LAB 2: Numeric Simulation

This lab introduces numeric simulation of dynamic systems described by differential equations. We consider a simple gradient-based method, a built-in function from the control system toolbox, and finally another built-in function for solving differential equations. We explore the pitfalls and advantages of each method.

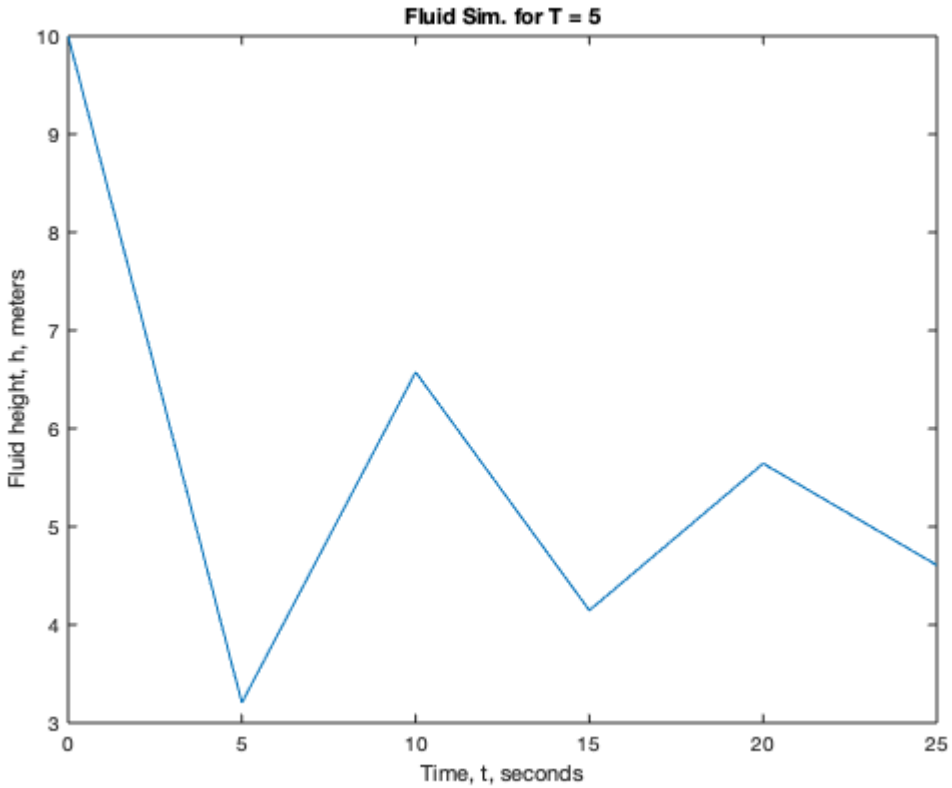
Table of Contents

Fluid Level Example

(20 pts) Free points as the code was already in the manual.

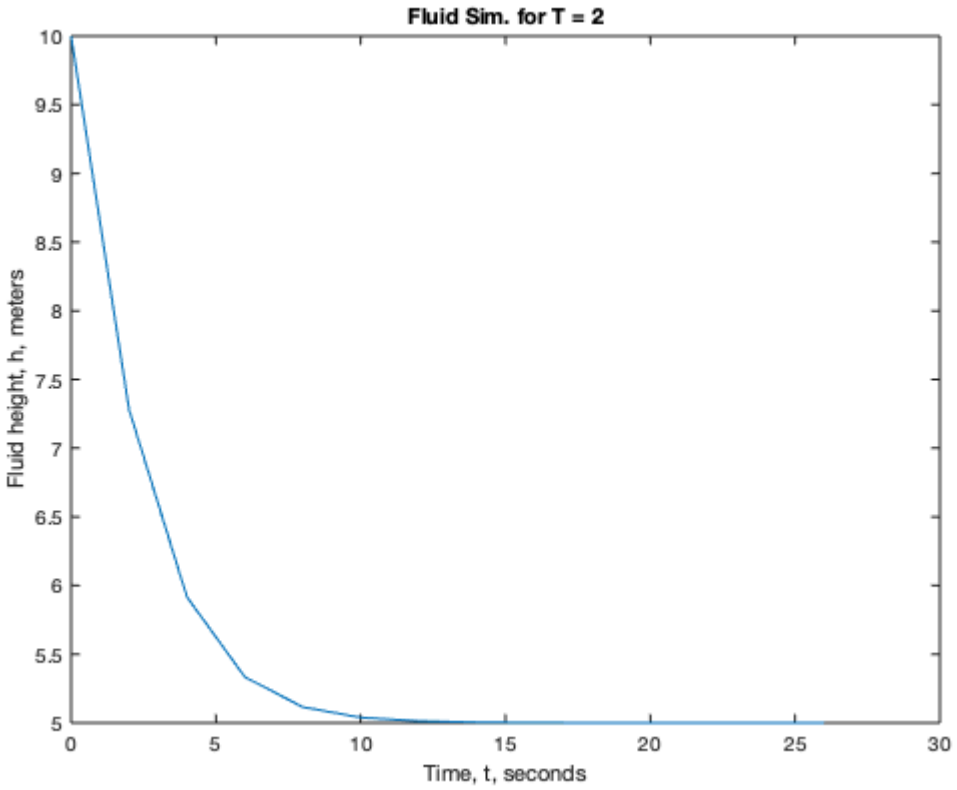
```
X0 = 10;
T = 5;
fld_lvl_sim(X0, T);
```

```
tm = 0.0379
```




```
T = 2;
fld_lvl_sim(X0, T);
```

```
tm = 0.0301
```



