Laboratory 2

Helen Du

SID: 862081856

TA: Wang Hu

January 19, 2021

MODELING AND SIMULATION OF DYNAMIC SYSTEMS EE 105

SECTION 023

Objective:

The purpose of this lab is to understand the process and pitfalls of the numeric solution of differential equations. We will be constructing and simulating a dynamic system on a computer, while using vector and matrix notation and implementations throughout the entire lab.

Experimental Procedure:

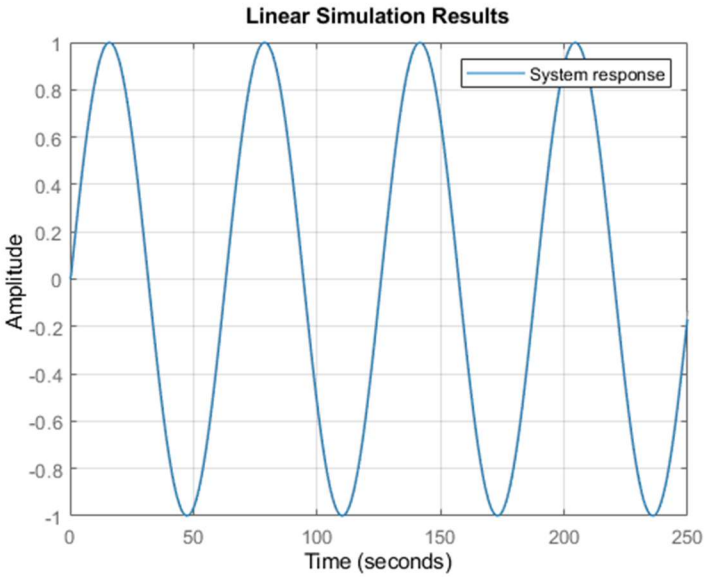
8.1: SIMULATE SYSTEM WITH LSIM

Since eqn. (6) is a linear system, its solution can be found using the 'lsim' function. Do this first, using a zero-initial condition with u(t) = 1.0sin(.1t), so that you know what the correct response is for the subsequent steps

The system is defined as $\frac{Y(s)}{U(s)} = \frac{16}{s^2 + 6s + 16} = H(s)$

And its matrix format is $A = \begin{bmatrix} 0 & 1 \\ -16 & -6 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 16 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, D = 0$

```
clear
% Define the transfer function of the system
H = tf(16,[1 6 16]);

% Define A,B,C,D
A = [0,1; -16,-6];
B = [0;16];
C = [1 0];
D = 0;

% Define state-space model
sys = ss(A,B,C,D);

% Define initial state
x0 = [0,0];

% Define the input u(t) = 1.0sin(0.1t) using sinusoidal signal
tau = 2*pi/0.1; % Period = 2*pi/0.1
% 0:Ts:Tf
Ts = 0.01; % Time step
Tf = 250; % Duration
[u,t] = gensig('sin',tau,Tf,Ts);

% Simulate the system
lsim(sys,u,t,x0);
grid on;
legend("System response");
```



8.2: SIMULATE SYSTEM WITH EULER METHOD

Part A

Write an m-file (call it f.m in the following) that implements the state space representation of the system. At least initially, assume that u(t) = 0. After you get your simulation running, then redefine u(t) = 1.0sin(.1t)

```
function dx = f(t,x,u)
    % x -- [2xn] column vector
    % u -- [1xn] vector
    A = [0,1; -16,-6];
    B = [0;16];
    dx = A*x + B*u;
end
```

Part B

Write an m-file to implement the recursion in (4-5) by calling 'Euler.m'.

```
function sim_t = Euler(t,x0,h,u,fig)
    % For the given initial condition x0 and step size
    % h this function uses Euler integration to
    % numerically solve the differential equation
    % of the fluid level system.
    % fig option: 1 turn on the figure
    %             0 disable figure plot
    % Function output: sim_t, Euler Simulation time cost
    tic;
    N = length(u); % The iteration steps based on the length of input signal
    % Initialize x
    x = zeros(length(x0),N); %The dimension of x in terms of dimension of x0
    x(:,1) = x0;
    for i=1:N
        dx = f(t(i),x(:,i),u(i));
        x(:,i+1) = x(:,i) + dx*h;
    end
    sim_t = toc;
    sim_t = sim_t*1000;
```

```
fprintf("Euler Simulation Took = %g ms\n", sim_t);
    if fig == 1
        figure
        plot(t,x(1,1:i));
        hold on
        plot(t,u);
        legend('System response', 'Input signal')
        str = sprintf('Euler Simulation: for step-size h = %g',h);
        grid on
        title(str);
        xlabel('Time, t, seconds');
        ylabel('y, meters');
    end
end
```

