1. Consider the eigenvalus BVP

$$u''(x) = \lambda u(x), \quad u(0) = u(1) = 0$$

which has eigenvalues $\lambda_n = -(n\pi)^2,\ n \in \mathbb{Z}^+$
a. Using finite differences, devise a numerical algorithm for computing these eigenvalues. Implement it in Matlab and discuss how good the approximation is in terms of the mesh size h. Plot the eigenfunctions for the first few eigenvalues.
b. Adapt your algorithm and code from a. to the eigenvalue BVP

$$u''(x) = \lambda u(x), \quad u(0) = 0, \quad u'(1) + u(1) = 0$$

for which the eigenvalues cannot be computed explicitly. Again, plot the first few eigenfunctions.
c. Consider the eigenvalue problem for the 2-dim Laplacian

$$\Delta u = \lambda u$$

in the unit square $[0,1] \times [0,1]$ with Dirichlett boundry conditions using the 5-point Laplacian and explain how well the eigenvalues $\lambda_{m,n} = -(m^2+n^2)\pi^2$ are approximated. Plot eigenfunctions for the first few eigenvalues.

Solution.
a. We use the finite difference formula

$$u'' \approx \frac{U^{n+1} - 2U^n + U^{n-1}}{h^2} = \lambda U^n$$

Hence,

$$U^{n+1} - (2 + h^2\lambda)U^n + U^{n-1} = 0$$

This translates into the matrix equation

$$\begin{pmatrix} -2-h^2\lambda & 1 & 0 & \cdots & 0 \\ 1 & -2-h^2\lambda & \ddots & \ddots & \vdots \\ 0 & 1 & \ddots & 1 & 0 \\ \vdots & \ddots & \ddots & -2-h^2\lambda & 1 \\ 0 & \cdots & 0 & 1 & -2-h^2\lambda \end{pmatrix} U = 0$$

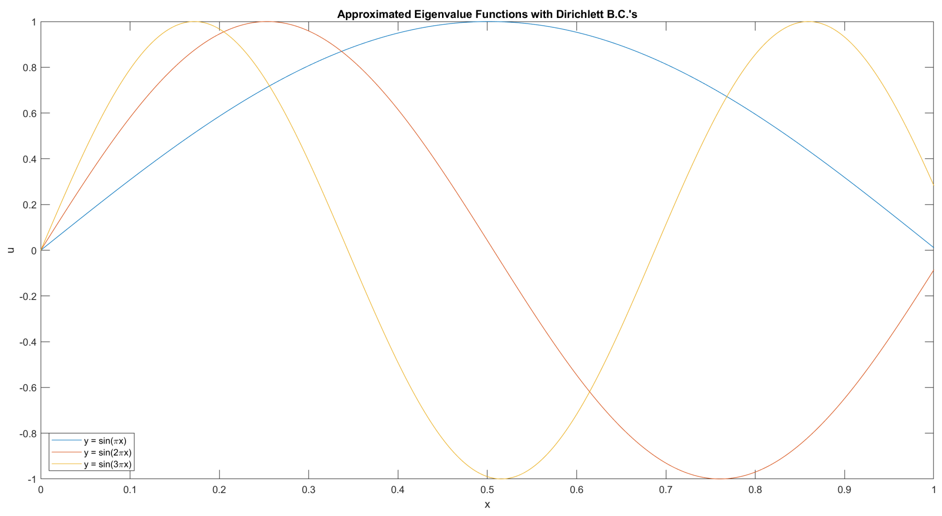
That is

$$(\begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & \ddots & \ddots & \vdots \\ 0 & 1 & \ddots & 1 & 0 \\ \vdots & \ddots & \ddots & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix} - h^2\lambda I)U$$
$$= (A - h^2\lambda I)U = 0$$

So, we have the eigenvalue problem

$$e_i = h^2\lambda$$

This gives us an apporximation to the true eigenvalues with $\frac{e_i}{h^2}$ where e_i is the i^{th} eigenvalue of A. Since $h = \frac{1}{n+1}$ where n is the number of steps in our interval, as n gets larger, h gets smaller, and so our approximation approaches the true eigenvalues with a smaller mesh size. This can be seen by changing the value for n in the code seen below. This being solved before, we have that the general solutions for this particular BVP are $u(x) = \sin(\sqrt{-\lambda}x)$. For the first 3 eigenvalues, we have the following eigenfunctions plotted on the interval $[0, 1]$.