EXPERIMENTAL PROCEDURE

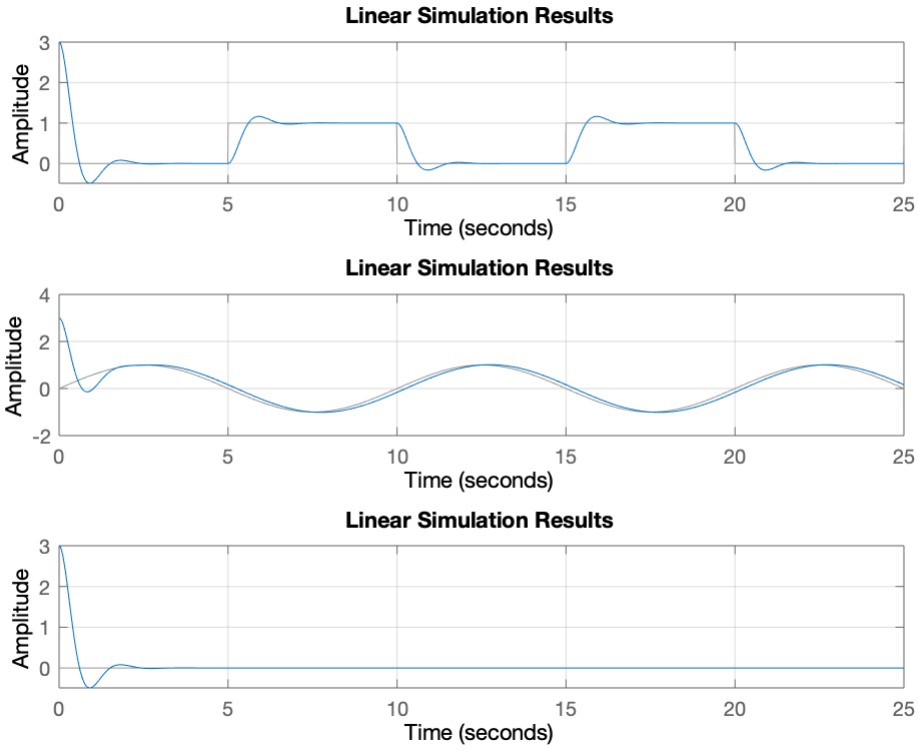
PART 1: SIMULATE SYSTEM WITH LSIM

(15 pts) Must plot at least zero response and sinusoidal response.

Here we simulate the system with three different input signals: square (could also be considered step), sine, and zero input. In all cases, the initial conditions are $[3.0; 0]$.

```
% Define the transfer function of the system (could be used for any system)
H = tf(16,[1 4 16]);

% Another way to define Linear systems
A = [0,1; -16,-4];
B = [0;16];
C = [1 0];
D = 0;
sys = ss(A,B,C,D);

% Square signal (essentially the step response of the sytem)
[u,t] = gensig('square',10,25,0.01);
x0 = [3,0];

figure; clf
% Simulate the system with square signal
subplot(311)
```

```
lsim(sys,u,t, x0);
grid on;

% Sinusoidal signal
[u,t] = gensig('sin',10,25,0.01);
%u = zeros(2501, 1);
% Simulate the system with sinusoidal
subplot(312)
lsim(sys,u,t, x0);
grid on;

% Zero input
u = zeros(size(u));
subplot(313)
lsim(sys,u,t, x0);
grid on;
```



