EE_105_021_23W Lab 2

Kiana Tristan

TOTAL POINTS

100 / 100

QUESTION 1

Section 1 10 pts

1.1 1.a) **4 / 4**

✓ **- 0 pts** *Correct*

1.2 1.b) **3 / 3**

✓ **- 0 pts** *Correct*

1.3 1.c) **3 / 3**

✓ **- 0 pts** *Correct*

QUESTION 2

Section 2 35 pts

2.1 2.a) **5 / 5**

✓ **- 0 pts** *Correct*

2.2 2.b) **5 / 5**

✓ **- 0 pts** *Correct*

2.3 2.c) **10 / 10**

✓ **- 0 pts** *Correct*

2.4 2.d) **15 / 15**

✓ **- 0 pts** *Correct*

QUESTION 3

Section 3 10 pts

3.1 3.b) **5 / 5**

✓ **- 0 pts** *Correct*

3.2 3.c) **5 / 5**

✓ **- 0 pts** *Correct*

QUESTION 4

4 Section 4 **10 / 10**

✓ **- 0 pts** *Correct*

QUESTION 5

5 Section 5 **10 / 10**

✓ **- 0 pts** *Correct*

QUESTION 6

6 Prelab (points assigned based on Prelab completion prior to lab) **25 / 25**

✓ **- 0 pts** *All questions complete before lab*

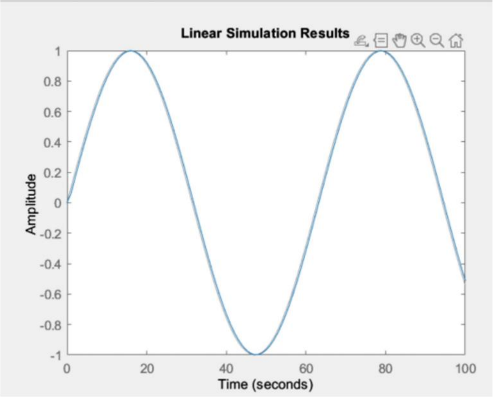
Lab #2

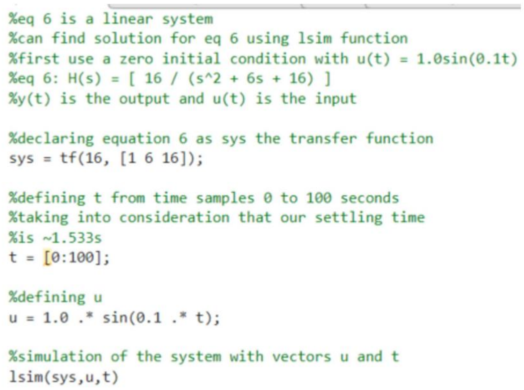
Thursday, January 26, 2023 6:09 PM

Kiana Tristan
EE 105
Lab 2: Euler's Approximation
1/26/23

Part 8.1:·

A)

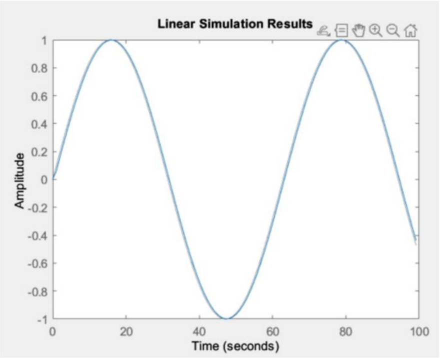


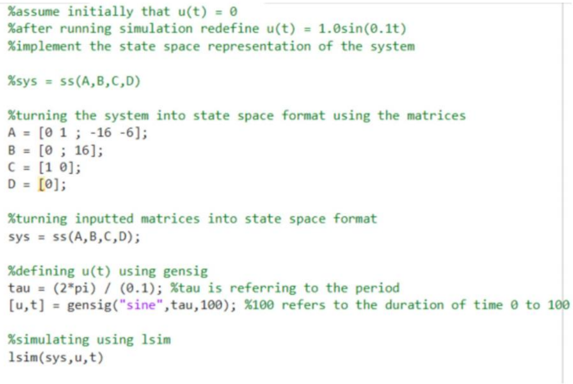


```
%eq 6 is a linear system
%can find solution for eq 6 using lsim function
%first use a zero initial condition with u(t) = 1.0sin(0.1t)
%eq 6: H(s) = [ 16 / (s^2 + 6s + 16) ]
%y(t) is the output and u(t) is the input

%declaring equation 6 as sys the transfer function
sys = tf(16, [1 6 16]);

%defining t from time samples 0 to 100 seconds
%taking into consideration that our settling time
%is ~1.533s
t = [0:100];

%defining u
u = 1.0 .* sin(0.1 .* t);

%simulation of the system with vectors u and t
lsim(sys,u,t)
```

Here, we can see how we can use the matlab function lsim to simulate the graph of the given transfer function, after defining the vector u and the time samples t. Here, we defined t to be from 0 to 100 seconds taking into consideration that out settling time is 1.533 seconds.

B and C)





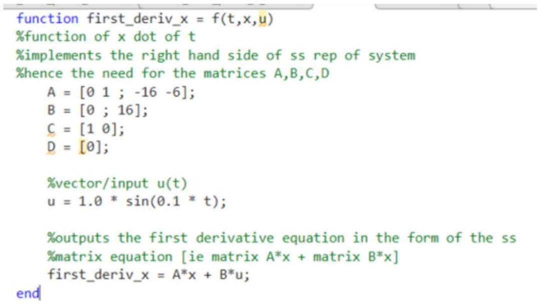
```
%assume initially that u(t) = 0
%after running simulation redefine u(t) = 1.0sin(0.1t)
%implement the state space representation of the system

%sys = ss(A,B,C,D)

%turning the system into state space format using the matrices
A = [0 1 ; -16 -6];
B = [0 ; 16];
C = [1 0];
D = [0];

%turning inputted matrices into state space format
sys = ss(A,B,C,D);

%defining u(t) using gensig
tau = (2*pi) / (0.1); %tau is referring to the period
[u,t] = gensig("sine",tau,100); %100 refers to the duration of time 0 to 100

%simulating using lsim
lsim(sys,u,t)
```

[part b] Using the gensig function we can now see a graph similar to the previous one that uses a time vector tau to evaluate the input u(t).

[part c] We take the values of the matrices found in the prelab (A,B,C,D) and turn them from the state space format to the sys format that the lsim function requires.

Part 8.2:

A) M-file called f.m



```
function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

This is the function for the right hand side of the ss rep of the sys. We take the matrices and create a vector u for the input u(t). Then do matrix multiplication with x for A*x and then add that to matrix B multiplied with u.

B and C)

1.1 1.a) **4 / 4**

✓ **- 0 pts** *Correct*

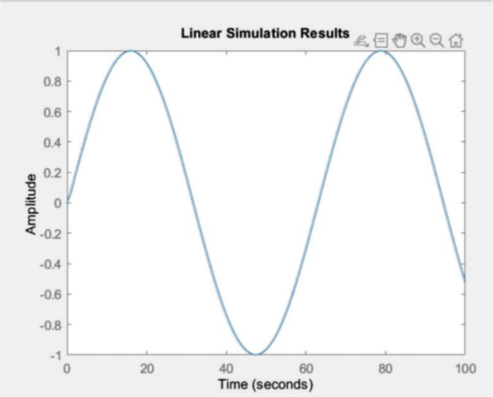
Lab #2

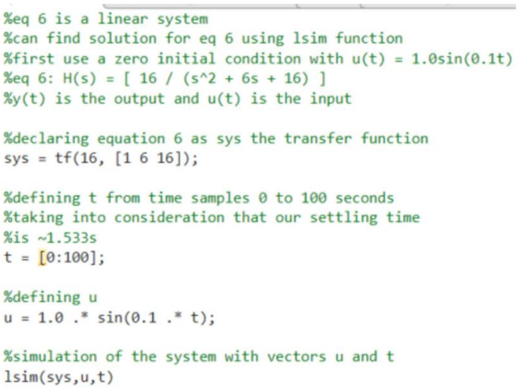
Thursday, January 26, 2023 6:09 PM

Kiana Tristan
EE 105
Lab 2: Euler's Approximation
1/26/23

Part 8.1:

A)

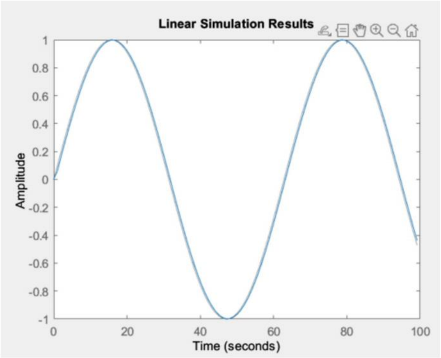


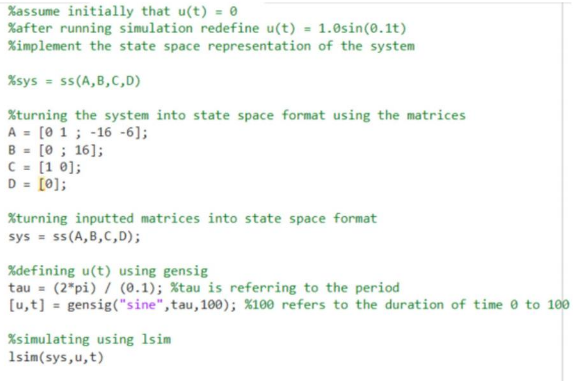


```
%eq 6 is a linear system
%can find solution for eq 6 using lsim function
%first use a zero initial condition with u(t) = 1.0sin(0.1t)
%eq 6: H(s) = [ 16 / (s^2 + 6s + 16) ]
%y(t) is the output and u(t) is the input

%declaring equation 6 as sys the transfer function
sys = tf(16, [1 6 16]);

%defining t from time samples 0 to 100 seconds
%taking into consideration that our settling time
%is ~1.533s
t = [0:100];

%defining u
u = 1.0 .* sin(0.1 .* t);

