UNIVERSITY OF CALIFORNIA, RIVERSIDE

BOURNS COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EE 105 Lab 2 Solution

Euler's

LUIS FERNANDO ENRIQUEZ-CONTRERAS

Contents

List of Figures

List of Tables

Listings

1 Introduction

This laboratory exercise aims to equip you with a firm understanding of the practicalities and potential challenges associated with numerically solving differential equations. Through hands-on simulations using MATLAB, you'll gain valuable experience in modeling and analyzing dynamic systems.

1.1 Objectives

- Master the concepts: Gain a comprehensive understanding of the key principles and methods involved in numerical differential equation solvers
- Develop simulation skills: Learn how to leverage MATLAB's capabilities to construct simulations of dynamic systems accurately and efficiently
- Identify potential pitfalls: Be aware of common issues and limitations that can arise when solving differential equations numerically, enabling you to approach numerical solutions with prudence and insight
- Apply your knowledge: Put your newfound skills into practice by constructing and analyzing your own dynamic system simulation in MATLAB

2 Pre-Lab

Given the transfer function:

$$H(s) = \frac{36}{s^2 + 3s + 36}$$

We can determine the following values:

$$\omega_n = 6$$
$$G = 1$$
$$\zeta = \frac{3}{12} = \frac{1}{4}$$
$$\sigma = \zeta\omega_n = \frac{6}{4} = 1.5$$
$$\omega_d = \omega_n\sqrt{1-\zeta} \approx 5.2$$
$$T_r = \frac{1.8}{\omega_n} \; T_p = \frac{\pi}{\omega_d} \quad T_s = \frac{4.6}{\sigma} \; M_p(\zeta) = e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}$$
$$T_r = 0.3 \; T_p \approx 0.60384 \quad T_s = 3.066\overline{6} \; M_p(\zeta) \approx 0.4329$$

The steady state response is given by:

$$
\begin{aligned}
H(s) &= \frac{36}{s^2+3s+36} \\
H(s)|_{s=j\omega} &= \frac{36}{36-\omega^2+3j\omega} \\
|H(j\omega)|^2 &= H(j\omega)H(j\omega)^* \\
&= \left(\frac{36}{36-\omega^2+3j\omega}\right)\left(\frac{36}{36-\omega^2-3j\omega}\right) \\
|H(j\omega)| &= \frac{36}{\sqrt{\left(\left(36-\omega^2\right)^2+9\omega^2\right)}} \approx 1.0002 \\
\angle H(j\omega) &= \left(\frac{36}{36-\omega^2+3j\omega}\right)\left(\frac{36-\omega^2-3j\omega}{36-\omega^2-3j\omega}\right) \\
&= \frac{36\left(36-\omega^2-3j\omega\right)}{\left(36-\omega^2\right)^2-9\omega^2} = 0.7856^\circ
\end{aligned}
$$

The steady state response is given by the following:

$$y(t) = 1.0002\sin\left(0.1t - 0.7856^\circ\right)$$

Given that $x = \begin{bmatrix} y & \dot{y} \end{bmatrix}^T$

$$
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \ddot{y}(t) \\
\frac{Y(s)}{U(s)} &= \frac{36}{s^2+3s+36} \\
Y(s)\left(s^2+3s+36\right) &= 36U(s) \\
s^2Y(s)+3sY(s)+36Y(s) &= 36U(s) \\
\mathscr{L}^{-1}\left[s^2Y(s)+3sY(s)+36Y(s)\right] &= \mathscr{L}^{-1}[36U(s)] \\
\ddot{y}(t)+3\dot{y}(t)+36y(t) &= 36u(t) \\
\ddot{y}(t) &= 36u(t)-3\dot{y}(t)-36y(t) \\
\ddot{y}(t) &= 36u(t)-3x_2-36x_1 \\
\dot{x}_2 &= 36u(t)-3x_2-36x_1 \\
\dot{x} &= [x_2; 36u(t)-3x_2-36x_1] \\
y &= [x_1]
\end{aligned}
$$

