# EE_105_023_23W Lab 2

Merrick Slane

TOTAL POINTS

**95 / 100**

QUESTION 1

## Section 1 10 pts

### *1.1* 1a **0 / 4**

✓ **+ 0 pts** *Didn't Calculate or use $$T_s$$*

### *1.2* 1b **3 / 3**

✓ **+ 3 pts** *Correct time vector t*

### *1.3* 1c **2 / 3**

✓ **+ 2 pts** *Used tf instead of ss or tf2ss*

QUESTION 2

## Section 2 35 pts

### *2.1* 2a **5 / 5**

✓ **+ 5 pts** *Correct f function*

### *2.2* 2b **5 / 5**

✓ **+ 5 pts** *Correct Euler recursion*

### *2.3* 2c **10 / 10**

✓ **+ 10 pts** *Correct for loop (Calling f.m)*

### *2.4* 2d **15 / 15**

✓ **+ 7.5 pts** *Simulated system using [3,0] initial conditions*

✓ **+ 7.5 pts** *Included table*

QUESTION 3

## Section 3 10 pts

### *3.1* 3b **5 / 5**

✓ **+ 5 pts** *Simulated ode23 correctly*

### *3.2* 3c **5 / 5**

✓ **+ 5 pts** *Correct plot*

QUESTION 4

## *4* Section 4 **10 / 10**

✓ **+ 10 pts** *Compared simulated response with calculated response*

QUESTION 5

## *5* Section 5 **10 / 10**

✓ **+ 10 pts** *simulated using different $$w$$ values*

QUESTION 6

## *6* Prelab **25 / 25**

✓ **+ 25 pts** *Full prelab*

# EE105 Lab 2

Merrick Slane

January 24, 2023

## 1 lsim Simulation

In order to validate the response of our simulations in the following steps, we simulate the linear system with the transfer function from the prelab and plot its response to $u(t) = 1.0sin(.1t)$ with a zero initial condition. To do this, we use the following code:

```
clear all;
sys = tf([16], [1,6,16]);
t = linspace(0,100,1000);
u = sin(0.1 * t);
lsim(sys,u,t);
```

The output from this code is shown in figure 1.

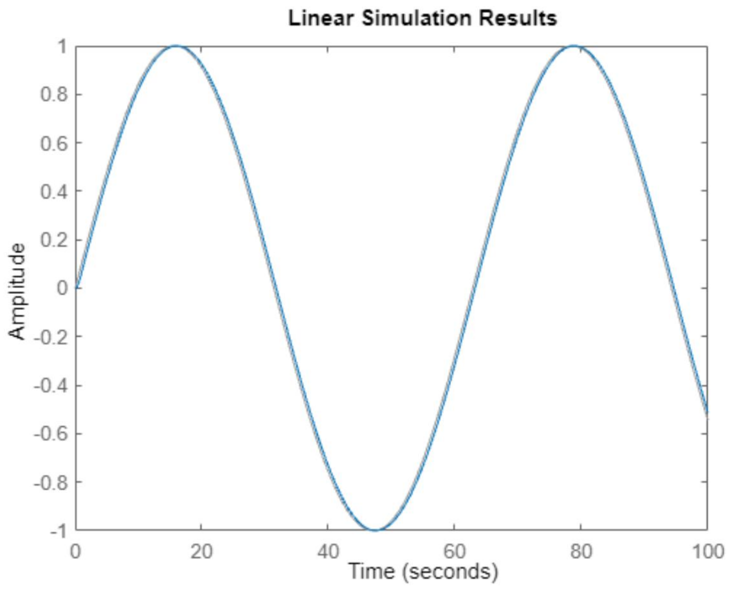


Figure 1: Linear System Simulation using lsim

1.1 1a **0 / 4**

✓ **+ 0 pts** *Didn't Calculate or use $$T_s$$*

# EE105 Lab 2

Merrick Slane

January 24, 2023

## 1 lsim Simulation

In order to validate the response of our simulations in the following steps, we simulate the linear system with the transfer function from the prelab and plot its response to $u(t) = 1.0sin(.1t)$ with a zero initial condition. To do this, we use the following code:

```
clear all;
sys = tf([16], [1,6,16]);
t = linspace(0,100,1000);
u = sin(0.1 * t);
lsim(sys,u,t);
```

The output from this code is shown in figure 1.