%simulation of the system with vectors u and t
lsim(sys,u,t)
```

Here, we can see how we can use the matlab function lsim to simulate the graph of the given transfer function, after defining the vector u and the time samples t. Here, we defined t to be from 0 to 100 seconds taking into consideration that out settling time is 1.533 seconds.

B and C)





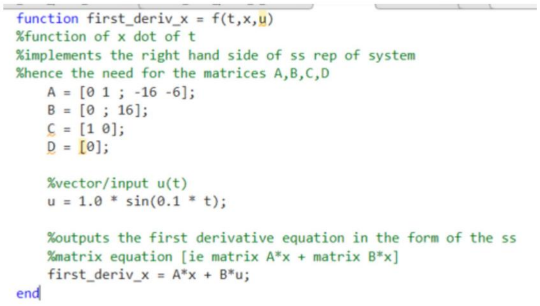
```
%assume initially that u(t) = 0
%after running simulation redefine u(t) = 1.0sin(0.1t)
%implement the state space representation of the system

%sys = ss(A,B,C,D)

%turning the system into state space format using the matrices
A = [0 1 ; -16 -6];
B = [0 ; 16];
C = [1 0];
D = [0];

%turning inputted matrices into state space format
sys = ss(A,B,C,D);

%defining u(t) using gensig
tau = (2*pi) / (0.1); %tau is referring to the period
[u,t] = gensig("sine",tau,100); %100 refers to the duration of time 0 to 100

%simulating using lsim
lsim(sys,u,t)
```

[part b] Using the gensig function we can now see a graph similar to the previous one that uses a time vector tau to evaluate the input u(t).

[part c] We take the values of the matrices found in the prelab (A,B,C,D) and turn them from the state space format to the sys format that the lsim function requires.

Part 8.2:

A) M-file called f.m



```
function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

This is the function for the right hand side of the ss rep of the sys. We take the matrices and create a vector u for the input u(t). Then do matrix multiplication with x for A*x and then add that to matrix B multiplied with u.

B and C)

1.2 1.b) **3 / 3**

✓ **- 0 pts** *Correct*

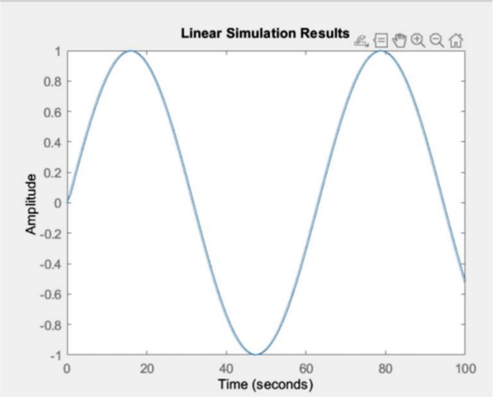
Lab #2

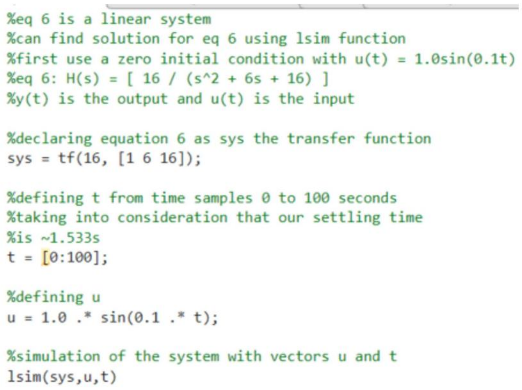
Thursday, January 26, 2023 6:09 PM

Kiana Tristan
EE 105
Lab 2: Euler's Approximation
1/26/23

Part 8.1:·

A)

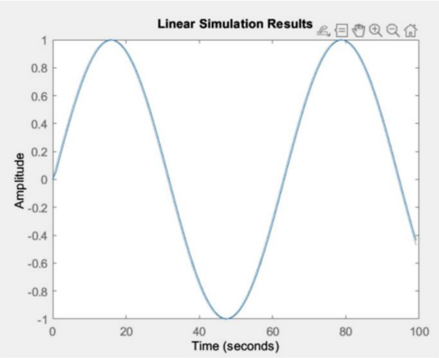


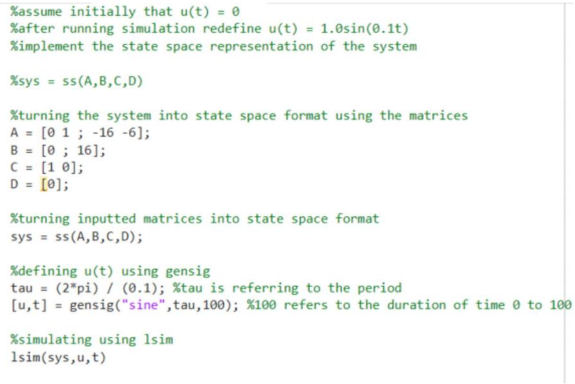


```
%eq 6 is a linear system
%can find solution for eq 6 using lsim function
%first use a zero initial condition with u(t) = 1.0sin(0.1t)
%eq 6: H(s) = [ 16 / (s^2 + 6s + 16) ]
%y(t) is the output and u(t) is the input

%declaring equation 6 as sys the transfer function
sys = tf(16, [1 6 16]);

%defining t from time samples 0 to 100 seconds
%taking into consideration that our settling time
%is ~1.533s
t = [0:100];

%defining u
u = 1.0 .* sin(0.1 .* t);

%simulation of the system with vectors u and t
lsim(sys,u,t)
```

Here, we can see how we can use the matlab function lsim to simulate the graph of the given transfer function, after defining the vector u and the time samples t. Here, we defined t to be from 0 to 100 seconds taking into consideration that out settling time is 1.533 seconds.

B and C)





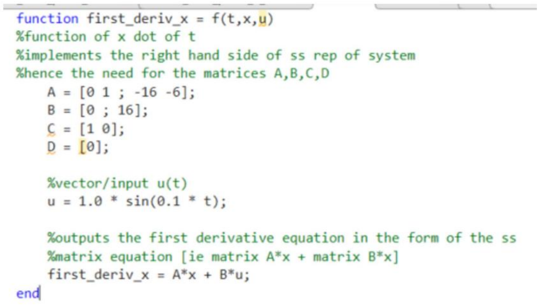
```
%assume initially that u(t) = 0
%after running simulation redefine u(t) = 1.0sin(0.1t)
%implement the state space representation of the system

%sys = ss(A,B,C,D)

%turning the system into state space format using the matrices
A = [0 1 ; -16 -6];
B = [0 ; 16];
C = [1 0];
D = [0];

%turning inputted matrices into state space format
sys = ss(A,B,C,D);

%defining u(t) using gensig
tau = (2*pi) / (0.1); %tau is referring to the period
[u,t] = gensig("sine",tau,100); %100 refers to the duration of time 0 to 100

%simulating using lsim
lsim(sys,u,t)
```

[part b] Using the gensig function we can now see a graph similar to the previous one that uses a time vector tau to evaluate the input u(t).

[part c] We take the values of the matrices found in the prelab (A,B,C,D) and turn them from the state space format to the sys format that the lsim function requires.

Part 8.2:

A) M-file called f.m



```
function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

This is the function for the right hand side of the ss rep of the sys. We take the matrices and create a vector u for the input u(t). Then do matrix multiplication with x for A*x and then add that to matrix B multiplied with u.

B and C)

1.3 1.c) **3 / 3**

✓ **- 0 pts** *Correct*

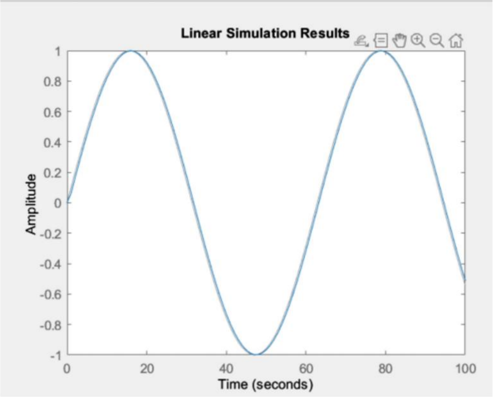
Lab #2

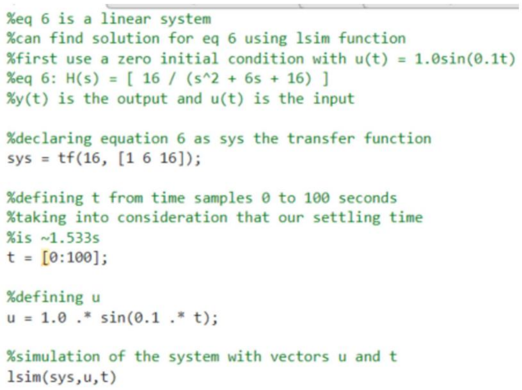
Thursday, January 26, 2023 6:09 PM

Kiana Tristan
EE 105
Lab 2: Euler's Approximation
1/26/23

Part 8.1:·

A)

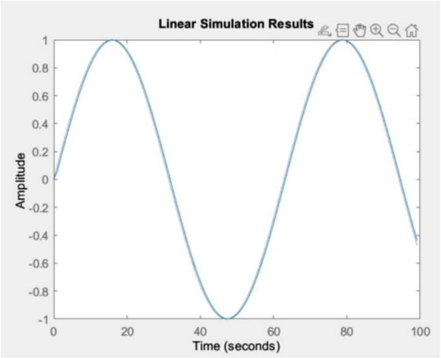


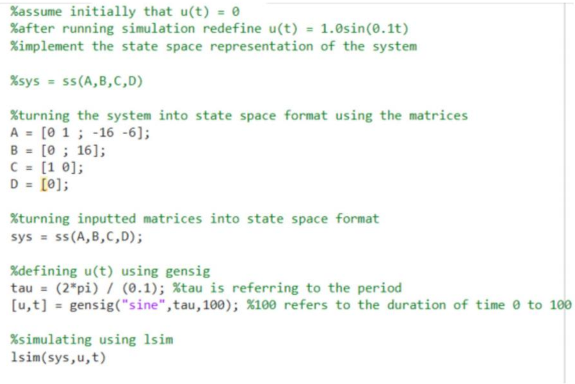