PART 2: SIMULATE SYSTEM WITH EULER METHOD

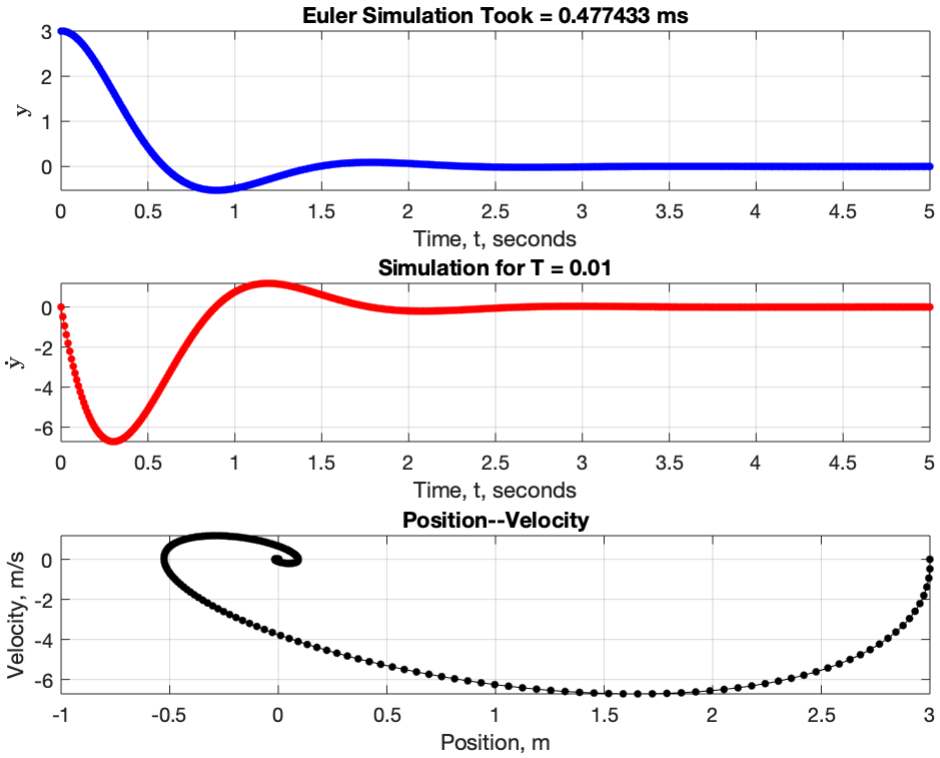
```
x0 = [3,0];
T = 0.01;
% Simulate with sinusoidal signal OR 0 (natural response)
% RESP = 0 : Natural
% RESP = 1 : Sinusoidal
```

With natural response (u(t) = 0)

(10 pts) Must plot figure.

```
RESP = 0;

N = round(5/T) + 1;
if RESP==0
    u = zeros(N,1);
elseif RESP ==1
[u,~] = gensig('sin',5,10,0.01);
else
    u= 0;
end
figure;
clf;
f_sim(x0, T, u, RESP);
```

```
Time elapsed 4.774e-01 ms
```



In the figure above, notice that the decay rate, $T_s \approx 2.3$ seconds. This corresponds to $\frac{4.6}{\sigma}$. Clearly, at steady-state the system settles at 0, it's input.