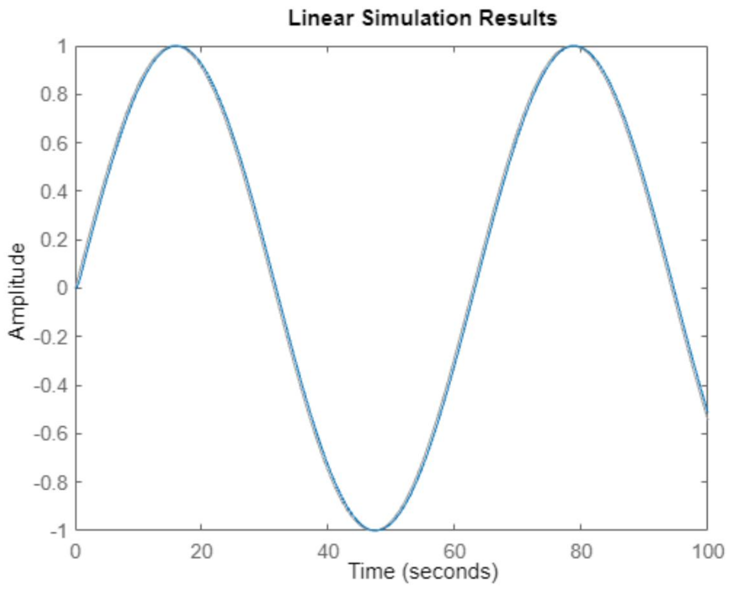


Figure 1: Linear System Simulation using lsim

## *1.2* 1b **3 / 3**

✓ **+ 3 pts** *Correct time vector t*

# EE105 Lab 2

Merrick Slane

January 24, 2023

## 1 lsim Simulation

In order to validate the response of our simulations in the following steps, we simulate the linear system with the transfer function from the prelab and plot its response to $u(t) = 1.0sin(.1t)$ with a zero initial condition. To do this, we use the following code:

```
clear all;
sys = tf([16], [1,6,16]);
t = linspace(0,100,1000);
u = sin(0.1 * t);
lsim(sys,u,t);
```

The output from this code is shown in figure 1.

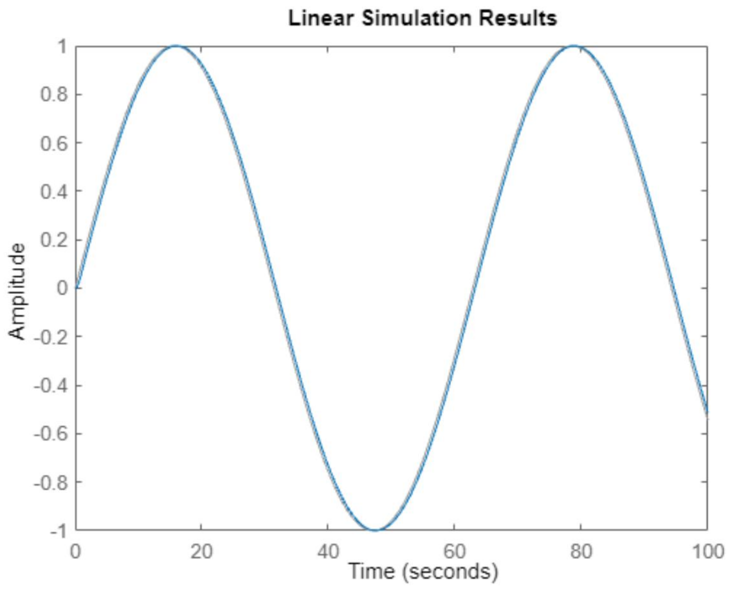


Figure 1: Linear System Simulation using lsim

*1.3* 1c **2 / 3**

✓ **+ 2 pts** *Used tf instead of ss or tf2ss*

## 2 Euler's Method

### 2.a State Space File

First, we implement the transfer function simulated in part 1 using a state-space model and save it in a matlab file named `f.m`:

```
function [dx,dy] = f(t,x)
    A = [0 1; -16 -6];
    B = [0; 16];
    C = [1 0];
    D = [0];

    u = sin(0.1 * t);

    dx = A*x + B*u;
    dy = C*x + D*u;
end
```

### 2.b Euler's Method Implementation

Next, we implement Euler's Method using the following function saved as `Euler.m`:

```
function x_2 = Euler(x,dx,T)
    x_2 = x + dx*T;
end
```

This function will be called recursively in the main script loop.

### 2.c Simulation

To simulate the response of this system using Euler's method, we implement the following code:

```
tic;
x = [3.0;0]; %initial conditions

h = 0.1; %time step
max = 100; %end time
N = max / h;
for i=1:N
    [dx,dy] = f(i*h,x); %get system from f.m
    x = Euler(x, dx, h); %recursively apply Euler's method
    x1(i) = x(1);
    x2(i) = x(2);
end
toc

subplot(2,1,1); %plot data on two separate graphs
plot(linspace(0,N*h,N), x1);
title("X1");
xlabel("Time");
ylabel("Amplitude");
subplot(2,1,2);
```

## *2.1* 2a **5 / 5**

✓ **+ 5 pts** *Correct f function*

## 2 Euler's Method

### 2.a State Space File

First, we implement the transfer function simulated in part 1 using a state-space model and save it in a matlab file named `f.m`:

```
function [dx,dy] = f(t,x)
    A = [0 1; -16 -6];
    B = [0; 16];
    C = [1 0];
    D = [0];

    u = sin(0.1 * t);

    dx = A*x + B*u;
    dy = C*x + D*u;
end
```

### 2.b Euler's Method Implementation

Next, we implement Euler's Method using the following function saved as `Euler.m`:

```
function x_2 = Euler(x,dx,T)
    x_2 = x + dx*T;
end
```

This function will be called recursively in the main script loop.