Part C

Using an initial condition of [3.0;0], simulate the system dynamics until steady state is (approximately) achieved. You will have to iterate on the simulation step size to get an accurate but reasonable length simulation.

Euler Simulation for h=1

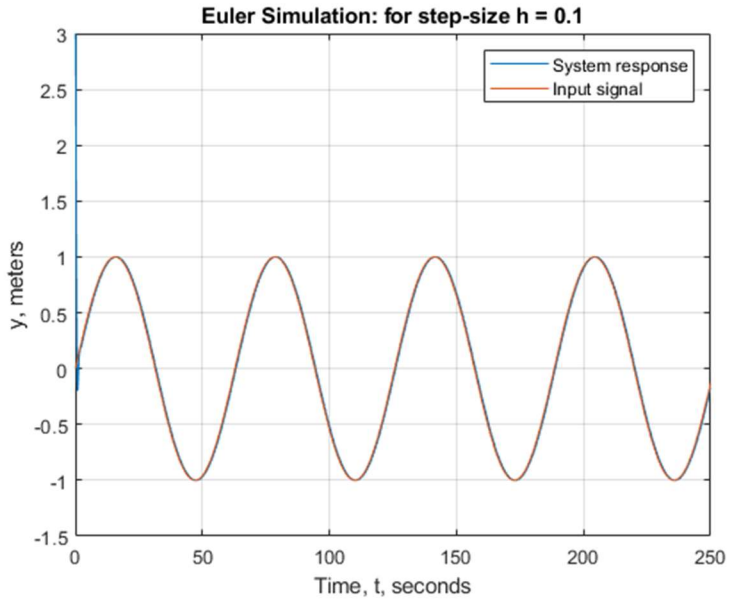
```
Ts = 1; % time step
[u,t] = gensig('sin',tau,Tf,Ts);
% Define initial state
x0 = [3,0];
% Define step-size
h = 1;
Euler(t,x0,h,u,0);
```

```
Euler Simulation Took 0.1628 ms for h = 1
```

Euler Simulation for h=0.1

```
Ts = 0.1; % time step
[u,t] = gensig('sin',tau,Tf,Ts);
h = 0.1;
Euler(t,x0,h,u,1);
```

```
Euler Simulation Took 0.9258 ms for h = 0.1
```



Euler Simulation for h=0.05

```
Ts = 0.05; % time step
[u,t] = gensig('sin',tau,Tf,Ts);
h = 0.05;
Euler(t,x0,h,u,0);
```

```
Euler Simulation Took 1.6868 ms for h = 0.05
```

Euler Simulation for h=0.01

```
Ts = 0.01; % time step
[u,t] = gensig('sin',tau,Tf,Ts);
h = 0.01;
Euler(t,x0,h,u,0);
```

```
Euler Simulation Took 7.6998 ms for h = 0.01
```

Compare the time that it took to achieve 1% steady-state (i.e., setting time Ts) with the value predicted by the pole locations.

$$\frac{Y(s)}{U(s)} = \frac{16}{s^2 + 6s + 16} = H(s)$$

```
roots([1 6 16])
```

```
ans = 2×1 complex
   -3.0000 + 2.6458i
   -3.0000 - 2.6458i
```

Because the roots are $-3 \pm 2.6458j$, we find that $\sigma = 3$. Using this, we can find $T_s = \frac{4.6}{\sigma} = \frac{4.6}{3} = 1.533 \sec$.

