22S-CSB150-EEB159 Lab1

LISA WANG

TOTAL POINTS

100 / 100

QUESTION 1

1 Ex.1 10 / 10

✓ **- 0 pts** *Correct*

- 1 pts Your code is correct, but you are supposed to show both `A` and `B` as outputs. The correct solution looks like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

- 1 pts Your loop structure is correct, but you forgot to square the entries of `A` before saving them in `B`. The actual solution looks like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

- 1 pts You were supposed to use a double for loop structure in this question to square the entries of the `A` matrix and save them in `B`. The solution should look something like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

QUESTION 2

2 Ex.2 15 / 15

✓ **- 0 pts** *Correct*

- 4 pts Technically, this function generated the correct output, but the point of the exercise is to write a **recursive** function. i.e. one that calls itself. The actual solution would be something like this:

```matlab
function N = factorial_recur(n)
if n == 1
N = 1
else
N = n * factorial_recur(n - 1)
end
```

- 0 pts Your code is correct so I gave you full

points but you were supposed to test and print 2 input values

- 15 pts No attempt

QUESTION 3

3 Ex.3 15 / 15

✓ **- 0 pts** *Correct*

- 4 pts Technically, you didn't use vectorization. The solution should look something like this:

```matlab
A = rand(3, 3);
tic
% for loop
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
toc


tic
% element-wise
B = A.^2
toc
```

- 4 pts You didn't use the `tic toc` function to show how much time you saved. The solution should look something like this:

```matlab
A = rand(3, 3);
tic
% for loop
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
toc


tic
% element-wise
B = A.^2
toc
```

Your solutions (exercise 1 and 3) are correct, but in this question specifically, you did B = A^2, which is technically incorrect. In this case, B is the square of the entire MATRIX A, not the elements.

QUESTION 4

4 Ex.4 15 / 15

✓ **- 0 pts** *Correct*

- 2 pts You need to find the indices in the format of row and column **together**. The correct solution should look something like this:

```matlab
[i, j] = find((A > (1/6)) & (A < (1/4)));
```

QUESTION 5

5 Ex.5 15 / 15

✓ **- 0 pts** *Correct*

- 5 pts You need to modify both the `logGrowth` and `ode45` functions to take extra parameters. It should look something like this:

```matlab
Function dNdt = logGrowth(t, N, r, k)
```

[t, N] = ode45(@(t, N) logGrowth(t, N, r, k), [t0 tf],
N0;
```

**- 1 pts** I'm not sure where you modified the `logGrowth` function.

**- 1 pts** You need to explain the plots

**- 15 pts** No attempt

QUESTION 6

## *6* Ex.6 **30 / 30**

✓ **- 0 pts** *Correct*

### 6.1 - 7 pts

**- 1 pts** Step size is `0.01` instead of `0.1`

**- 3 pts** Need to plot analytical solution

**- 7 pts** incomplete

### 6.2 - 10 pts

**- 10 pts** incomplete

**- 5 pts** attempted, but incorrect

### 6.3 - 13 pts

**- 3 pts** Need to calculate max error and describe the relationship between error and step size

**- 4 pts** General method correct, but plot is incorrect. Because the output vector of analytical solution and numerical solution have different dimension, you should transfer one of the vectors and then subtract.

**- 13 pts** third part of exercise not complete.

**- 10 pts** for error, need to plot for each case, max error calculation not correct ("container" should be vector not a value), and need to explain the relationship between step size and error.

**- 2 pts** Relationship incorrect

**- 3 pts** Need axes and titles on plots
```

Lisa Wang
UID: 105502901
Lab 1 Submission

1. Exercise on matrices (10pt). Define a random matrix A of size 3-by-3. Use a double "for loop" to calculate the square of the entries in A and store the values in another matrix B (Hint: type "help rand" in Matlab if you don't know how to define a random matrix). Your solution should show both A and B.

Code:

```
A = rand(3,3)
```

A =

0.6551	0.4984	0.5853
0.1626	0.9597	0.2238
0.1190	0.3404	0.7513

```
>> for i = 1:3
for j = 1:3
B(i,j) = A(i,j).^2;
end
end
>> B
```

B =

0.4292	0.2484	0.3425
0.0264	0.9211	0.0501
0.0142	0.1159	0.5644

2. Exercise (15pt): Finish the following recursive code (fill in the # part) that gives the factorial of a positive number n, using the recursive formula $n! = n(n-1)...1$. Then test 2 input values of your choice using the code.

```
function N = factorial_recur(n)

if #
    N = 1;
else
    N = #;
end
```

```
function N = factorial_recur(n)
if n == 0
```

1 Ex.1 10 / 10

✓ **- 0 pts** *Correct*

- 1 pts Your code is correct, but you are supposed to show both `A` and `B` as outputs. The correct solution looks like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

- 1 pts Your loop structure is correct, but you forgot to square the entries of `A` before saving them in `B`. The actual solution looks like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

- 1 pts You were supposed to use a double for loop structure in this question to square the entries of the `A` matrix and save them in `B`. The solution should look something like this:

```matlab
A = rand(3, 3);
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
```

Lisa Wang
UID: 105502901
Lab 1 Submission

1. Exercise on matrices (10pt). Define a random matrix A of size 3-by-3. Use a double "for loop" to calculate the square of the entries in A and store the values in another matrix B (Hint: type "help rand" in Matlab if you don't know how to define a random matrix). Your solution should show both A and B.

Code:

```
A = rand(3,3)
```

A =

0.6551	0.4984	0.5853
0.1626	0.9597	0.2238
0.1190	0.3404	0.7513

```
>> for i = 1:3
for j = 1:3
B(i,j) = A(i,j).^2;
end
end
>> B
```

B =

0.4292	0.2484	0.3425
0.0264	0.9211	0.0501
0.0142	0.1159	0.5644

2. Exercise (15pt): Finish the following recursive code (fill in the # part) that gives the factorial of a positive number n, using the recursive formula $n! = n(n-1)...1$. Then test 2 input values of your choice using the code.

```
function N = factorial_recur(n)

if #
    N = 1;
else
    N = #;
end
```

```
function N = factorial_recur(n)
if n == 0
```

```
N = 1;
else
N = n*factorial_recur(n-1);
end
```

Displayed solution:

```
factorial_recur(2)

ans =
    2

>> factorial_recur(5)

ans =
   120
```

3. Exercise (15pt): Re-do the for loop exercise (Ex.1) using element-wise operator (.^) and see how much time you've saved (hint: use tic, toc function above to measure time).

Code:

```
 tic
B = A^2
toc

 tic
B = A.^2
toc
```

the elapsed time has been 5.365 seconds and the second elapsed time has been 2.728 seconds, I hypothesize that since it is only a squared (exponential power of 2) power, the effect of a vectorized Matlab data structure is more apparent.

4. Exercise(15pt): Build a 5-by-5 random matrix A using the command

```
>> A = rand(5)
```

Find the row and column indices of entries whose value is smaller than 1/4 and bigger than 1/6 (hint: type "help find" to learn more about find function if necessary. Can also use `ind2sub` function to convert linear index to matrix index).

Code:

```
A = rand(5)

A =
```

2 Ex.2 15 / 15

✓ **- 0 pts** *Correct*

- 4 pts Technically, this function generated the correct output, but the point of the exercise is to write a **recursive** function. i.e. one that calls itself. The actual solution would be something like this:

```matlab
function N = factorial_recur(n)
if n == 1
N = 1
else
N = n * factorial_recur(n - 1)
end
```

- 0 pts Your code is correct so I gave you full points but you were supposed to test and print 2 input values

- 15 pts No attempt

```
N = 1;
else
N = n*factorial_recur(n-1);
end
```

Displayed solution:

```
factorial_recur(2)

ans =
    2

>> factorial_recur(5)

ans =
   120
```

3. Exercise (15pt): Re-do the for loop exercise (Ex.1) using element-wise operator (.^) and see how much time you've saved (hint: use tic, toc function above to measure time).