### 2.c Simulation

To simulate the response of this system using Euler's method, we implement the following code:

```
tic;
x = [3.0;0]; %initial conditions

h = 0.1; %time step
max = 100; %end time
N = max / h;
for i=1:N
    [dx,dy] = f(i*h,x); %get system from f.m
    x = Euler(x, dx, h); %recursively apply Euler's method
    x1(i) = x(1);
    x2(i) = x(2);
end
toc

subplot(2,1,1); %plot data on two separate graphs
plot(linspace(0,N*h,N), x1);
title("X1");
xlabel("Time");
ylabel("Amplitude");
subplot(2,1,2);
```

*2.2* 2b **5 / 5**

✓ **+ 5 pts** *Correct Euler recursion*

## 2 Euler's Method

### 2.a State Space File

First, we implement the transfer function simulated in part 1 using a state-space model and save it in a matlab file named `f.m`:

```
function [dx,dy] = f(t,x)
    A = [0 1; -16 -6];
    B = [0; 16];
    C = [1 0];
    D = [0];

    u = sin(0.1 * t);

    dx = A*x + B*u;
    dy = C*x + D*u;
end
```

### 2.b Euler's Method Implementation

Next, we implement Euler's Method using the following function saved as `Euler.m`:

```
function x_2 = Euler(x,dx,T)
    x_2 = x + dx*T;
end
```

This function will be called recursively in the main script loop.

### 2.c Simulation

To simulate the response of this system using Euler's method, we implement the following code:

```
tic;
x = [3.0;0]; %initial conditions

h = 0.1; %time step
max = 100; %end time
N = max / h;
for i=1:N
    [dx,dy] = f(i*h,x); %get system from f.m
    x = Euler(x, dx, h); %recursively apply Euler's method
    x1(i) = x(1);
    x2(i) = x(2);
end
toc

subplot(2,1,1); %plot data on two separate graphs
plot(linspace(0,N*h,N), x1);
title("X1");
xlabel("Time");
ylabel("Amplitude");
subplot(2,1,2);
```

*2.3* 2c **10 / 10**

✓ **+ 10 pts** *Correct for loop (Calling f.m)*

## 2 Euler's Method

### 2.a State Space File

First, we implement the transfer function simulated in part 1 using a state-space model and save it in a matlab file named `f.m`:

```
function [dx,dy] = f(t,x)
    A = [0 1; -16 -6];
    B = [0; 16];
    C = [1 0];
    D = [0];

    u = sin(0.1 * t);

    dx = A*x + B*u;
    dy = C*x + D*u;
end
```

### 2.b Euler's Method Implementation

Next, we implement Euler's Method using the following function saved as `Euler.m`:

```
function x_2 = Euler(x,dx,T)
    x_2 = x + dx*T;
end
```

This function will be called recursively in the main script loop.

### 2.c Simulation

To simulate the response of this system using Euler's method, we implement the following code:

```
tic;
x = [3.0;0]; %initial conditions

h = 0.1; %time step
max = 100; %end time
N = max / h;
for i=1:N
    [dx,dy] = f(i*h,x); %get system from f.m
    x = Euler(x, dx, h); %recursively apply Euler's method
    x1(i) = x(1);
    x2(i) = x(2);
end
toc

subplot(2,1,1); %plot data on two separate graphs
plot(linspace(0,N*h,N), x1);
title("X1");
xlabel("Time");
ylabel("Amplitude");
subplot(2,1,2);
```

```
plot(linspace(0,N*h,N),x2,'r');
title("X2");
xlabel("Time");
ylabel("Amplitude");
```

For h = 0.4, the simulation diverges to infinity as shown in Figure 2

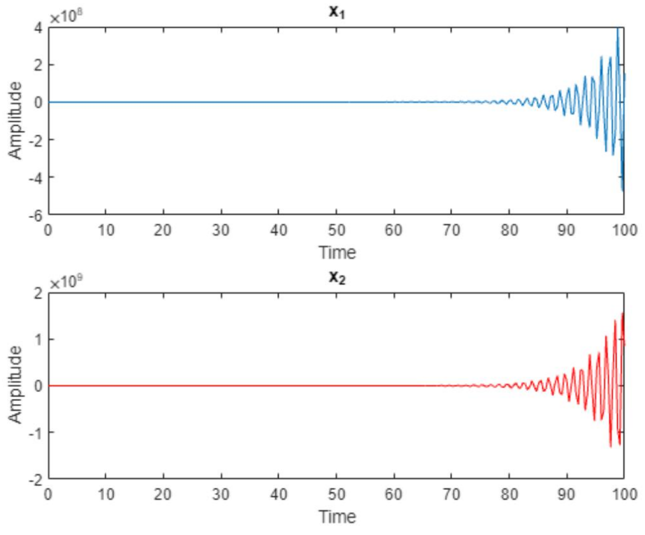


Figure 2: Euler's Method Simulation for h=0.4

For h = 0.25, the simulation converges after an initial oscillation as shown in Figure 3

