MA3650 Assignment

Ally Preston

November 2018

1 Part 1

The logistic equation is a differential equation that calculates the rate in change of population $\dot{X}(t)$ as a function of the current population $X(t)$:

$$\dot{X}(t) = rX(t)\left(1 - \frac{X(t)}{K}\right), \qquad t > 0. \tag{1}$$

The constant r is the proportion of population change per time, as it is the birth minus death rate. When the population is under no external stress, such as competition from another species or group, lack of available resources etc. Where

$$r > 0 \tag{2}$$

means an overall population increase and

$$r < 0 \tag{3}$$

means an overall population decrease.
The constant K is the carrying capacity of the location. If the population is below the carrying capacity then the population will grow, if it exceeds the carrying capacity then the overall population will fall until it reaches the equilibrium K.
We can simplify the Logistic equation (1) into the following form.

$$\frac{\dot{X}(t)}{X(t)(K - X(t))} = \frac{r}{K}. \tag{4}$$

Upon integrating both sides with respect to t we obtain,

$$\int \frac{\dot{X}(t)}{X(t)(K-X(t))}dt = \int \frac{r}{K}dt = \frac{1}{K}\left(\int \frac{\dot{X}(t)}{X(t)} + \int \frac{\dot{X}(t)}{K - X(t)}dt\right) = \frac{1}{K}(\ln|X(t)| + \ln|K - X(t)|) = \frac{rt}{K} + C. \tag{5}$$

Which simplifies to

$$\frac{X(t)}{(K - X(t))} = Ae^{rt} \quad \text{or} \quad X(t) = \frac{AKe^{rt}}{1 + Ae^{rt}}, \qquad A = e^{C}. \tag{6}$$

We can then find our value of A in terms of our initial value X_0

$$A = \frac{X_0}{K - X_0}. \tag{7}$$

This then gives us the solution for $X(t)$ with initial value X_0,

$$X(t) = \frac{X_0 K e^{rt}}{K - X_0 + X_0 e^{rt}}. \tag{8}$$

If we plot the graph for the Logistic equation in MATLAB for different starting values of X_0 for a $r = 0.05$ and $K = 20$ all the lines converge to the carrying capacity at their own rate. The output for the MATLAB script is shown in Figure 1.

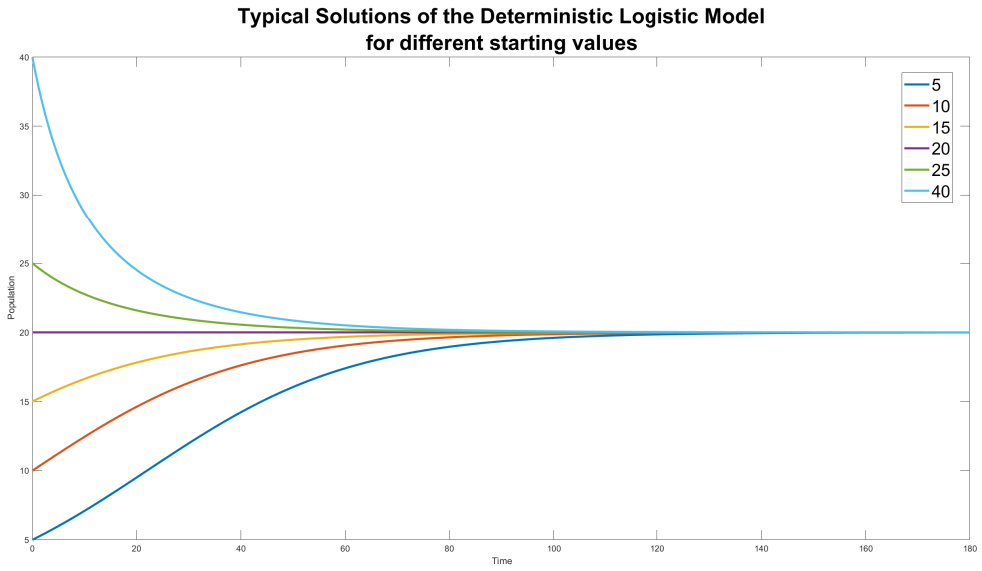


Figure 1: Simulations for the deterministic equation with varying starting values

We can improve the simulation of the equation with the form

$$d\dot{X}(t) = rX(t)\left(1 - \frac{X(t)}{K}\right)dt + \sigma X(t)dW, \qquad t > 0, \tag{9}$$

where $dW = \varepsilon\sqrt{dt}$, $\varepsilon \in N(0,\sigma)$ and σ is the volatility of our population. This allows us to simulate subtle changes in our environment variables. The effects of different values of σ are shown in Figures 2 and 3 for time periods of 90 and 180 respectively. Both graphs have the initial conditions $r = 0.05,\ K = 20$ and $X_0 = 5$.

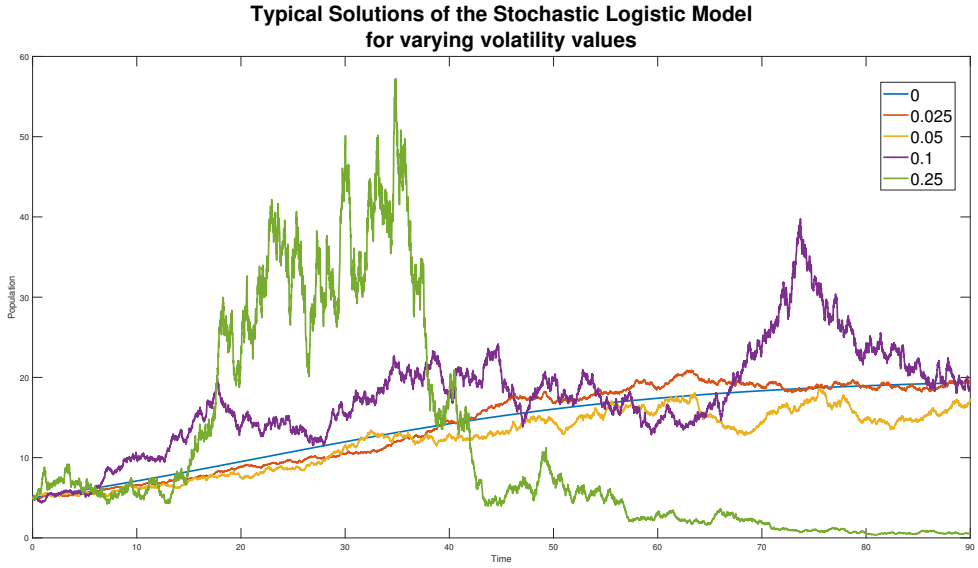


Figure 2: Solutions for the Stochastic equation with time of approximately 3 months

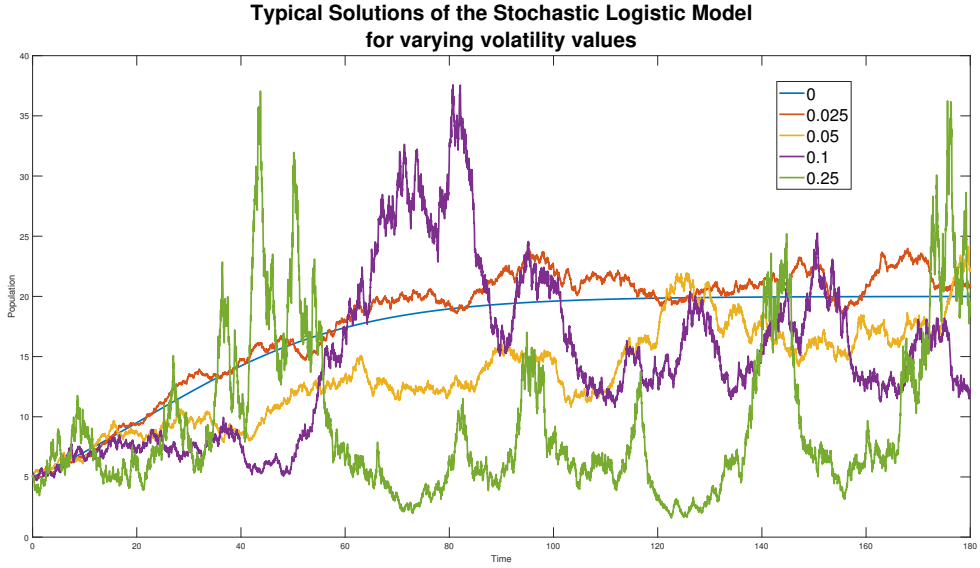


Figure 3: Solutions for the Stochastic equation with time of approximately 6 months

2 Part 2

For this part we are looking at MATLAB simulations of the transport and 2D heat equations.