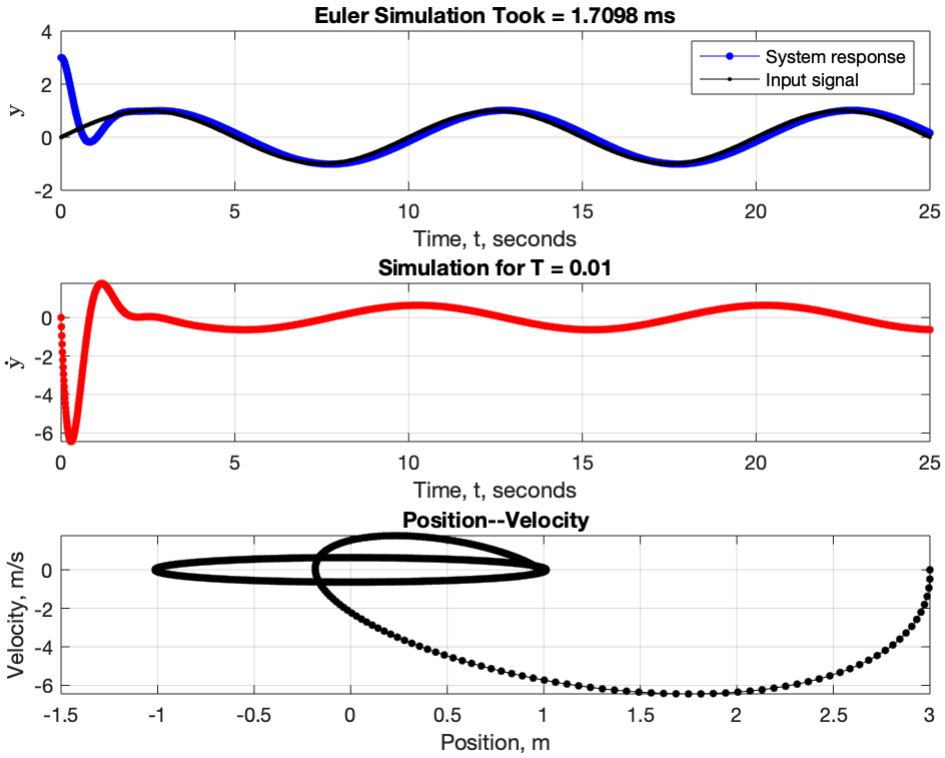
With u(t) = 1.0sin(0.1t)

(10 pts) Must plot figure.

Here, the syntax of the gensig function is gensig(type, τ, T_f, T_s) where $\tau = \frac{1}{\omega}$, T_f: duration, T_s:signal spacing (dt). For us, $\tau = \frac{1}{0.1} = 10$.

```
RESP = 1;

N = round(5/T) + 1;
if RESP==0
    u = zeros(N,1);
elseif RESP ==1
[u,~] = gensig('sin',10,25,0.01);
else
    u= 0;
end
figure;
clf;
f_sim(x0, T, u, RESP);
```

```
Time elapsed 1.710e+00 ms
```



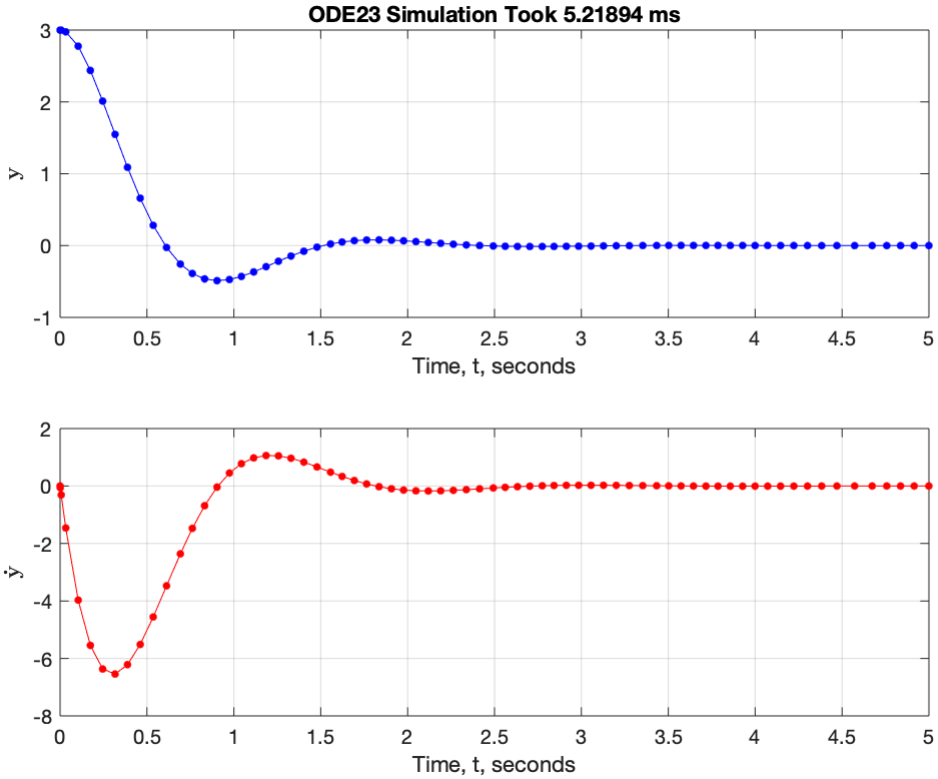
Notice that the amplitude of the system remains the same as that of the input. Also, the the system response has a slight lead on the input signal. Again, note that the system's stead-state response is sinusoidal like the input.