8.3 + 8.4: SIMULATE SYSTEM WITH ODE23 FOR u(t) = sin(0.1t)

Part A

Type 'help ode23'

```
help ode23
```

```
  ode23  Solve non-stiff differential equations, low order method.
     [TOUT,YOUT] = ode23(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates
     the system of differential equations y' = f(t,y) from time T0 to TFINAL
     with initial conditions Y0. ODEFUN is a function handle. For a scalar T
     and a vector Y, ODEFUN(T,Y) must return a column vector corresponding
     to f(t,y). Each row in the solution array YOUT corresponds to a time
     returned in the column vector TOUT.  To obtain solutions at specific
     times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN =
     [T0 T1 ... TFINAL].

     ...
```

Part B

Use ode23 with input function 'f.m' to simulate the system using the same initial condition as 8.2.

```
tspan = [0 250]; % Interval of integration
x0 = [3;0]; % Initial condition
% u(t) = sin(0.1t) defined in f_ode23
omg = 0.1;
tic
[t_out,y] = ode23(@(t,x) f_ode23(t,x,omg), tspan, x0);
tm = toc;
tm = tm*1000;
fprintf("ODE23 Simulation Took %g ms\n", tm);
```

```
ODE23 Simulation Took 9.6487 ms
```

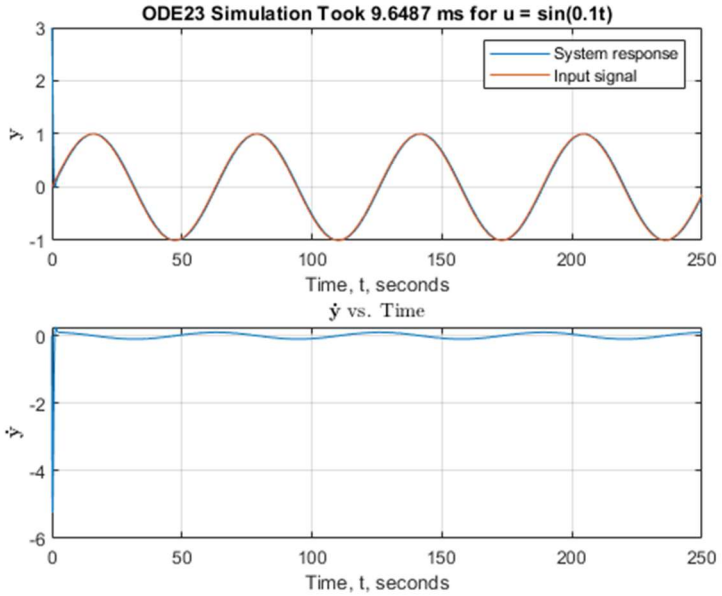
Discuss and compare the computation time required for each approach.

Using the ODE23 method took around 9-10 ms, whereas using Euler's Method took around 6-7 ms with an h value of 0.01.

Part C

Plot the output of ode23 versus time and compare with the simulation you wrote in 8.2

```
figure
subplot(2, 1, 1)
plot(t_out, y(:,1))
hold on

% tau = 2*pi/0.1;
% Ts = 0.01;
% Tf = 250;
% [u,t] = gensig('sin', tau, Tf, Ts);
plot(t,u)

legend('System response', 'Input signal')
str = sprintf('ODE23 Simulation Took %g ms for u = sin(0.1t)', tm);
title (str)
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(2, 1, 2)
plot(t_out, y(:,2))
title ('$\bf \dot{y}$ vs. Time', 'Interpreter', 'latex')
xlabel('Time, t, seconds');
ylabel('$ \bf \dot{y}$', 'Interpreter', 'latex');
grid on;
```