The system is linear, therefore we can solve for A, B, C, D

$$A = \begin{bmatrix} 0 & 1 \\ -36 & -3 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 36 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad D = [0]$$

3 Linear System Function

Listing 1: lsim Simulation of the State Space Model with domain of 0-100 seconds

```
clear all;
clc;
% Define the transfer function of the system
H = tf(36,[1 3 36]);
% Define A,B,C,D
A = [0,1; -36,-3];
B = [0;36];
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) usinkg sinusoidal signal
tau = 2*pi/0.1; % Period = 2*pi/0.1
% 0:Ts:Tf
Ts = 0.01; % Time step
Tf = 100; % Duration
[u,t] = gensig('sin',tau,Tf,Ts);
% Simulate the system
lsim(sys,u,t,x0);
grid on;
legend(System response);
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/lsim_run_100s.pdf', 'ContentType','vector');
```

This code generates the plot for Figure 1.

Listing 2: lsim Simulation of the State Space Model with domain of 0-4 seconds

```
clear all;
clc;
% Define the transfer function of the system
H = tf(36,[1 3 36]);
% Define A,B,C,D
A = [0,1; -36,-3];
B = [0;36];
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 = 4; % Duration
[u,t] = gensig('sin',tau,Tf,Ts);
% Simulate the system
lsim(sys,u,t,x0);
grid on;
legend(System response);
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/lsim_run_4s.pdf', 'ContentType','vector');
```

This code generates the plot for Figure 2.