PART 3: SIMULATE SYSTEM WITH ODE23

(10 pts) Must plot figure.

Simulate the natural response using ode23.

```
tspan = [0 5];
x0 = [3;0];
u = 0;
tic

[t,y] = ode23(@(t,x) f(t,x,u), tspan, x0);

tm = toc;
tm = tm*1000;

figure;
clf;

subplot(211)
plot(t,y(:,1), '.-b', 'MarkerSize', 12);
str = sprintf('ODE23 Simulation Took %g ms', tm);
title (str)
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(212)
plot(t,y(:,2), '.-r', 'MarkerSize', 12);
xlabel('Time, t, seconds');
ylabel('$ \bf \dot{y}$', 'Interpreter', 'latex');
grid on;
```



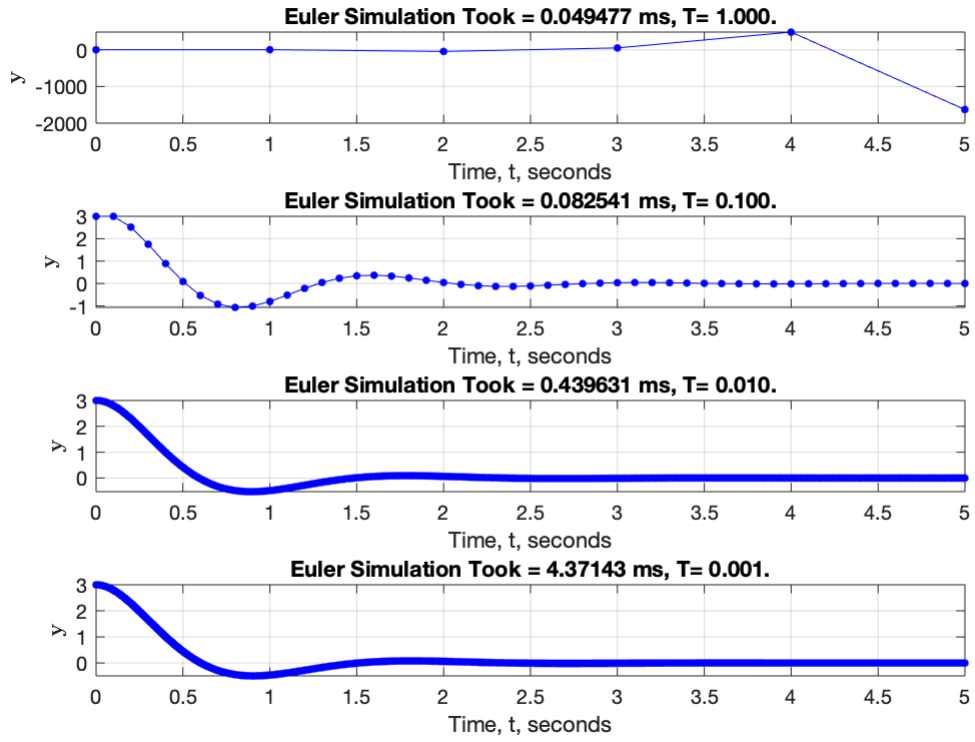
TABLE

CODE

```
cnt = 1;
X = zeros(5,4); % for values of y_k
K = zeros(5,4); % for values of k
TOC = zeros(4,1); % for values of tm (elapsed time)
figure;
for h = [1, 0.1, 0.01, 0.001]

    RESP = 0;    % just to select the sine input
    T = 5;      % duration of simulation
    N = round(T/h) + 1;
    u = zeros(N,1);
    subplot(4,1,cnt);

    [x,tm] = f_sim_mod(x0, h, u, RESP);  % 'f_sim_mod' is a modified 'f_sim' function
    for j=1:5
        X(j, cnt) = x(1,j/h);
        K(j, cnt) = j/h;
        TOC(cnt) = tm;
    end
    cnt = cnt + 1;
end
```

```
Time elapsed 4.948e-02 ms
```

```
Time elapsed 8.254e-02 ms

Time elapsed 4.396e-01 ms

Time elapsed 4.371e+00 ms
```



RESULTS

(10 pts) for table (extra points possible for discussion of results.)