Code:

```
 tic
B = A^2
toc

 tic
B = A.^2
toc
```

the elapsed time has been 5.365 seconds and the second elapsed time has been 2.728 seconds, I hypothesize that since it is only a squared (exponential power of 2) power, the effect of a vectorized Matlab data structure is more apparent.

4. Exercise(15pt): Build a 5-by-5 random matrix A using the command

```
>> A = rand(5)
```

Find the row and column indices of entries whose value is smaller than 1/4 and bigger than 1/6 (hint: type "help find" to learn more about find function if necessary. Can also use `ind2sub` function to convert linear index to matrix index).

Code:

```
A = rand(5)

A =
```

3 Ex.3 15 / 15

✓ **- 0 pts** *Correct*

- 4 pts Technically, you didn't use vectorization. The solution should look something like this:

```matlab
A = rand(3, 3);
tic
% for loop
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
toc

tic
% element-wise
B = A.^2
toc
```

- 4 pts You didn't use the `tic toc` function to show how much time you saved. The solution should look something like this:

```matlab
A = rand(3, 3);
tic
% for loop
for i = 1:3,
for j = 1:3,
B(i, j) = A(i, j)^2;
end
end
toc

tic
% element-wise
```

```
B = A.^2
toc
```

Your solutions (exercise 1 and 3) are correct, but in this question specifically, you did B = A^2, which is technically incorrect. In this case, B is the square of the entire MATRIX A, not the elements.

```
N = 1;
else
N = n*factorial_recur(n-1);
end
```

Displayed solution:

```
factorial_recur(2)

ans =
    2

>> factorial_recur(5)

ans =
   120
```

3. Exercise (15pt): Re-do the for loop exercise (Ex.1) using element-wise operator (.^) and see how much time you've saved (hint: use tic, toc function above to measure time).

Code:

```
 tic
B = A^2
toc

 tic
B = A.^2
toc
```

the elapsed time has been 5.365 seconds and the second elapsed time has been 2.728 seconds, I hypothesize that since it is only a squared (exponential power of 2) power, the effect of a vectorized Matlab data structure is more apparent.

4. Exercise(15pt): Build a 5-by-5 random matrix A using the command

```
>> A = rand(5)
```

Find the row and column indices of entries whose value is smaller than 1/4 and bigger than 1/6 (hint: type "help find" to learn more about find function if necessary. Can also use `ind2sub` function to convert linear index to matrix index).

Code:

```
A = rand(5)

A =
```

```
   0.3804   0.7792   0.0119   0.5285   0.6892
   0.5678   0.9340   0.3371   0.1656   0.7482
   0.0759   0.1299   0.1622   0.6020   0.4505
   0.0540   0.5688   0.7943   0.2630   0.0838
   0.5308   0.4694   0.3112   0.6541   0.2290

>> find(A >1/6 & A < 1/4)
ans =
   25
>> ind = find(A > 1/6 & A < 1/4)
ind =
   25
>> [r,c] = ind2sub([5,5], ind)
r =
    5
c =
    5
```

Solution : in this set of random numbers, the index of the number that satisfy the condition is at [5,5]

5. Exercise (15pt): read the ode45 documentation and modify your logistic growth model to accept r and K as parameters. Vary the value of K over 1 order of magnitude and compare the results. You should generate 2 plots with different K values and explain your findings.

Code:

```
% Numerical solution of the logistic equation
t0 = 0; % initial time
tf = 50; % final time
N0 = 1; % Initial population size
for i =1:2
   K = 100*(10^i);
[T, vNint] = ode45(@(t,N) logGrowth(t,N, r, K), [t0 tf], N0); % Numerically
integrate
% Actual solution
r= 0.5;
K = 100;
vNact = (N0*exp(r*T))./(1+ N0*(exp(r*T)-1)/K); % Actual solution
% Plot results
figure;
hold on
plot(T, vNint) % Plot numerically integrated solution
plot(T, vNact, 'ro') % Plot actual solution
xlabel('Time'), ylabel('Population size'), title('Logistic model')
print -dpdf logPlot.pdf
```

4 Ex.4 **15 / 15**

✓ **- 0 pts** *Correct*

- 2 pts You need to find the indices in the format of row and column **together**. The correct solution should look something like this:

```matlab
[i, j] = find((A > (1/6)) & (A < (1/4)));
```

```
   0.3804   0.7792   0.0119   0.5285   0.6892
   0.5678   0.9340   0.3371   0.1656   0.7482
   0.0759   0.1299   0.1622   0.6020   0.4505
   0.0540   0.5688   0.7943   0.2630   0.0838
   0.5308   0.4694   0.3112   0.6541   0.2290

>> find(A >1/6 & A < 1/4)
ans =
   25
>> ind = find(A > 1/6 & A < 1/4)
ind =
   25
>> [r,c] = ind2sub([5,5], ind)
r =
    5
c =
    5
```

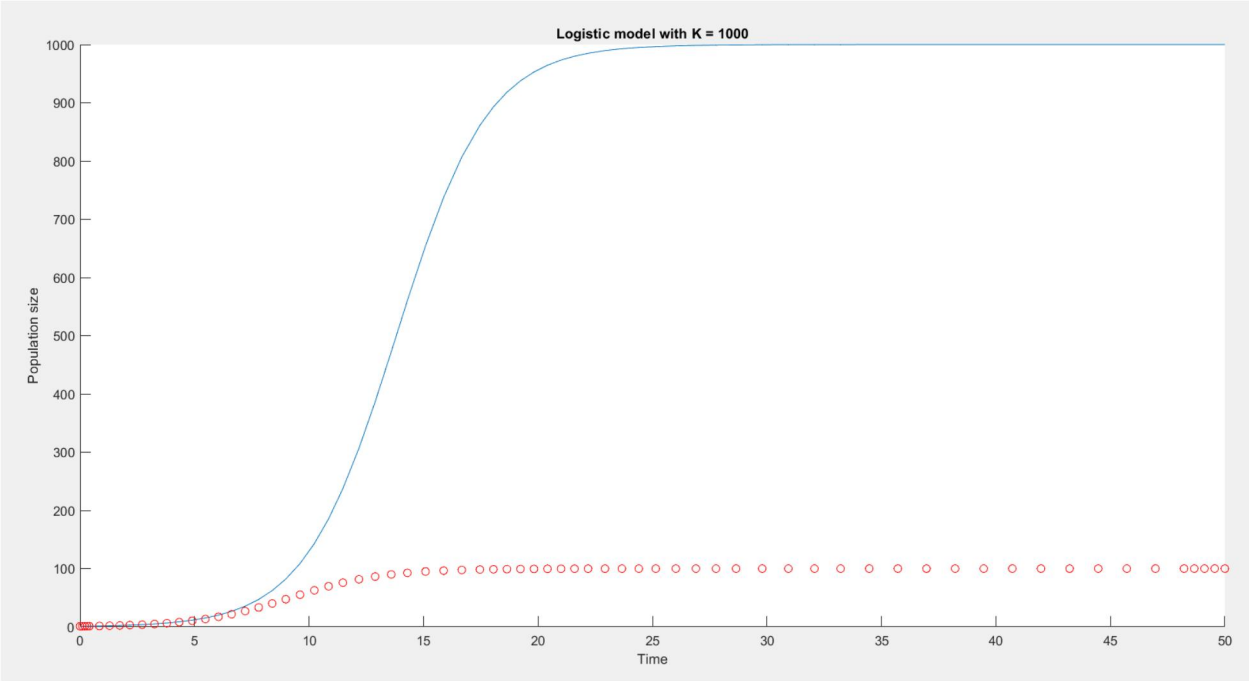
Solution : in this set of random numbers, the index of the number that satisfy the condition is at [5,5]