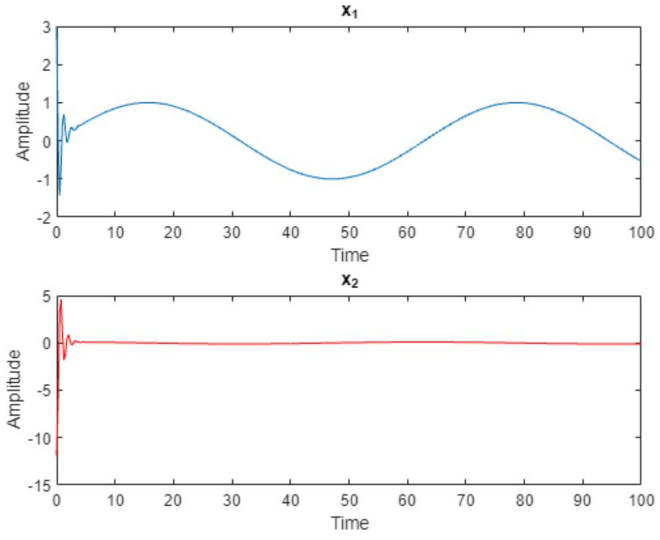


Figure 3: Euler's Method Simulation for h=0.25

Finally, for h = 0.05, we see that the simulation appears nearly identical to what is expected as seen in Figure 4

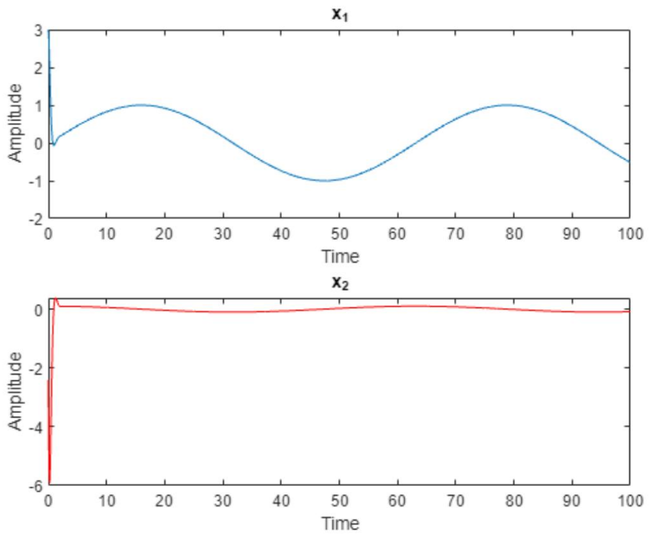


Figure 4: Euler's Method Simulation for h=0.05

Simulation time and accuracy for varying values of *h* are compared in Table 1 below.

Table 1: Table of selected values from Euler's Method simulation with step size h

| | $x_1$ | | | |
|---|---|---|---|---|
| Time (t) | h = 0.5 | h = 0.2 | h = 0.1 | h = 0.05 |
| 0 | 3 | 3 | 3 | 3 |
| 1 | 3.199 | 0.097 | -0.105 | -0.070 |
| 2 | -20.603 | 0.152 | 0.191 | 0.178 |
| 10 | $-2.9 \cdot 10^3$ | 0.843 | 0.832 | 0.826 |
| 20 | $-6.6 \cdot 10^6$ | 0.909 | 0.917 | 0.920 |
| 99 | $3.7 \cdot 10^{30}$ | -0.460 | -0.442 | -0.433 |
| Execution Time (s) | 0.017 | 0.019 | 0.018 | 0.019 |

## 3 ode23

Next, we simulate the same system using matlab's built-in `ode23` function and plot it to compare the result with our approximation from Euler's method. The code to do this is as follows:

```
clear all;
tic;
[t,y] = ode23(@f, [0,100], [3.0,0]); %ode23 from t=0 to t=100 and an intial condition of (3,0)
toc
subplot(1,1,1);
plot(t,y);
title("MATLAB ode23 Simulation");
xlabel("Time");
ylabel("Amplitude");
```

## *2.4* 2d **15 / 15**

✓ **+ 7.5 pts** *Simulated system using [3,0] initial conditions*

✓ **+ 7.5 pts** *Included table*

Finally, for h = 0.05, we see that the simulation appears nearly identical to what is expected as seen in Figure 4

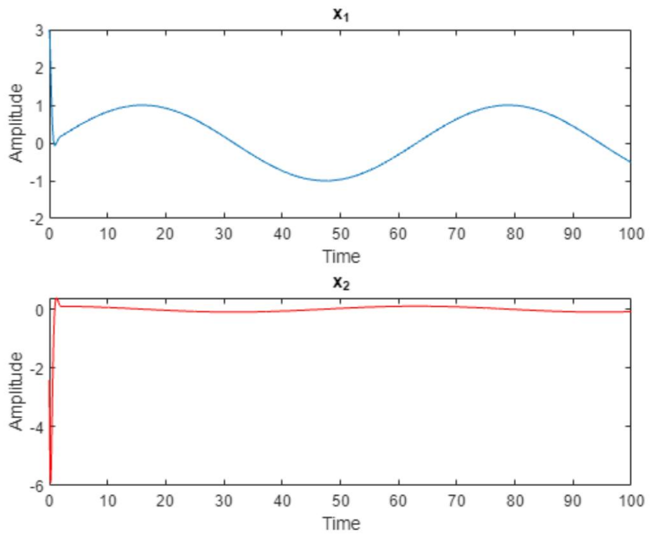


Figure 4: Euler's Method Simulation for h=0.05

Simulation time and accuracy for varying values of $h$ are compared in Table 1 below.

