Brief Theoretical & Methodical Foundations

For this project we will be working with several PDEs. We will be working with the Diffusion Equation, Advection Equation, Convection-Diffusion Equation, and the Viscous Burgers Equation. In which we solve some of them using explicit method or Implicit methods. There are boundary and initial conditions that are given for each equation. Then depending on how we need to treat the problem we will need to figure out how to discretize the problem. For example, if the problem gives you Dirichlet conditions versus Neumann conditions than your Δx will be different for each one. Then you will decide which numerical method to implicit and also which discretization.

$$Diffusion: u_t = u_{xx}$$
$$Advection: u_t + au_x = 0$$
$$Convection - Diffusion: u_t + au_x = du_{xx}$$
$$Viscous\ Burgers: u_t + uu_x = du_{xx}$$

Then to discretize in terms of space we use the 2nd Order central difference discretization and also the first order which results in us getting the $T_{\Delta x}$ and $S_{\Delta x}$ matrices which will be used to update our solution. Not only this but we use such methods as the Explicit Euler

$$u^{m+1} = u^m + \Delta t T_{\Delta x} u^m$$

which is based off of a semi-discretization: meaning that we want to approximate this in terms of space and not time. Following the next method is an implicit one that will be discretized in both time and space which is the Crank-Nicolson method

$$u^{m+1} = u^m + \frac{\Delta t}{2}(T_{\Delta x} u^m + T_{\Delta x} u^{m+1}) .$$

Continuing another method that we will use is the Lax-Wendroff method

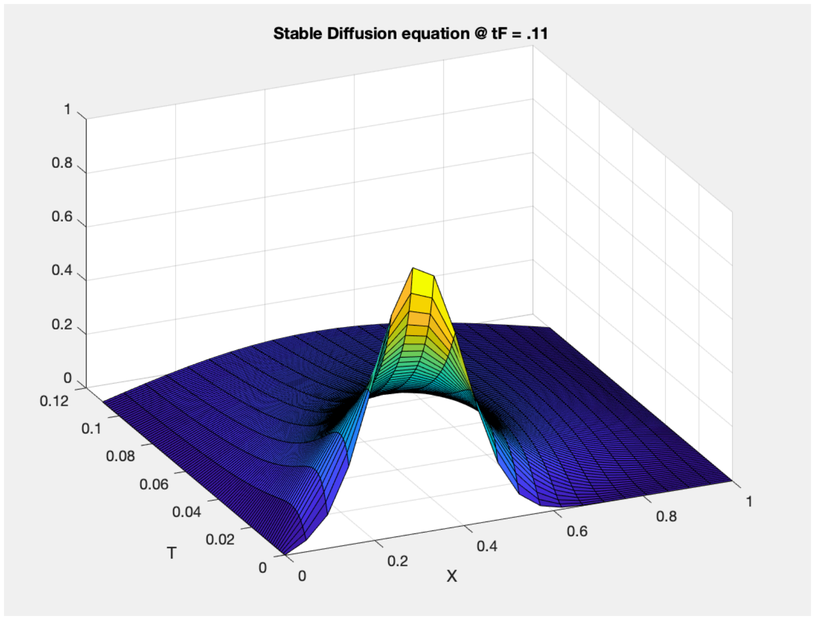
$$u_j{}^{n+1} = \frac{a\mu}{2}(1 + a\mu)u_{j-1}{}^n + (1 - a^2\mu^2)u_j{}^n - \frac{a\mu}{2}(1 - a\mu)u_{j+1}{}^n \text{ where } \mu = \frac{\Delta t}{\Delta x}.$$

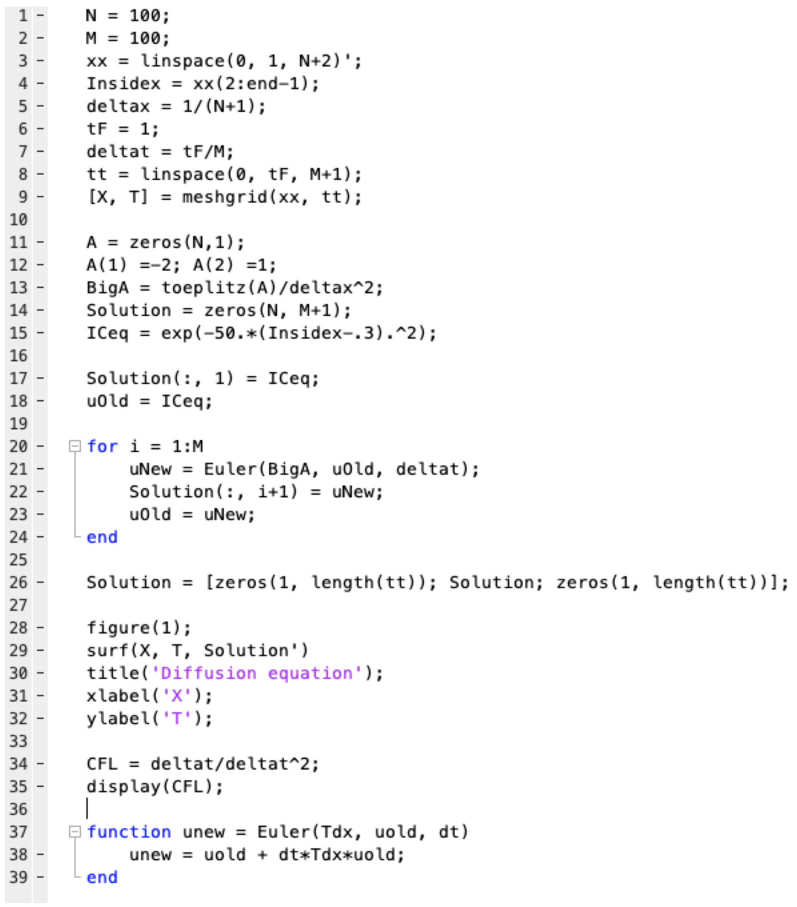
These general methods and knowing the discretization will lead to the use of implementing them for more complex cases which require to do several steps. Another thing to know is the Method of Lines discretization which adds the tridiagonal matrix with the skew-symmetric one which will lead to using the Crank-Nicolson method for more complex problems such as the Viscous Burgers problem.

$$\text{Method of lines: } u' = (dT_{\Delta x} - a\, S_{\Delta x})u$$

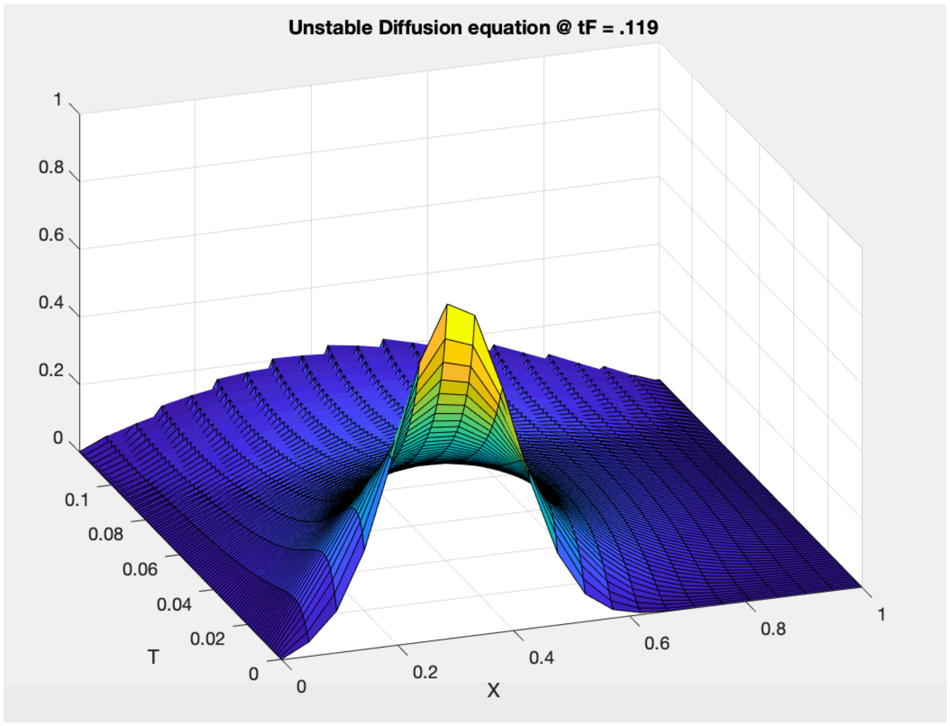
This is the general list of the theory/methods that we will be using for the rest of the project.