2.1 Question 1

For this question we look at a MATLAB function transport which runs a simulation of the transport equation,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \tag{10}$$

on the domain $\mathbb{R} \times [0, \infty]$ with initial conditions for the length of the system (X_{min} and X_{max} respectively).We also specify the upper limit for time with t_{max}, our constant c, and the size of the steps we are going to use dt and dx.This code then simulates the movement of something being transported through a plant structure at different times by modelling it as a function of $f(x)$ which is also defined in our initial conditions.
Given the initial condition for the system in the form $u(x,0) = f(x)$, a solution would be in the form

$$u(x,t) = f(x-ct), \tag{11}$$

because,

$$\frac{\partial u}{\partial t} = -cf'(x-ct), \frac{\partial u}{\partial x} = f'(x-ct). \tag{12}$$

Upon substituting this into equation (1) we get

$$[-cf'(x-ct)] + c[f'(x-ct)] = 0. \tag{13}$$

All the graphs show 4 plots with the times at 0, $\frac{1}{3}$ of the way through, $\frac{2}{3}$ of the way through and at the end point t_{max}.

For the first simulation I chose to run for a distance of 0 to 6 for a time of 1 with dx and dt being 0.05 and 0.01 respectively. I chose my C to be 2 and using the function $f(x) = sin(x)$.

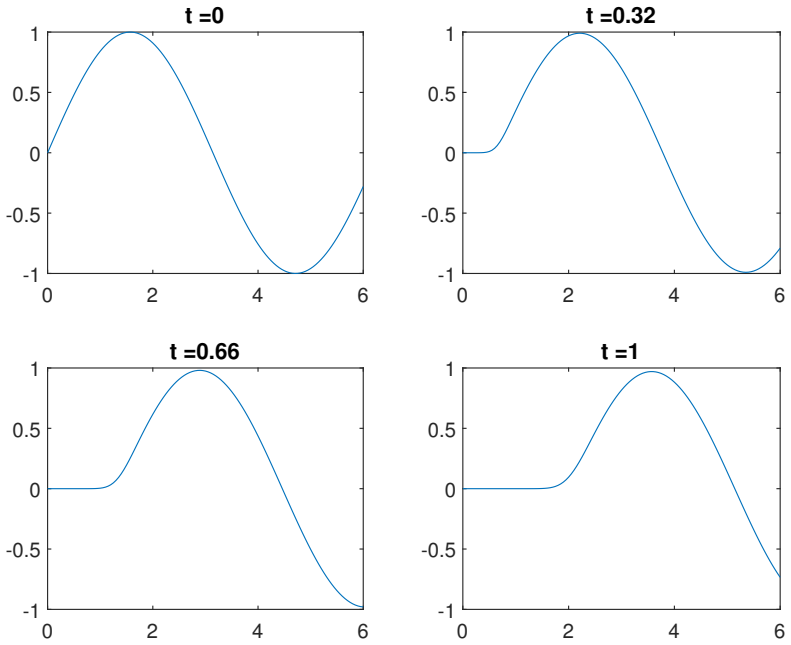


Figure 4: model of transport equation for the simulation using the function f(x) = sin(x)

For the second simulation I chose to run for a smaller range of x from 1 to 2 and t being 0.5 with dx and dt being 0.025 and 0.001 respectively. I then chose c to be 1 and used the function $f(x) = \frac{1}{x}$.

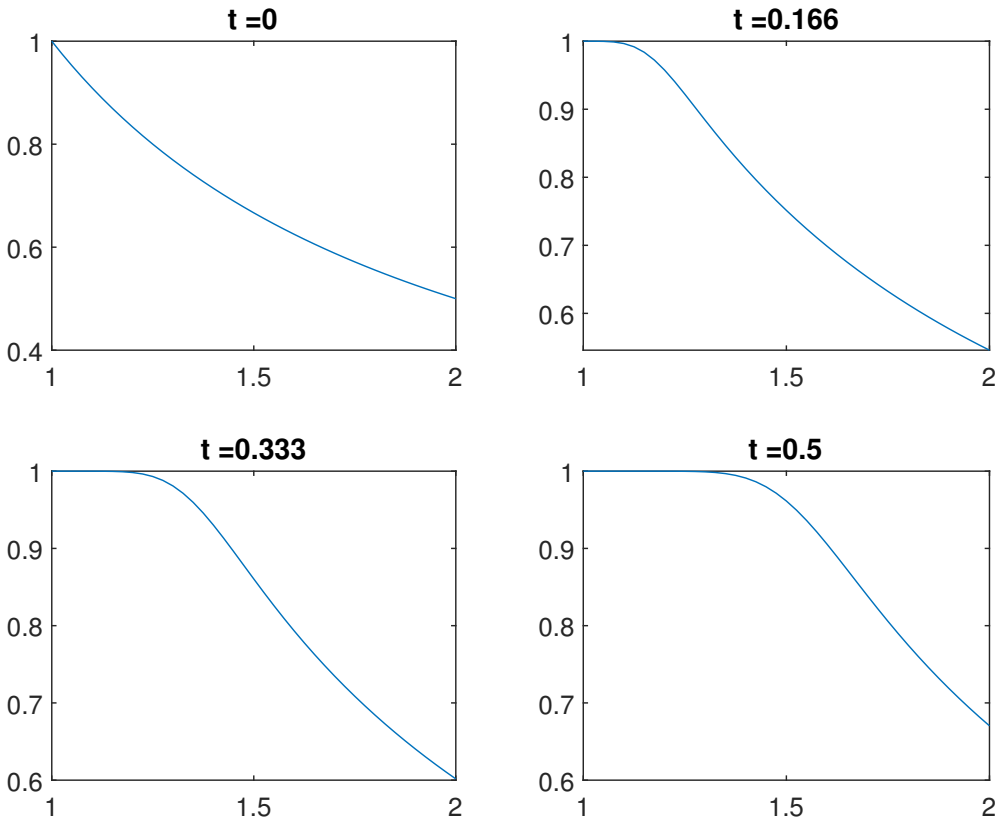


Figure 5: model of transport equation for the simulation using the function $f(x) = \frac{1}{x}$

.

2.2 Question 2

The Heat2D programme is a simulation of a square with side length 1 being heated by solving the differential equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, on the domain of a unit square $[0,1] \times [0,1] \times [0, T_{\max}]$.