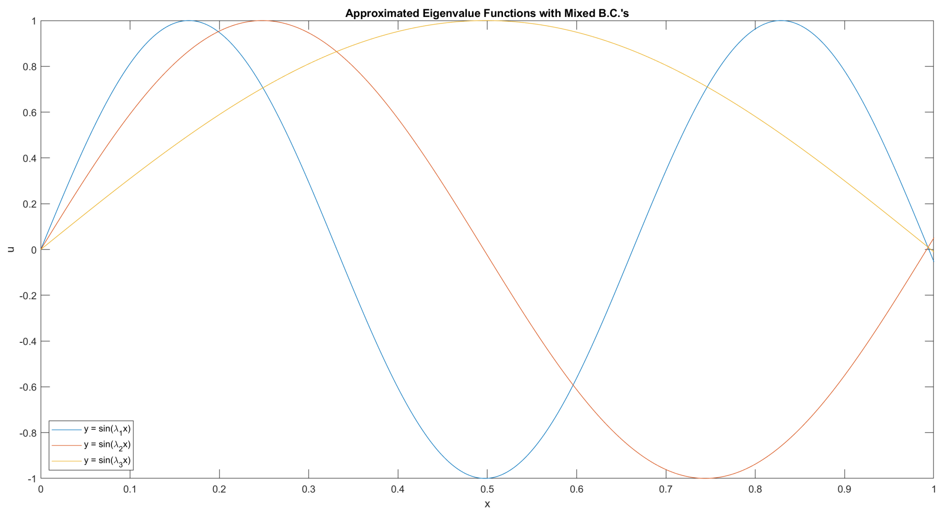


b. We see that this problem will have similar general solutions, just with different eigenvalues to satisfy the new boundry condition $u'(1) + u(1) = 0$. So, the adapted code utilizing the matrix equation