Task 1.1





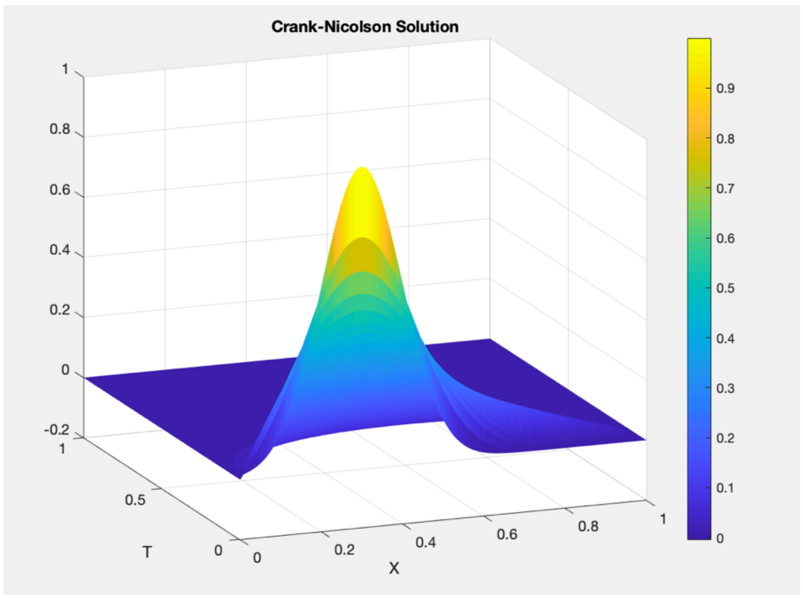
```
N = 100;
M = 100;
xx = linspace(0, 1, N+2)';
Insidex = xx(2:end-1);
deltax = 1/(N+1);
tF = 1;
deltat = tF/M;
tt = linspace(0, tF, M+1);
[X, T] = meshgrid(xx, tt);

A = zeros(N,1);
A(1) =-2; A(2) =1;
BigA = toeplitz(A)/deltax^2;
Solution = zeros(N, M+1);
ICeq = exp(-50.*(Insidex-.3).^2);

Solution(:, 1) = ICeq;
uOld = ICeq;

for i = 1:M
    uNew = Euler(BigA, uOld, deltat);
    Solution(:, i+1) = uNew;
    uOld = uNew;
end

Solution = [zeros(1, length(tt)); Solution; zeros(1, length(tt))];

figure(1);
surf(X, T, Solution')
title('Diffusion equation');
xlabel('X');
ylabel('T');

CFL = deltat/deltat^2;
display(CFL);

function unew = Euler(Tdx, uold, dt)
    unew = uold + dt*Tdx*uold;
end
```

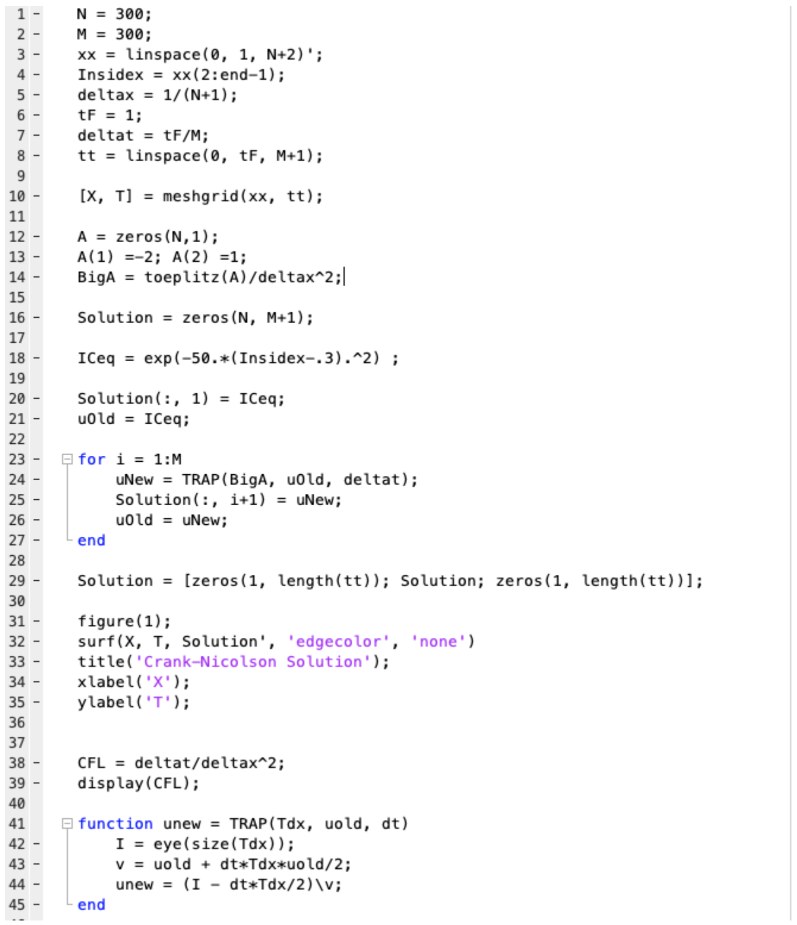


For the Diffusion equation problem here, I decided to use the equation $g(x) = e^{-50(x-.3)^2}$ with space interval [0, tF]. I decided to use $N = 20$ and $M = 100$ where M is the number of time steps and N is the number of mesh points on the space interval. When I decided to use higher values such as $N = 200$ and $M = 200$ the values would cause an error. Then, using the conditions that I provided I received CFL of .5248 for tF of .119 and CFL of .4851 for tF of .11 which shows one case under the limit of .5 and one over it to show the difference of stability vs instability. Thus, we are able to see how stability changes by altering the end points of time. From the graphs we can see how the solution is still stable at tF = .11 but then the condition breaks for when tF = .119. From my code you can see how I implemented the semi-discretization. I set up my 2nd Order Central Difference matrix as BigA, I then used the Euler method to update my matrix through the Euler function. Messing around a bit with the N and M values, I noticed that as M increases you get a somewhat more spread out graph of oscillations but in order to do so, I had to lower my N value.

Task 1.2

Now implementing the Crank-Nicolson method to the same problem. I noticed the difference in methods instantaneously. Not just by looking at the graph but also because with the semi-discretization Euler method, I was not able to input N or M greater than 200 but with this method I was able to go to at least 800 for both N and M. Now using the same problem and also the same interval but with $N = 300$ and $M = 300$ I received a CFL of 302.0033. Now comparing both graphs together you can see how the Crank-Nicolson gives more information versus the Euler Method graph as it is a better method to showcase this. You receive here more solutions on x as you are able to implement more time steps and mesh points resulting in a more complete solution.

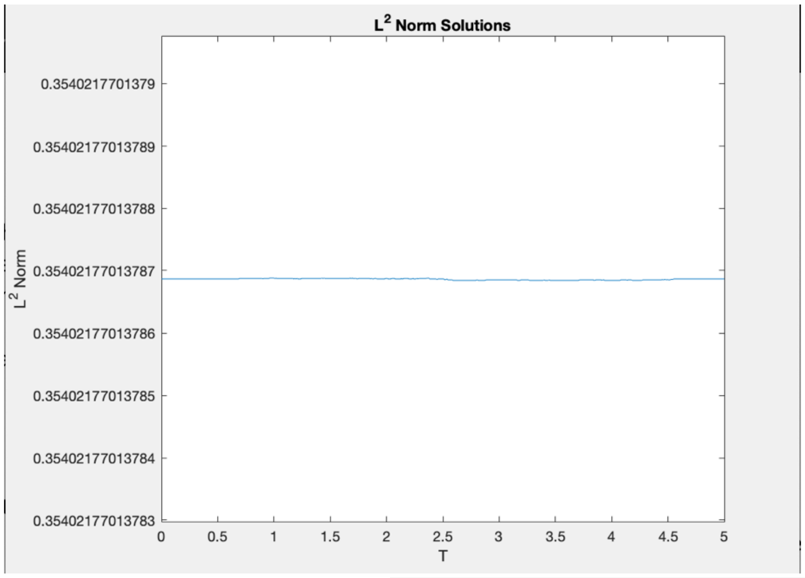


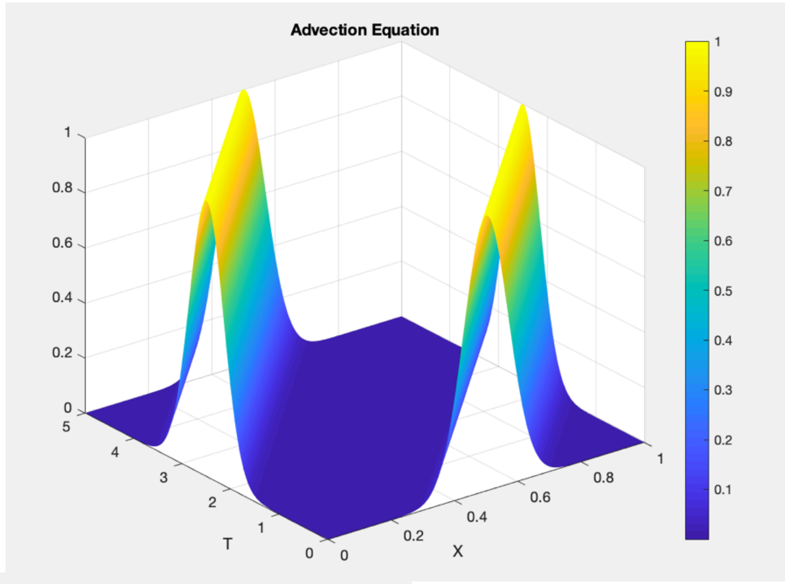

```
N = 300;
M = 300;
xx = linspace(0, 1, N+2)';
Insidex = xx(2:end-1);
deltax = 1/(N+1);
tF = 1;
deltat = tF/M;
tt = linspace(0, tF, M+1);

[X, T] = meshgrid(xx, tt);

A = zeros(N,1);
A(1) =-2; A(2) =1;
BigA = toeplitz(A)/deltax^2;

Solution = zeros(N, M+1);

ICeq = exp(-50.*(Insidex-.3).^2) ;

Solution(:, 1) = ICeq;
uOld = ICeq;

for i = 1:M
    uNew = TRAP(BigA, uOld, deltat);
    Solution(:, i+1) = uNew;
    uOld = uNew;
end

Solution = [zeros(1, length(tt)); Solution; zeros(1, length(tt))];

figure(1);
surf(X, T, Solution', 'edgecolor', 'none')
title('Crank-Nicolson Solution');
xlabel('X');
ylabel('T');


CFL = deltat/deltax^2;
display(CFL);

function unew = TRAP(Tdx, uold, dt)
    I = eye(size(Tdx));
    v = uold + dt*Tdx*uold/2;
    unew = (I - dt*Tdx/2)\v;
end
```