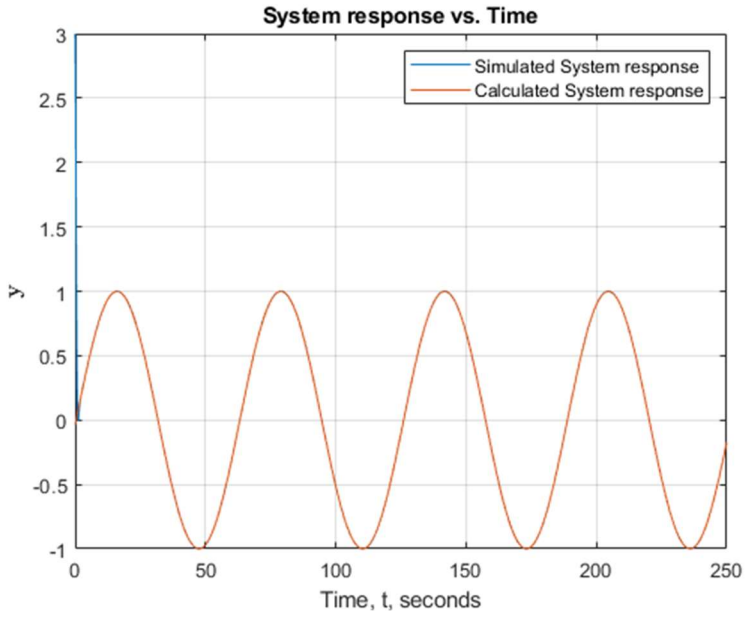


Is there any difference? How much CPU time was required and how does it compare with that required in 8.2?

In terms of the graph, there is not much difference. However, the ODE23 Simulation typically took much longer than Euler's Method with 10-11 ms compared to 6-7 ms (when h = 0.01).

Quantitatively compare the simulated response with the answer from the prelab. How well do they compare?

```
figure
plot(t_out, y(:,1))
hold on
func=0.9993.*sin(0.1.*t-0.0374); % Answer from Pre-Lab
plot(t, func)
legend('Simulated System response', 'Calculated System response')
title ('System response vs. Time')
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on
```



As we can see in the graph above, the simulated system response almost perfectly resembles the calculated system response once it has been stabilized after the first two seconds.

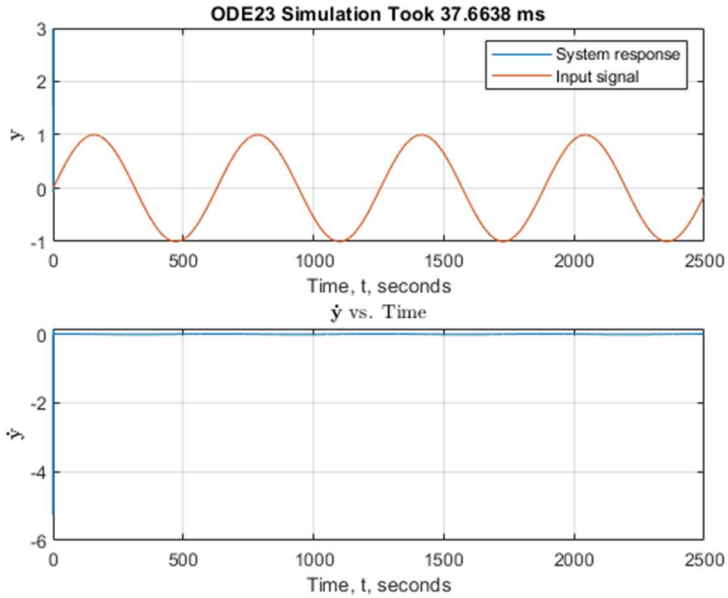
8.5: SIMULATE SYSTEM WITH MULTIPLE ω's

Simulating System with ω=0.01

```
tspan = [0 2500]; % Interval of integration
x0 = [3;0]; % Initial condition
% u(t) = sin(0.1t) defined in f_ode23
omg = 0.01;
tic
[t_out,y] = ode23(@(t,x) f_ode23(t,x,omg), tspan, x0);
tm = toc;
tm = tm*1000;
fprintf("ODE23 Simulation Took %g ms\n", tm);
```

```
ODE23 Simulation Took 37.6638 ms
```

```
figure
subplot(2, 1, 1)
plot(t_out, y(:,1))
hold on

tau = 2*pi/0.01;
Ts = 0.01;
Tf = 2500;
[u,t] = gensig('sin', tau, Tf, Ts);
plot(t,u)

legend('System response', 'Input signal')
str = sprintf('ODE23 Simulation Took %g ms', tm);
title (str)
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(2, 1, 2)
plot(t_out, y(:,2))
title ('$\bf \dot{y}$ vs. Time', 'Interpreter', 'latex')
xlabel('Time, t, seconds');
ylabel('$ \bf \dot{y}$', 'Interpreter', 'latex');
grid on;
```