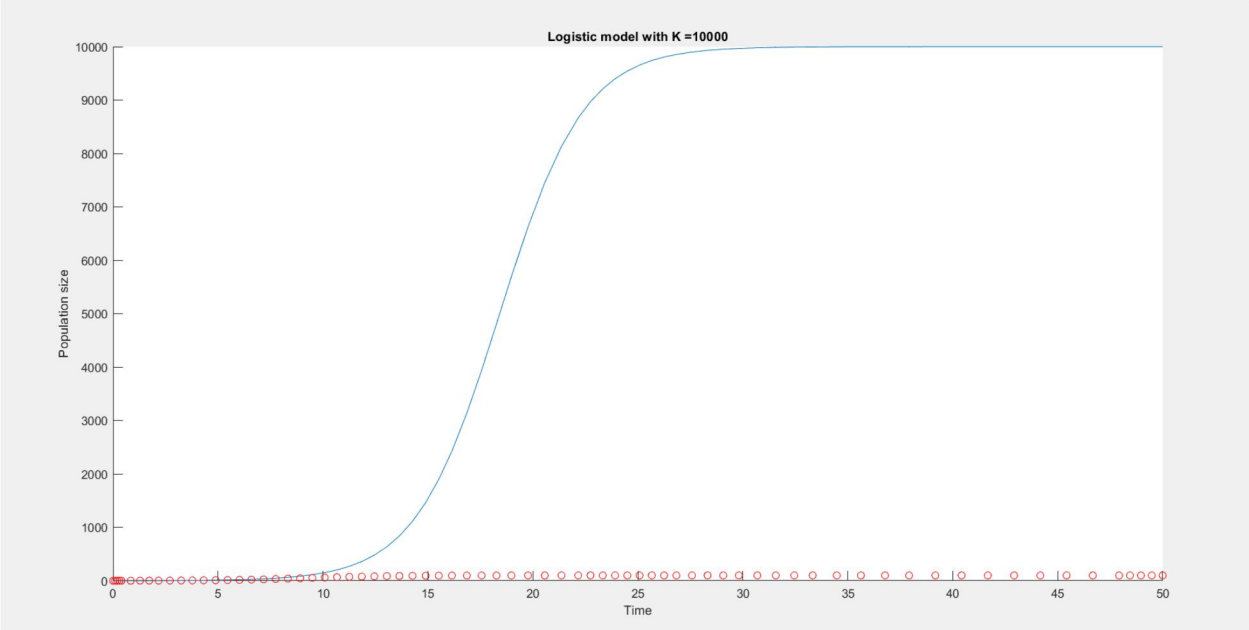
5. Exercise (15pt): read the ode45 documentation and modify your logistic growth model to accept r and K as parameters. Vary the value of K over 1 order of magnitude and compare the results. You should generate 2 plots with different K values and explain your findings.

Code:

```
% Numerical solution of the logistic equation
t0 = 0; % initial time
tf = 50; % final time
N0 = 1; % Initial population size
for i =1:2
   K = 100*(10^i);
[T, vNint] = ode45(@(t,N) logGrowth(t,N, r, K), [t0 tf], N0); % Numerically
integrate
% Actual solution
r= 0.5;
K = 100;
vNact = (N0*exp(r*T))./(1+ N0*(exp(r*T)-1)/K); % Actual solution
% Plot results
figure;
hold on
plot(T, vNint) % Plot numerically integrated solution
plot(T, vNact, 'ro') % Plot actual solution
xlabel('Time'), ylabel('Population size'), title('Logistic model')
print -dpdf logPlot.pdf
```

```
end
```





Explanation: A logistic model is one that is commonly used in ecology, to model that species and growth of organisms are often restricted to resources and habitat size. The K here is a parameter, or commonly referred to as the carrying capacity, which changes the logistics equation's progression. The one horizontal asymptotes are different because of this K.

5 Ex.5 15 / 15

✓ **- 0 pts** *Correct*

- 5 pts You need to modify both the `logGrowth` and `ode45` functions to take extra parameters. It should look something like this:

```matlab
Function dNdt = logGrowth(t, N, r, k)
[t, N] = ode45(@(t, N) logGrowth(t, N, r, k), [t0 tf], N0;
```

- 1 pts I'm not sure where you modified the `logGrowth` function.