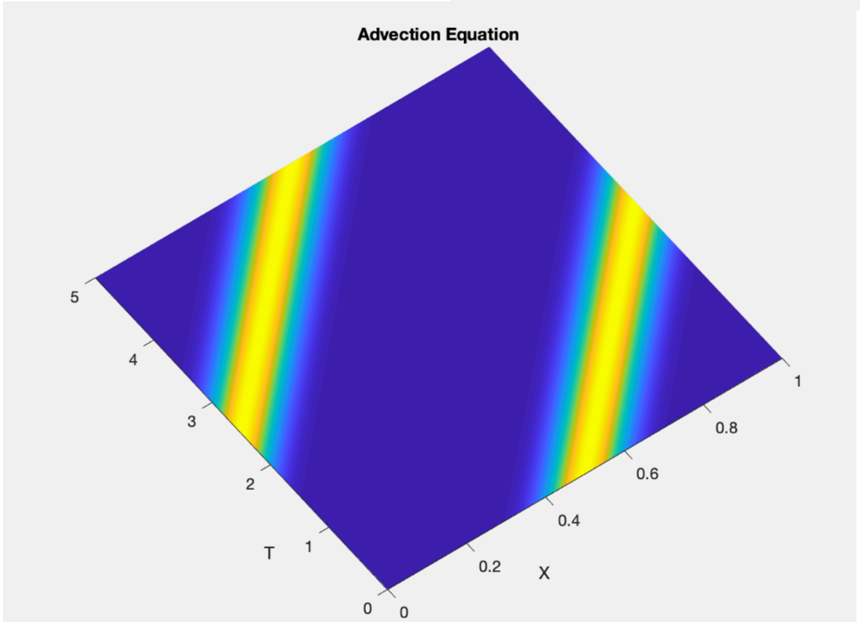
Looking at both graphs they are displaying the same thing just oriented at different positions to show the graph in a more readable way. Now looking at my code, we can see how the Trapezoidal (Crank-Nicolson) method is used to take my matrix to update it implicitly as it also using the "value" that it is looking for. Thus, this is also one of the reasons it is able to give a better result than the Euler method as that was an Explicit Euler method that was being implemented. Also, using 'edgecolor' is important because when you choose higher N and M, the graph starts to become black if this is not implemented.

Task 2

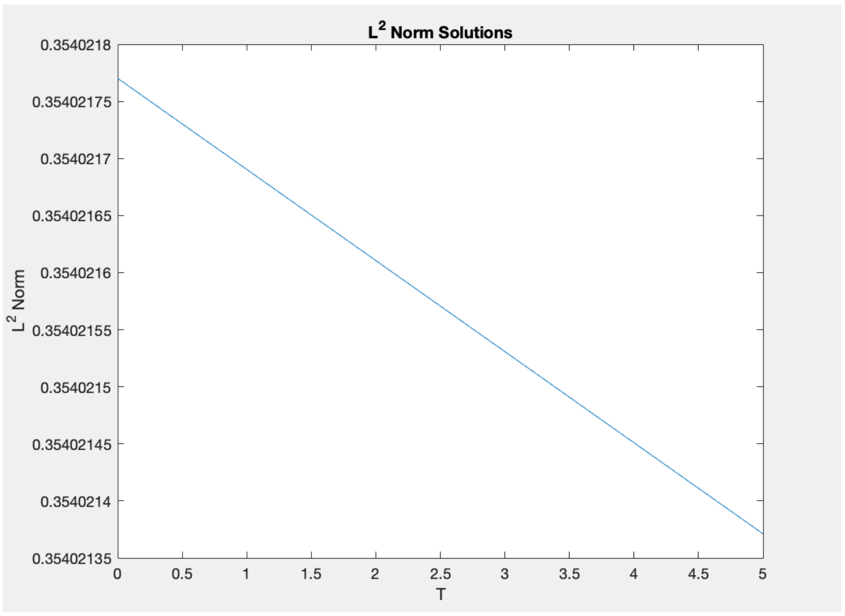
For this this task, I used the linear advection equation $u_t + au_x = 0$ with the space interval [0,1] and initial condition $g(x) = e^{-100(x-.5)^2}$ in which I implemented the Lax-Wendroff method in order to solve the linear advection equation. After verifying my code is able to run this method for both positive and negative a. The following three graphs will be for $g(x) = e^{-100(x-.5)^2}$ with the same space interval as above but with a time interval of [0,5], Courant value of 1, $N = 800$ and $M = 800$.

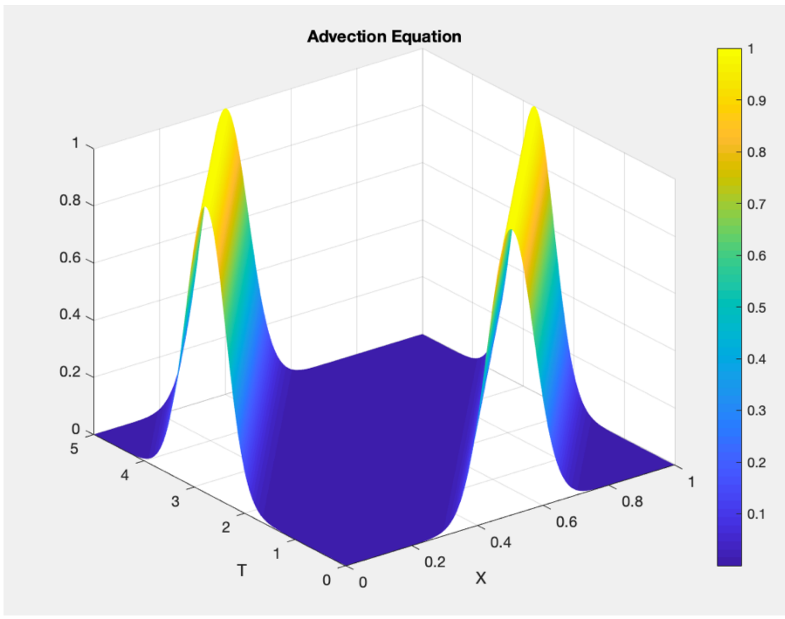


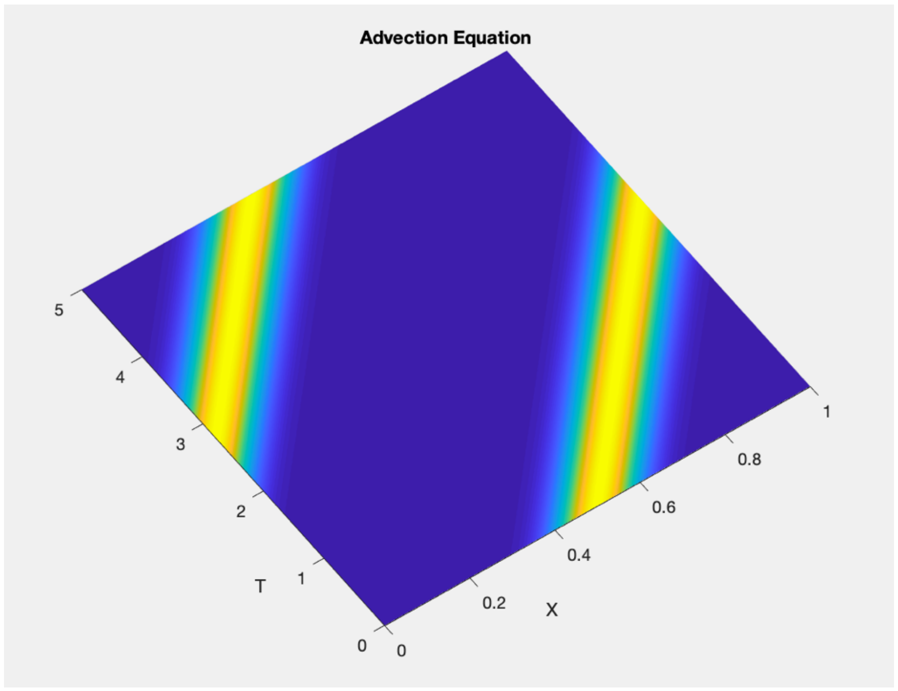




The following three graphs will be for $g(x) = e^{-100(x-.5)^2}$ with the same space interval as above but with a time interval of [0,5], Courant value of .9, $N = 800$ and $M = 800$.