2.2.1 1a

We can split the program up into separate stages of input, execution and output phases in the following way.

```
INPUT
function U = Heat2D(dt, dx, dy, Tmax, Tsnap, value, bounds)
%% This section uses the input to calculate the intervals
% make sure steps are consistent
Nx = round(1/dx);
dx = 1/Nx;
Ny = round(1/dy);
dy = 1/Ny;
Nt = round(Tmax/dt);
dt = Tmax/Nt;
rhox = dt/dx^2;
rhoy = dt/dy^2;
%%Just in case our steps are too large
if  rhox + rhoy > 0.5
    fprintf(1,'Warning: bad selection of steps\n');
end
%% sets up initial inputs that will always be constant
C1 = 1-2*rhox-2*rhoy;
EXECUTION
Layers = zeros(2, 1+Nx, 1+Ny);
tpast = 1;
tnow = 2;
iTsnap = Tsnap/dt;
[X, Y] = meshgrid(0:dx:1, 0:dy:1);
% set up initial conditions and plot
Layers(tpast, (1+round(bounds(1)/dx)):(1+round(bounds(2)/dx)), ...
    (1+round(bounds(3)/dy)):(1+round(bounds(4)/dy))) = value;
U = shiftdim(Layers(tpast,:,:));
%% Creates a figure and calls the surf function to make it colour
%coordinated
figure;
surf(X,Y,U);
title('t=0','Fontsize',12);
% Carry out iterations
for t=1:Nt
    for i=2:Nx
        for j=2:Ny
            Layers(tnow,i,j) = C1*Layers(tpast,i,j) + ...
```

```
                rhox*(Layers(tpast,i+1,j) + Layers(tpast,i-1,j)) + ...
                rhoy*(Layers(tpast,i,j+1) + Layers(tpast,i,j-1));
        end
    end
OUTPUT
%% Plot function for the 2D graph style
    % Plot if required
    if find(iTsnap == t)
        U = shiftdim(Layers(tnow,:,:));
        figure;
        surf(X,Y,U);
        title(['t=', num2str(Tsnap(1)) ],'Fontsize',12);
        Tsnap(1) = [];
    end
    % Swap layers
    tnow = 1+mod(t+1,2);
    tpast = 1+mod(t,2);
end
```

2.2.2 1b

We can split the input phase into the separate phases of user inputs and calculated inputs, as the line with function U = Heat2D(dt, dx, dy, Tmax, Tsnap, value, bounds) is the specified user parameters, whereas the other lines in the input phase are calculated from these conditions. The execution phase can be split as well into the iteration phases, in the for loop, and the setup phase where it sets all the variables so the iteration can run.

2.2.3 2

The inputs for the program are the following as shown in the table below.

Variable	Purpose	Range
dt	The step interval in time	Very small, preferably smaller than $\frac{Tmax}{1000}$
dx	The step interval in the x direction	Very small, less that $0.5dt^2$
dy	The step interval in the y direction	Very small, less that $0.5dt^2$
$Tmax$	The maximum value of time we are calculating to	Can be any positive number, but not too large
$Tsnap$	The set of times which will give us outputs	A vector holding numbers that are smaller than $Tmax$
value	The level which we are heating our system to	Any positive number
bounds	The upper and lower bounds for our x and y	A 4 by 1 vector which has elements in the range of 0 to 1

2.2.4 3

The lines in the code detailing the following,

```
Nx = round(1/dx);
dx = 1/Nx;
Ny = round(1/dy);
dy = 1/Ny;
Nt = round(Tmax/dt);
dt = Tmax/Nt.
```

The purpose of these lines is as we chose our values for dx,dy and dt the steps may be difficult to calculate if they cannot be expressed as rational fractions. To minimize this issue we get MATLAB to round the denominator of all the fractions to the nearest whole number, therefore giving us a fraction in the form $\frac{1}{k}, k \in \mathbb{Z}$.

2.2.5 4

The code has 2 lines detailing a relationship between $\frac{dt}{dx^2} + \frac{dt}{dy^2} < 0.5$ if this condition is not met then it will print the message 'Warning: bad selection of steps'. The purpose of this line of code is so that it will inform the user if the steps are too big as for this system to work the combined total of both of these ratios must be less than a half, otherwise the system will misbehave and not compute the difference properly.

2.2.6 5

The line Layers = zeros(2, 1+Nx, 1+Ny); creates an array for storing all our values computed during the loop. It creates a series of 1 + Ny 2 by 1 + Nx matrices in MATLAB which are then used to house our values during calculations. If this was not done our last step will be different as the system will make sure it ends at t_{max} or the upper limits of bounds.

2.2.7 6

The lines detailing tpast = 1 and tnow = 2 define our original starting point for t_0 and t_1 respectively, we have to start at 1 instead of 0 because MATLAB arrays start with an index of 1.
The line iTsnap = Tsnap/dt; tells us how many steps of dt we will need to get to our values of T we were wanting to output as graphs. The line [X, Y] = meshgrid(0:dx:1, 0:dy:1); sets up a Matrix which will simulate our unit square with all points for the steps of dx and dy filled in. This is then used in our plotting function surf

2.2.8 7

The lines,

```
Layers(tpast, (1+round(bounds(1)/dx)):(1+round(bounds(2)/dx)), ...
(1+round(bounds(3)/dy)):(1+round(bounds(4)/dy))) = value;
U = shiftdim(Layers(tpast,:,:));
figure; surf(X,Y,U); title('t=0','Fontsize',12);
```

define values into our set up layer array. For values that are within our bounds in the initial parameters it sets those equal to our value of heat we defined initially. This then is shifted to the left 1 by the line U = ... This is because we had to add 1 to all our indices when fetching the values as our values could start from the 0 but MATLAB arrays start with index 1. So once we have shifted everything to how the simulation should run we create a figure and plot the values on it in 3D using the surf function.

2.2.9 8

The main execution phase of the program is in the following lines,

```
for i=2:Nx
    for j=2:Ny
        Layers(tnow,i,j) = C1*Layers(tpast,i,j) + ...
        rhox*(Layers(tpast,i+1,j) + Layers(tpast,i-1,j)) + ...
        rhoy*(Layers(tpast,i,j+1) + Layers(tpast,i,j-1));
    end
end.
```

The purpose of these lines is to compute the value of u for our system using our previous values of t, x and y. We can arrive at these values by expressing $\frac{\partial u}{\partial t}$ using forward difference, and $\frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}$ using the second order difference. We want to rewrite this equation using $\rho_x = \frac{dt}{dx^2}$, $\rho_y = \frac{dt}{dy^2}$ and $C1 = 1 - 2\rho_x - 2\rho_y$ we calculated earlier in the code.

$$\frac{v_{i,j,k+1} - v_{i,j,k}}{dt} = \frac{v_{i+1,j,k} + v_{i-1,j,k} - 2v_{i,j,k}}{(dx)^2} + \frac{v_{i,j+1,k} + v_{i,j-1,k} - 2v_{i,j,k}}{} \tag{14}$$

$$v_{i,j,k+1} = v_{i,j,k} + \rho_x(v_{i+1,j,k} + v_{i-1,j,k} - 2v_{i,j,k}) + \rho_y(v_{i,j+1,k} + v_{i,j-1,k} - 2v_{i,j,k}) \tag{15}$$

$$v_{i,j,k+1} = v_{i,j,k} - 2\rho_x v_{i,j,k} - 2\rho_y v_{i,j,k} + 2\rho_x(v_{i+1,j,k} + v_{i-1,j,k}) + \rho_y(v_{i,j+1,k} + v_{i,j-1,k}) \tag{16}$$

$$v_{i,j,k+1} = C1 v_{i,j,k} + 2\rho_x(v_{i+1,j,k} + v_{i-1,j,k}) + \rho_y(v_{i,j+1,k} + v_{i,j-1,k}). \tag{17}$$

This the expresses our new calculated value of $v_{i,j,k+1}$. These variables are represented by the layers(t,x,y) variables in our code. This process is repeated for all values in both the x and y directions.