```
%eq 6 is a linear system
%can find solution for eq 6 using lsim function
%first use a zero initial condition with u(t) = 1.0sin(0.1t)
%eq 6: H(s) = [ 16 / (s^2 + 6s + 16) ]
%y(t) is the output and u(t) is the input

%declaring equation 6 as sys the transfer function
sys = tf(16, [1 6 16]);

%defining t from time samples 0 to 100 seconds
%taking into consideration that our settling time
%is ~1.533s
t = [0:100];

%defining u
u = 1.0 .* sin(0.1 .* t);

%simulation of the system with vectors u and t
lsim(sys,u,t)
```

Here, we can see how we can use the matlab function lsim to simulate the graph of the given transfer function, after defining the vector u and the time samples t. Here, we defined t to be from 0 to 100 seconds taking into consideration that out settling time is 1.533 seconds.

B and C)





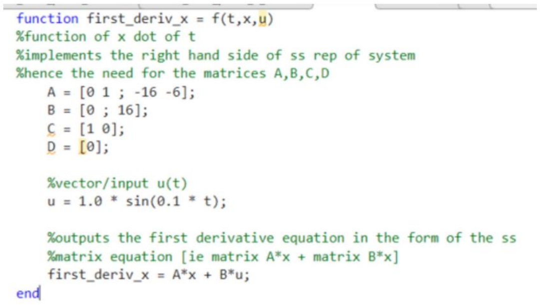
```
%assume initially that u(t) = 0
%after running simulation redefine u(t) = 1.0sin(0.1t)
%implement the state space representation of the system

%sys = ss(A,B,C,D)

%turning the system into state space format using the matrices
A = [0 1 ; -16 -6];
B = [0 ; 16];
C = [1 0];
D = [0];

%turning inputted matrices into state space format
sys = ss(A,B,C,D);

%defining u(t) using gensig
tau = (2*pi) / (0.1); %tau is referring to the period
[u,t] = gensig("sine",tau,100); %100 refers to the duration of time 0 to 100

%simulating using lsim
lsim(sys,u,t)
```

[part b] Using the gensig function we can now see a graph similar to the previous one that uses a time vector tau to evaluate the input u(t).

[part c] We take the values of the matrices found in the prelab (A,B,C,D) and turn them from the state space format to the sys format that the lsim function requires.

Part 8.2:

A) M-file called f.m



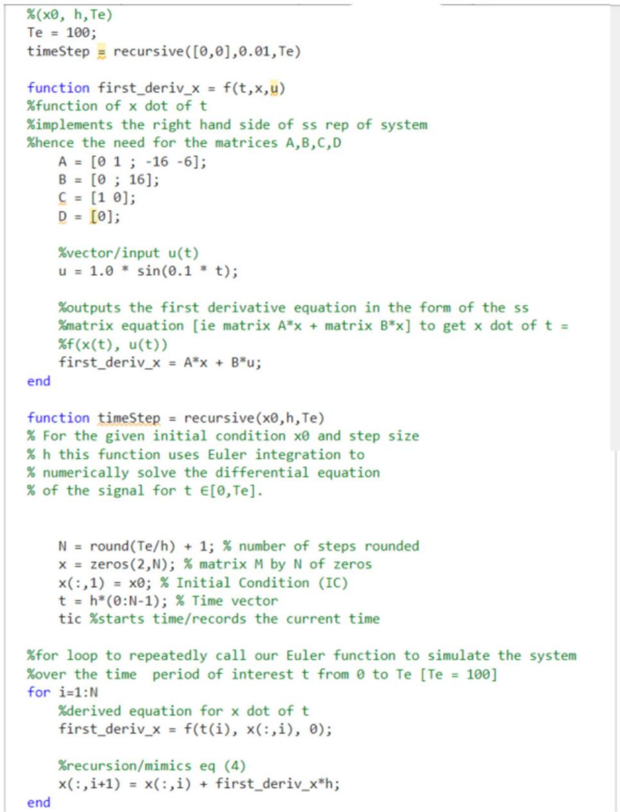
```
function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

This is the function for the right hand side of the ss rep of the sys. We take the matrices and create a vector u for the input u(t). Then do matrix multiplication with x for A*x and then add that to matrix B multiplied with u.

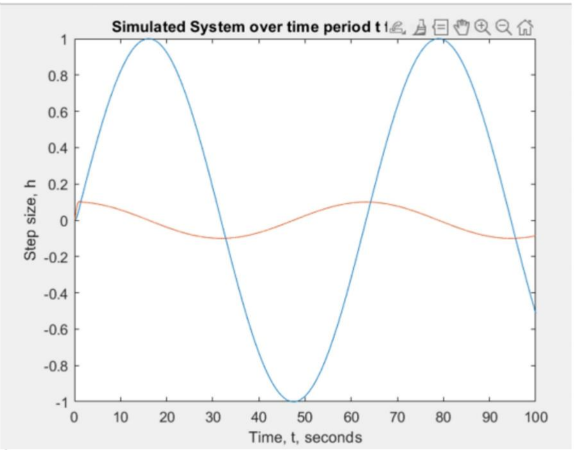
B and C)

2.1 2.a) **5 / 5**

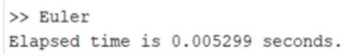
✓ **- 0 pts** *Correct*



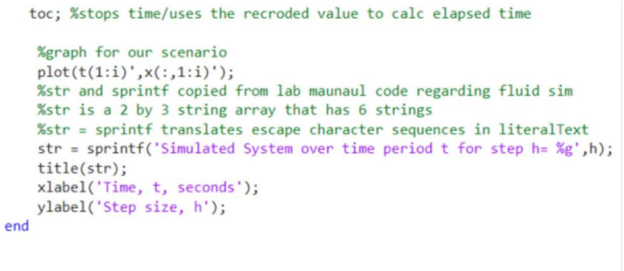
```
%(x0, h,Te)
Te = 100;
timeStep = recursive([0,0],0.01,Te)

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x] to get x dot of t =
    %f(x(t), u(t))
    first_deriv_x = A*x + B*u;
end

function timeStep = recursive(x0,h,Te)
% For the given initial condition x0 and step size
% h this function uses Euler integration to
% numerically solve the differential equation
% of the signal for t ∈[0,Te].


    N = round(Te/h) + 1; % number of steps rounded
    x = zeros(2,N); % matrix M by N of zeros
    x(:,1) = x0; % Initial Condition (IC)
    t = h*(0:N-1); % Time vector
    tic %starts time/records the current time

%for loop to repeatedly call our Euler function to simulate the system
%over the time  period of interest t from 0 to Te [Te = 100]
for i=1:N
    %derived equation for x dot of t
    first_deriv_x = f(t(i), x(:,i), 0);

    %recursion/mimics eq (4)
    x(:,i+1) = x(:,i) + first_deriv_x*h;
end
```



The m file called Euler.m and the for loop that repeatedly calls Euler to simulate the system over the time period from [0,Te] where here Te = 100. Here we also have initial condition set as [0,0] and h set as 0.01. With these conditions, this results in our graph which is very similar to the part 1 graphs with an expanded secondary line in orange.

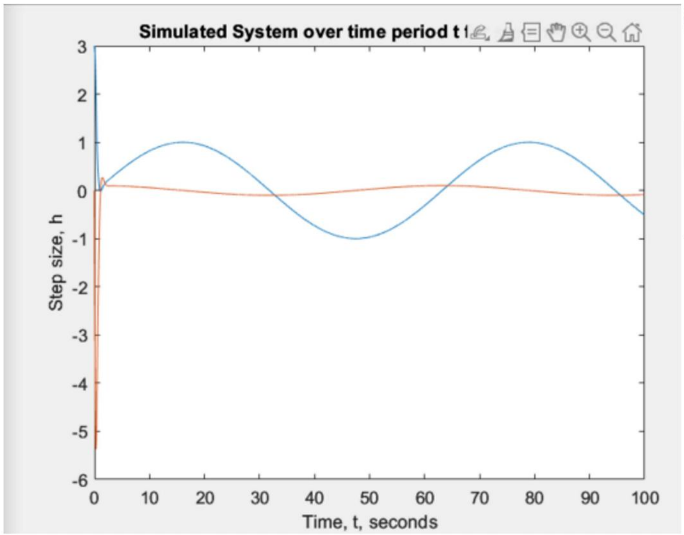


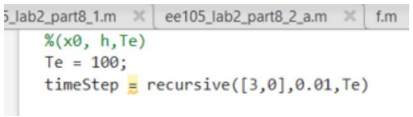
```
>> Euler
Elapsed time is 0.005299 seconds.
```



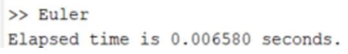
```
   toc; %stops time/uses the recroded value to calc elapsed time

    %graph for our scenario
    plot(t(1:i)',x(:,1:i)');
    %str and sprintf copied from lab maunaul code regarding fluid sim
    %str is a 2 by 3 string array that has 6 strings
    %str = sprintf translates escape character sequences in literalText
    str = sprintf('Simulated System over time period t for step h= %g',h);
    title(str);
    xlabel('Time, t, seconds');
    ylabel('Step size, h');
end
```

D)





```
%(x0, h,Te)
Te = 100;
timeStep = recursive([3,0],0.01,Te)
```

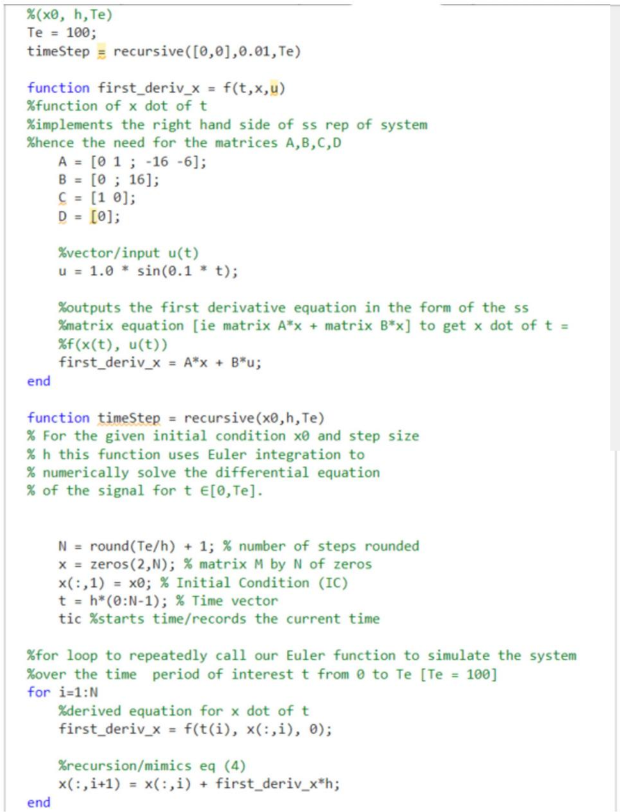


```
>> Euler
Elapsed time is 0.006580 seconds.
```