Table 1: Table of selected values from Euler's Method simulation with step size h

| | $x_1$ | | | |
|---|---|---|---|---|
| Time (t) | h = 0.5 | h = 0.2 | h = 0.1 | h = 0.05 |
| 0 | 3 | 3 | 3 | 3 |
| 1 | 3.199 | 0.097 | -0.105 | -0.070 |
| 2 | -20.603 | 0.152 | 0.191 | 0.178 |
| 10 | $-2.9 \cdot 10^3$ | 0.843 | 0.832 | 0.826 |
| 20 | $-6.6 \cdot 10^6$ | 0.909 | 0.917 | 0.920 |
| 99 | $3.7 \cdot 10^{30}$ | -0.460 | -0.442 | -0.433 |
| Execution Time (s) | 0.017 | 0.019 | 0.018 | 0.019 |

## 3 ode23

Next, we simulate the same system using matlab's built-in `ode23` function and plot it to compare the result with our approximation from Euler's method. The code to do this is as follows:

```
clear all;
tic;
[t,y] = ode23(@f, [0,100], [3.0,0]); %ode23 from t=0 to t=100 and an intial condition of (3,0)
toc
subplot(1,1,1);
plot(t,y);
title("MATLAB ode23 Simulation");
xlabel("Time");
ylabel("Amplitude");
```

*3.1* 3b **5 / 5**

✓ **+ 5 pts** *Simulated ode23 correctly*

As we observe in Figure 5, the result of this simulation is remarkably close to the result of our manual step size simulation from part 2.

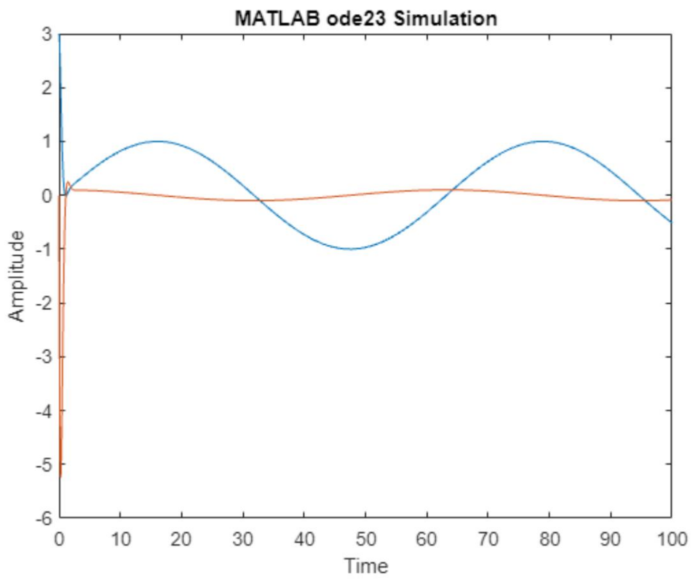


Figure 5: ode23 simulation from t=0 to t=100 and initial conditions [3.0,0]

`tic` and `toc` report the simulation time for this method to be 0.062 seconds, which is over three times slower than Euler's method. It is evident that in this instance we are trading computational time for simulation accuracy.

## 4 Comparison with Calculated Response

From the prelab, it was predicted that the final steady-state response of the system would be $y(t) = 0.999\sin(0.1t - 0.09967)$. We can verify this by plotting this against the output from `lsim` using the following code:

```
t = linspace(0,100,1000);
y = 0.999*sin(0.1*t+0.09967);

plot(t,y,'r');
hold on;
sys = tf([16], [1,6,16]);
u = sin(0.1 * t);
lsim(sys,u,t);
```

As can be see in Figure 6, this prediction is reasonably close to the actual output, although the phase of the sine wave is slightly off from the actual result

*3.2* 3c **5 / 5**

✓ **+ 5 pts** *Correct plot*

As we observe in Figure 5, the result of this simulation is remarkably close to the result of our manual step size simulation from part 2.

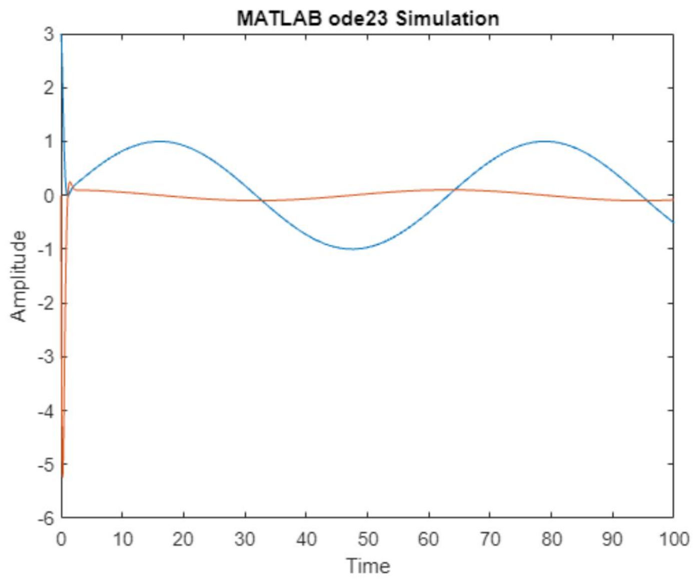


Figure 5: ode23 simulation from t=0 to t=100 and initial conditions [3.0,0]

`tic` and `toc` report the simulation time for this method to be 0.062 seconds, which is over three times slower than Euler's method. It is evident that in this instance we are trading computational time for simulation accuracy.