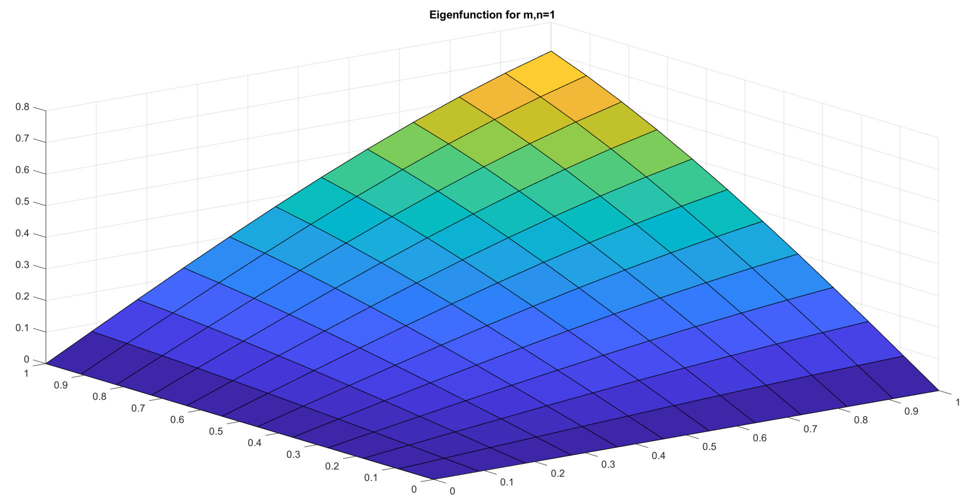
$$AU = \lambda BU$$

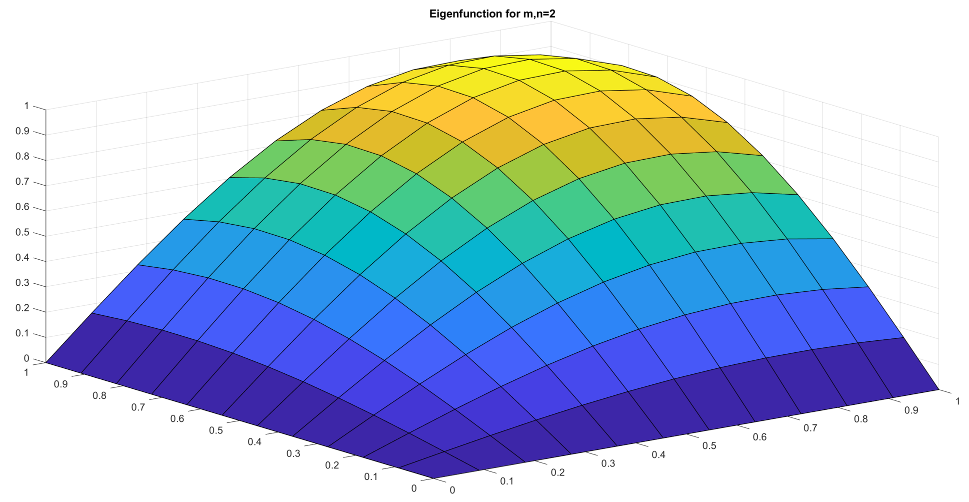
$$B = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

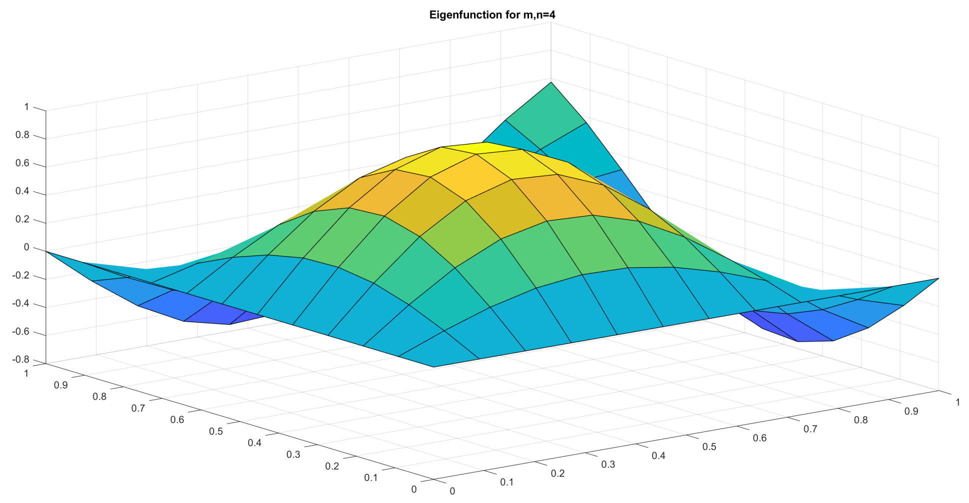
to find the eigenvalues of both A and B gives us the following 3 eigenfunctions, with the code below. Again, we also see that as the discretization increases, and our mesh size decreases. This yeilds better and better approximations and can be seen by chaging the values for n in the code below.



c. For the closed unit square Laplace equation with Dirichlett boundry conditions, we utilized the code from the 5-point Laplacian. For this approximation, we made the distinction between the discretization on x and on y. This allows us to find more distinct eigenvalues and also see how having unequal mesh sizes impairs the approximations. So, for this, we used $m = n$ so that the approximations were closer, and as in the previous two questions, we have that the larger m, n, the better the approximations become. The values for m and n can be changed in the code below.