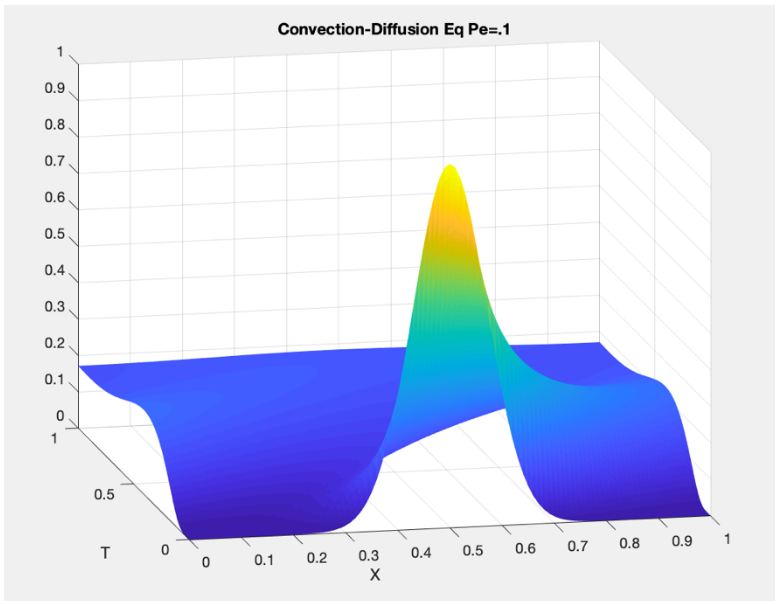




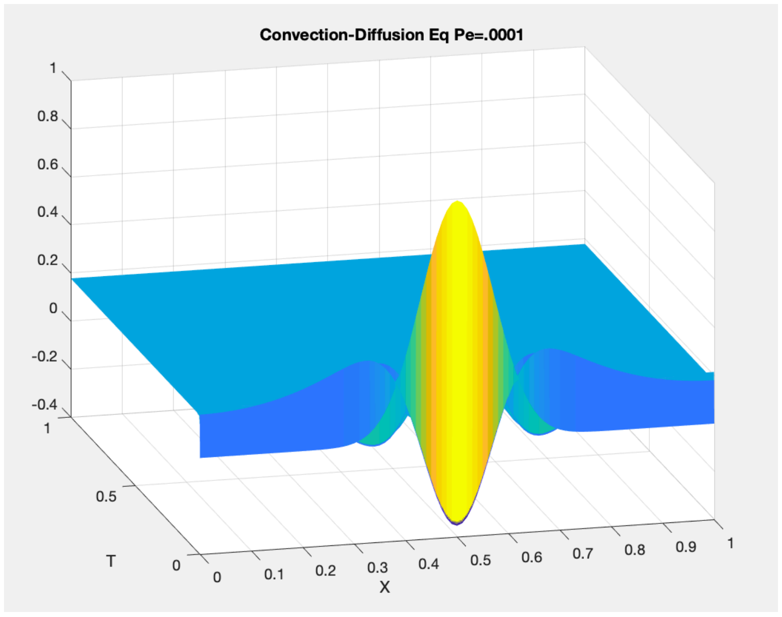
The difference in the graphs is very small as we can see on the lefthand side that there is a small change between them . Also the change in the courant is very small but we can see the difference in the norm graphs. When applying a higher CFL such as 1.1 the solution starts to become unstable thus the difference between .9 and 1 for the CFL is that one value is closer to becoming unstable as it is closer to the limit. Here 1 is at the limit for stability as the CFL condition is less than or equal to one. The norm is being preserved except when CFL is at 1 but it is preserved at .9 as it is shown above. The ampliutude for both CFL seems to be constant as they both remain the same and do not decrease.

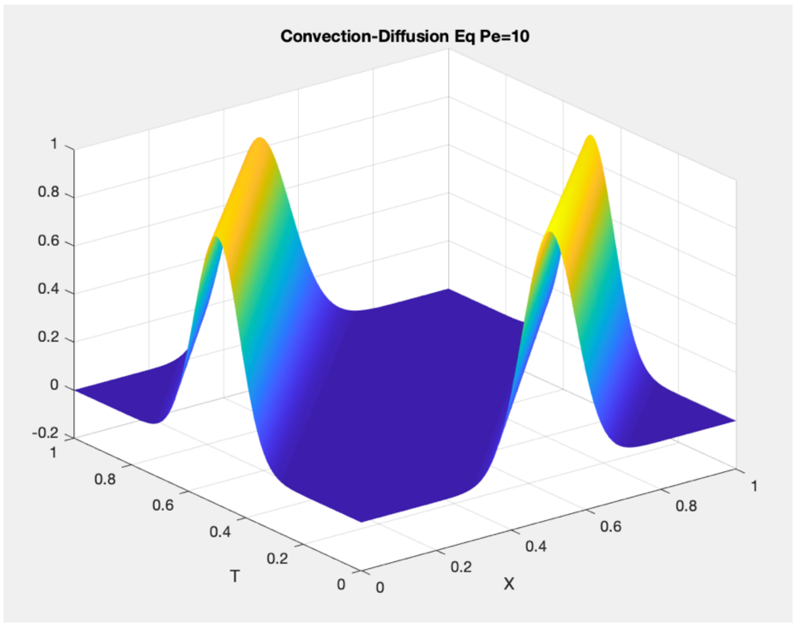
Task 3

For this task we implement the linear convection-diffusion equation $u_t + au_x = du_{xx}$ where I use $(x) = e^{-100(x-.5)^2}$ with the same space interval as above but with a time and space interval of [0,1], $N = 800$ and $M = 800$, a = 1, d=.1. By looking at the ratio $Pe = \left|\frac{a}{d}\right| \Delta x$, we get a convection dominated solution for when Pe is large and a diffusion dominated solution for when Pe is small. Then also implementing, $N = 1000$ and $M = 1000$, we will get the following graph.

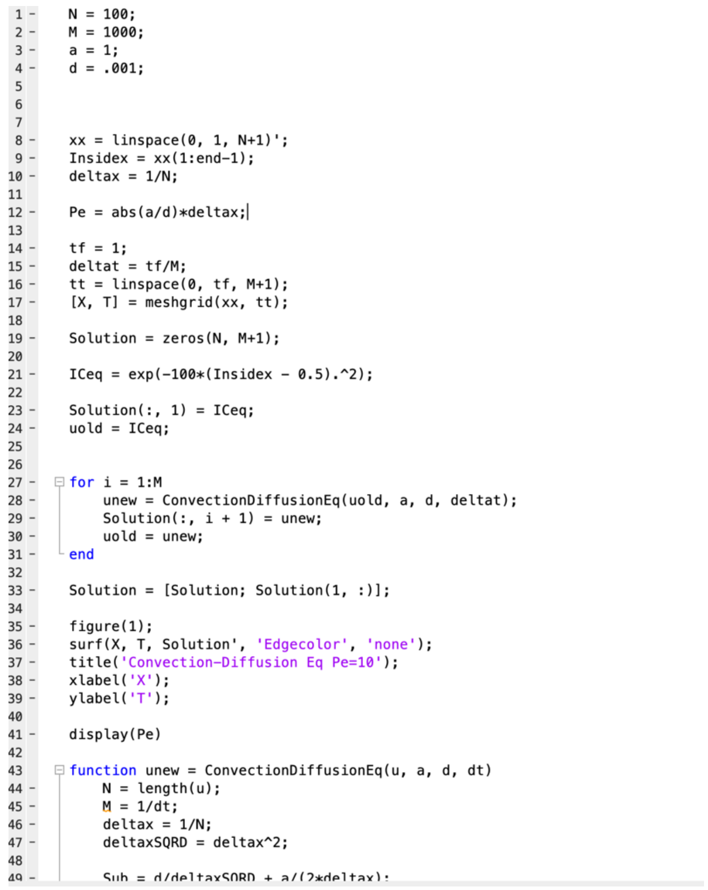


Here the Pe value is .1, which isn't that large of a value, but we can see the convection part of the problem starting to take over compared to the diffusion part as we would need that value to become smaller for this to happen.

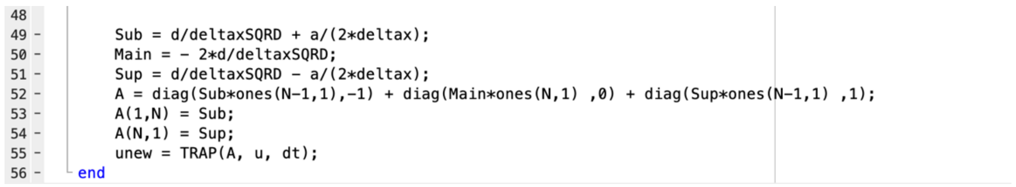




Now the above left graph has Pe value of .0001 and the above right graph has the Pe value of 10 showing how the values of the Pe ratio affect where it will be more of a convection or diffusion as is shown above.



```
N = 100;
M = 1000;
a = 1;
d = .001;



xx = linspace(0, 1, N+1)';
Insidex = xx(1:end-1);
deltax = 1/N;

Pe = abs(a/d)*deltax;

tf = 1;
deltat = tf/M;
tt = linspace(0, tf, M+1);
[X, T] = meshgrid(xx, tt);

Solution = zeros(N, M+1);

ICeq = exp(-100*(Insidex - 0.5).^2);

Solution(:, 1) = ICeq;
uold = ICeq;


for i = 1:M
    unew = ConvectionDiffusionEq(uold, a, d, deltat);
    Solution(:, i + 1) = unew;
    uold = unew;
end

Solution = [Solution; Solution(1, :)];

figure(1);
surf(X, T, Solution', 'Edgecolor', 'none');
title('Convection-Diffusion Eq Pe=10');
xlabel('X');
ylabel('T');

display(Pe)

function unew = ConvectionDiffusionEq(u, a, d, dt)
    N = length(u);
    M = 1/dt;
    deltax = 1/N;
    deltaxSQRD = deltax^2;
```



```

    Sub = d/deltaxSQRD + a/(2*deltax);
    Main = - 2*d/deltaxSQRD;
    Sup = d/deltaxSQRD - a/(2*deltax);
    A = diag(Sub*ones(N-1,1),-1) + diag(Main*ones(N,1) ,0) + diag(Sup*ones(N-1,1) ,1);
    A(1,N) = Sub;
    A(N,1) = Sup;
    unew = TRAP(A, u, dt);
end
```

Now regarding my code, the important thing is to make sure that the setting up of the function 'ConvectionDiffusionEq' is set up correctly as this is what will be the defining part of the problem as the matrices for the discretization need to be done right or else the solution will be wrong. Thus, using the Method of lines, we will get an equation $u' = (dT_{\Delta x} - aS_{\Delta x})u$ where the inside of the parenthesis will be the matrix setup for my code.