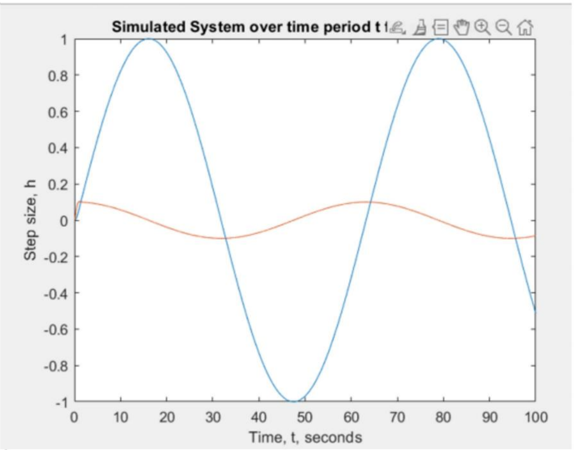
Here we have the initial condition for x0 = [3,0] with the same step size and Te = 100. We have a smaller amplitude and the graph doesn't appear to begin at [0,0] but at 3. Our elapsed time is also longer than the previous initial conditions of [0,0].

2.2 2.b) **5 / 5**

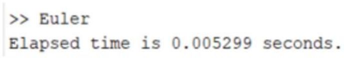
✓ **- 0 pts** *Correct*



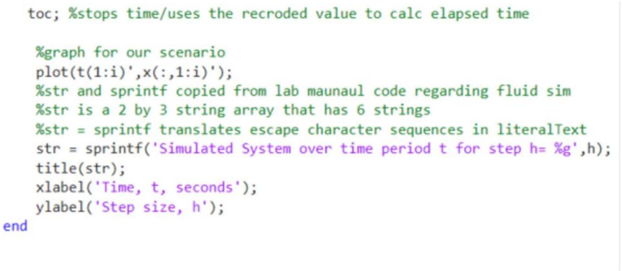
```
%(x0, h,Te)
Te = 100;
timeStep = recursive([0,0],0.01,Te)

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x] to get x dot of t =
    %f(x(t), u(t))
    first_deriv_x = A*x + B*u;
end

function timeStep = recursive(x0,h,Te)
% For the given initial condition x0 and step size
% h this function uses Euler integration to
% numerically solve the differential equation
% of the signal for t ∈[0,Te].


    N = round(Te/h) + 1; % number of steps rounded
    x = zeros(2,N); % matrix M by N of zeros
    x(:,1) = x0; % Initial Condition (IC)
    t = h*(0:N-1); % Time vector
    tic %starts time/records the current time

%for loop to repeatedly call our Euler function to simulate the system
%over the time  period of interest t from 0 to Te [Te = 100]
for i=1:N
    %derived equation for x dot of t
    first_deriv_x = f(t(i), x(:,i), 0);

    %recursion/mimics eq (4)
    x(:,i+1) = x(:,i) + first_deriv_x*h;
end
```



The m file called Euler.m and the for loop that repeatedly calls Euler to simulate the system over the time period from [0,Te] where here Te = 100. Here we also have initial condition set as [0,0] and h set as 0.01. With these conditions, this results in our graph which is very similar to the part 1 graphs with an expanded secondary line in orange.

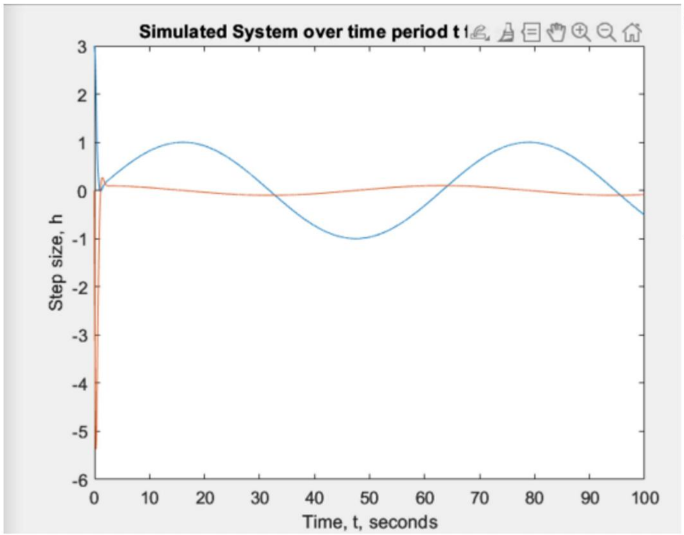


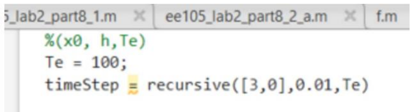
```
>> Euler
Elapsed time is 0.005299 seconds.
```



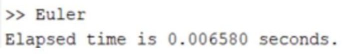
```
  toc; %stops time/uses the recroded value to calc elapsed time

   %graph for our scenario
   plot(t(1:i)',x(:,1:i)');
   %str and sprintf copied from lab maunaul code regarding fluid sim
   %str is a 2 by 3 string array that has 6 strings
   %str = sprintf translates escape character sequences in literalText
   str = sprintf('Simulated System over time period t for step h= %g',h);
   title(str);
   xlabel('Time, t, seconds');
   ylabel('Step size, h');
end
```

D)





```
%(x0, h,Te)
Te = 100;
timeStep = recursive([3,0],0.01,Te)
```

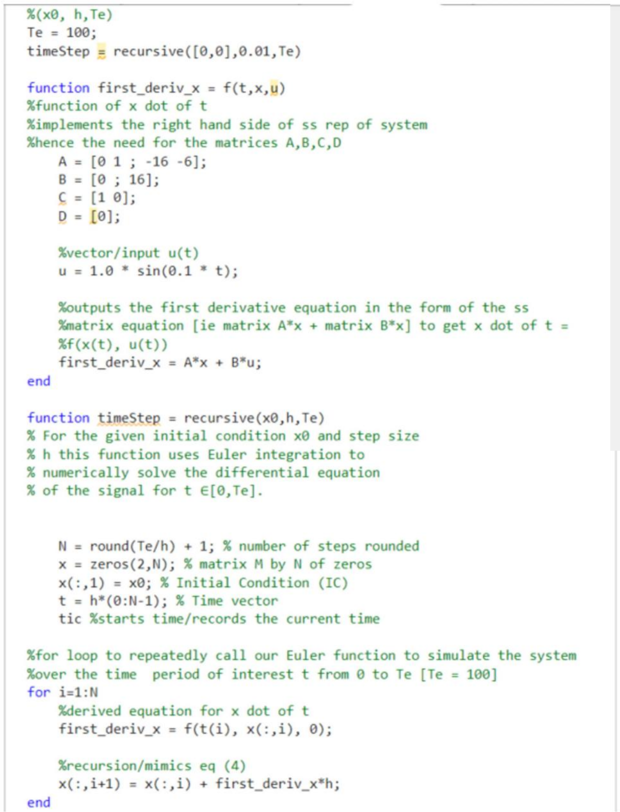


```
>> Euler
Elapsed time is 0.006580 seconds.
```

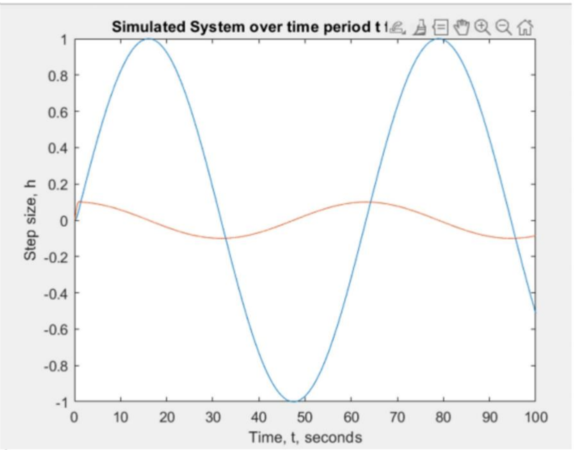
Here we have the initial condition for x0 = [3,0] with the same step size and Te = 100. We have a smaller amplitude and the graph doesn't appear to begin at [0,0] but at 3. Our elapsed time is also longer than the previous initial conditions of [0,0].

2.3 2.c) **10 / 10**

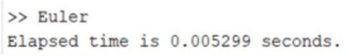
✓ **- 0 pts** *Correct*



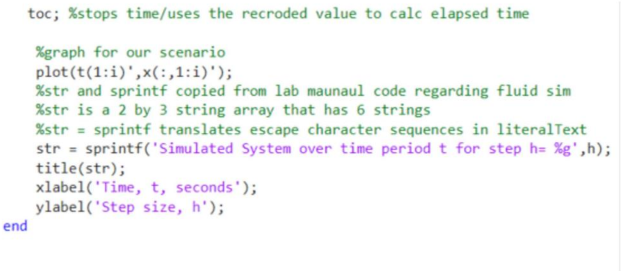
```
%(x0, h,Te)
Te = 100;
timeStep = recursive([0,0],0.01,Te)

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x] to get x dot of t =
    %f(x(t), u(t))
    first_deriv_x = A*x + B*u;
end

function timeStep = recursive(x0,h,Te)
% For the given initial condition x0 and step size
% h this function uses Euler integration to
% numerically solve the differential equation
% of the signal for t ∈[0,Te].


    N = round(Te/h) + 1; % number of steps rounded
    x = zeros(2,N); % matrix M by N of zeros
    x(:,1) = x0; % Initial Condition (IC)
    t = h*(0:N-1); % Time vector
    tic %starts time/records the current time

%for loop to repeatedly call our Euler function to simulate the system
%over the time  period of interest t from 0 to Te [Te = 100]
for i=1:N
    %derived equation for x dot of t
    first_deriv_x = f(t(i), x(:,i), 0);

    %recursion/mimics eq (4)
    x(:,i+1) = x(:,i) + first_deriv_x*h;
end
```



The m file called Euler.m and the for loop that repeatedly calls Euler to simulate the system over the time period from [0,Te] where here Te = 100. Here we also have initial condition set as [0,0] and h set as 0.01. With these conditions, this results in our graph which is very similar to the part 1 graphs with an expanded secondary line in orange.