## 4 Comparison with Calculated Response

From the prelab, it was predicted that the final steady-state response of the system would be $y(t) = 0.999\sin(0.1t - 0.09967)$. We can verify this by plotting this against the output from `lsim` using the following code:

```
t = linspace(0,100,1000);
y = 0.999*sin(0.1*t+0.09967);

plot(t,y,'r');
hold on;
sys = tf([16], [1,6,16]);
u = sin(0.1 * t);
lsim(sys,u,t);
```

As can be see in Figure 6, this prediction is reasonably close to the actual output, although the phase of the sine wave is slightly off from the actual result

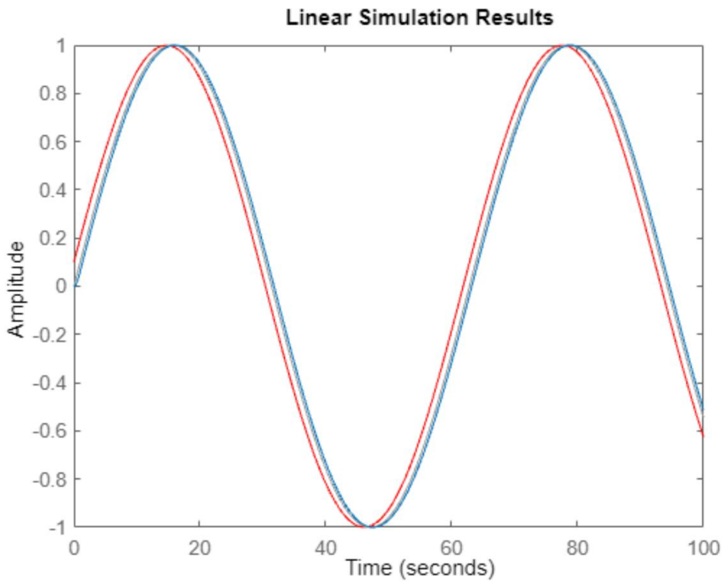


Figure 6: Comparison between calculated (red) and simulated (blue) system response

## 5 Simulation for Different $\omega$

To determine the frequency response of this system, we simulate it and plot the time-domain steady-state output for four different values of $\omega$: 0.04, 4, 16, and 64

```
t = linspace(0,500,1000);
sys = tf([16], [1,6,16]);
u = sin(0.04 * t);
subplot(2,2,1);
lsim(sys,u,t);

t = linspace(0,10,1000);
u = sin(4 * t);
subplot(2,2,2);
lsim(sys,u,t);

t = linspace(0,5,1000);
u = sin(16 * t);
subplot(2,2,3);
lsim(sys,u,t);

t = linspace(0,1,1000);
u = sin(64 * t);
subplot(2,2,4);
lsim(sys,u,t);
```

## *4* Section 4 **10 / 10**

✓ **+ 10 pts** *Compared simulated response with calculated response*

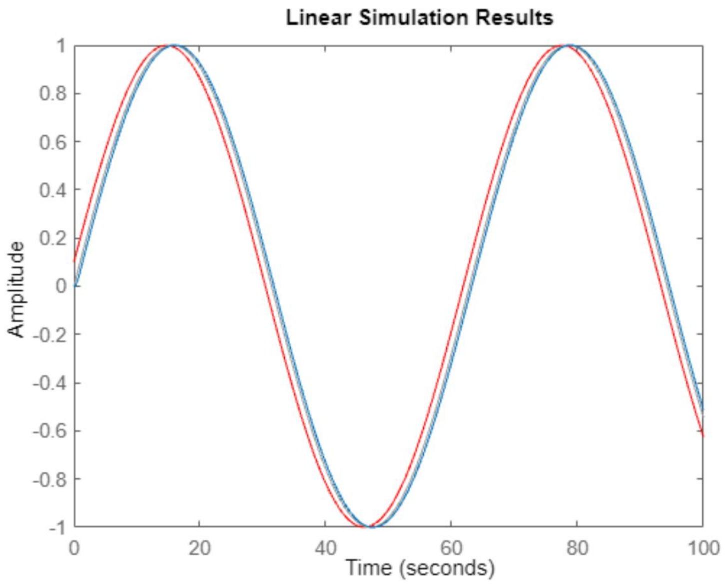


Figure 6: Comparison between calculated (red) and simulated (blue) system response

## 5 Simulation for Different $\omega$

To determine the frequency response of this system, we simulate it and plot the time-domain steady-state output for four different values of $\omega$: 0.04, 4, 16, and 64

```
t = linspace(0,500,1000);
sys = tf([16], [1,6,16]);
u = sin(0.04 * t);
subplot(2,2,1);
lsim(sys,u,t);

t = linspace(0,10,1000);
u = sin(4 * t);
subplot(2,2,2);
lsim(sys,u,t);

t = linspace(0,5,1000);
u = sin(16 * t);
subplot(2,2,3);
lsim(sys,u,t);

t = linspace(0,1,1000);
u = sin(64 * t);
subplot(2,2,4);
lsim(sys,u,t);
```

As we can see in Figure 7, the amplitude of the steady-state response decreases for higher frequencies and the phase shift becomes more pronounced as the frequency approaches $\omega_0$. This means that this is the transfer function of a low-pass filter.