- 1 pts You need to explain the plots

- 15 pts No attempt

6. Exercise (30pt): Given the ode and initial condition:

$$\frac{dy}{dt} = cos(20t) \tag{3}$$

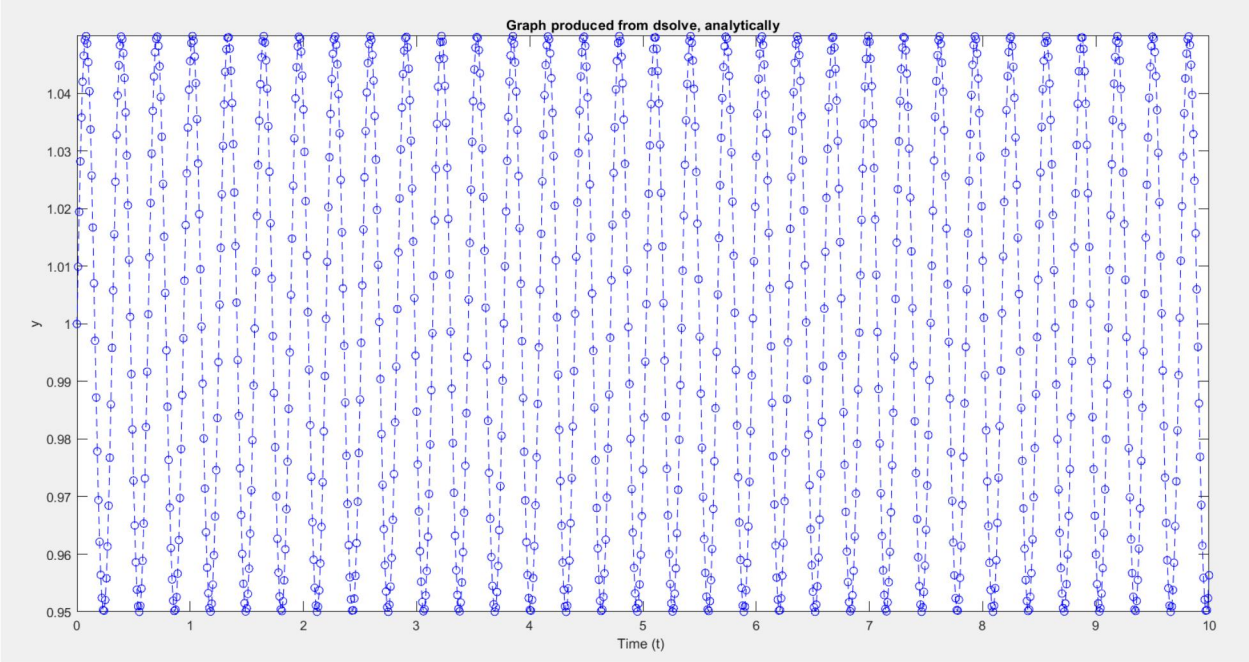
$$y(0) = 1 \tag{4}$$

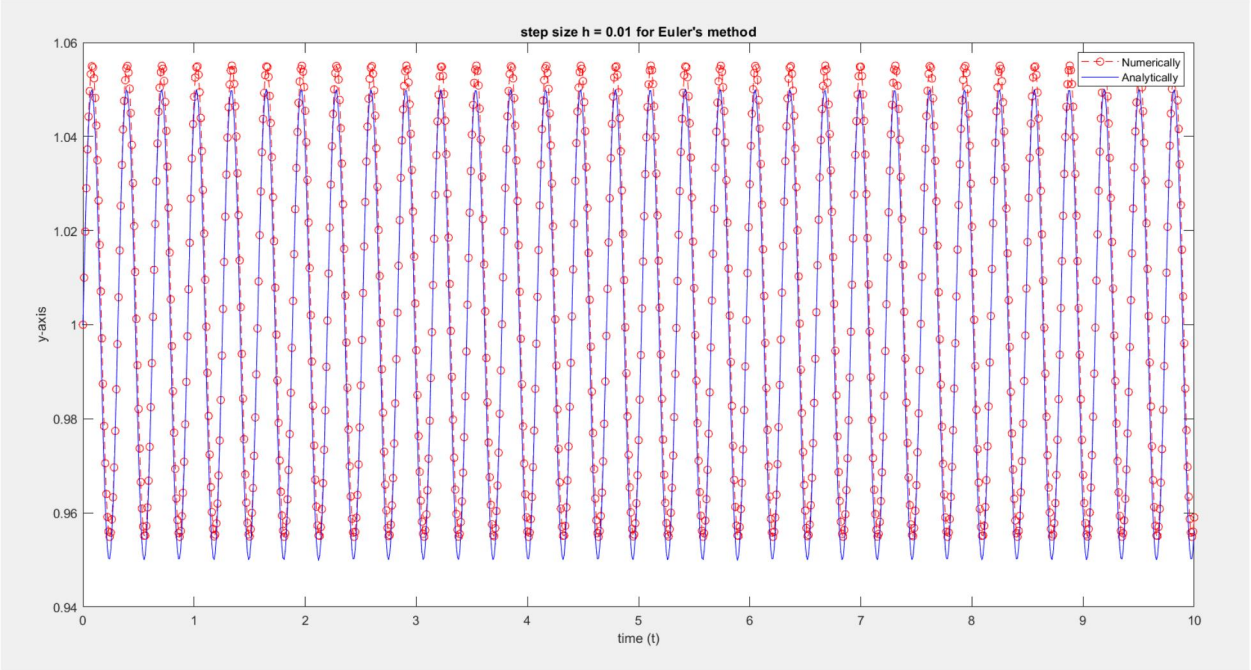
- Solve for y(t) analytically using `dsolve()` in Matlab, and plot the solution over (0,10) with step size $\triangle t = 0.01$.
- Program the ODE in Matlab and solve with step size $\triangle t = 0.01$, 0.1, and 1 (where needed) over (0,10) using (**3 plots EACH for Euler's method and Runge-Kutta 4th order. One for each step size.**):
  - Euler's method
  - Runge-Kutta 4th order
- For each method and step size, plot the error between analytical and numerical solutions (error means the analytical solution output minus numerical solution output (or vise versa)). Calculate max error over (0,10). Explain the relationship between step size and error for each method.

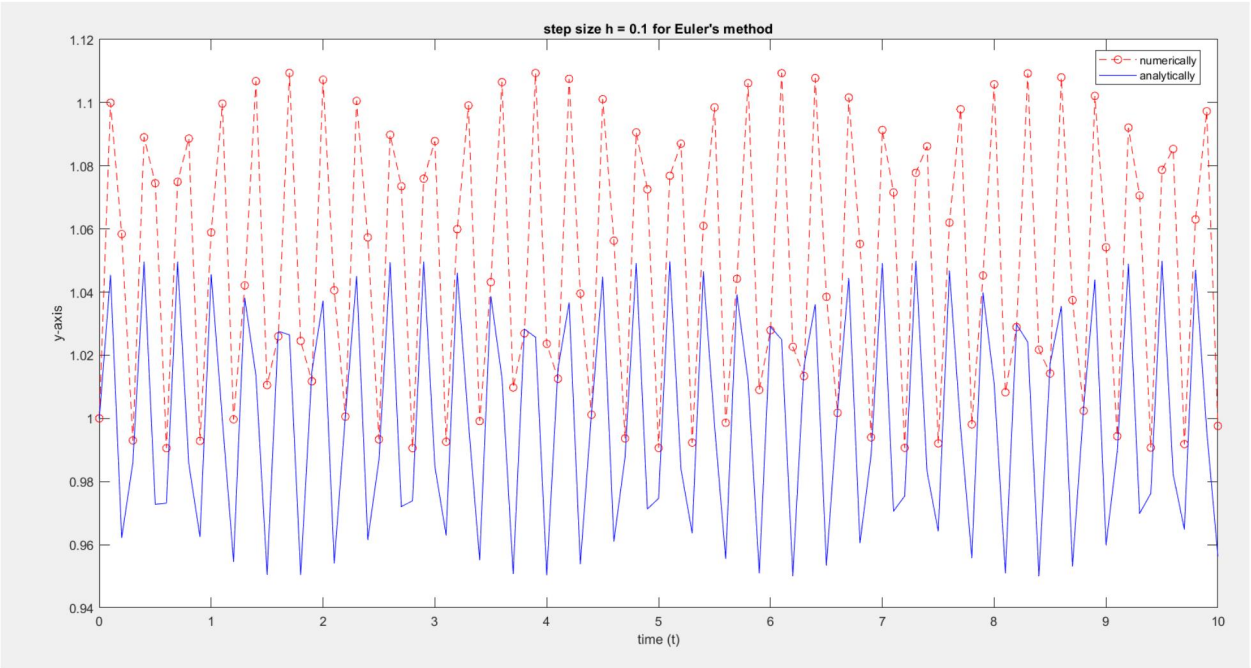
```
%exercise 6
syms y(t)
eqn = diff(y,t) == cos(20*t);
cond = y(0) == 1;
ySol(t) = dsolve(eqn, cond);
t = 0:0.01:10;
plot(t, ySol(t), 'b--o') %Use a blue line
xlabel('t'), ylabel('y') %label the axises

%cosineode.m code
function dydt = cosineode(t, y)
dydt = cos(20*t);

% RK4 code
t0 = 0; %Set initial time
tend = 10; %Set end time
for i = 1:3
h = 0.01*(10.^i);  %for loop incrementing step size
y0 = 1; %Set initial condition to y(0) = 1
out = RK4(@cosineode,t0,h,tend,y0); % Solve numerically here.
t = 0:h:10;
figure
plot(t, out, 'r--o', t, sin(20*t)/20 + 1, 'b-')

%Exercise 6 Euler method
t0 =0;
tend = 10;
for i = 0:2
   h = 0.01*(10.^i);
y0 =1;
```