2.2.10 9

The lines,

```
if find(iTsnap == t)
    U = shiftdim(Layers(tnow,:,:));
    figure; surf(X,Y,U);
    title(['t=', num2str(Tsnap(1)) ],'Fontsize',12);
    Tsnap(1) = [];
end
```

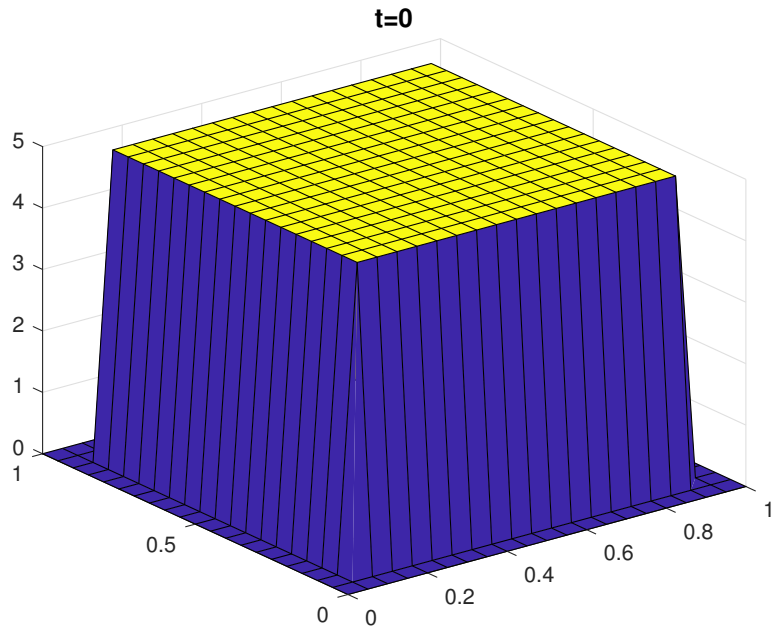
are used for the plotting our function when our function is at the values we wanted to see Tsnap. If the number of interations we have completed, t, is equal to the number of steps we required to reach a Tsnap value, iTsnap, then it runs our plotting function similar to how it was run in question 7. The code then deletes the entry for Tsnap to indicate that the value has been plotted.

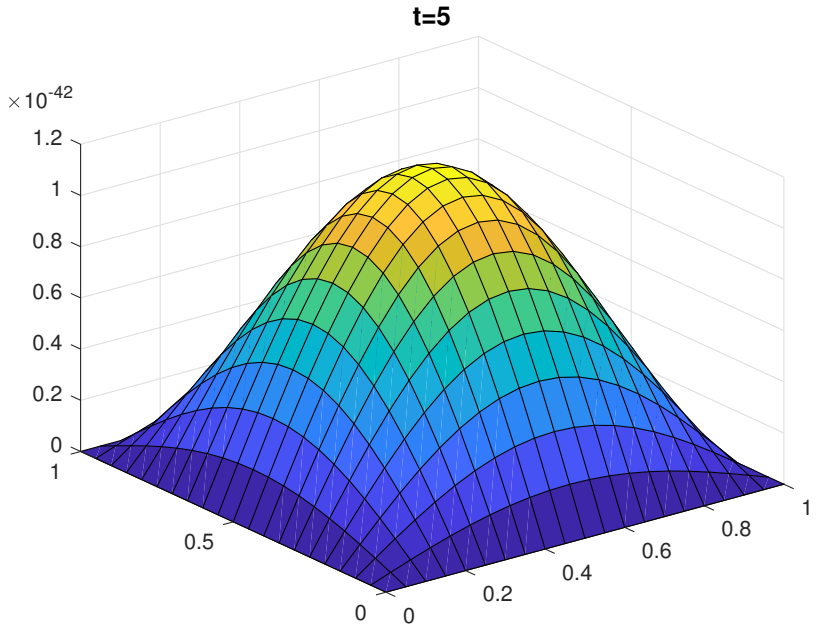
2.2.11 10

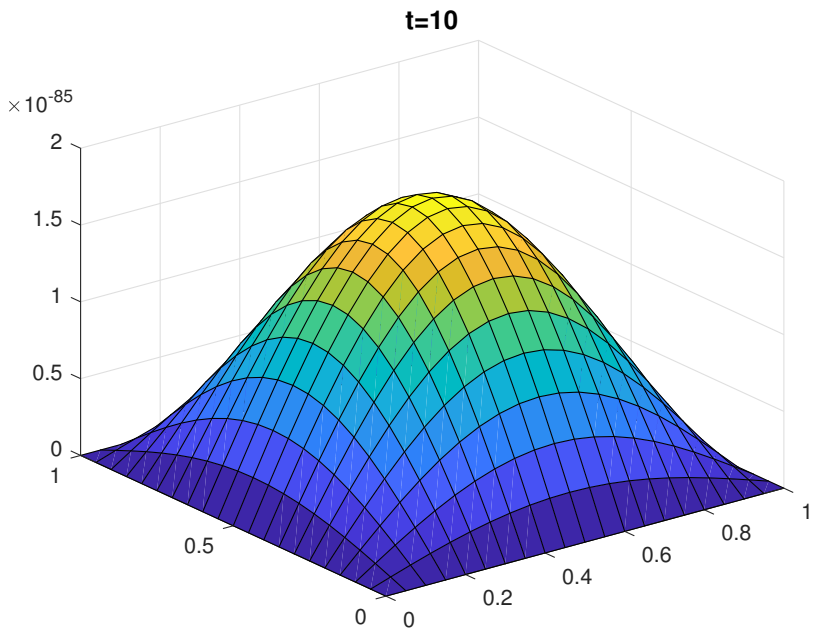
The loop for t=1:Nt runs through and calculates the new values for the next step in time using the method detailed in part 7. I then goes through and fills in the layer. If it is one of the values we would like to see, detailed in our inputs in Tsnap it then plots and displays the outputs.
The lines tnow = 1+mod(t+1,2); tpast = 1+mod(t,2); essentially switch round the values of tnow and tpast, because the code only has set up the storage of 2 layers each iteration rewrites the layer that was used previously to save space. This then means we either have to go up in the layers, or down in the layer depending on which one is being overwritten.

2.3 Question 3

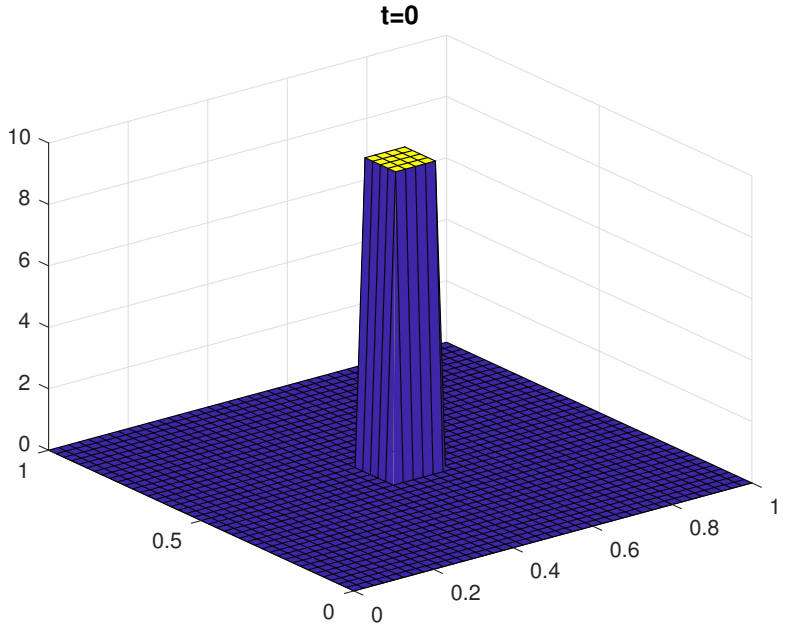
For this question we are running a code for two different initial conditions, for the first run I am running for a unit square sides from 0.1 to 0.9 being heated. The intervals are dt = 0.0001; dx = 0.05; dy = 0.05; Tmax = 10. We will get outputs for time at 0, 5, 10 and using a heating value of 5. We then obtain the following graphs shown below.