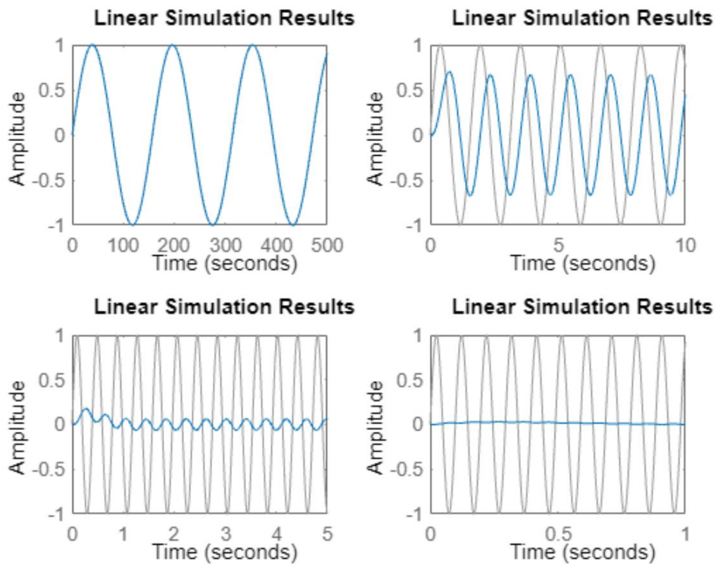


Figure 7: Steady-state response of the system for different $\omega$

To better understand the frequency response of the system, we may create a Bode plot of the transfer function using matlab's `bode()` function as shown in Figure 8.