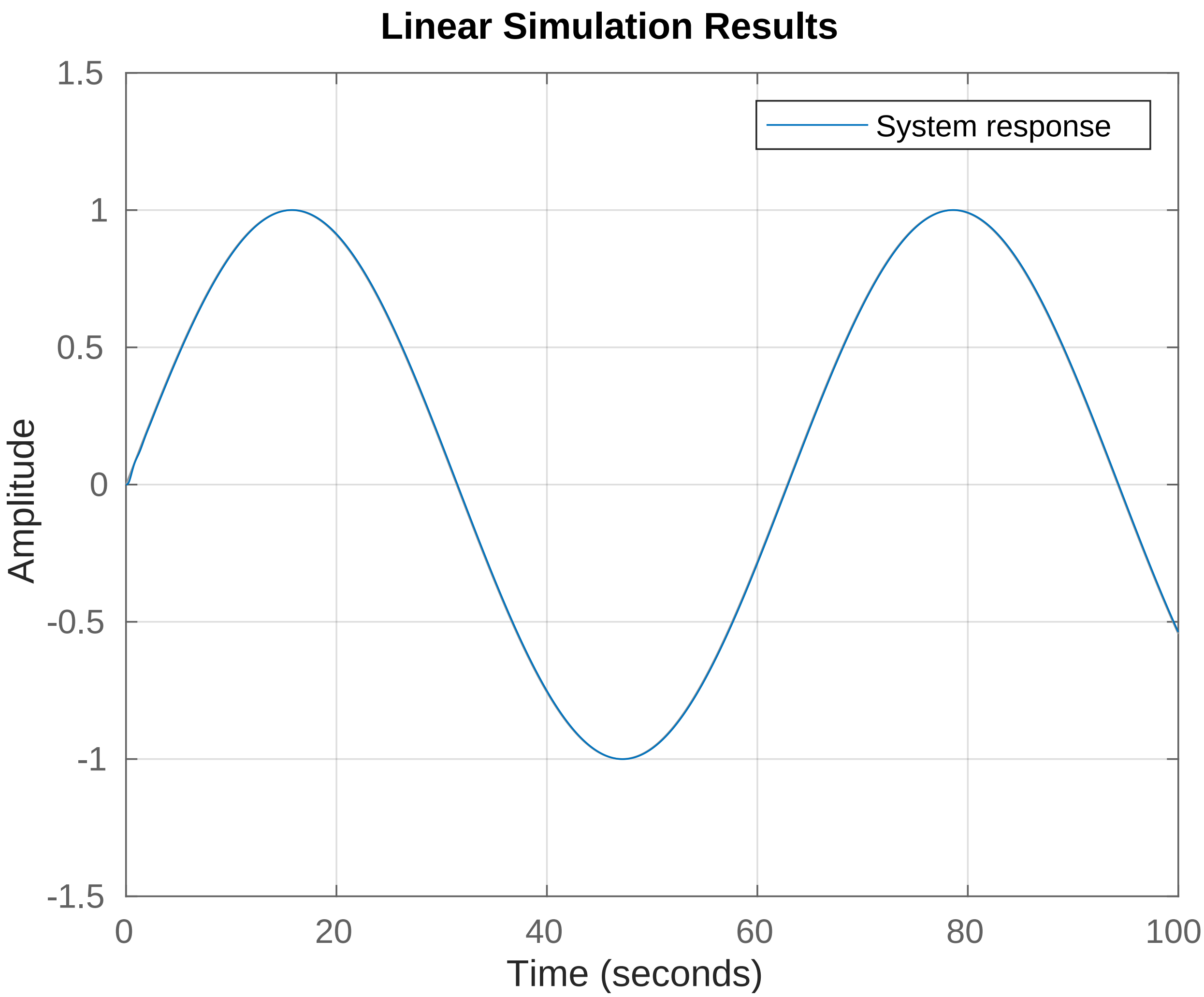

Figure 1: lsim Simulation of the State Space Model with a domain of 0-100 seconds

This is the plot for Listing 1.