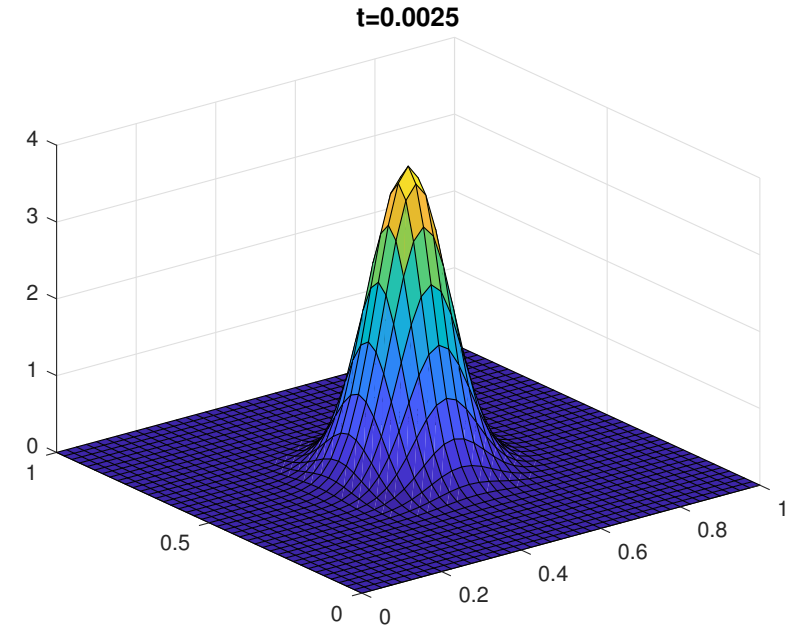


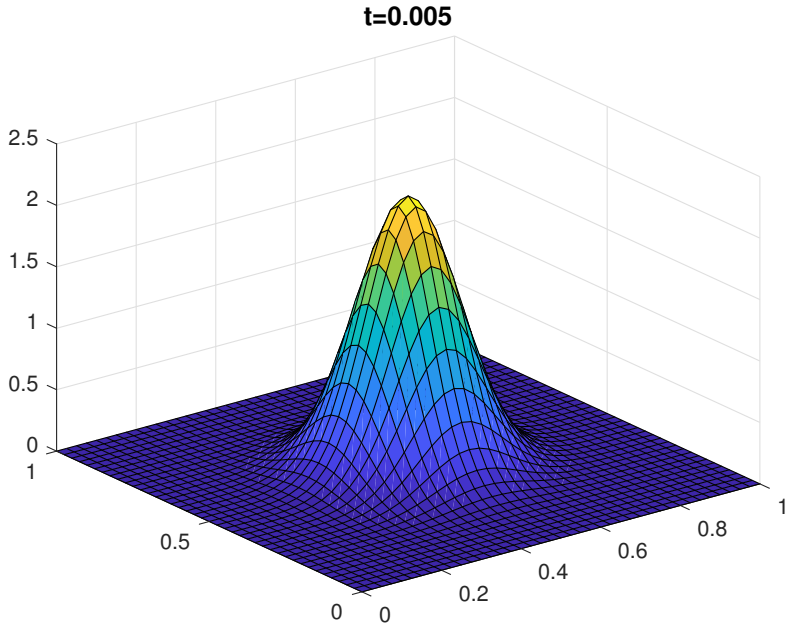


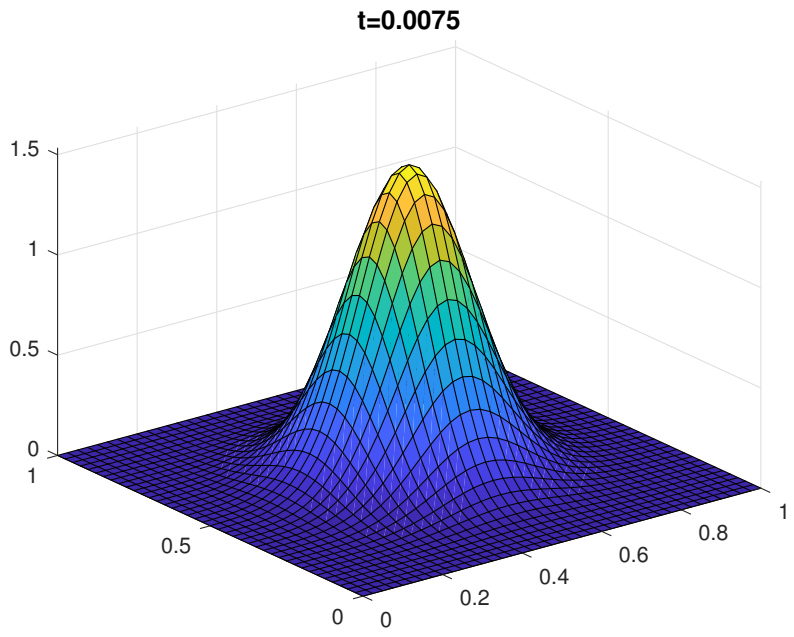


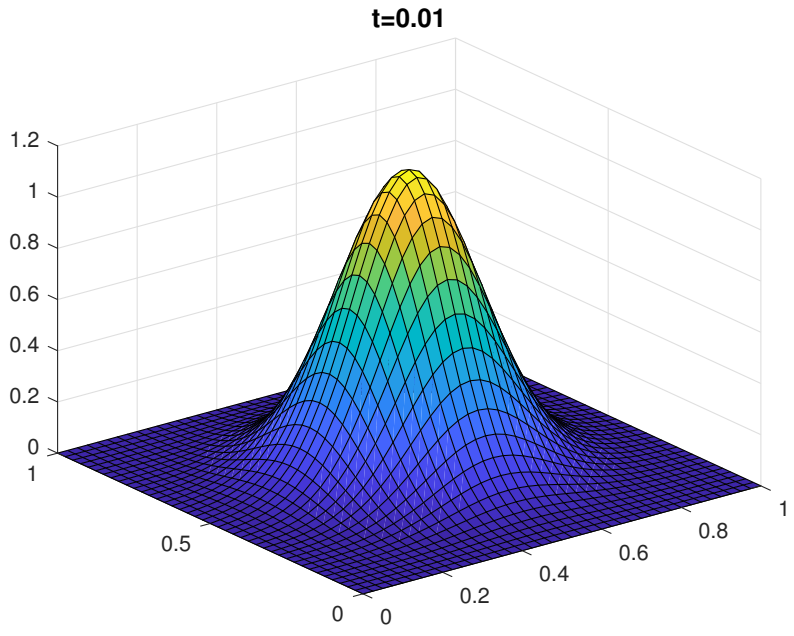
For the second run I decided to run for a smaller initial range for x and y between 0.45 and 0.55, then also using a much smaller interval of time using intervals of 0.0025, 0.005, 0.0075 and 0.01 as Tmax. The value for our steps are increased as well to dt = 0.00005;dx = 0.025; dy = 0.025. The initial heating value will be increased to 10.