Eigenfunction for m,n=1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Eigenfunction for m,n=2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Eigenfunction for m,n=4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

```matlab
%Stephanie Klumpe
%Midterm
%Problem 1
%Part a
%Using finite differences to solve an eigenvalue problem posed in ...
    the form
%of a second order ODE u''=lambda*u
clear;
close all;
clc;
fprintf('Problem 1\n')
fprintf('Part a\n');
n=10;%n=25; %n=50;
x=linspace(0,1,10000);
h=1/(n+1);
A=full(gallery('tridiag',n,1,-2,1));
mu=eig(A)/h^2
E=[];
for i=1:n
    eval=-(i*pi)^2;
    E=[eval,E];
end
fprintf('The true eigenvalues are:\n');
disp(E')
figure
for j=0:2
    mj=-mu(n-j);
    utrue=sin(sqrt(mj)*x);
    plot(x,utrue);
    hold on
end
hold off
title('Approximated Eigenvalue Functions with Dirichlett B.C.''s');
xlabel('x');
ylabel('u');
legend({'y = sin(\pix)','y = sin(2\pix)','y = ...
    sin(3\pix)'},'Location','southwest')
```

```matlab
%Stephanie Klumpe
%Midterm
%Problem 1
%Part b
%Using finite differences to solve an eigenvalue problem posed in ...
    the form
%of a second order ODE u''=lambda*u with mixed boundry conditions
clear;
close all;
clc;
fprintf('Problem 1\n')
fprintf('Part b\n');
n=10;%n=25; %n=50;
x=linspace(0,1,10000);
h=1/(n+1);
A=full(gallery('tridiag',n,1,-2,1));
I=eye(n-1,n);
R=zeros(1,n);
B=[I;R];
eta=eig(A,B)/h^2
figure
for k=1:3
    lam=-eta(n-k);
    u2true=sin(sqrt(lam)*x);
    plot(x,u2true);
    hold on
end
title('Approximated Eigenvalue Functions with Mixed B.C.''s');
xlabel('x');
ylabel('u');
legend({'y = sin(\lambda_1x)','y = sin(\lambda_2x)','y = ...
    sin(\lambda_3x)'},'Location','southwest')
```

```
%Stephanie Klumpe
%Midterm
%Problem 1
%Part c
%Using the 5 point Laplacian to solve an eigenvalue problem posed ...
    in the
%form \Δ(u)=lambda*u in the closed unit square with Dirichlet boundry
%conditions.
clear;
close all;
clc;
fprintf('Problem 1\n')
fprintf('Part c\n');
n=10;
%n=25;
%n=50;
m=10;
%m=25;
%m=50;
x=linspace(0,1,n+2);
y=linspace(0,1,m+2);
[X, Y] = meshgrid(x, y);
hx=1/(n+1);
hy=1/(m+1);
Ix = speye(n);
Iy=speye(m);
ex = ones(n,1);
ey=ones(m,1);
T = spdiags([ex -4*ex ex],[-1 0 1],n,n);
S = spdiags([ey ey],[-1 1],m,m);
A = (kron(Iy,T) + kron(S,Ix)) / (hx*hy);
mu=eig(A)
E=[];
for i=1:n
    for j=1:m
        eval=-(i^2+j^2)*pi^2;
        E=[eval,E];
    end
end
fprintf('The true eigenvalues are:\n');
disp(E')
figure;
u11=sin(X).*sin(Y);
surf(X,Y,u11);
title('Eigenfunction for m,n=1');
figure;
u22=sin(2*X).*sin(2*Y);
surf(X,Y,u22);
title('Eigenfunction for m,n=2');
figure;
u12=sin(X).*sin(2*Y);
surf(X,Y,u12);
title('Eigenfunction for n=1,m=2');
```

```
figure;
u21=sin(2*X).*sin(Y);
surf(X,Y,u21);
title('Eigenfunction for n=2,m=1');
figure;
u44=sin(4*X).*sin(4*Y);
surf(X,Y,u44);
title('Eigenfunction for m,n=4');
```

2. An implicit r-step LMM of the form

$$U^{n+r} = U^{n+r-2} + h\sum_{j=0}^{r} \beta_j f(t_{n+j}, U^{n+j}), \quad \beta_r \neq 0$$

is known as the implicit Milne Method.
a. Derive the 2-step third order accurate Milne Method, starting from the relation

$$u(t_{n+2}) = u(t_n) + \int_{t_n}^{t_{n+2}} f(s, u(s))ds$$

and following the procedure described in exercise 5.10.b. in LeVeque.
b. Use the implicit Milne's method to solve numerically the ODE

$$u'(t) = 2(t+1), \quad u(1) = 3$$

Solution.
a. From the prcedure in LeVeque, we will continue with Newton's Method for the derivation. Since it is recommended that we approximate up to a quadratic, indeed since this is a two step this makes sense, that is precisely what we will do. First note that will we have a quadratic equation of the from $p(t) = A + B(t+h) + C(t+h)t$. By Newton's method, we have that

$$A = f(U^n) \;\; B = \frac{f(U^{n+1}) - f(U^n)}{t_{n+1} - t_n}$$

$$C = \frac{\frac{f(U^{n+2})-f(U^{n+1})}{t_{n+2}-t_{n+1}} - \frac{f(U^{n+1})-f(U^n)}{t_{n+1}-t_n}}{t_{n+2} - t_n}$$