The values saved in the variables above were used to create the table below (in MS Excel).

	h= 1 s.		h= 0.1 s.		h= 0.01 s.		h= 0.0
Time, t (s)	***k***	x_k	***k***	x_k	***k***	x_k	***k***
0	0	3	0	3	0	3	0
1	1	3	10	-1.00744	100	-0.49172	1000
2	2	3	20	0.14707	200	0.06863	2000
3	3	-45	30	0.02107	300	-0.00811	3000
4	4	51	40	-0.01572	400	0.00075	4000
5	5	483	50	0.00331	500	-0.00003	5000
CPUTIME (ms)	***0.049***		***0.083***		***0.440***		***4.3***

The first thing to notice from this here table is that for the step size of 1 second (h = 1), the algorithm fails to converge. This can also be seen in the figure above. It is due to the nature of the function, the fact that it decreases and increases in within the time of the step size (within 0 and 1 second, for example). Our

simple algorithm does not know what direction to go to, and therefore we cannot approximate (simulate) the function with this step size. One can also expect any step size larger than 1 to fail to approximate the function. However, when we decrease the step size to 0.1, the algorithm performs better and the approximation is good enough. Decreasing the step size to 0.01 does even better.

There is no accuracy gained by decreasing the step size from 0.01 to 0.001, even though the algorithm takes 10x longer to run (50x compared to h=0.1). As engineers, we should carefully weigh the trade-offs with advantages when designing algorithms. In this case, one could just do with h=0.01, or even h=0.1 when trading speed for accuracy.

CONCLUSION

(5 pts)

We have learned of three different methods to simulate linear dynamic systems: the simple Euler method, lsim, and ode23. When simulating linear dynamic systems, one must always choose the step size according to the function being simulated. However, utilities like MATLAB have built-in functions (such as ode23) to automatically adjust step sizes to achieve a desires accuracy. In practice, we are encouraged to use these buil-in functions if we can, as they are very optimized (also it's not good to reinvent the wheel.)

Functions Used

M-FILE TO SIMULATE THE SYSTEM

```
function f_sim(x0,T, u, RESP)

tic; %flops
N = length(u);
t = zeros(1,N);
x = zeros(2,N);
x(1:2,1) = x0;
t = T*(0:N-1);
for i = 1:N
dx = f(t(:)',x(:,i),u(i));
x(:,i+1) = x(:,i) + dx*T;
end
tm = toc;
tm = tm*1000;

fprintf("Time elapsed %0.3d ms \n \n", tm);

subplot(311)
plot(t(1:i)', x(1,1:i)', '.-b', 'MarkerSize', 12);
if RESP
hold on;
plot(t(1:i)', u(1:i)', '.-k', 'MarkerSize', 6);
hold off;
legend('System response', 'Input signal')
end

str = sprintf('Euler Simulation Took = %g ms', tm);
title(str);
```

```
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(312)
plot(t(1:i)', x(2,1:i)', '.-r', 'MarkerSize', 12);
str = sprintf('Simulation for T = %g', T);
title(str);
xlabel('Time, t, seconds');
ylabel('$\bf \dot{y}$', 'Interpreter', 'latex');
grid on;


subplot(313)
plot(x(1,1:i), x(2,1:i), '.-k', 'MarkerSize', 12);
str = sprintf('Position--Velocity');
title(str);
xlabel('Position, m');
ylabel('Velocity, m/s');
grid on;

end
```

MODIFIED M-FILE TO SIMULATE THE SYSTEM

```
function [x,tm] = f_sim_mod(x0,T, u, RESP)

tic; %flops
N = length(u);
t = zeros(1,N);
x = zeros(2,N);
x(1:2,1) = x0;
t = T*(0:N-1);
for i = 1:N
dx = f(t(:)',x(:,i),u(i));
x(:,i+1) = x(:,i) + dx*T;
end
tm = toc;
tm = tm*1000;

fprintf("Time elapsed %0.3d ms \n \n", tm);

plot(t(1:i)', x(1,1:i)', '.-b', 'MarkerSize', 12);
if RESP
hold on;
plot(t(1:i)', u(1:i)', '.-k', 'MarkerSize', 6);
hold off;
legend('System response', 'Input signal')
end

str = sprintf('Euler Simulation Took = %g ms, T= %.3f.', tm, T);
title(str);
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;


end
```

SYSTEM DEFINTION

```
function dx = f(t, x,u)
% x -- [2xn] column vector
% u -- [1xn] vector
% t -- [1xn] vector
A = [0,1; -16,-4];
B = [0;16];
C = [1 0];
D = 0;

dx = A*x + B*u;
end
```

(20 pts) for completing the pre-lab.