Task 4

The Viscous Burgers equation will be the topic of discussion here. It is very similar to the linear convection-diffusion problem except that this problem will be non-linear and that you replace the a from the linear-diffusion problem with a u such as $u_t + uu_x = du_{xx}$. In order to solve this equation, we will need to use an implicit method because of the fact that it is non-linear. To begin with this, we implement the Lax-Wendroff method and then apply the Trapezoidal method (Implicit). Now to show the derivation of the method that we must implement: We will want to take the time derivative out of the Inviscid Burger Eq.

$$u_{tt} = -uu_{xt} - u_t u_x$$

where we will then want to get the space derivative leading to

$$u_{tx} = -uu_{xx} - u_x u_x = -uu_{xx} - {u_x}^2.$$

Now replacing u_{tx} for u_{xt} in the u_{tt} formula where we can now get a formula in terms of space only.

$$u_{tt} = 2u{u_x}^2 - u^2 u_{xx}.$$

Now using the Taylor series expansion and inserting it into the appropriate values we will get the equation:

$$u(t + \Delta t, x) = u + \Delta t(-uu_x) + \frac{\Delta t^2}{2}(2u{u_x}^2 + u^2 u_{xx})$$

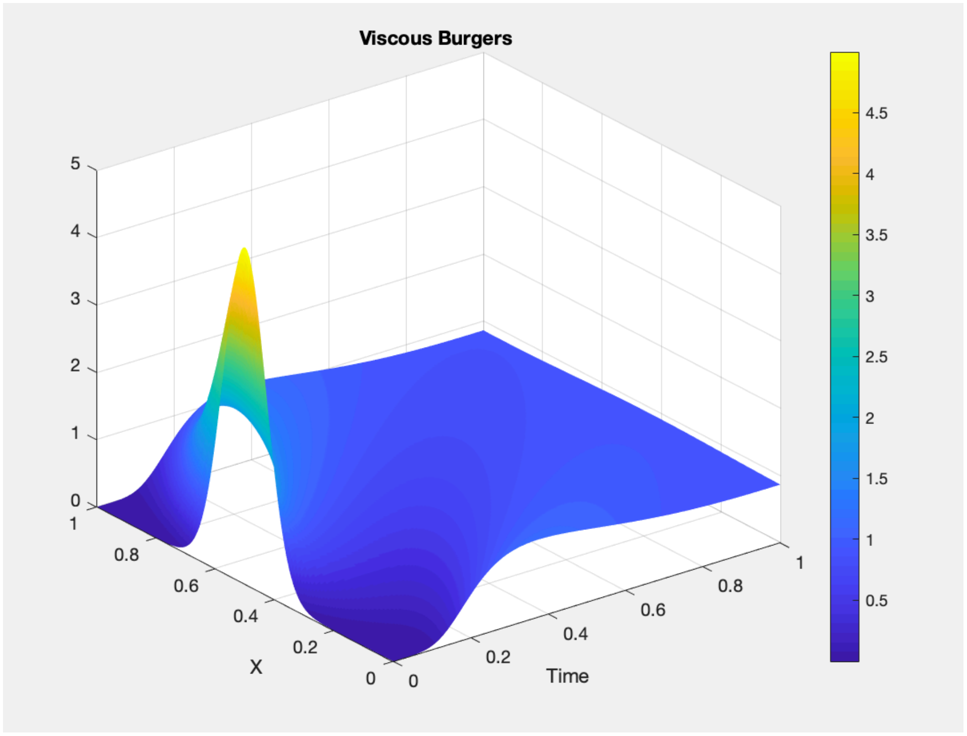
and now using the central finite difference we get u_x = S_x and $u_{xx} = T_{\Delta x}$ where S is a Skew Symmetric matrix and T is a Toeplitz matrix which leads to the Lax-Wendroff scheme of

$${u_l}^{n+1} = {u_l}^n(1 - \Delta t S_x + \Delta t^2 S^2 + \frac{\Delta t^2}{2} T_{\Delta x} {u_l}^n)$$

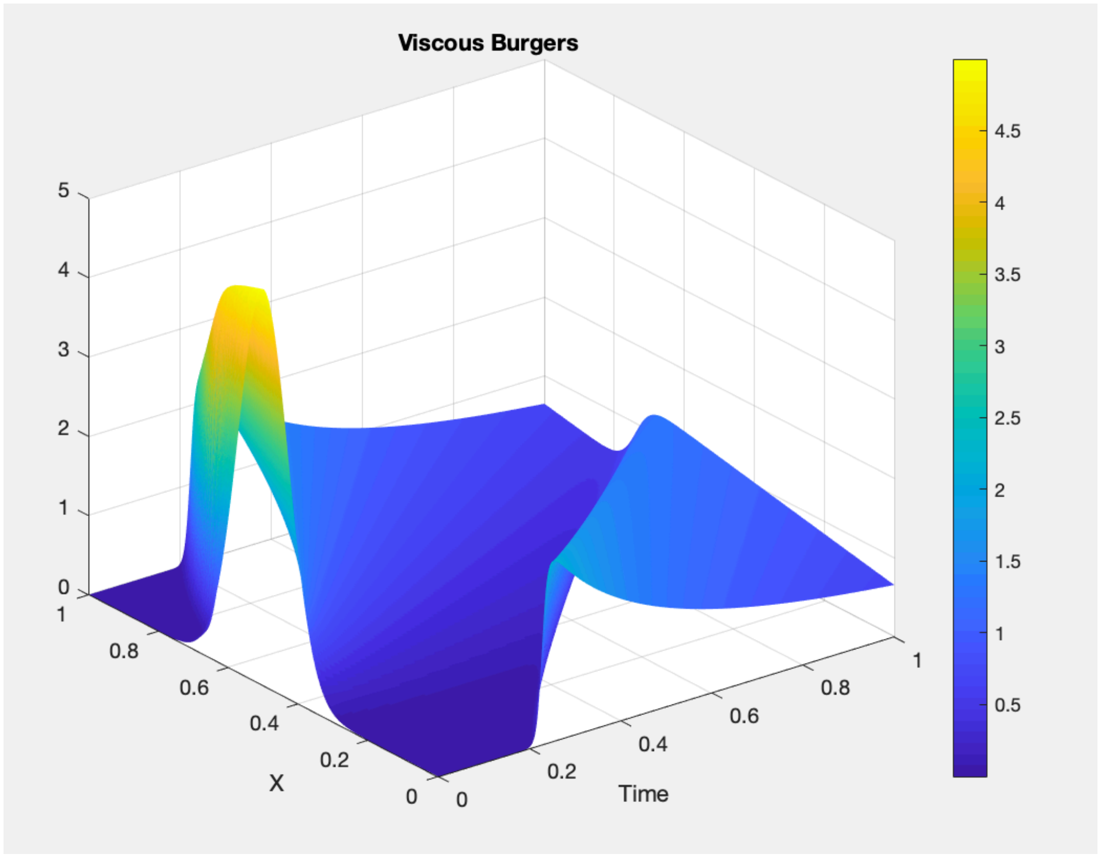
which will ultimately lead to

$$\left(I - d\frac{\Delta t}{2} T_{\Delta x}\right) unew = LW(uold) + d\frac{\Delta t}{2} T_{\Delta x} uold$$

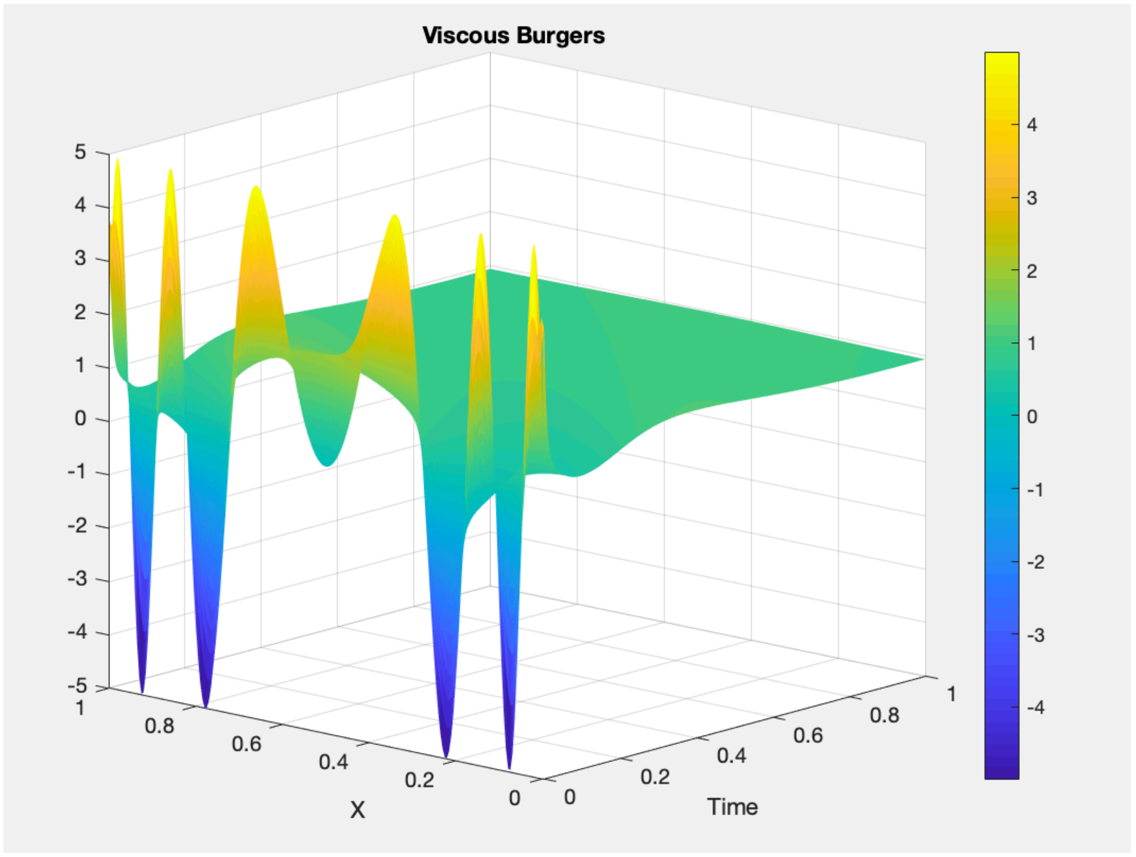
Now the Initial function that I will be using will be $g(x) = 5e^{-100(x-.5)^2}$ with N= 200 and $M = 1000$ where d=.1 to begin with to show that the solver is producing the correct results. Now the following graph will pertain to that diffusivity value of .1.