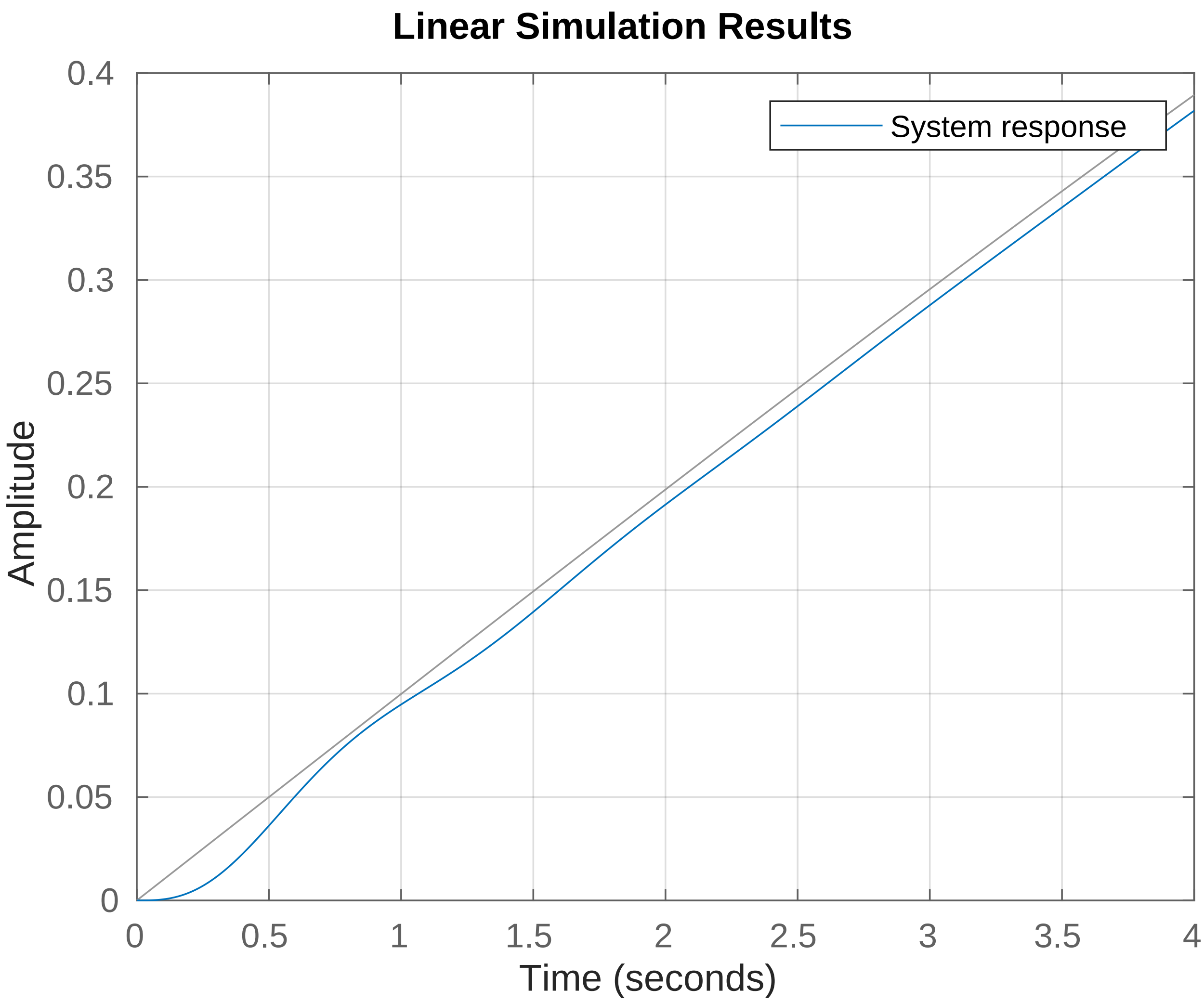

Figure 2: lsim Simulation of the State Space Model with a domain of 0-4 seconds

This is the plot for Listing 2.

4 Euler Integration Routine

Listing 3: Run the Euler function for different step sizes h

```
% 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 = 1; % Time step
Tf = 10; % Duration
% Define initial state
x0 = [3,0];
% Define step-size
for h = [1, 0.1, 0.05, 0.01, 0.001]
    %[u,t] = gensig('sin',tau,Tf,h);
    N = (Tf/h);
    t = h*(0:N-1);
    u = sin(0.1*t);
    filename = append('Fig/Euler_plot_h_', string(h),'.pdf');
    filename_csv = append('Euler_plot_h_', string(h),'.csv');
    sim_t = Euler(t,x0,h,u, filename, filename_csv);
end
```

Listing 4: Function to run Euler recursion

```
function sim_t = Euler(t,x0,h,u, filename, filename_csv)
% For the given initial condition x0 and step size
% h this function uses Euler integration to
% numerically solve the differential equation
% of the transfer function.
% Function output: sim_t, Euler Simulation time cost
tic; % start the clock
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; % IC
for i=1:N
    [dx] = f(x(:,i),u(i));
    x(:,i+1) = x(:,i) + dx*h;
end
sim_t = toc; % end the clock
sim_t = sim_t*1000;
```

```matlab
display(append('Euler Simulation Took = ', string(sim_t) ,'s'));
figure
subplot(2, 1, 1);
plot(t,x(1,1:i));
hold on;
plot(t,u);
legend('System response', 'Input signal');
plot_title = {[append('Euler Simulation: for step-size h = ', string(h))] [append('Simulation took ', string(sim_t), ' ms')] ['$\bf y$ vs. Time']};
grid on;
title(plot_title, 'Interpreter', 'latex');
xlabel('Time (s)');
ylabel('$ \bf y$', 'Interpreter', 'latex');
subplot(2, 1, 2);
plot (t,x(2,1:i));
title ('y prime vs. Time', 'Interpreter', 'latex')
xlabel('Time (S)');
ylabel('y prime', 'Interpreter', 'latex');
grid on;
writematrix(x(1,1:i),filename_csv);
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, filename,'ContentType','vector');
end
```

Listing 5: Function for the State Space of the representation of the system

```matlab
function [dx] = f(x,u)
% x -- [2xn] column vector
% u -- [1xn] vector
A = [0,1; -36,-3];
B = [0;36];
dx = A*x + B*u;
end
```