t=0.0025
4
3
2
1
0
1
0.5
0
0
0.2
0.4
0.6
0.8
1


t=0.005
2.5
2
1.5
1
0.5
0
1
0.5
0
0
0.2
0.4
0.6
0.8
1


t=0.0075
1.5
1
0.5
0
1
0.5
0
0
0.2
0.4
0.6
0.8
1


t=0.01
1.2
1
0.8
0.6
0.4
0.2
0
1
0.5
0
0
0.2
0.4
0.6
0.8
1

2.4 Additional Questions

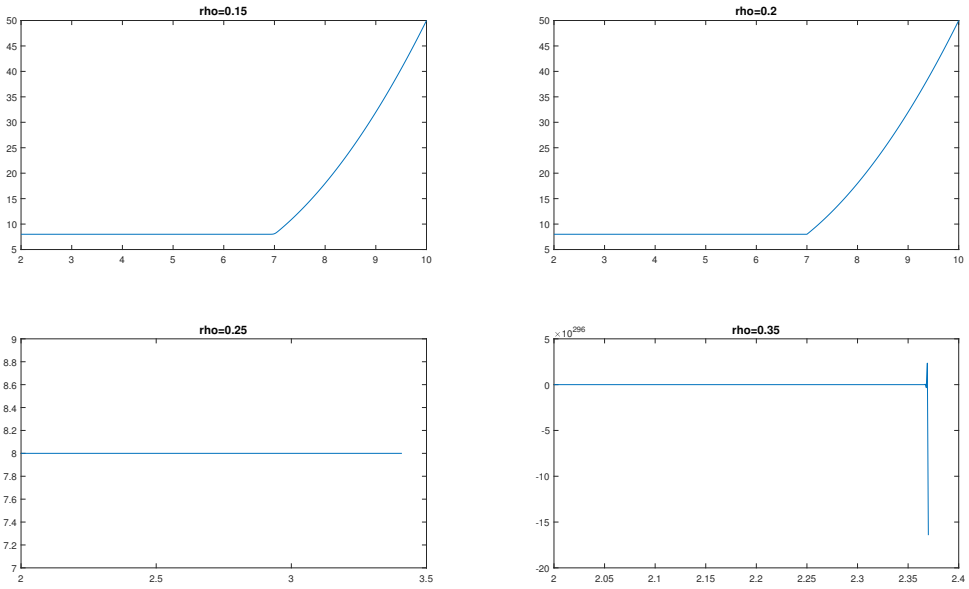
2.4.1 1

For the advection equation we have the constant $\rho = \frac{dt}{dx}$, which is used in our calculations phases. We needed $\rho \leq \frac{1}{c}$ for our scheme to be stable, this can be shown using the following MATLAB script.

```
    xmin = 2;
xmax = 10;
tmax = 1;
c = 5;
f0 = @(x) 2*x.^2;
Vdx = values to change
Vdt = values to change
count = 1;
for loop1 = 1:length(Vdx)
    for loop2 = 1:length(Vdt)
        dx = Vdx(loop1);
        dt = Vdt(loop2);
        rho = dt/dx;
        N = ceil((xmax - xmin) / dx);
        xmax = xmin + N*dx; %makes sure our steps will take us from
        the minimum to maximum values for x exactly
        M = ceil(tmax/dt); %makes sure that the number of steps will
        exactly take us to the maximum time
        k1 = 1 - rho*c;
        k2 = rho*c;
        %uses forward difference for t and backward difference for x
        solution = zeros(N+1,M+1); % a vector to store all our answer outputs
        vetx = xmin:dx:xmax;
        % a vector for all of our x values whilst plotting
        for i=1:N+1 %for all of our values of x
            solution(i,1) = feval(f0,vetx(i)); %calculate the initial
            conditions for our first row
        end
        fixedvalue = solution(1,1);
        %first value is value we use to calculate the values from
        % this is needed because of finite domain
        for j=1:M %for the interval for time
        solution(:,j+1) = k1*solution(:,j) +
        k2*[ fixedvalue ; solution(1:N,j)];
        %uses a form of the forward difference using our earlier ratios
        end
        time = floor(length(solution(1,:)));
        %sets the time we are plotting to be the end of the solution
        % TransportPlot(xmin, dx, xmax, times, solution) %calls the function
        transport plot to plot the different times in figures
        %imported the entirety of the function into this code for simplicity
        %editting
```

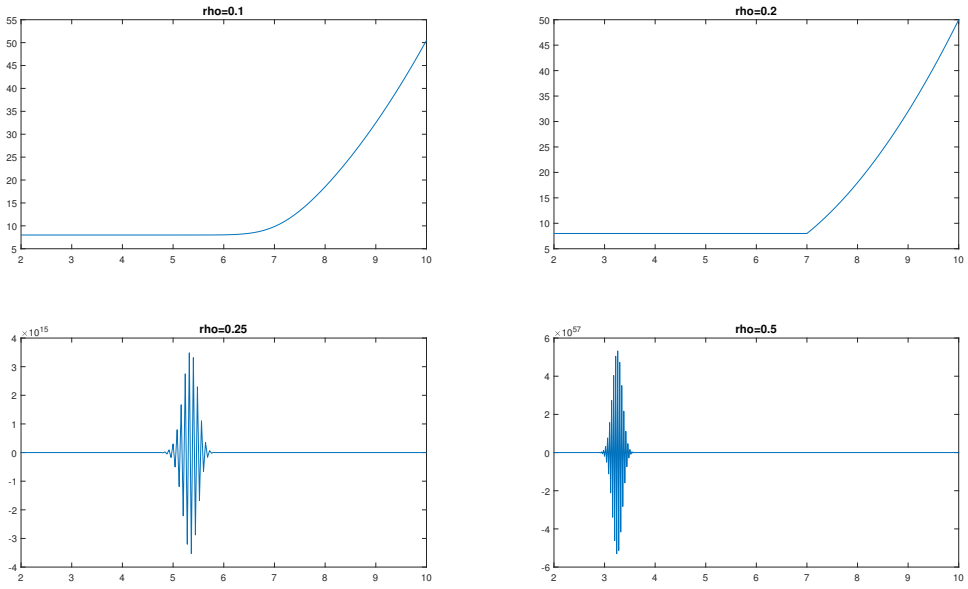
```
        %plots the results for varying amounts of time on seperate subplots
        subplot(2,2,count)
        plot(xmin:dx:xmax, solution(:,time))
        %axis([xmin xmax min(solution(:,time)) max(solution(:,times(4)))])
        %sets the axis to fit the range
        %axis lines are commented out because automatic settings for matlab
        %actually are preferable
        title(['rho=', num2str(rho) ],'Fontsize',12);
        count = count+1;
    end
end
```

This code runs a scheme from $x \in [2,10], t_{max} = 1$ and $c = 5$ with the function $f(x) = 2x^2$. We then can alter our values of dx and dt to vary our ρ. Using the values of $dx = [0.001]$ for the first run and using the values of $dt = [0.00015, 0.0002, 0.00025, 0.00035]$ we obtain the following plots.



We can see that the function works where $\rho \leq \frac{1}{c}$ as shown in the first and second plots. However when this is not met the system breaks down and completely goes wrong, giving the third and fourth plots.

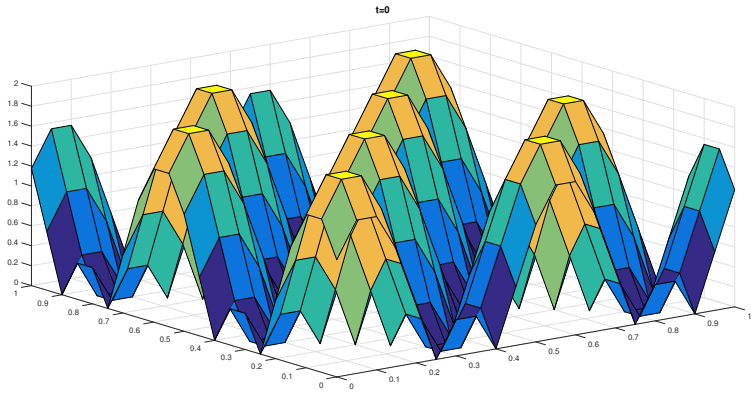
We can also obtain different values of ρ by changing the values of dx in the same way, using the same initial conditions but this time having $dt = 0.01$ and $dx = [0.1, 0.05, 0.04, 0.02]$

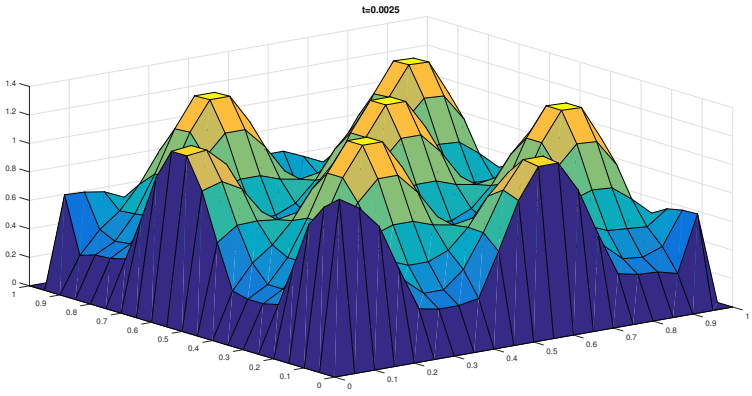


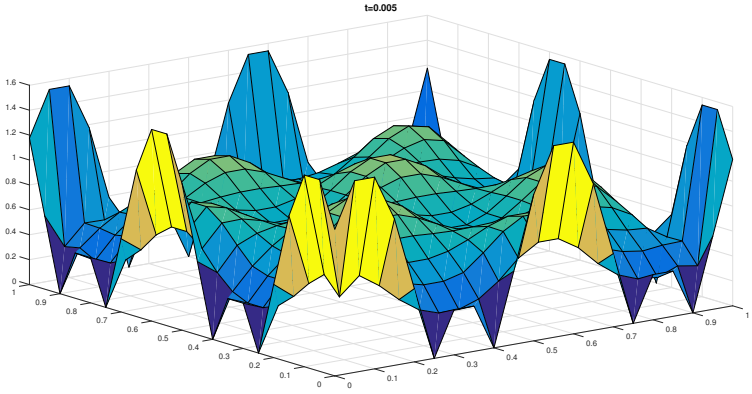
Which again obtains similar results. The third and fourth plots are less 'broken' if still not correct solutions to the system.