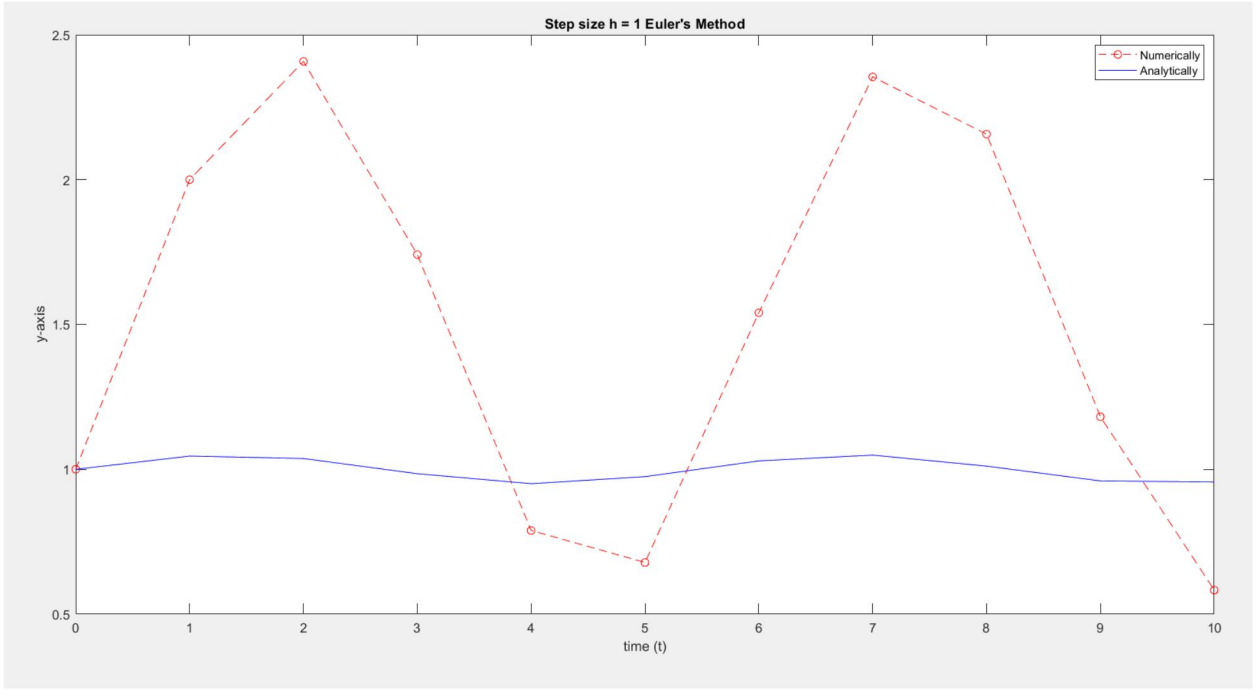
```
out = eul(@cosineode, t0, h, tend, y0);
t = 0:h:10;
plot(t, out, 'r--o', t, sin(20*t)/20 + 1, 'b-')
```

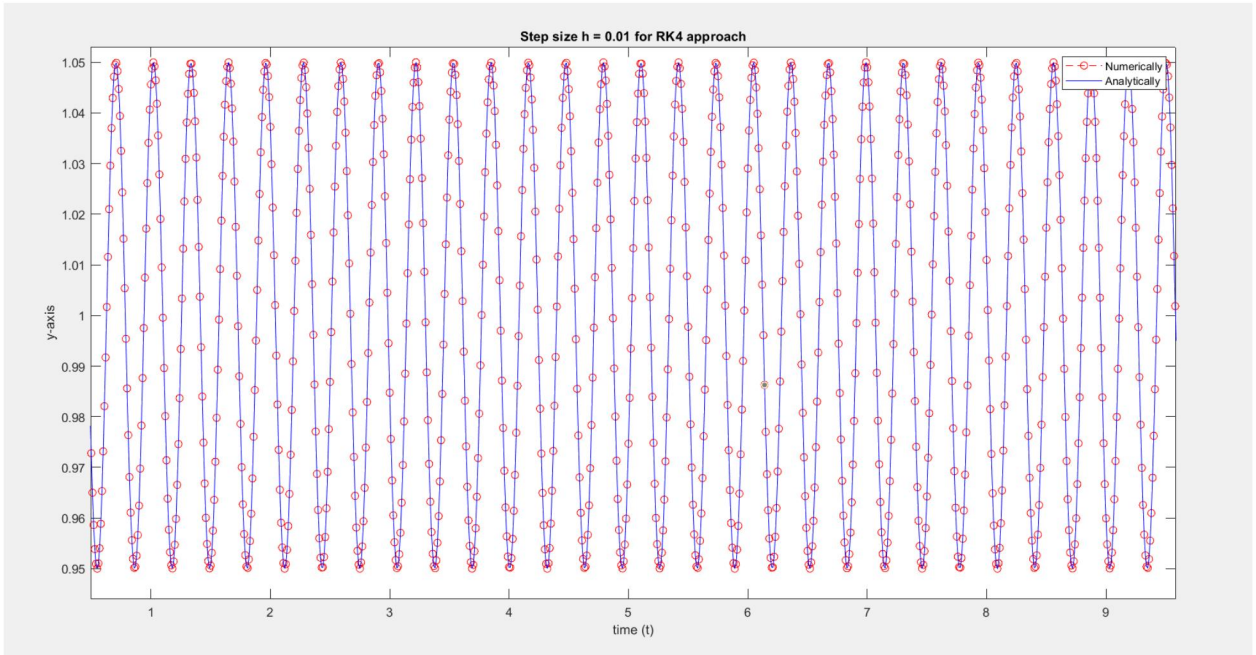
```
function dNdt=logGrowth(t,N,r, K)
%function dNdt=logGrowth(t,N)
%
% logGrowth gives the growth rate of a population of size N at time t
% Usage:dNdt=logGrowth(t,N)
dNdt =r*N.*(1-N/K);
```