The following will be of diffusivity of .001



Now to show another graph that could be considered interesting as it has some waves. Here the function that I used is $g(x) = 5\sin(20\pi(x - .5)^2)$. Which produces the following:



Now the first two equations looked more as a combination of convection-diffusion graphs whereas this one looks a bit more interesting because of the dips and inclines that it is producing. As the diffusivity for the first two graphs grew the problem started to become a bit unstable and when graphing for higher diffusivity this was apparent.

.

References

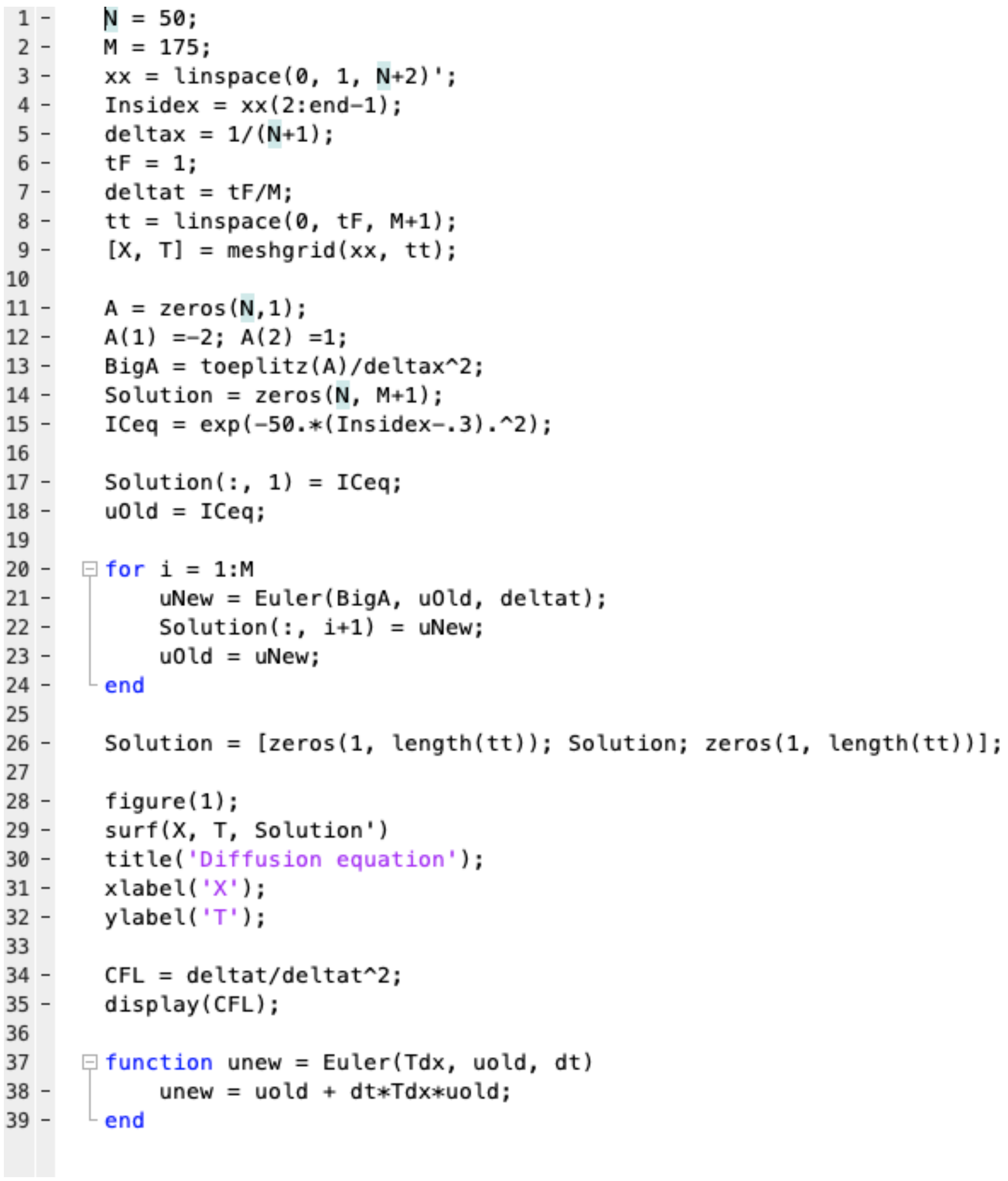
Burden, Richard L., et al. Numerical Analysis. Cengage Learning, 2016.

"Lax-Wendroff Method." *Lax-Wendroff Method - Encyclopedia of Mathematics*, www.encyclopediaofmath.org/index.php/Lax-Wendroff_method.

Chapra, Steven C., and Raymond P. Canale. *Numerical Methods for Engineers: with Software and Programming Applications*. McGraw-Hill, 2003.