Since we are considering uniform time steps, we have that $t_{n+2} - t_{n+1} = t_{n+1} - t_n = h$ and that $t_{n+2} - t_n = 2h$. Hence, $B = \frac{f(U^{n+1})-f(U^n)}{h}$ and $C = \frac{f(U^{n+2})-2f(U^{n+1})+f(U^n)}{2h^2}$. So, integrating $p(t)$ from $t_n = -h$ to $t_{n+2} = h$ yields

$$\int_{t_n}^{t_{n+2}} p(t)d$$

$$= 2Ah + 2Bh^2 + \frac{2}{3}Ch^3 = 2hf(U^n) + 2h^2\frac{f(U^{n+1}) - f(U^n)}{h}$$

$$+\frac{2}{3}h^3\frac{f(U^{n+2}) - 2f(U^{n+1}) + f(U^n)}{2h^2}$$

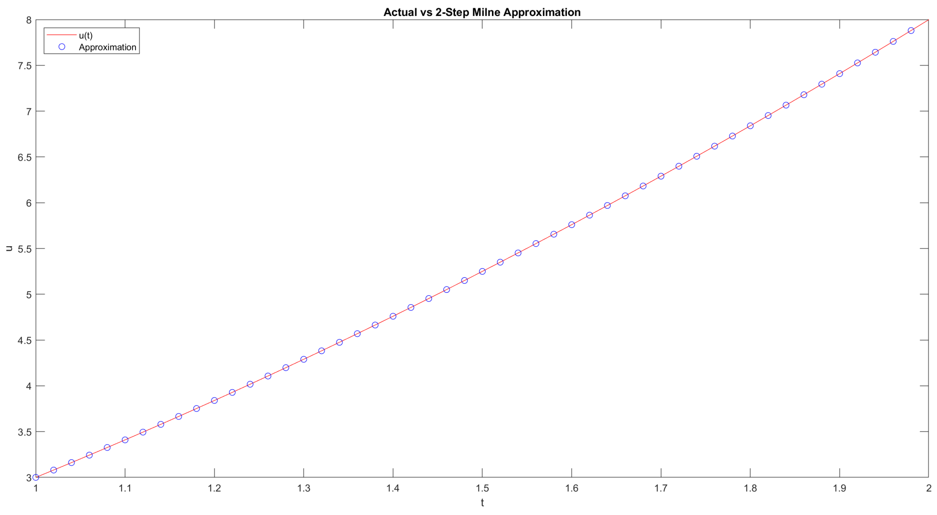
$$= 2hf(U^n) + 2h(f(U^{n+1}) - f(U^n)) + \frac{1}{3}h(f(U^{n+2}) - 2f(U^{n+1}) + f(U^n))$$

$$= \frac{h}{3}(f(U^n) + 4f(U^{n+1}) + f(U^{n+2}))$$

Therefore, the 2-step third order accurate Milne Method is

$$U^{n+2} = U^n + \frac{h}{3}(f(U^n) + 4f(U^{n+1}) + f(U^{n+2}))$$

b. Since this ODE is only dependent on time and not on space, we can compute the exact solution to be $u(t) = t^2 + 2t$. The code to numerically solve is given below.



```
%Stephanie Klumpe
%Midterm
%Problem 2
%Solving the ODE u'(t)=2(t+1), u(1)=3 using the 2 step Milne's ...
    method. In
%order to get the first step so that we have the first two ...
    conditions, we
%impliment the forward Euler's method. After that, it is Milne's ...
    method.
clear;
close all;
clc;
fprintf('Problem 2\n');
%n=10;
n=50;
%n=100;
t=linspace(1,2);
h=1/n;
utrue=t.^2+2*t;
up=3;
uc=3+h*4;
tp=1;
plot(t,utrue,'r');
hold on
for i=1:n
    un=up+(h/3)*(2*(tp+1)+4*2*(tp+h+1)+2*(tp+(2*h)+1));
    plot(tp,up,'bo');
    hold on
    tc=tp+h;
    tp=tc;
    up=uc;
    uc=un;
end
title('Actual vs 2-Step Milne Approximation');
xlabel('t');
ylabel('u');
legend('u(t)','Approximation','Location','northwest');
```

3. Consider the implicit Runge-Kutta method

$$U^* = U^n + \frac{h}{2} f(t_n + \frac{h}{2}, U^*)$$
$$U^{n+1} = U^n + hf(t_n + \frac{h}{2}, U^*)$$

a. Determine the order of accuracy of this method.
b. Determine the absolute stability region. Is it A-stable? L-stable?
c. Use this method to solve the initial value problem $u'(t) = 2(t+1)$, $u(1) = 3$