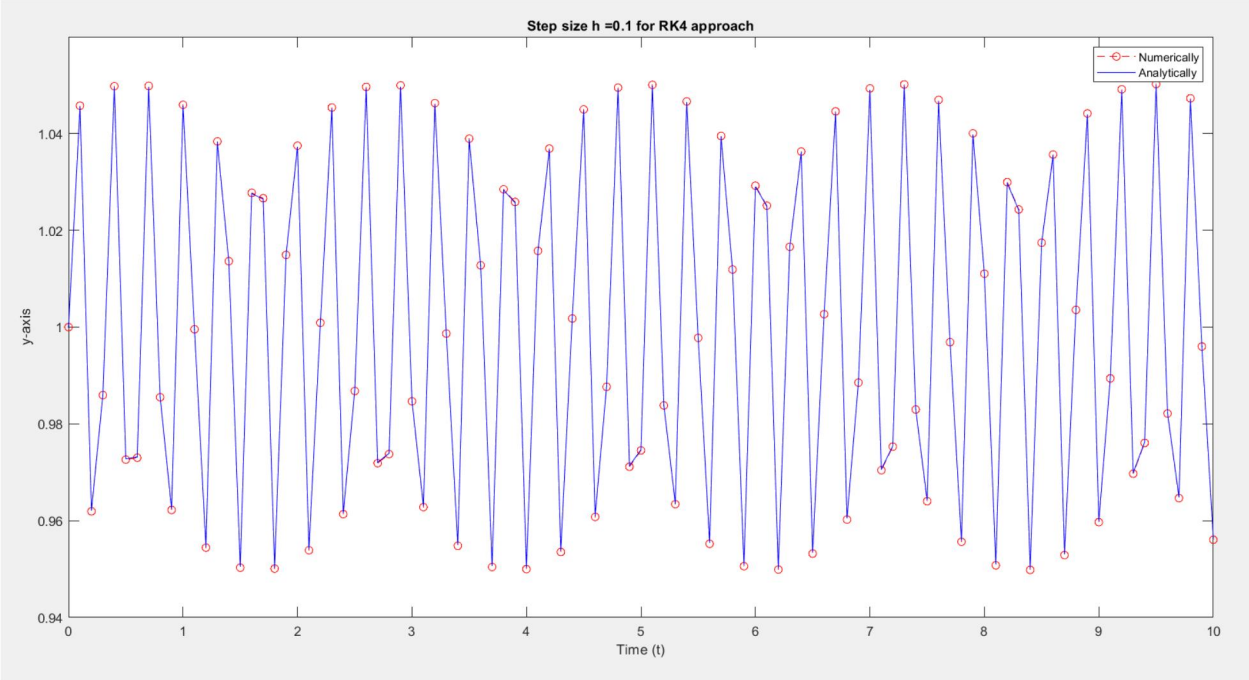


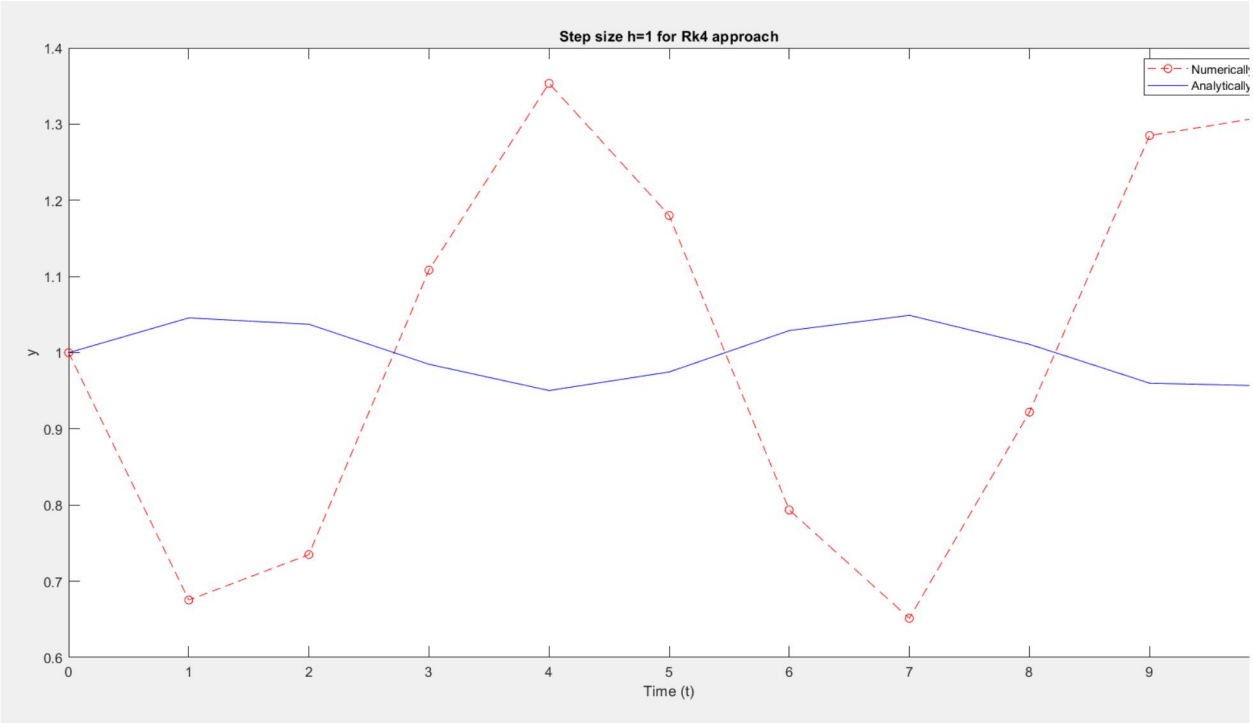

step size h = 0.01 for Euler's method
Numerically
Analytically
y-axis
time (t)
1.06
1.04
1.02
1
0.98
0.96
0.94
0
1
2
3
4
5
6
7
8
9
10


step size h = 0.1 for Euler's method
numerically
analytically
y-axis
time (t)
1.12
1.1
1.08
1.06
1.04
1.02
1
0.98
0.96
0.94
0
1
2
3
4
5
6
7
8
9
10


Step size h = 1 Euler's Method
Numerically
Analytically
y-axis
time (t)
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
9
10


Step size h = 0.01 for RK4 approach
Numerically
Analytically
y-axis
time (t)
1.05
1.04
1.03
1.02
1.01
1
0.99
0.98
0.97
0.96
0.95
1
2
3
4
5
6
7
8
9