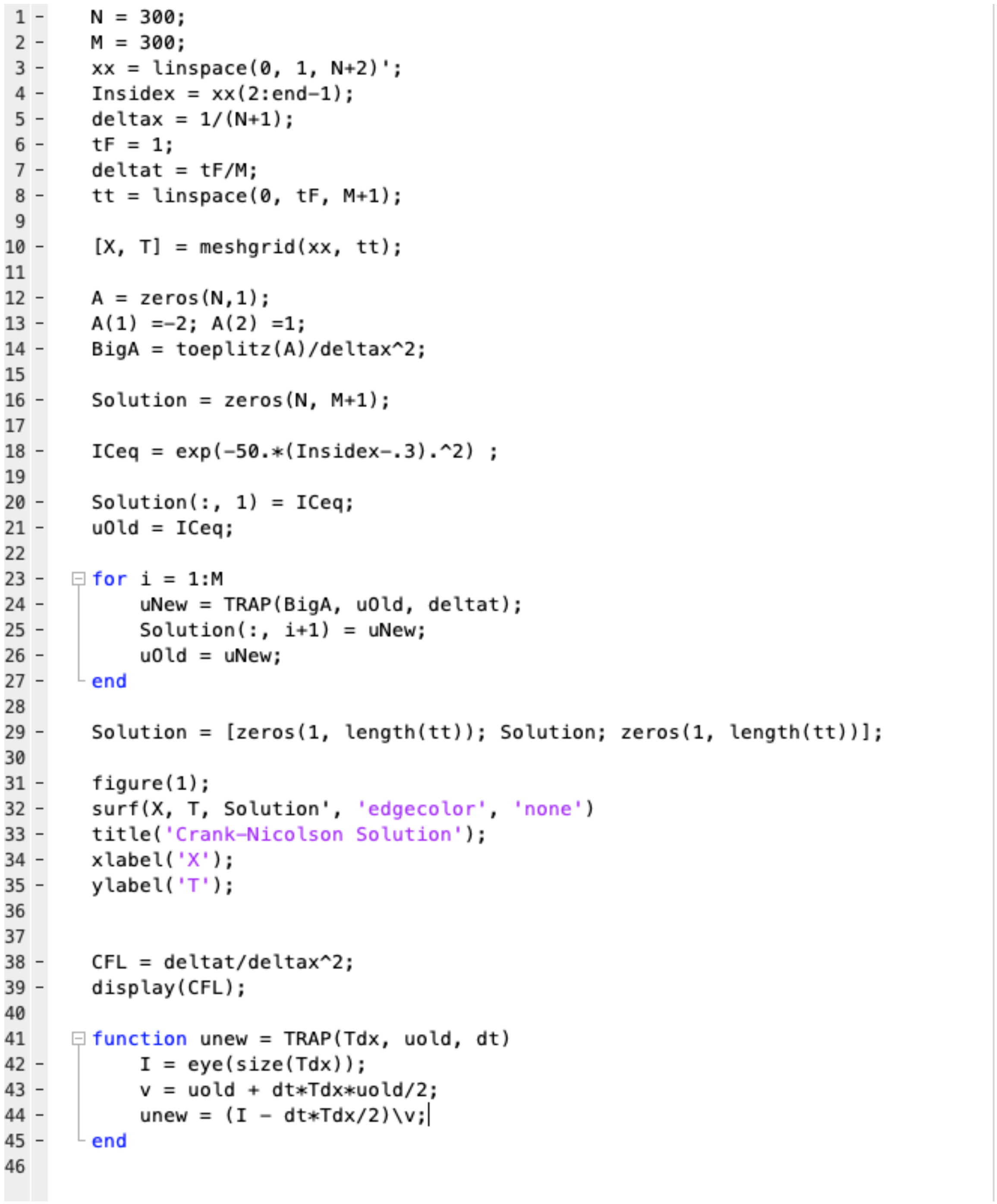
Appendix

Task 1.1



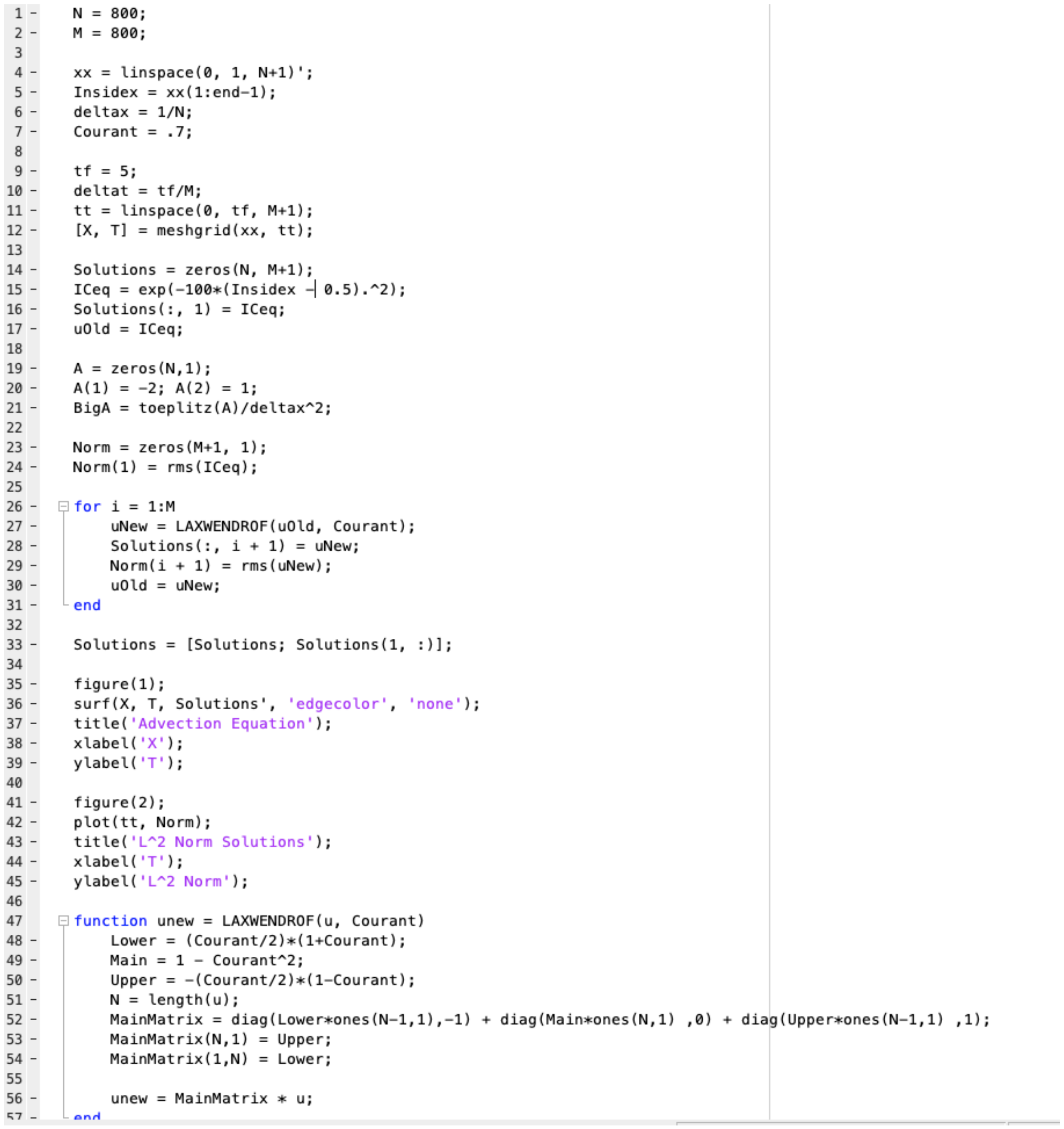
```
N = 50;
M = 175;
xx = linspace(0, 1, N+2)';
Insidex = xx(2:end-1);
deltax = 1/(N+1);
tF = 1;
deltat = tF/M;
tt = linspace(0, tF, M+1);
[X, T] = meshgrid(xx, tt);

A = zeros(N,1);
A(1) =-2; A(2) =1;
BigA = toeplitz(A)/deltax^2;
Solution = zeros(N, M+1);
ICeq = exp(-50.*(Insidex-.3).^2);

Solution(:, 1) = ICeq;
uOld = ICeq;

for i = 1:M
    uNew = Euler(BigA, uOld, deltat);
    Solution(:, i+1) = uNew;
    uOld = uNew;
end

Solution = [zeros(1, length(tt)); Solution; zeros(1, length(tt))];

figure(1);
surf(X, T, Solution')
title('Diffusion equation');
xlabel('X');
ylabel('T');

CFL = deltat/deltat^2;
display(CFL);

function unew = Euler(Tdx, uold, dt)
    unew = uold + dt*Tdx*uold;
end
```

Task 1.2



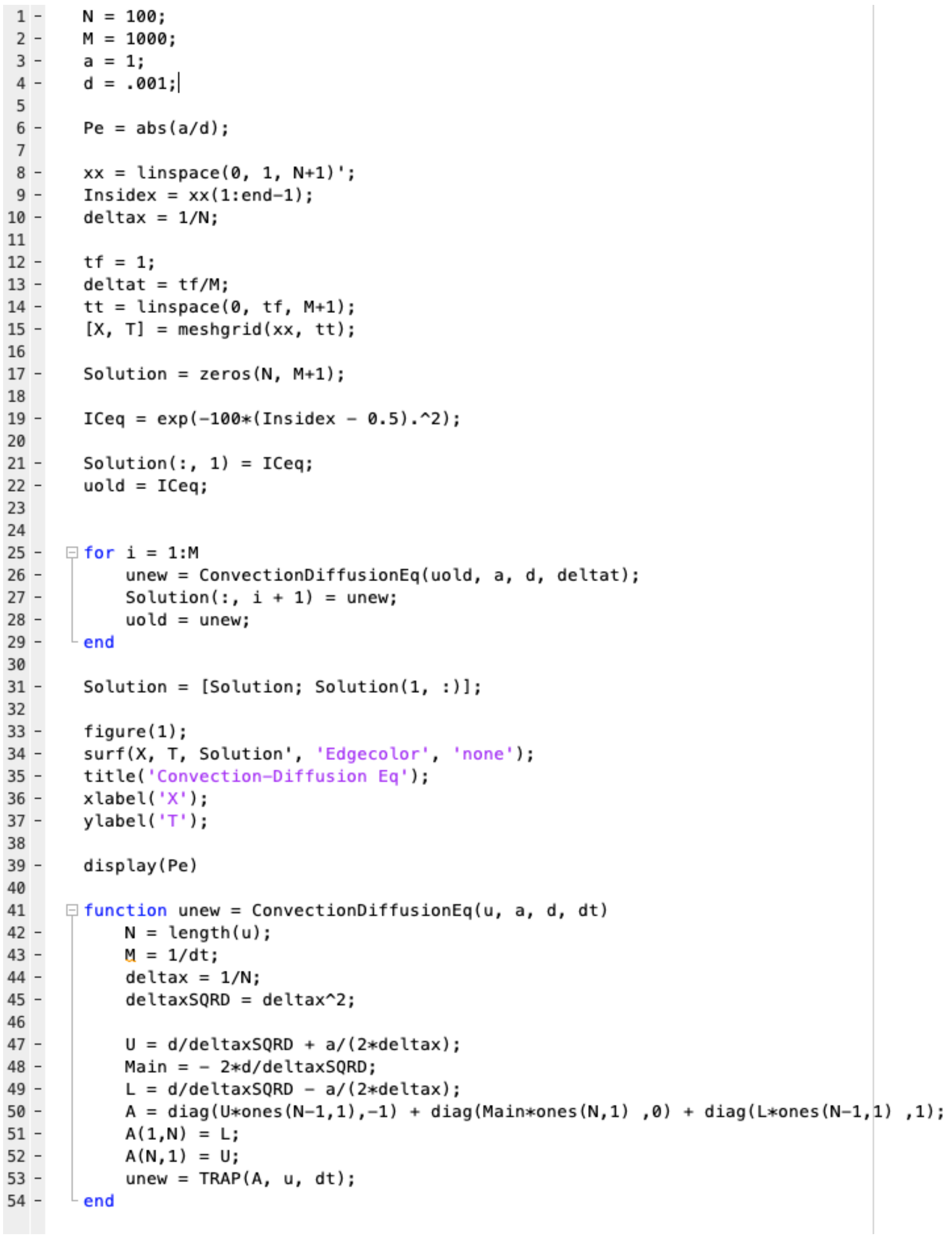
```
N = 300;
M = 300;
xx = linspace(0, 1, N+2)';
Insidex = xx(2:end-1);
deltax = 1/(N+1);
tF = 1;
deltat = tF/M;
tt = linspace(0, tF, M+1);

[X, T] = meshgrid(xx, tt);

A = zeros(N,1);
A(1) =-2; A(2) =1;
BigA = toeplitz(A)/deltax^2;

Solution = zeros(N, M+1);

ICeq = exp(-50.*(Insidex-.3).^2) ;

Solution(:, 1) = ICeq;
uOld = ICeq;

for i = 1:M
    uNew = TRAP(BigA, uOld, deltat);
    Solution(:, i+1) = uNew;
    uOld = uNew;
end

Solution = [zeros(1, length(tt)); Solution; zeros(1, length(tt))];

figure(1);
surf(X, T, Solution', 'edgecolor', 'none')
title('Crank-Nicolson Solution');
xlabel('X');
ylabel('T');


CFL = deltat/deltax^2;
display(CFL);

function unew = TRAP(Tdx, uold, dt)
    I = eye(size(Tdx));
    v = uold + dt*Tdx*uold/2;
    unew = (I - dt*Tdx/2)\v;
end
```