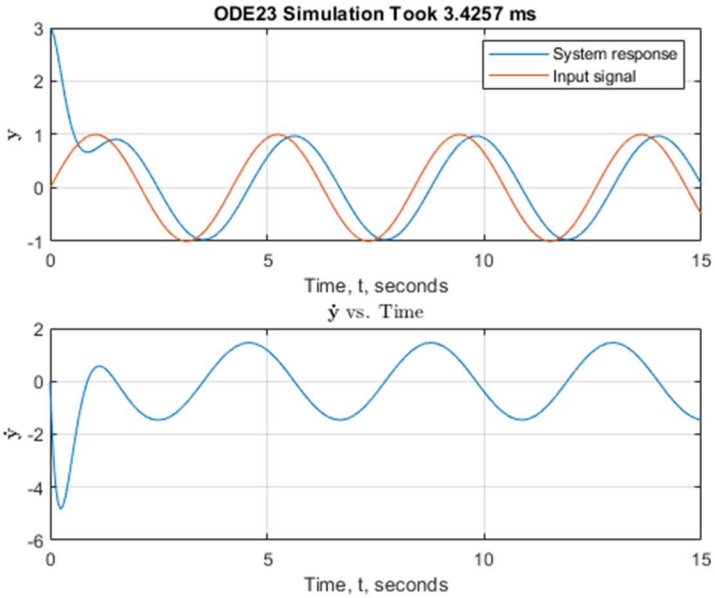
Simulating System with ω=1.5

```
tspan = [0 15]; % Interval of integration
x0 = [3;0]; % Initial condition
% u(t) = sin(0.1t) defined in f_ode23
omg = 1.5;
tic
[t_out,y] = ode23(@(t,x) f_ode23(t,x,omg), tspan, x0);
tm = toc;
tm = tm*1000;
fprintf("ODE23 Simulation Took %g ms\n", tm);
```

```
ODE23 Simulation Took 3.4257 ms
```

```
figure
subplot(2, 1, 1)
plot(t_out, y(:,1))
hold on

tau = 2*pi/1.5;
Ts = 0.01;
Tf = 15;
[u,t] = gensig('sin', tau, Tf, Ts);
plot(t,u)

legend('System response', 'Input signal')
str = sprintf('ODE23 Simulation Took %g ms', tm);
title (str)
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(2, 1, 2)
plot(t_out, y(:,2))
title ('$\bf \dot{y}$ vs. Time', 'Interpreter', 'latex')
xlabel('Time, t, seconds');
```

```
ylabel('$ \bf \dot{y}$', 'Interpreter', 'latex');
grid on;
```



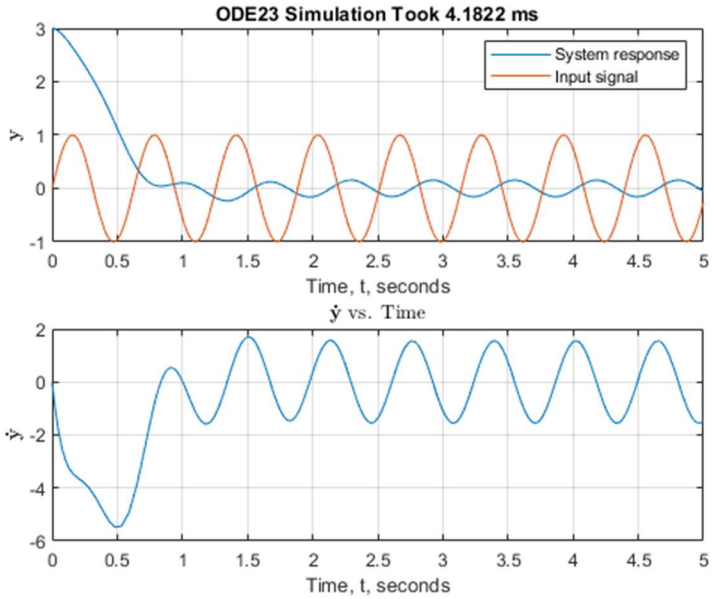
Simulating System with ω=10

```
tspan = [0 5]; % Interval of integration
x0 = [3;0]; % Initial condition
% u(t) = sin(0.1t) defined in f_ode23
omg = 10;
tic
[t_out,y] = ode23(@(t,x) f_ode23(t,x,omg), tspan, x0);
tm = toc;
tm = tm*1000;
fprintf("ODE23 Simulation Took %g ms\n", tm);
```

```
ODE23 Simulation Took 4.1822 ms
```