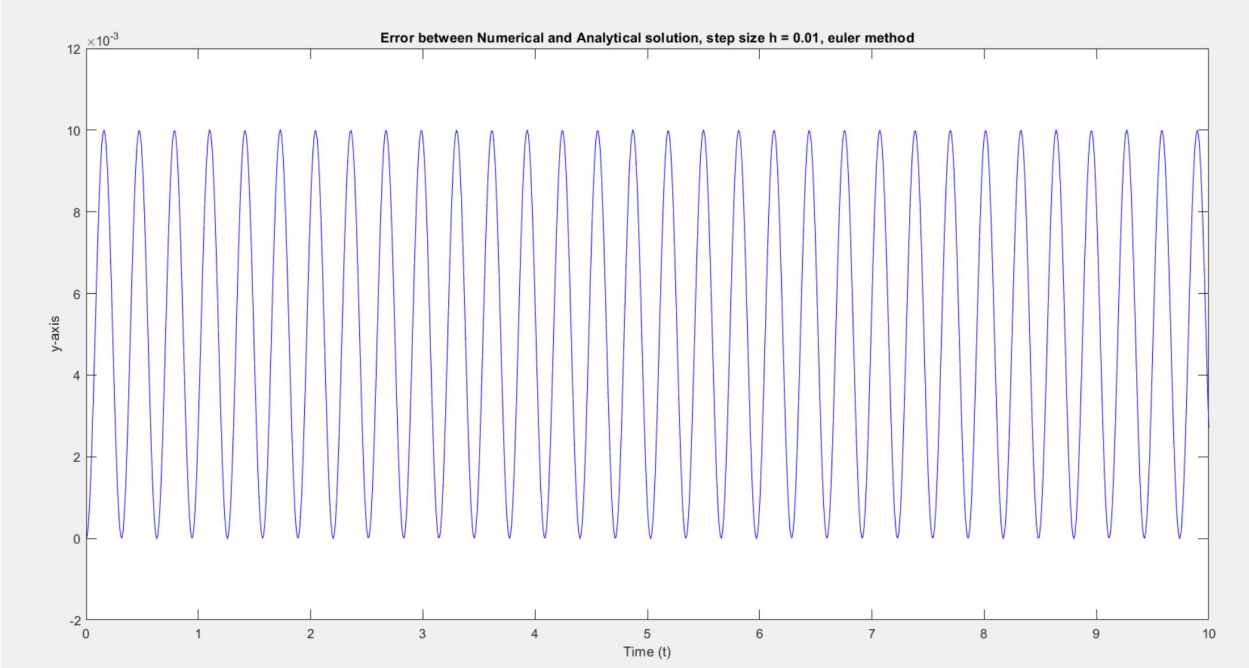
-max error question

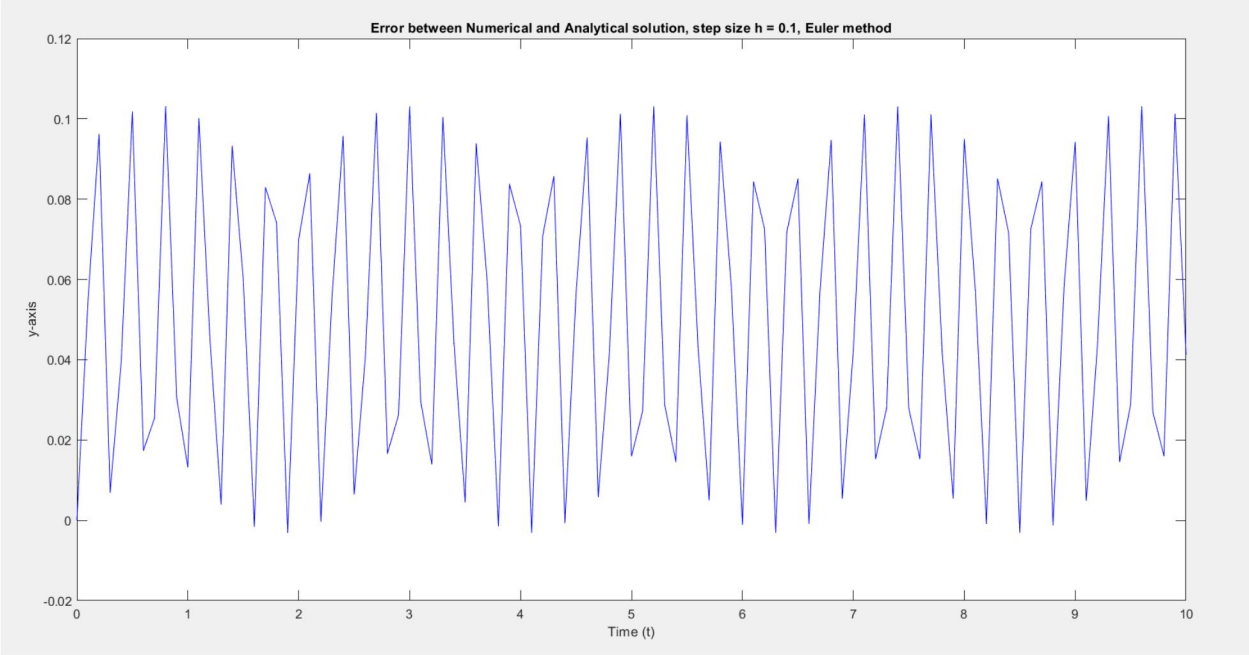
```
%Exercise 6 Euler method
t0 =0;
tend = 10;
for i = 0:2
   h = 0.01*(10.^i);
y0 =1;
```

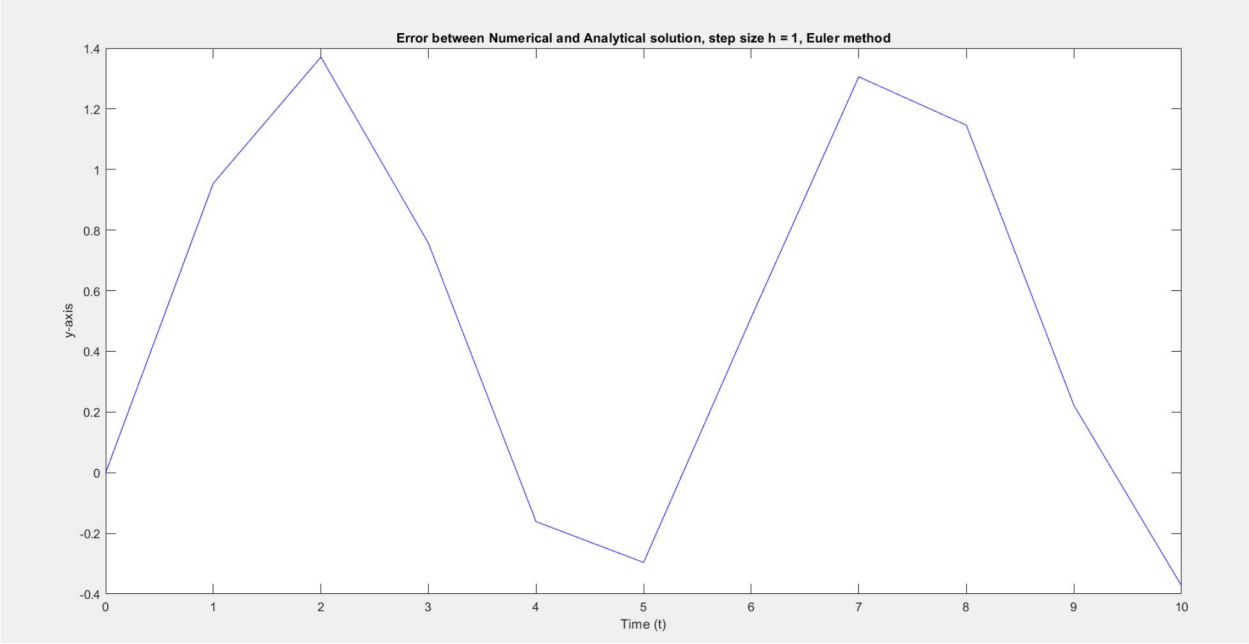
```
out = eul(@cosineode, t0, h, tend, y0);
t = 0:h:10;
%plot(t, out, 'r--o', t, sin(20*t)/20 + 1, 'b-')
figure
error = (out)'-(sin(20*t)/20 + 1);
plot(t, error, 'b-')
max_value= max(max(error));
fprintf('%i\n', max_value);
end
```

```
>> exercise6eul
1.000272e-02
1.031059e-01
1.370826e+00
>>
```

These are the respective errors between numerical and analytical solutions, with ascending order of step sizes for euler method.