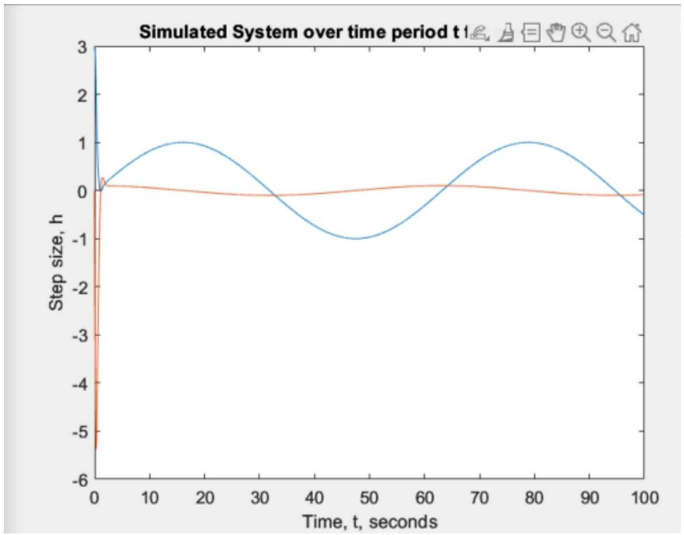


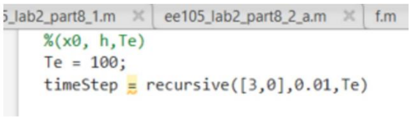
```
>> Euler
Elapsed time is 0.005299 seconds.
```



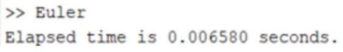
```
toc; %stops time/uses the recroded value to calc elapsed time

 %graph for our scenario
 plot(t(1:i)',x(:,1:i)');
 %str and sprintf copied from lab maunaul code regarding fluid sim
 %str is a 2 by 3 string array that has 6 strings
 %str = sprintf translates escape character sequences in literalText
 str = sprintf('Simulated System over time period t for step h= %g',h);
 title(str);
 xlabel('Time, t, seconds');
 ylabel('Step size, h');
end
```

D)



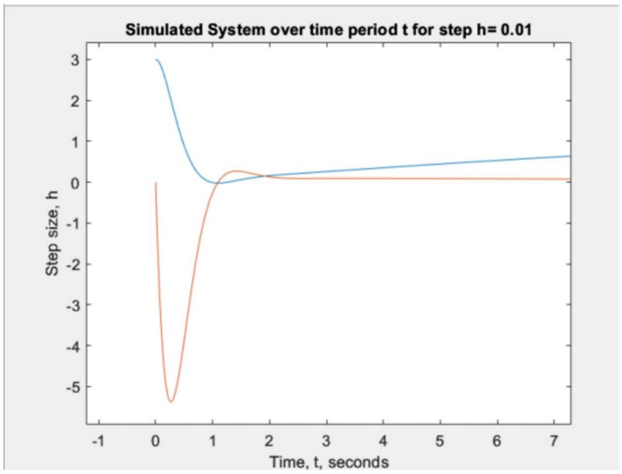


```
%(x0, h,Te)
Te = 100;
timeStep = recursive([3,0],0.01,Te)
```

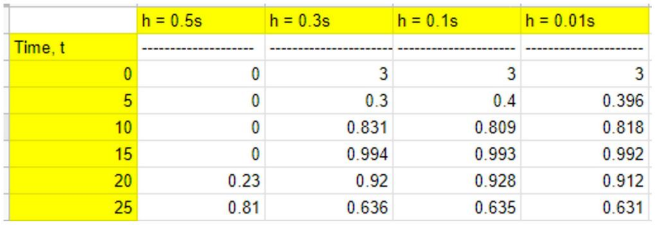


```
>> Euler
Elapsed time is 0.006580 seconds.
```

Here we have the initial condition for x0 = [3,0] with the same step size and Te = 100. We have a smaller amplitude and the graph doesn't appear to begin at [0,0] but at 3. Our elapsed time is also longer than the previous initial conditions of [0,0].



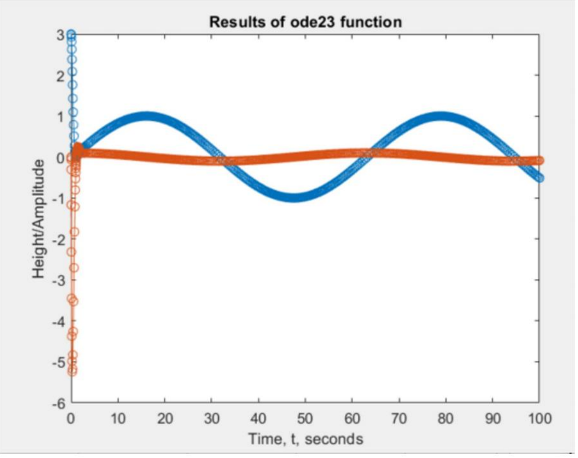
Here we can see the settling time more clearly and we can see that our settling time is close to 2. Comparing this with our prelab value for settling time of Ts = 1.533, we can see that our approximated settling time is fairly accurate.

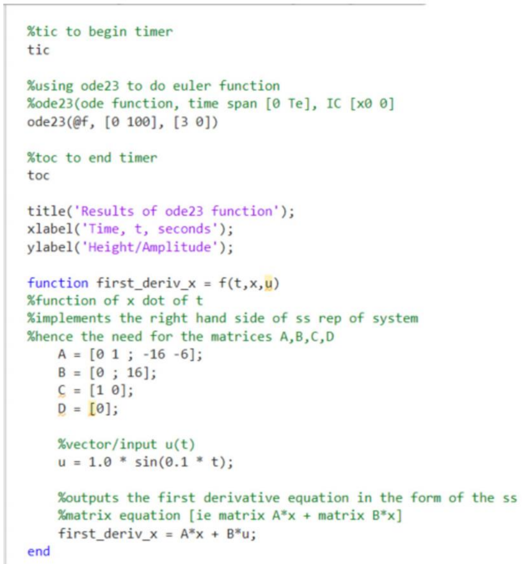


	h = 0.5s	h = 0.3s	h = 0.1s	h = 0.01s
Time, t	--------------------	---------------------	--------------------	---------------------
0	0	3	3	3
5	0	0.3	0.4	0.396
10	0	0.831	0.809	0.818
15	0	0.994	0.993	0.992
20	0.23	0.92	0.928	0.912
25	0.81	0.636	0.635	0.631

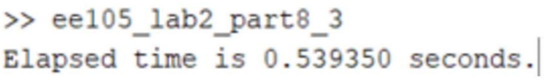
Here is our table of values for changing step size and time. We can see that as the step size decreases, we get more accurate values. I would assume that step size of h = 0.01s would be the most accurate due to the fact that it gives more "sample" selections compared to the larger step sizes. *Note: I changed Te from 100 to Te = 25 to make for easier sifting of values and kept x0 = [3,0].

Part 8.3:
A, B and C)





```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 100], [3 0])

%toc to end timer
toc

title('Results of ode23 function');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

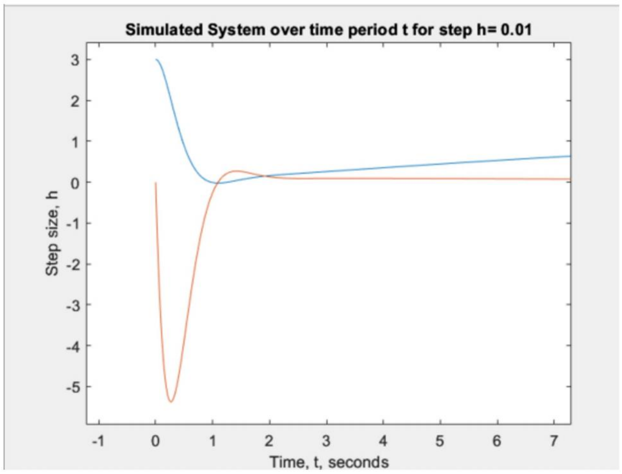