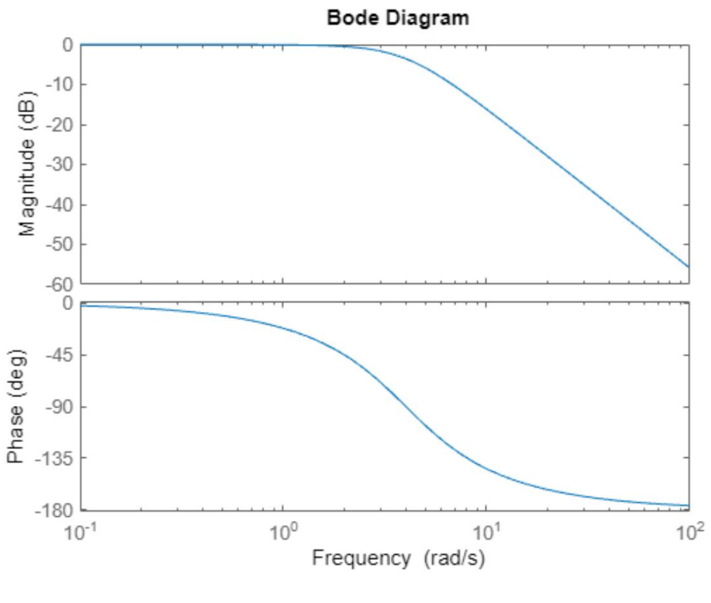


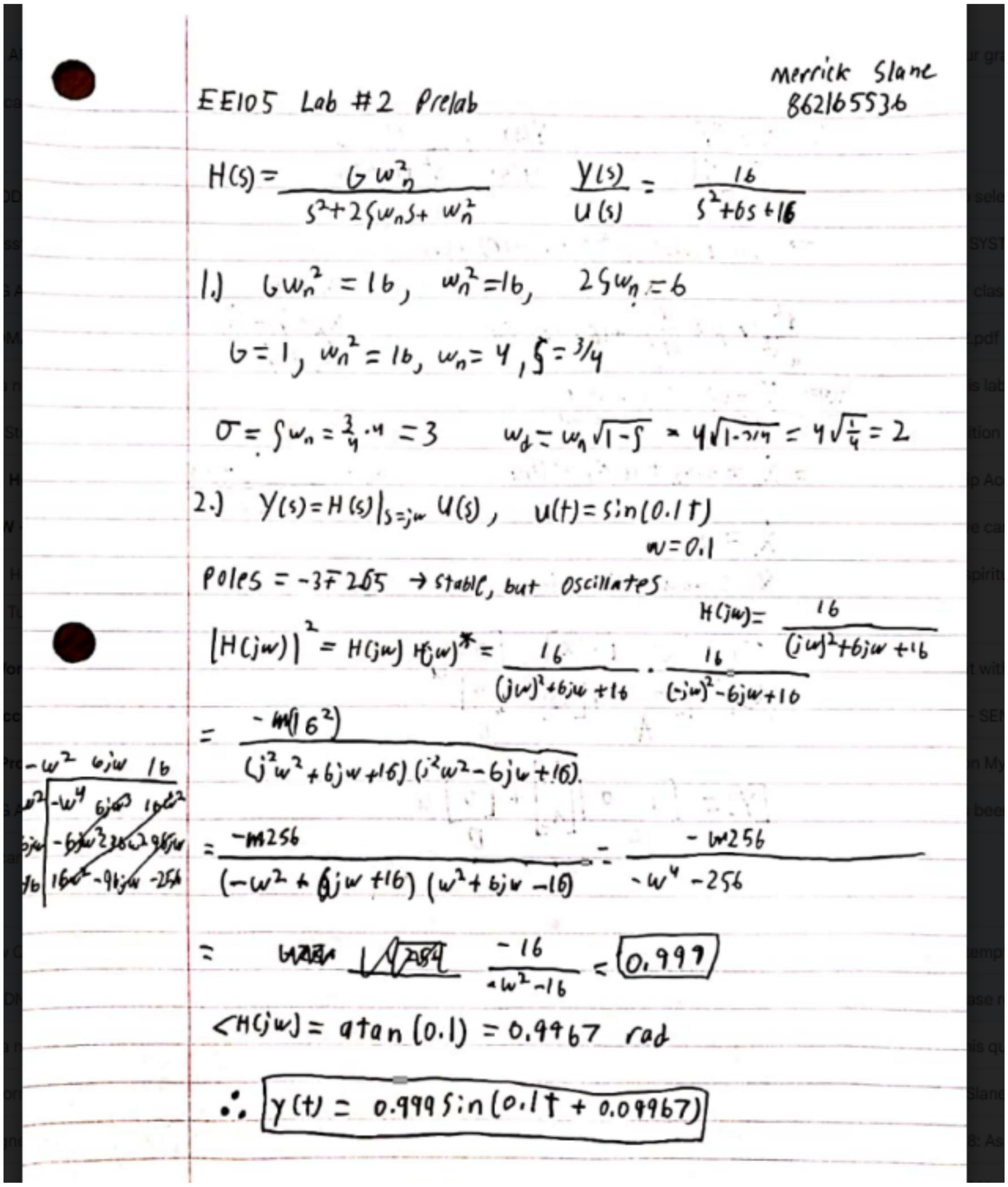
Figure 8: Bode plot of the system

This Bode plot confirms our assertion that this system is a low pass filter.

## *5* Section 5 **10 / 10**

✓ **+ 10 pts** *simulated using different $$w$$ values*

## 6 Prelab



EE105 Lab #2 Prelab

Merrick Slane
862165536

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

1.) $G\omega_n^2 = 16, \quad \omega_n^2 = 16, \quad 2\zeta\omega_n = 6$

$G = 1, \; \omega_n^2 = 16, \; \omega_n = 4, \; \zeta = 3/4$

$\sigma = \zeta\omega_n = \frac{3}{4}\cdot 4 = 3 \qquad \omega_d = \omega_n\sqrt{1-\zeta} = 4\sqrt{1-3/4} = 4\sqrt{\tfrac{1}{4}} = 2$

2.) $Y(s) = H(s)\big|_{s=j\omega} U(s), \quad u(t) = \sin(0.1t)$
$\omega = 0.1$

Poles $= -3 \mp 2j5 \rightarrow$ stable, but oscillates

$H(j\omega) = \frac{16}{(j\omega)^2+6j\omega+16}$

$|H(j\omega)|^2 = H(j\omega)\,H(j\omega)^* = \frac{16}{(j\omega)^2+6j\omega+16} \cdot \frac{16}{(-j\omega)^2-6j\omega+16}$

$= \frac{-\cancel{\omega}(16^2)}{(j^2\omega^2+6j\omega+16)(j^2\omega^2-6j\omega+16)}$

$$\begin{array}{c|ccc} & -\omega^2 & 6j\omega & 16 \\ \hline -\omega^2 & -\omega^4 & 6j\omega^3 & 16\omega^2 \\ 6j\omega & -6j\omega^3 & 36\omega^2 & 96j\omega \\ 16 & 16\omega^2 & -96j\omega & -256 \end{array}$$

$= \frac{-\cancel{\omega}256}{(-\omega^2+6j\omega+16)(\omega^2+6j\omega-16)} = \frac{-\omega 256}{-\omega^4-256}$

$= \cancel{[illegible]} \; \cancel{\sqrt{284}} \quad \frac{-16}{-\omega^2-16} \approx \boxed{0.997}$

$\angle H(j\omega) = \operatorname{atan}(0.1) = 0.0967$ rad

$\therefore \boxed{y(t) = 0.999\sin(0.1t + 0.09967)}$

3.) $x = \begin{bmatrix} y \\ \dot{y} \end{bmatrix}$ $\qquad \frac{Y(s)}{U(s)} = \frac{16}{s^2+6s+16}$

$\rightarrow \quad 16\,u(s) = (s^2+6s+16)\,Y(s)$

$$\frac{d^2}{dt^2}y + 6\frac{d}{dt}y + 16y = 16u$$

$x_1 = y$

$x_2 = \dot{y} = \dot{x}_1$

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

$\boxed{f(s) > \quad \ddot{y} + 6\dot{y} + 16y = 16u}$

$= \dot{x}_2 + 6x_2 + 16x_1 = 16u$ $\qquad x_2 = \frac{16u - \dot{x}_2 - 16x_1}{6}$

$\dot{x}_1 = x_2$

$\dot{x}_2 = 16u - 6x_2 - 16x_1$ $\qquad \boxed{n = 2}$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -16 & -6 \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 16 \end{bmatrix}}_{B} u$$

$$[y] = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_{D} u$$

## 6 Prelab 25 / 25

✓ **+ 25 pts** *Full prelab*