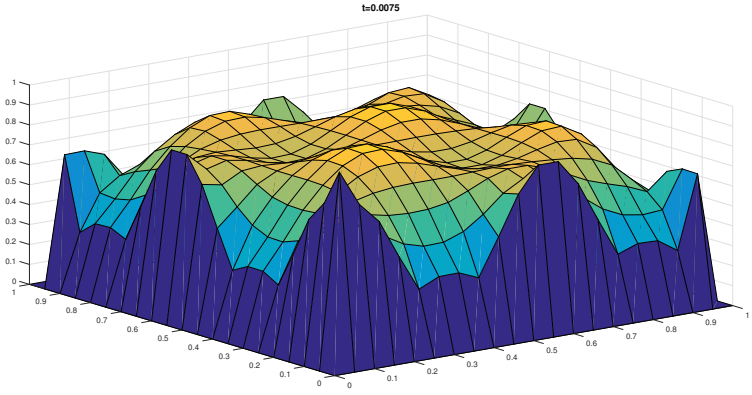
2.4.2 3

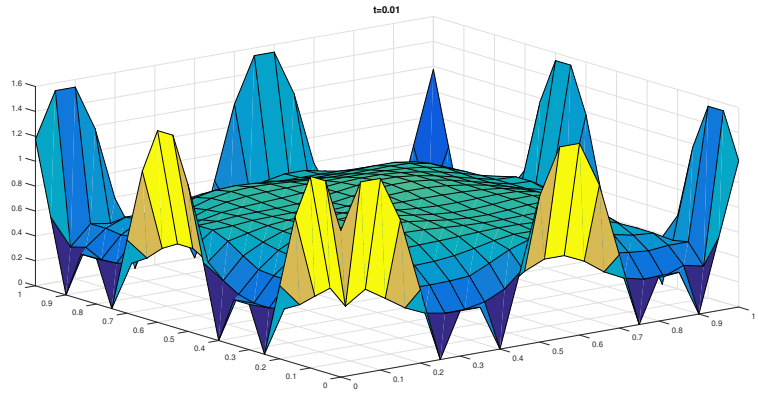
For this exercise I will be considering an extension of the 2D heat equation where $U_0(x) \neq C$ where $C \in \mathbb{R}$ so therefore it will be an explicit function of x and y. I will be using the function $u(x,y) = |\sin 4\pi x + \cos 4\pi y|$ for my modeling in this question. For the first step in this problem I modified the Heat2D program to run for a function $u(x,y)$ instead of a single value, this was then not set to repeat at the start of each iteration as the original program did, as there is no range which . I the ran the program using the values of $dt = 0.0005, dx, dt = 0.05, T_{max} = 0.01$ and the bounds being the edge of our system. This then yields the results shown in the following graphs below, showing the heat levels at intervals through the range.




t=0.0025


t=0.005


t=0.0075



As we can see the heat 'falls' from the peaks and raises the overall temperature, as seen from the the evolution of the graphs as time increases, as this is only a small set of time we can assume the system would become uniformly hot over the course of the model.

3 Part 3

For this section we will be looking at the pattern of butterfly wings influenced by the presence of two chemicals. We have the initial equations as the following.

$$u_t = \gamma\Big(a - u - h(u,v)\Big) + u_{xx} + u_{yy}, \tag{18}$$

$$v_t = \gamma\Big(\alpha(b-v) - h(u,v)\Big) + d(v_{xx} + v_{yy}). \tag{19}$$

Where $h(u,v) = \frac{\rho uv}{1+u+Ku^2}$. Acting within a rectangular domain $x \in [0,1]$ and $y \in [0,Y]$ where Y is a constant. We have the boundary conditions along the edge of the rectangle,

$$u_x(0,y) = u_x(1,y) = 0 = u_y(x,0) = u_y(x,Y).$$

We can find constant solutions for $u = \hat{u}$ and $v = \hat{v}$ in terms of our parameter constants. As $\hat{u}$ and $\hat{v}$ are constants their derivatives are 0, therefore our equations can simplify down. Working with equations (18) and (19) we get the following,

$$a - \hat{u} - h(u,v) = 0, \tag{20}$$

and

$$\alpha(b-\hat{v}) - h(u,v) = 0. \tag{21}$$

By eliminating $h(u,v)$ we get an equation in terms of u and v only,

$$a - \hat{u} = \alpha(b-\hat{v}), \tag{22}$$

leading to making $\hat{v}$ the subject,

$$\hat{v} = \frac{\hat{u}-a}{\alpha} + b. \tag{23}$$

We can then put this into equation (20) to eliminate $\hat{v}$

$$a - \hat{u} - \frac{\rho\hat{u}(\frac{\hat{u}-a}{\alpha} + b)}{1+\hat{u}+K\hat{u}^2} = 0. \tag{24}$$

Which can then be rearranged to give the following cubic equation,

$$\hat{u}^3\alpha k + \hat{u}^2(\rho + \alpha - \alpha a k) + \hat{u}(\alpha - \alpha a - \rho - a\rho ab) - \alpha a = 0 \tag{25}$$

Which we can solve using the constants for our system for $\alpha, \gamma, \rho, a, b, d$ and k. For my model I will be choosing them to have the values summarised in the table below.

α	γ	ρ	a	b	d	k
$\frac{3}{7}$	0.1	$\frac{1}{5}$	$\frac{24}{7}$	6	80	$\frac{1}{3}$

This can then be substitued into (25) and solved to obtain $\hat{u} = 3$ and then use equation (22) to obtain $\hat{v} = 5$.

Whilst we are modeling our system for this part the areas of the concentration of the chemical u will yield one of 4 colours. We will be assuming that the intervals are in the regions of

$u < \frac{\hat{u}}{2}$, $\frac{\hat{u}}{2} < u < \hat{u}$, $\hat{u} < u < \frac{3\hat{u}}{2}$ and $\frac{3\hat{u}}{2} < u$. Each of these regions will cause a different colour in our system.

I will be modelling the scheme using the forward differences for both u_t and v_t, as well as using the second order differences for u_{xx}, u_{yy}, v_{xx} and v_{yy}.