```
>> ee105_lab2_part8_3
Elapsed time is 0.539350 seconds.
```

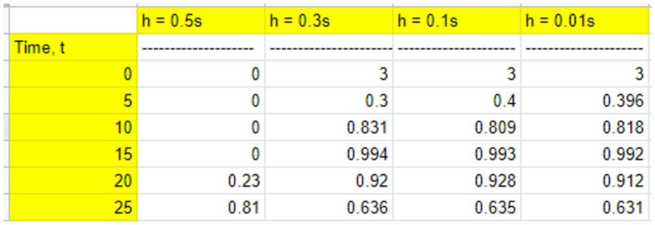
Here we can see we get a similar looking graph compared to part 2d. Our Te = 100 and our IC is [3.0, 0]. The lines of the graph appear thicker using the ode function. This is because there are dots that are connected by lines to create the simulation. For the code, it's much simpler to use ode23 compared to that of Euler. We just need to start the timer with tic, enter in our parameters for ode23, and end the timer with toc. It is also easier since the step size is automatically adjusted by the ode23 function itself. In terms of elapsed time, the ode23 function is slower compared to the elapsed time of the Euler function we wrote.

2.4 2.d) **15 / 15**

✓ **- 0 pts** *Correct*



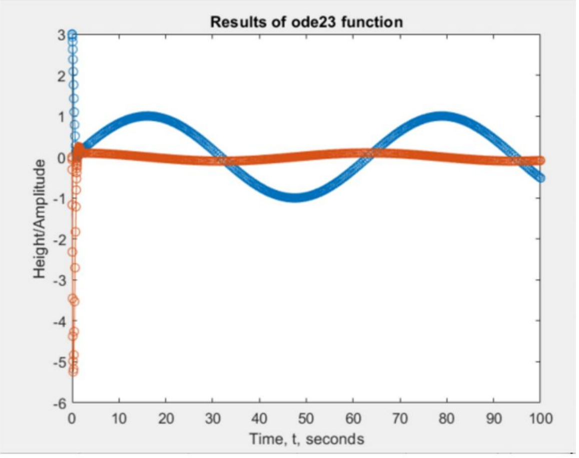
Here we can see the settling time more clearly and we can see that our settling time is close to 2. Comparing this with our prelab value for settling time of Ts = 1.533, we can see that our approximated settling time is fairly accurate.

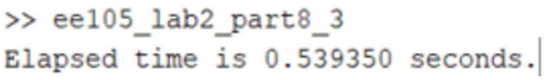


	h = 0.5s	h = 0.3s	h = 0.1s	h = 0.01s
Time, t	--------------------	----------------------	---------------------	---------------------
0	0	3	3	3
5	0	0.3	0.4	0.396
10	0	0.831	0.809	0.818
15	0	0.994	0.993	0.992
20	0.23	0.92	0.928	0.912
25	0.81	0.636	0.635	0.631

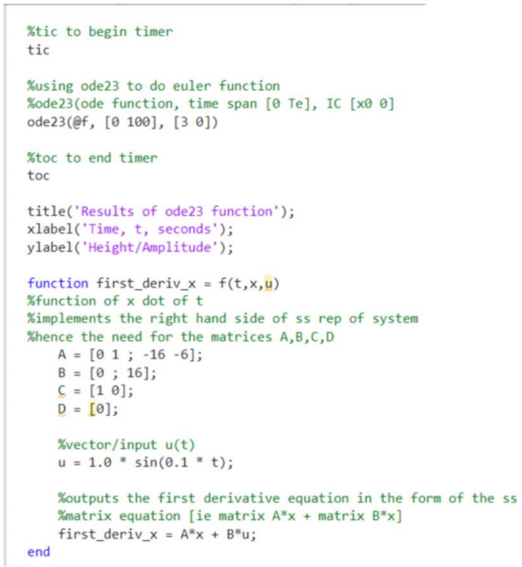
Here is our table of values for changing step size and time. We can see that as the step size decreases, we get more accurate values. I would assume that step size of h = 0.01s would be the most accurate due to the fact that it gives more "sample" selections compared to the larger step sizes. *Note: I changed Te from 100 to Te = 25 to make for easier sifting of values and kept x0 = [3,0].

Part 8.3:
A, B and C)





```
>> ee105_lab2_part8_3
Elapsed time is 0.539350 seconds.
```

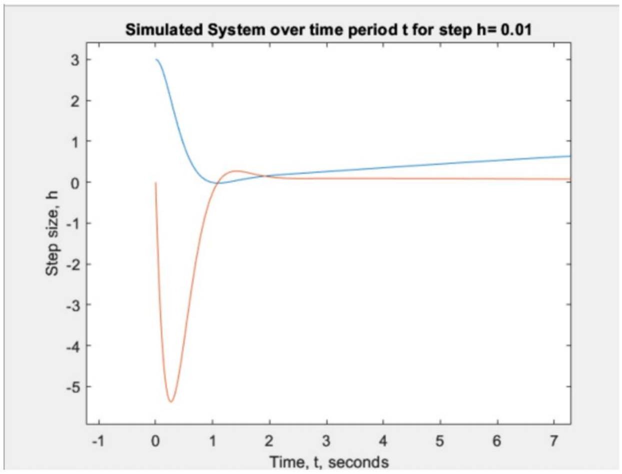


```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 100], [3 0])

%toc to end timer
toc

title('Results of ode23 function');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

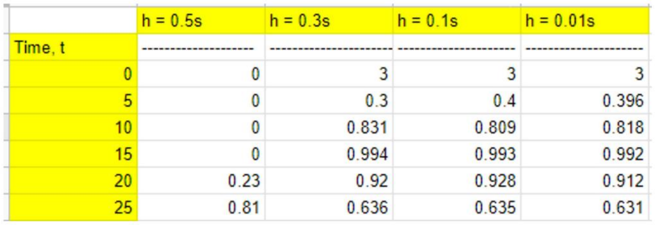
Here we can see we get a similar looking graph compared to part 2d. Our Te = 100 and our IC is [3.0, 0]. The lines of the graph appear thicker using the ode function. This is because there are dots that are connected by lines to create the simulation. For the code, it's much simpler to use ode23 compared to that of Euler. We just need to start the timer with tic, enter in our parameters for ode23, and end the timer with toc. It is also easier since the step size is automatically adjusted by the ode23 function itself. In terms of elapsed time, the ode23 function is slower compared to the elapsed time of the Euler function we wrote.

3.1 3.b) **5 / 5**

✓ **- 0 pts** *Correct*



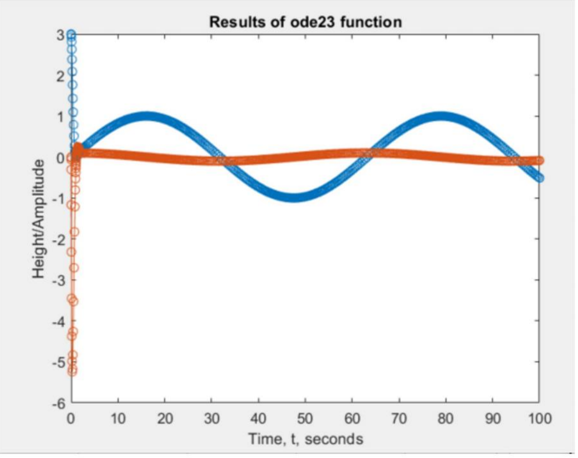
Here we can see the settling time more clearly and we can see that our settling time is close to 2. Comparing this with our prelab value for settling time of Ts = 1.533, we can see that our approximated settling time is fairly accurate.

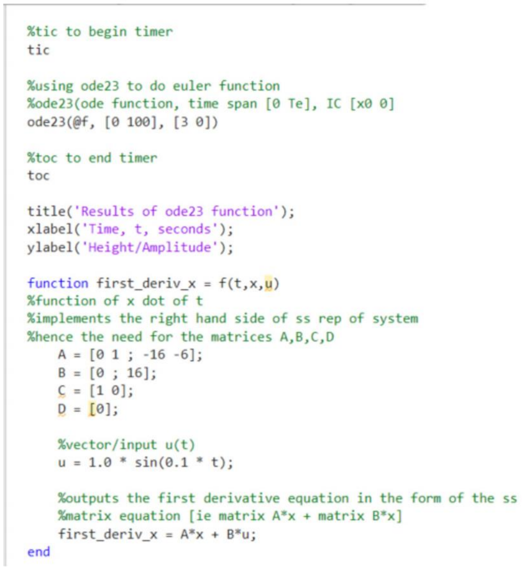


	h = 0.5s	h = 0.3s	h = 0.1s	h = 0.01s
Time, t	--------------------	---------------------	--------------------	---------------------
0	0	3	3	3
5	0	0.3	0.4	0.396
10	0	0.831	0.809	0.818
15	0	0.994	0.993	0.992
20	0.23	0.92	0.928	0.912
25	0.81	0.636	0.635	0.631

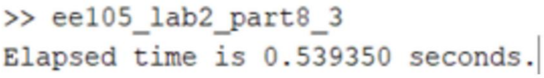
Here is our table of values for changing step size and time. We can see that as the step size decreases, we get more accurate values. I would assume that step size of h = 0.01s would be the most accurate due to the fact that it gives more "sample" selections compared to the larger step sizes. *Note: I changed Te from 100 to Te = 25 to make for easier sifting of values and kept x0 = [3,0].

Part 8.3:
A, B and C)





```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 100], [3 0])

%toc to end timer
toc

title('Results of ode23 function');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t)
    u = 1.0 * sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```



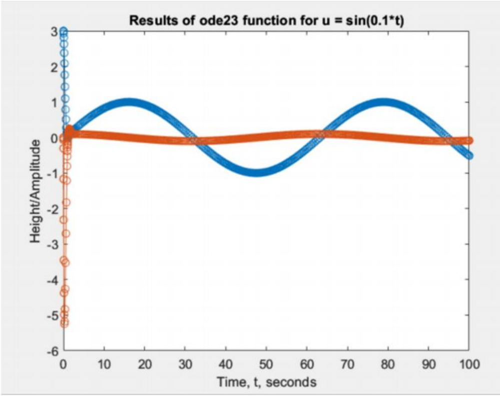
```
>> ee105_lab2_part8_3
Elapsed time is 0.539350 seconds.
```

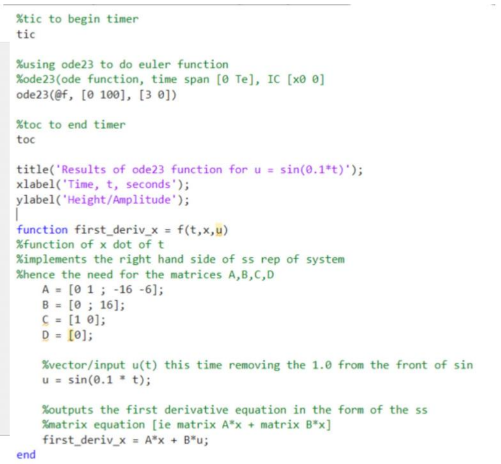
Here we can see we get a similar looking graph compared to part 2d. Our Te = 100 and our IC is [3.0, 0]. The lines of the graph appear thicker using the ode function. This is because there are dots that are connected by lines to create the simulation. For the code, it's much simpler to use ode23 compared to that of Euler. We just need to start the timer with tic, enter in our parameters for ode23, and end the timer with toc. It is also easier since the step size is automatically adjusted by the ode23 function itself. In terms of elapsed time, the ode23 function is slower compared to the elapsed time of the Euler function we wrote.

3.2 3.c) **5 / 5**

✓ **- 0 pts** *Correct*

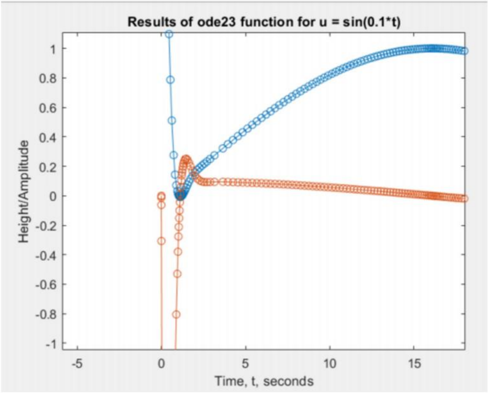
Part 8.4:





```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 100], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(0.1*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

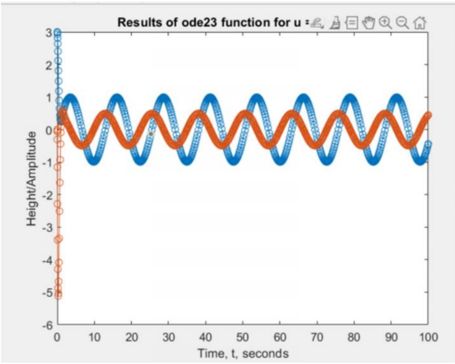
Simulating the system with keeping the same x0 and Te values, but removing the 1.0 from the front of u(t) = 1.0sin(1.0 * t), we can see that there is essentially no change in the simulation and data. The only change would be from the code itself to remove the 1.0. In comparison to the prelab, our settling time

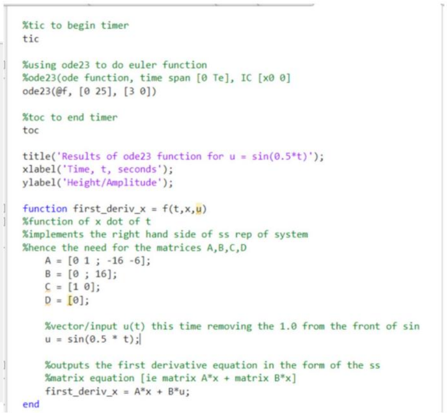


Our settling time appears to be around 2s which is close to our predicted Ts from the prelab of Ts = 1.533. Our rising time in the prelab we calculated Tr = 0.45, and from the graph we can see that our Tr is approx Tr = 1.18s which does not match our value from the prelab.

Part 8.5:

Wn = 0.5



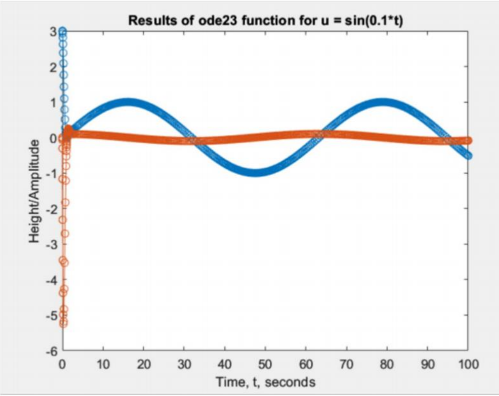


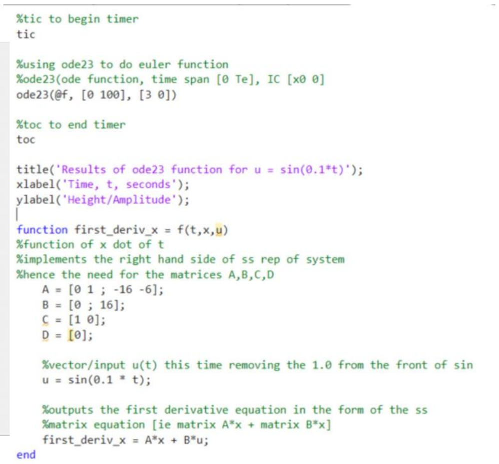
```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 25], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(0.5*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(0.5 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

4 Section 4 10 / 10

✓ **- 0 pts** *Correct*

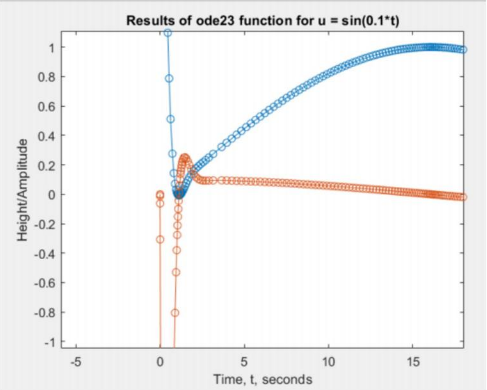
Part 8.4:





```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 100], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(0.1*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(0.1 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

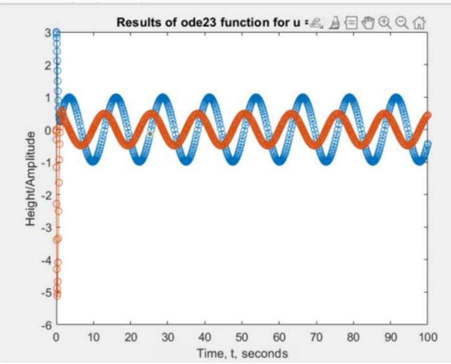
Simulating the system with keeping the same x0 and Te values, but removing the 1.0 from the front of u(t) = 1.0sin(1.0 * t), we can see that there is essentially no change in the simulation and data. The only change would be from the code itself to remove the 1.0. In comparison to the prelab, our settling time

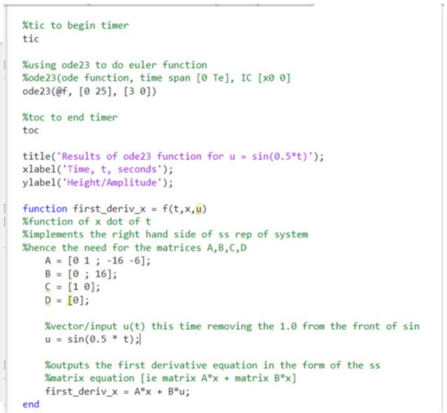


Our settling time appears to be around 2s which is close to our predicted Ts from the prelab of Ts = 1.533. Our rising time in the prelab we calculated Tr = 0.45, and from the graph we can see that our Tr is approx Tr = 1.18s which does not match our value from the prelab.

Part 8.5:

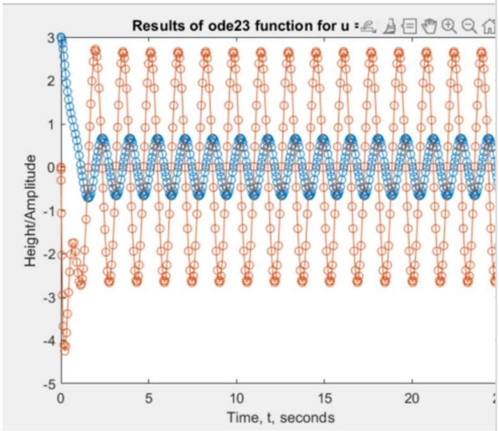
Wn = 0.5

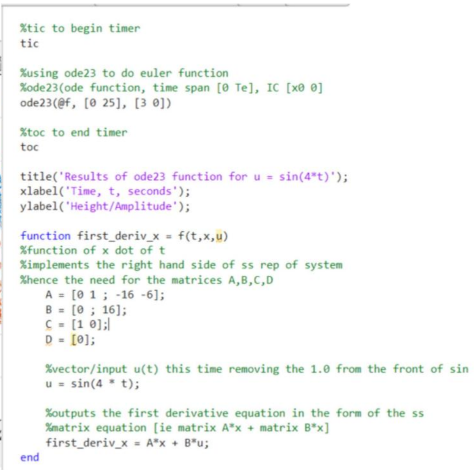




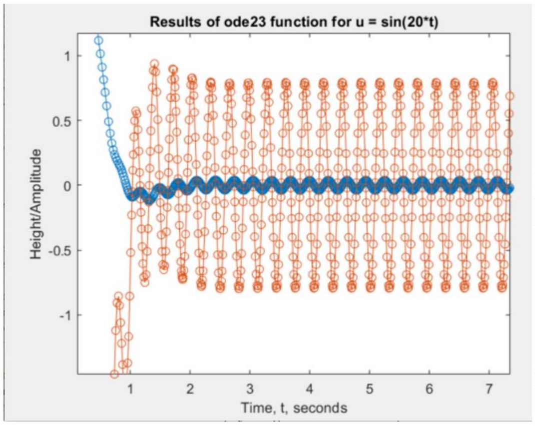
```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 25], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(0.5*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(0.5 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

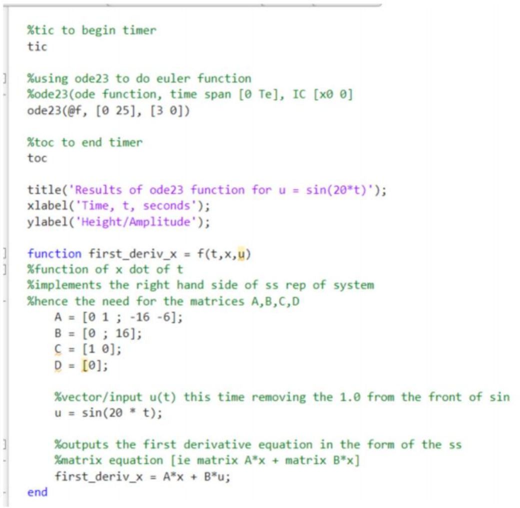
Wn = 4




```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 25], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(4*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(4 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