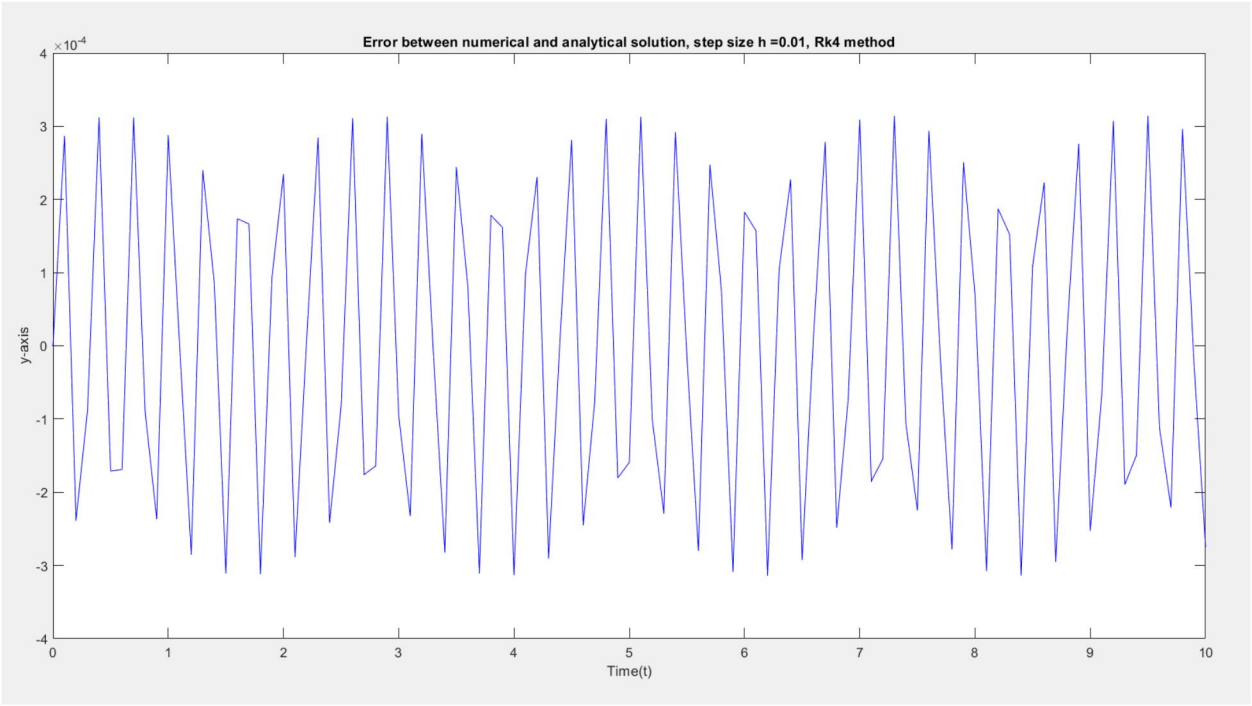


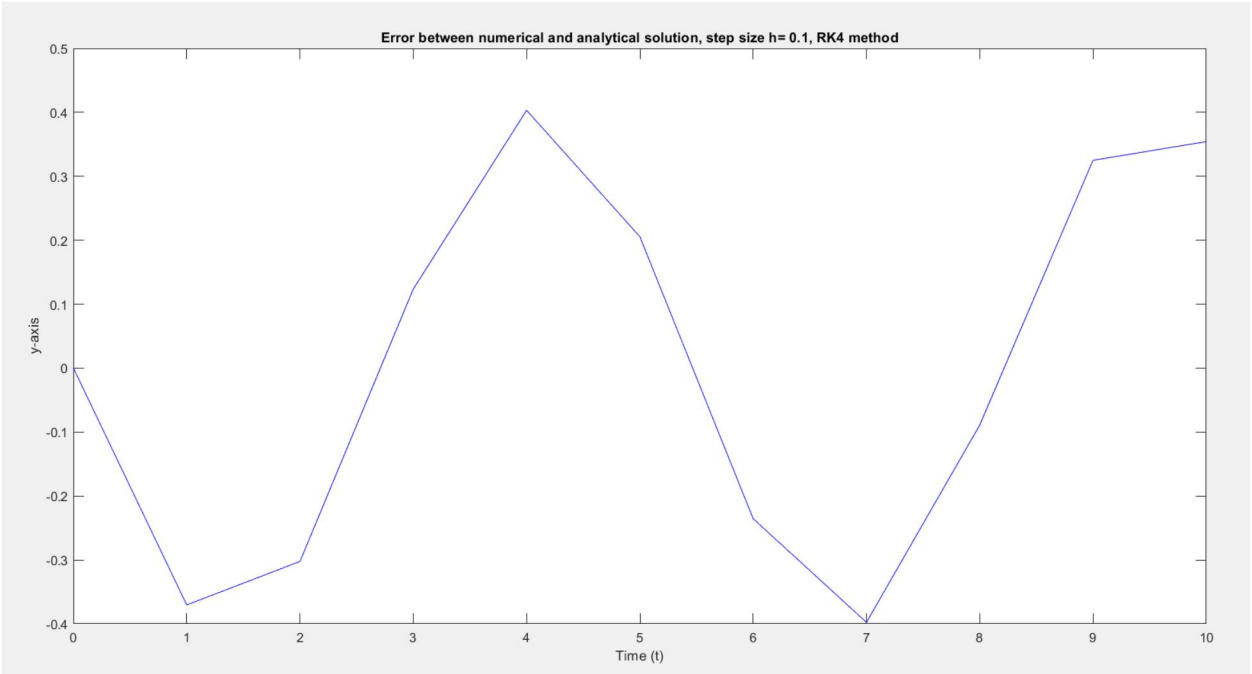


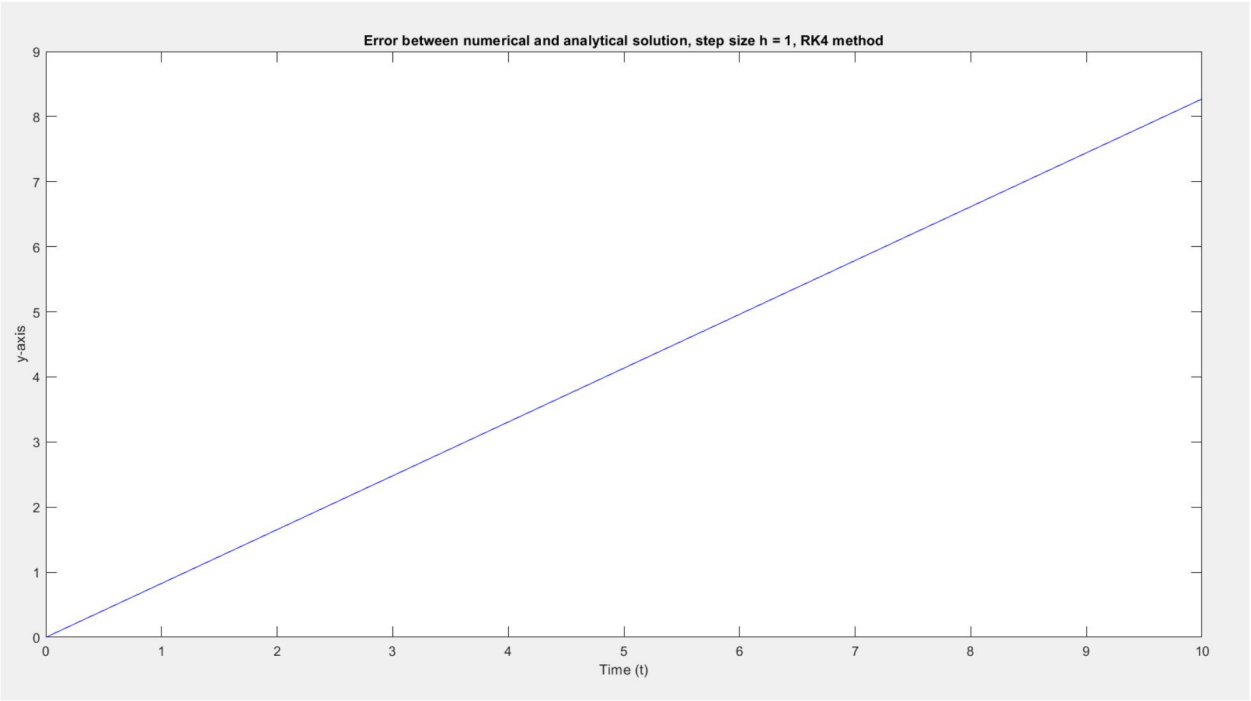
Rk4 figures

```
>> exercise6rungekutta
3.140212e-04
4.031837e-01
8.271103e+00
```

These are the respective errors between numerical and analytical solutions, with ascending order of step sizes for Rk4 method.


Error between numerical and analytical solution, step size h =0.01, Rk4 method
×10^-4
y-axis
Time(t)


Error between numerical and analytical solution, step size h= 0.1, RK4 method
y-axis
Time (t)



The relationship between step size and error is that as step size decreases, the error also decreases. This is similar to the definition of a derivative when we infinitely approach the segment.

6 Ex.6 **30 / 30**

✓ **- 0 pts** *Correct*

6.1 - 7 pts

- 1 pts Step size is `0.01` instead of `0.1`

- 3 pts Need to plot analytical solution

- 7 pts incomplete

6.2 - 10 pts

- 10 pts incomplete

- 5 pts attempted, but incorrect

6.3 - 13 pts

- 3 pts Need to calculate max error and describe the relationship between error and step size

- 4 pts General method correct, but plot is incorrect. Because the output vector of analytical solution and numerical solution have different dimension, you should transfer one of the vectors and then subtract.

- 13 pts third part of exercise not complete.

- 10 pts for error, need to plot for each case, max error calculation not correct ("container" should be vector not a value), and need to explain the relationship between step size and error.

- 2 pts Relationship incorrect

- 3 pts Need axes and titles on plots