Task 2



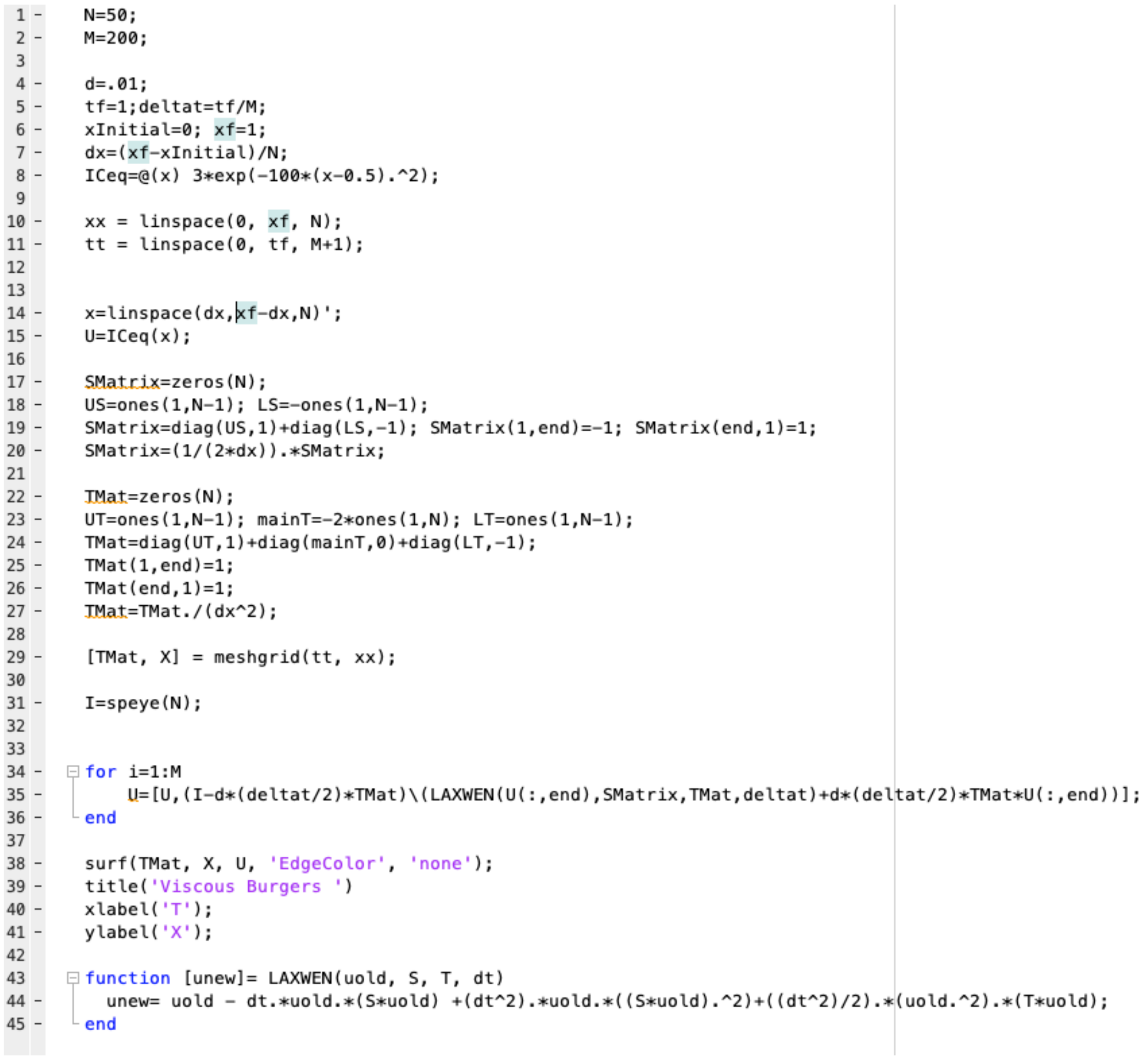
```
N = 800;
M = 800;

xx = linspace(0, 1, N+1)';
Insidex = xx(1:end-1);
deltax = 1/N;
Courant = .7;

tf = 5;
deltat = tf/M;
tt = linspace(0, tf, M+1);
[X, T] = meshgrid(xx, tt);

Solutions = zeros(N, M+1);
ICeq = exp(-100*(Insidex - 0.5).^2);
Solutions(:, 1) = ICeq;
uOld = ICeq;

A = zeros(N,1);
A(1) = -2; A(2) = 1;
BigA = toeplitz(A)/deltax^2;

Norm = zeros(M+1, 1);
Norm(1) = rms(ICeq);

for i = 1:M
    uNew = LAXWENDROF(uOld, Courant);
    Solutions(:, i + 1) = uNew;
    Norm(i + 1) = rms(uNew);
    uOld = uNew;
end

Solutions = [Solutions; Solutions(1, :)];

figure(1);
surf(X, T, Solutions', 'edgecolor', 'none');
title('Advection Equation');
xlabel('X');
ylabel('T');

figure(2);
plot(tt, Norm);
title('L^2 Norm Solutions');
xlabel('T');
ylabel('L^2 Norm');

function unew = LAXWENDROF(u, Courant)
    Lower = (Courant/2)*(1+Courant);
    Main = 1 - Courant^2;
    Upper = -(Courant/2)*(1-Courant);
    N = length(u);
    MainMatrix = diag(Lower*ones(N-1,1),-1) + diag(Main*ones(N,1) ,0) + diag(Upper*ones(N-1,1) ,1);
    MainMatrix(N,1) = Upper;
    MainMatrix(1,N) = Lower;

    unew = MainMatrix * u;
end
```

Task 3



```matlab
N = 100;
M = 1000;
a = 1;
d = .001;

Pe = abs(a/d);

xx = linspace(0, 1, N+1)';
Insidex = xx(1:end-1);
deltax = 1/N;

tf = 1;
deltat = tf/M;
tt = linspace(0, tf, M+1);
[X, T] = meshgrid(xx, tt);

Solution = zeros(N, M+1);

ICeq = exp(-100*(Insidex - 0.5).^2);

Solution(:, 1) = ICeq;
uold = ICeq;


for i = 1:M
    unew = ConvectionDiffusionEq(uold, a, d, deltat);
    Solution(:, i + 1) = unew;
    uold = unew;
end

Solution = [Solution; Solution(1, :)];

figure(1);
surf(X, T, Solution', 'Edgecolor', 'none');
title('Convection-Diffusion Eq');
xlabel('X');
ylabel('T');

display(Pe)

function unew = ConvectionDiffusionEq(u, a, d, dt)
    N = length(u);
    M = 1/dt;
    deltax = 1/N;
    deltaxSQRD = deltax^2;

    U = d/deltaxSQRD + a/(2*deltax);
    Main = - 2*d/deltaxSQRD;
    L = d/deltaxSQRD - a/(2*deltax);
    A = diag(U*ones(N-1,1),-1) + diag(Main*ones(N,1) ,0) + diag(L*ones(N-1,1) ,1);
    A(1,N) = L;
    A(N,1) = U;
    unew = TRAP(A, u, dt);
end
```

Task 4



```
N=50;
M=200;

d=.01;
tf=1;deltat=tf/M;
xInitial=0; xf=1;
dx=(xf-xInitial)/N;
ICeq=@(x) 3*exp(-100*(x-0.5).^2);

xx = linspace(0, xf, N);
tt = linspace(0, tf, M+1);


x=linspace(dx,xf-dx,N)';
U=ICeq(x);

SMatrix=zeros(N);
US=ones(1,N-1); LS=-ones(1,N-1);
SMatrix=diag(US,1)+diag(LS,-1); SMatrix(1,end)=-1; SMatrix(end,1)=1;
SMatrix=(1/(2*dx)).*SMatrix;

TMat=zeros(N);
UT=ones(1,N-1); mainT=-2*ones(1,N); LT=ones(1,N-1);
TMat=diag(UT,1)+diag(mainT,0)+diag(LT,-1);
TMat(1,end)=1;
TMat(end,1)=1;
TMat=TMat./(dx^2);

[TMat, X] = meshgrid(tt, xx);

I=speye(N);


for i=1:M
    U=[U,(I-d*(deltat/2)*TMat)\(LAXWEN(U(:,end),SMatrix,TMat,deltat)+d*(deltat/2)*TMat*U(:,end))];
end

surf(TMat, X, U, 'EdgeColor', 'none');
title('Viscous Burgers ')
xlabel('T');
ylabel('X');

function [unew]= LAXWEN(uold, S, T, dt)
  unew= uold - dt.*uold.*(S*uold) +(dt^2).*uold.*((S*uold).^2)+((dt^2)/2).*(uold.^2).*(T*uold);
end
```