Solution.
a. Using $f(u) = \lambda u$ on U^* first gives us

$$u_* = u_n + \frac{h}{2}\lambda u_*$$

So,

$$u_* = \frac{2u_n}{2 - h\lambda}$$

Plugging this into the corrector yields

$$U^{n+1} = U^n + hf(t_n + \frac{h}{2}, \frac{2U^n}{2-z}) \;\; z = h\lambda$$

Again using $f(u) = \lambda u$, we get

$$u_{n+1} = u_n + h\lambda \frac{2u_n}{2-z}$$

Whence,

$$\frac{u_{n+1-u_n}}{h} = \lambda \frac{2u_n}{2-z}$$

Hence, our truncation error becomes

$$\tau = \frac{u_{n+1-u_n}}{h} - \lambda \frac{2u_n}{2-z} = \frac{1}{h}u_{n+1} - \frac{1}{h}u_n - \lambda \frac{2u_n}{2-z}$$

Expanding u_{n+1} about u_n gives us

$$\tau = \frac{1}{h}(u_n + hu_n' + \frac{h^2}{2}u_n'' + \mathcal{O}(h^3)) - \frac{1}{h}u_n - \lambda \frac{2u_n}{2-z}$$

$$= u_n' + \frac{h}{2}u_n'' + \mathcal{O}(h^2) - \lambda\frac{2u_n}{2-z}$$

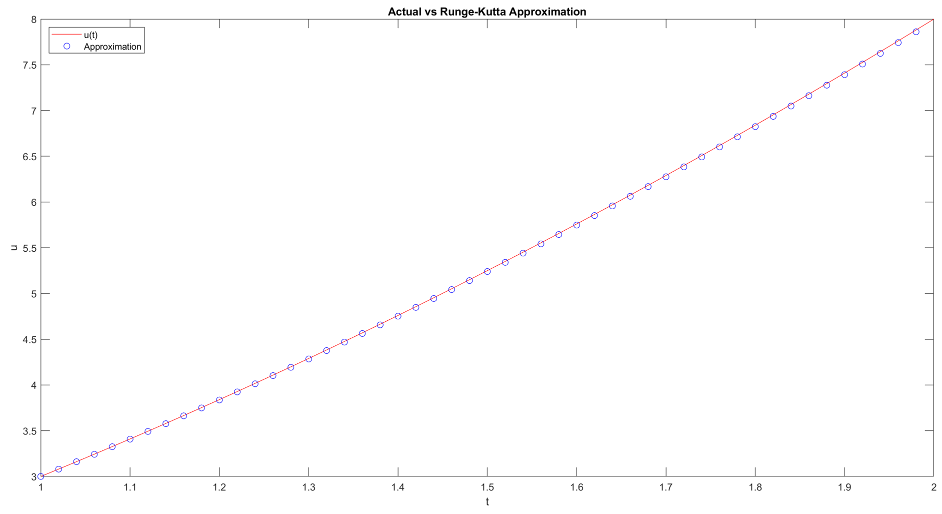
Using $u' = \lambda u$, we have

$$\tau = \lambda u_n + \frac{h\lambda^2}{2}u_n - \lambda\frac{2u_n}{2-z} + \mathcal{O}(h^2)$$
$$= -\frac{h\lambda^2 u_n}{2-z} + \frac{h\lambda^2}{2}u_n + \mathcal{O}(h^2)$$
$$= -\frac{h\lambda^2 u_n}{2-z} + (\frac{1-\frac{1}{2}h\lambda}{1-\frac{1}{2}z})\frac{h\lambda^2}{2}u_n + \mathcal{O}(h^2)$$
$$= -\frac{\frac{1}{2}z^2\lambda}{2-z}u_n + \mathcal{O}(h^2)$$

Thus, we see that this implicit Runge-Kutta method is of the order of h^2

b. From above, we see that $R(z) = 1 + \frac{2z}{1-z} = \frac{2+z}{2-z}$. So, the stability region is the set $\{z \in \mathbb{C}| \ |\frac{2+z}{2-z}| \leq 1\}$. Clearly, $\lim_{z\to\infty}|R(z) = 1|$, so this method is not L-stable. We also have that $R^{-1}(e^{i\theta}) = \frac{2(e^{i\theta}-1)}{e^{i\theta}+1}$ and since for all $\theta \in [0, 2\pi]$, $Re(R^{-1})$ is defined, we have that the method is A-stable.

c. Since the ODE in question is only dependent on t, we have that the code comes down to an explicit single step method. The code is implemented below and just as above in problem 2, the true solution is $u(t) = t^2 + 2t$.