Wn = 20



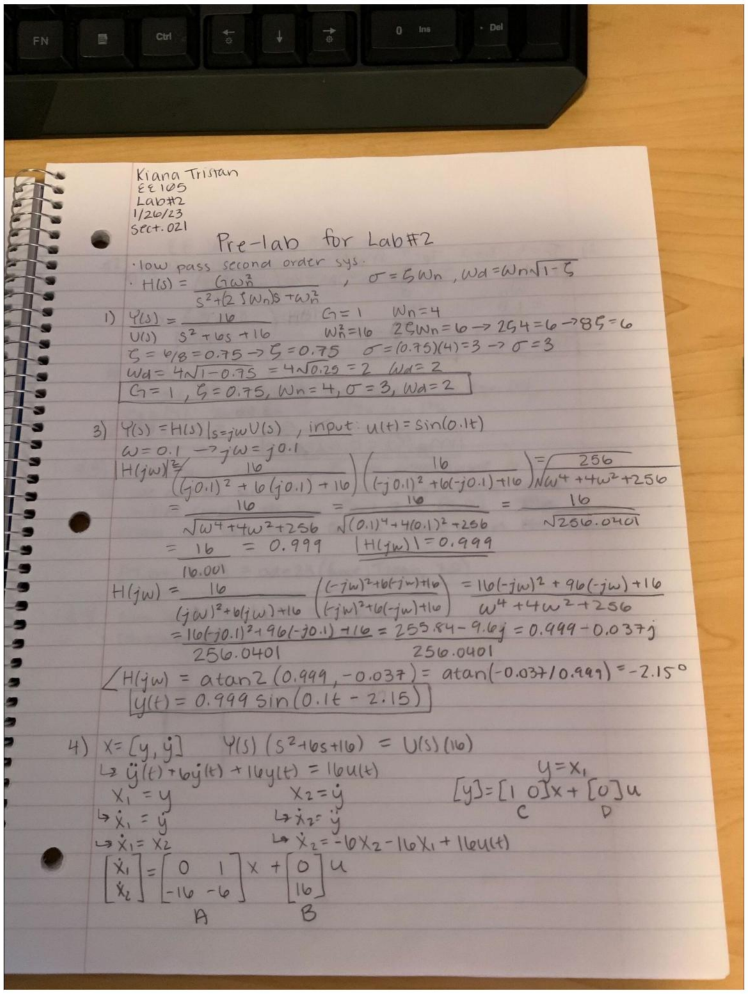

```
%tic to begin timer
tic

%using ode23 to do euler function
%ode23(ode function, time span [0 Te], IC [x0 0]
ode23(@f, [0 25], [3 0])

%toc to end timer
toc

title('Results of ode23 function for u = sin(20*t)');
xlabel('Time, t, seconds');
ylabel('Height/Amplitude');

function first_deriv_x = f(t,x,u)
%function of x dot of t
%implements the right hand side of ss rep of system
%hence the need for the matrices A,B,C,D
    A = [0 1 ; -16 -6];
    B = [0 ; 16];
    C = [1 0];
    D = [0];

    %vector/input u(t) this time removing the 1.0 from the front of sin
    u = sin(20 * t);

    %outputs the first derivative equation in the form of the ss
    %matrix equation [ie matrix A*x + matrix B*x]
    first_deriv_x = A*x + B*u;
end
```

After changing the value of Wn, we can see that the frequency of the overall graph itself changes. When wn = 0.5 we see a more defined and uniform graph. When we see wn = 4, amplitude increases. Same for wn = 20, amplitude decreases and frequency increases.

5 Section 5 10 / 10

✓ **- 0 pts** *Correct*

Pre-lab



Kiana Tristan
EE 105
Lab#2
1/26/23
Sect. 021

Pre-lab for Lab#2

- low pass second order sys.
- $H(s) = \frac{G\omega_n^2}{s^2+(2\zeta W_n)s+\omega_n^2}$, $\sigma = \zeta W_n$, $W_d = W_n\sqrt{1-\zeta}$

1) $\frac{Y(s)}{U(s)} = \frac{16}{s^2+6s+16}$ $G=1$ $W_n=4$

$W_n^2=16$ $2\zeta W_n = 6 \rightarrow 2\zeta 4 = 6 \rightarrow 8\zeta = 6$

$\zeta = 6/8 = 0.75 \rightarrow \zeta = 0.75$ $\sigma = (0.75)(4) = 3 \rightarrow \sigma = 3$

$W_d = 4\sqrt{1-0.75} = 4\sqrt{0.25} = 2$ $W_d = 2$

$\boxed{G=1,\ \zeta=0.75,\ W_n=4,\ \sigma=3,\ W_d=2}$

3) $Y(s) = H(s)|_{s=j\omega}U(s)$, input: $u(t) = \sin(0.1t)$

$\omega = 0.1 \rightarrow j\omega = j0.1$

$$|H(j\omega)|^2 = \left(\frac{16}{(j0.1)^2+6(j0.1)+16}\right)\left(\frac{16}{(-j0.1)^2+6(-j0.1)+16}\right) = \sqrt{\frac{256}{\omega^4+4\omega^2+256}}$$

$$= \frac{16}{\sqrt{\omega^4+4\omega^2+256}} = \frac{16}{\sqrt{(0.1)^4+4(0.1)^2+256}} = \frac{16}{\sqrt{256.0401}}$$

$$= \frac{16}{16.001} = 0.999$$ $\underline{\underline{|H(j\omega)| = 0.999}}$

$$H(j\omega) = \frac{16}{(j\omega)^2+6(j\omega)+16}\left(\frac{(-j\omega)^2+6(-j\omega)+16}{(-j\omega)^2+6(-j\omega)+16}\right) = \frac{16(-j\omega)^2+96(-j\omega)+16}{\omega^4+4\omega^2+256}$$

$$= \frac{16(-j0.1)^2+96(-j0.1)+16}{256.0401} = \frac{255.84-9.6j}{256.0401} = 0.999-0.037j$$

$\angle H(j\omega) = \text{atan2}(0.999, -0.037) = \text{atan}(-0.037/0.999) = -2.15°$

$\boxed{y(t) = 0.999\sin(0.1t-2.15)}$

4) $x = [y, \dot{y}]$ $Y(s)(s^2+6s+16) = U(s)(16)$

$\hookrightarrow \ddot{y}(t) + 6\dot{y}(t) + 16y(t) = 16u(t)$

$x_1 = y$ $x_2 = \dot{y}$

$\hookrightarrow \dot{x}_1 = \dot{y}$ $\hookrightarrow \dot{x}_2 = \ddot{y}$

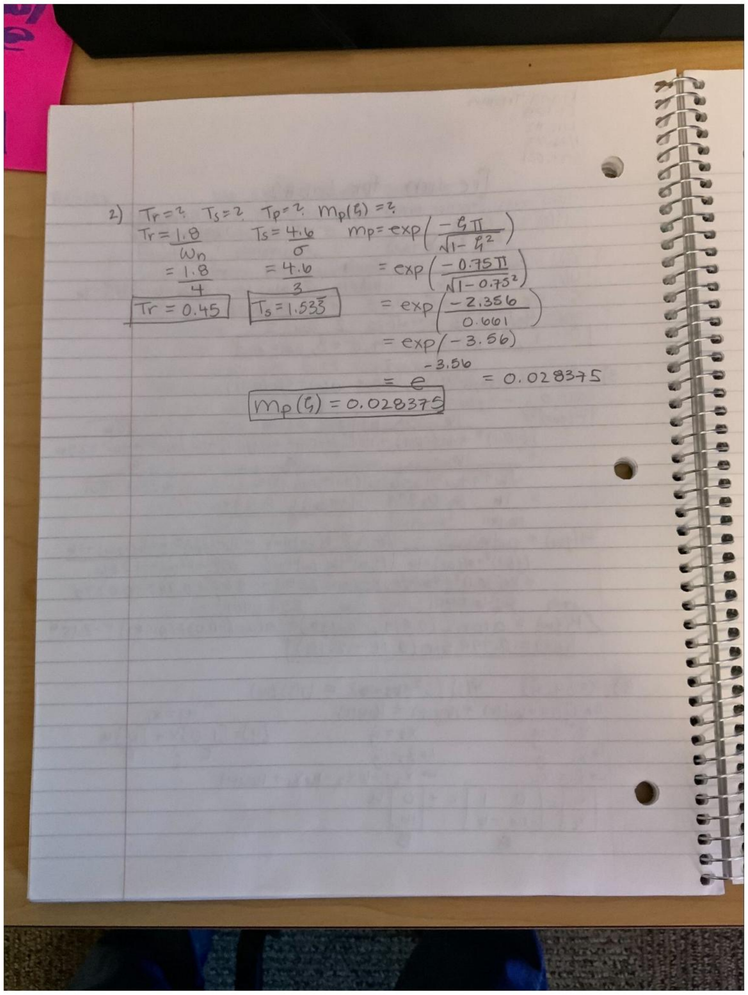
$\hookrightarrow \dot{x}_1 = x_2$ $\hookrightarrow \dot{x}_2 = -6x_2 - 16x_1 + 16u(t)$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \underbrace{\begin{bmatrix}0 & 1\\ -16 & -6\end{bmatrix}}_{A}x + \underbrace{\begin{bmatrix}0\\ 16\end{bmatrix}}_{B}u$$

$y = x_1$

$$[y] = \underbrace{[1\ \ 0]}_{C}x + \underbrace{[0]}_{D}u$$

Next page of prelab ---&gt;



2) $T_r = ?$ $T_s = ?$ $T_p = ?$ $m_p(\zeta) = ?$

$$T_r = \frac{1.8}{\omega_n} = \frac{1.8}{4}$$

$$\boxed{T_r = 0.45}$$

$$T_s = \frac{4.6}{\sigma} = \frac{4.6}{3}$$

$$\boxed{T_s = 1.53\overline{3}}$$

$$m_p = \exp\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right) = \exp\left(\frac{-0.75\pi}{\sqrt{1-0.75^2}}\right) = \exp\left(\frac{-2.356}{0.661}\right) = \exp(-3.56) = e^{-3.56} = 0.028375$$

$$\boxed{m_p(\zeta) = 0.028375}$$

6 Prelab (points assigned based on Prelab completion prior to lab) **25 / 25**

✓ **- 0 pts** *All questions complete before lab*