Temperature distribution at t = 0.15
figure()
surf(x,y,T(:,:,M))
view(2)
axis('equal')
title('time=0.15 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')
```

Listing 15: Numerical 2D Heat Equation With Mixed BC— Q3_Neumann_BC.m

```
% ---------------------------------------------------------------------------
% This script calculates heat diffusion on a square domain using Mixed Boundary Conditions.
% ---------------------------------------------------------------------------

clear all; clc; close all;

% Parameter definitions --------------------------------------------------
N = 30; % numbers of nodes along both x and y directions
L = 1; % Length of square domain
alpha = 1;  % thermal diffusivity
dt = 0.0002; % time step
dx = L/N; % grid spacing in x direction
dy = dx; % grid spacing in y direction (same as x direction)
finalTime = 0.15; % final time given in the problem
M = int16(finalTime/dt); % number of time steps required

% Initialization ----------------------------------------------------------
% Solution Grid
x=linspace(0,L,N);
y=linspace(0,L,N);
[X,Y]=meshgrid(x,y);
T = ones([size(X) M]); % all time steps are stored for demo purposes

% Initial Conditions
T(:,:,1) = sin(4*pi*X).*cos(4*pi*Y); % at (x,y,t=0)

% Mixed boundary conditions (these could also be applied in the loop)
T(1,:,:) = 1;                                   % at (x,y=0,t)
T(end,:,:) = -1;                                % at (x,y=1,t)



% PDE Solution ------------------------------------------------------------
% Space: second order derivatives
% Time: explicit time marching, i.e, k+1 values are determined by k values
for k=1:M-1  % time step up to specified final time
    for i=2:length(y)-1 % x node step
        for j=2:length(x)-1 % y node step
            % Solving for temperature at next time step
            T(i,j,k+1) = ((((T(i,j+1,k)-2*T(i,j,k)+T(i,j-1,k))/dx^2)....
            +((T(i+1,j,k)-2*T(i,j,k)+T(i-1,j,k))/dy^2))*dt)+T(i,j,k);
        end
    end
    T(1:end,1,k+1) = T(1:end,2,k);
    T(1:end,length(y),k+1) = T(1:end,length(y)-1,k+1) + 2*dx;
end %

% Visualization -----------------------------------------------------------
% Temperature distribution at t = 0
figure()
surf(x,y,T(:,:,1))
view(2)
axis('equal')
title('time=0 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')

% Temperature distribution at t = 0.05
figure()
surf(x,y,T(:,:,250))
view(2)
axis('equal')
title('time=0.05 s')
h = colorbar();
```

```
title(h,'Temperature')
xlabel('X')
ylabel('Y')


% Temperature distribution at t = 0.1
figure()
surf(x,y,T(:,:,500))
view(2)
axis('equal')
title('time=0.10 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')

% Temperature distribution at t = 0.15
figure()
surf(x,y,T(:,:,M))
view(2)
axis('equal')
title('time=0.15 s')
h = colorbar();
title(h,'Temperature')
xlabel('X')
ylabel('Y')
```