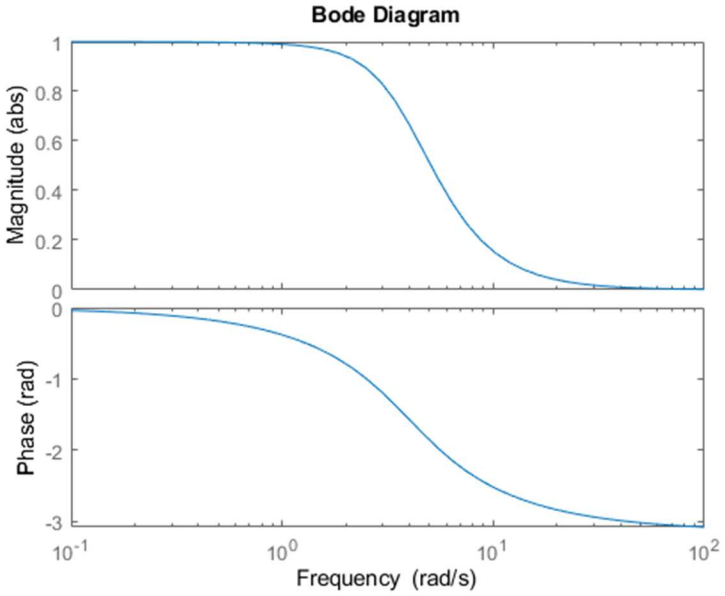
```
figure
subplot(2, 1, 1)
plot(t_out, y(:,1))
hold on

tau = 2*pi/10;
Ts = 0.01;
Tf = 5;
[u,t] = gensig('sin', tau, Tf, Ts);
plot(t,u)

legend('System response', 'Input signal')
str = sprintf('ODE23 Simulation Took %g ms', tm);
title (str)
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;

subplot(2, 1, 2)
plot(t_out, y(:,2))
```

```
title ('$\bf \dot{y}$ vs. Time', 'Interpreter', 'latex')
xlabel('Time, t, seconds');
ylabel('$ \bf \dot{y}$', 'Interpreter', 'latex');
grid on;
```



Discuss and compare the amplitude and phase shift of the output relative to the input. How do the steady state values compare with those predicted by frequency response (try using the 'Bode' function)?

```
H = tf(16,[1 6 16]);
opts = bodeoptions('cstprefs');
opts.MagUnits = 'abs';
opts.PhaseUnits = 'rad';
figure
bode(H, opts);
```



```
w = [0.01, 1.5, 10];
[mag,phase] = bode(sys, w);
phase = deg2rad(phase);
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', w(1), mag(1), phase(1));
```

```
At omega = 0.01, the amplitude is 0.999999 and the phase is -0.00375001
```

```
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', w(2), mag(2), phase(2));
```

```
At omega = 1.5, the amplitude is 0.973616 and the phase is -0.579564
```

```
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', w(3), mag(3), phase(3));
```

```
At omega = 10, the amplitude is 0.154997 and the phase is -2.52134
```

As shown in the bode plot, the amplitude of the system response decreases as frequency increases, while phase shift increases as frequency increases. This can be seen in the demonstrations above as the system response almost perfectly matches the input signal in the graph where $\omega = 0.01$. As ω increases, we can see that the amplitude is the least and the phase shift is the greatest when $\omega = 10$.

Conclusion:

Overall, the objective of this lab was accomplished. We now understand the process and pitfalls of the numeric solution of differential equations. We also successfully constructed and simulated a dynamic system on a computer using vector and matrix notation and implementations throughout the entire lab.