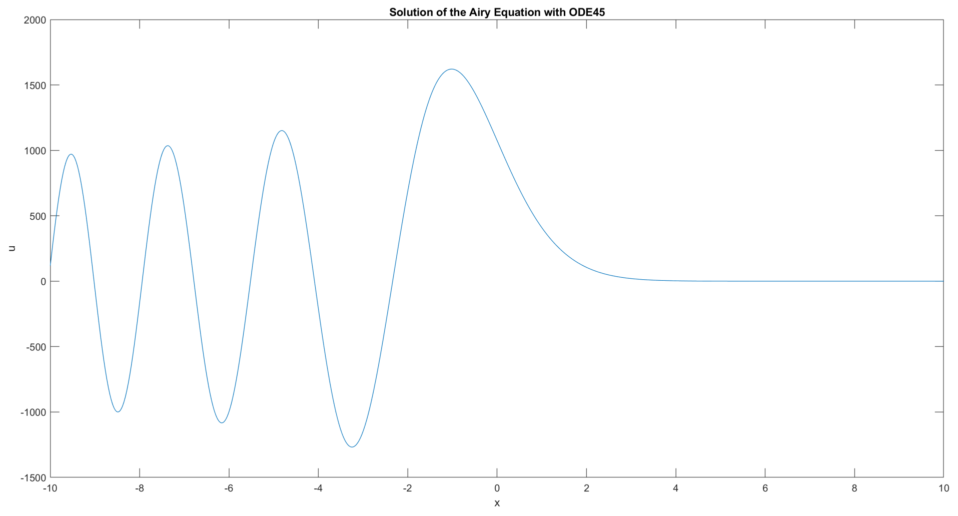
```
%Stephanie Klumpe
%Midterm
%Problem 3
%Solving the ODE u'(t)=2(t+1), u(1)=3 using the implicit Runge-Kutta
%method on the interval [1,2].
clear;
close all;
clc;
fprintf('Problem 3\n');
%n=10;
n=50;
%n=100;
t=linspace(1,2);
h=1/(n);
utrue=t.^2+2*t;
up=3;
tp=1;
plot(t,utrue,'r');
hold on
for i=1:n
    uc=up+h*(2*(tp+1));
    plot(tp,up,'bo');
    hold on
    tc=tp+h;
    tp=tc;
    up=uc;
end
title('Actual vs Runge-Kutta Approximation');
xlabel('t');
ylabel('u');
legend('u(t)','Approximation','Location','northwest');
```

4. Consider the second order problem

$$u''(x) = xu(x), \quad x \geq 0$$

which has a unique solution $u(x)$ with $u(\infty) = 0$. Numerically find the solution approaching as either a BVP or an IVP.

Solution.
To solve the Airy equation, we took the approach of an IVP. Here, we set the initial values to be close to zero, on the order of about half of the epsilon in Matlab. That is, we set the initial conditions to be on the order of 10^{-6}. From there, we simply used two already defined Matlab functions to approximate the solution to the equation. We used an in name only modified ode function as well as ode45. The approximate solution has the condition $u(\infty) = 0$ as desired which can be seen below. SInce the solution is unique and the desired condition is met, we have a decent approximation.



```
%Stephanie Klumpe
%Midterm
%Problem 4
%Using ode45, we solve the Airy equation. Here, we used a prebuilt ode
%function from mathworks while changing the variables to better ...
    fit the
%problem u''=xu.
clear;
close all;
clc;
fprintf('Problem 4\n');
xspan=linspace(10,-10,1000);
u0=[1e-6 1e-6];
[x,u]=ode45(@(x,u) odeairy(x,u), xspan, u0);
plot(x,u(:,1))
title('Solution of the Airy Equation with ODE45');
xlabel('x');
ylabel('u');
function dudx = odeairy(x,u)
  dudx = zeros(2,1);
  dudx(1) = u(2);
  dudx(2) = x.*u(1);
end
```