This then leads to the implicit scheme for equation (18) being in the form,

$$u_{i,j,t+1} = \gamma dt(a - u_{i,j,t} - h(u_{i,j,t}, v_{i,j,t})) + \frac{dt}{(dx)^2}(u_{i+1,j,t} + u_{i-1,j,t}) + \dots$$
$$\frac{dt}{(dy)^2}(u_{i,j+1,t} + u_{i,j-1,t}) + u_{i,j,t}(1 - 2\frac{dt}{(dx)^2} - 2\frac{dt}{(dy)^2}),$$

and equations (19) in the form,

$$v_{i,j,t+1} = \gamma dt(\alpha(b - v_{i,j,t}) - h(u_{i,j,t}, v_{i,j,t})) + d\frac{dt}{(dx)^2}(v_{i+1,j,t} + v_{i-1,j,t}) + \dots$$
$$d\frac{dt}{(dy)^2}(v_{i,j+1,t} + v_{i,j-1,t}) + v_{i,j,t}(1 - 2d\frac{dt}{(dx)^2} - 2d\frac{dt}{(dy)^2}).$$

Where we can choose our step sizes to give a stable system, in my code I will be using $dx = 0.05, dy = 0.05$ and $dt = 0.00025$. We can then implement this scheme into MATLAB and obtain an approximation for the concentrations of chemical u and chemical v at a specified time t using the constants defined earlier. Which then can be plotted at the final time to show us the concentration of the chemicals. We will then collect the regions and show definitive regions for each colour using the regions we defined earlier. This is then done using the following stages of code.

```
cx = dt/dx^2;
cy = dt/dy^2; %sets up our ratios for later

if cx + cy > 0.5
    disp('Warning, bad step sizes')
end
```

To check whether our step size in our system is stable.

```
 numbert = tmax/dt;
numberx = 1/dx;
numbery = y/dy; %sets up number of steps for t, x and y

Lu = 0.75*ones(1+numberx, 1+numbery, 2);
Lv = 0.75*ones(1+numberx, 1+numbery, 2);
[X, Y] = meshgrid(0:dx:1, 0:dy,y);
```

To set up our layers with appropriate step sizes for our system for our calculation.

```
for k = 1:numbert
for i = 2:numberx
    for j = 2:numbery
        huv = h(Lu(i,j,1),Lv(i,j,1), rho, K);
```

```
            Lu(i,j, 2) = Lu(i,j,1)*(1-2*cx - 2*cy) + cx*(Lu(i+1,j,1)
            + Lu(i-1,j,1)) + cy*(Lu(i,j+1,1) + Lu(i,j-1,1)) +
            dt*gamma*(a - Lu(i,j,1) - huv);
            Lv(i,j, 2) = Lv(i,j,1)*(1-2*d*cx - 2*d*cy) + d*cx*(Lv(i+1,j,1)
            + Lv(i-1,j,1)) + d*cy*(Lv(i,j+1,1) + Lv(i,j-1,1)) +
            dt*gamma*(alpha*(b-Lv(i,j,1)) - huv);
                Lu(1,j,2) = Lu(2,j,1);
                Lu(1+numberx,j,2) = Lu(numberx, j,1);
                Lv(1,j,2) = Lv(2,j,1);
                Lv(1+numberx,j,2) = Lv(numberx, j,1);
        end
        Lu(i,1,2) = Lu(i,2,1);
        Lu(i,1+numbery,2) = Lu(i,numbery,1);
        Lv(i,1,2) = Lv(i,2,1);
        Lv(i,1+numbery,2) = Lv(i,numbery,1);
    end
    Lu(1,1,2) = Lu(2,2,1);
    Lu(1+numberx,1+numbery,2) = Lu(numberx,numbery,1);
    Lu(1,1+numbery,2) = Lu(2,numbery,1);
    Lu(1+numberx,1,2) = Lu(numberx,2,1);
    Lv(1,1,2) = Lv(2,2,1);
    Lv(1+numberx,1+numbery,2) = Lv(numberx,numbery,1);
    Lv(1,1+numbery,2) = Lv(2,numbery,1);
    Lv(1+numberx,1,2) = Lv(numberx,2,1);
    Lu(:,:,1) = Lu(:,:,2);
    Lv(:,:,1) = Lv(:,:,2);
end

function huv = h(u,v,rho,k)
huv = (rho*u*v)/(1+u+k*u^2);
end
```

Which runs our iterations for our equations, as well as fixing our boundary conditions so that they have a derivative of zero as specified in our boundary conditions.
After this we can then assign specific values to our matrix Lu to be used in the plotting by assigning specific values to each entry depending on what range it is in.

```
    for loop = 1:numel(final)
    if final(loop) < uhat/2
        final(loop) = 1;
    elseif final(loop) < uhat
        final(loop) = 2;
    elseif final(loop) < 1.5*uhat
        final(loop) = 3;
    elseif final(loop) > 1.5*uhat
        final(loop) = 4;
    end
end
```

After this we use the function imagesc(final) to display the final resultant pattern for the system. For the variables stated earlier the wing pattern shown below is obtained for $Y = 2$ and $Y = 3$.

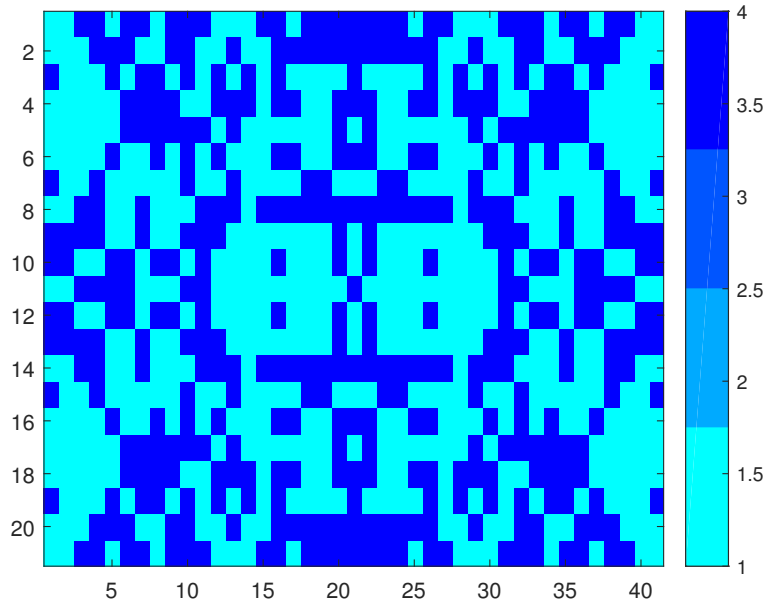


Figure 6: Butterfly wing produced by using our mathematical model when Y = 2

In both both of the images the darker regions are the higher concentrations of the chemical u and lighter colour is a lower concentration. There unfortunately is no intermediate range of the chemical due to the system being very sensitive and only demonstrating patterns with more than two colours for specific values of d and γ, however I could not find any such patterns from my system.

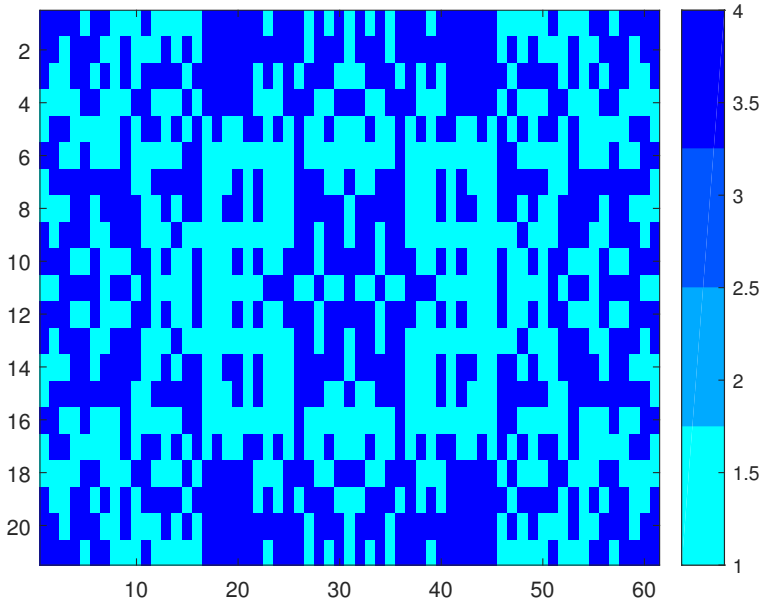


Figure 7: Butterfly wing produced by using our mathematical model when Y = 3