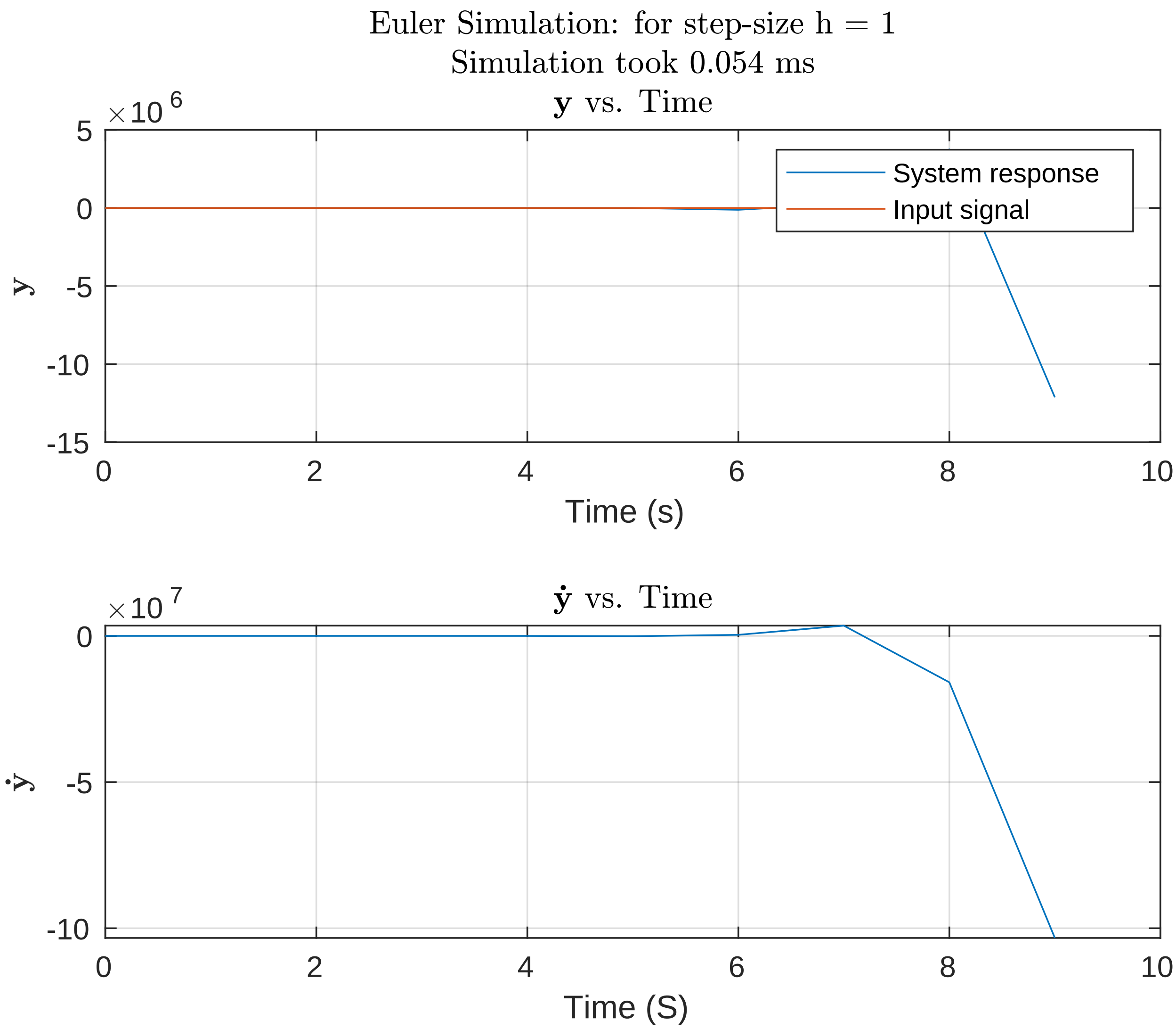

Figure 3: Euler Plot for $h = 1$

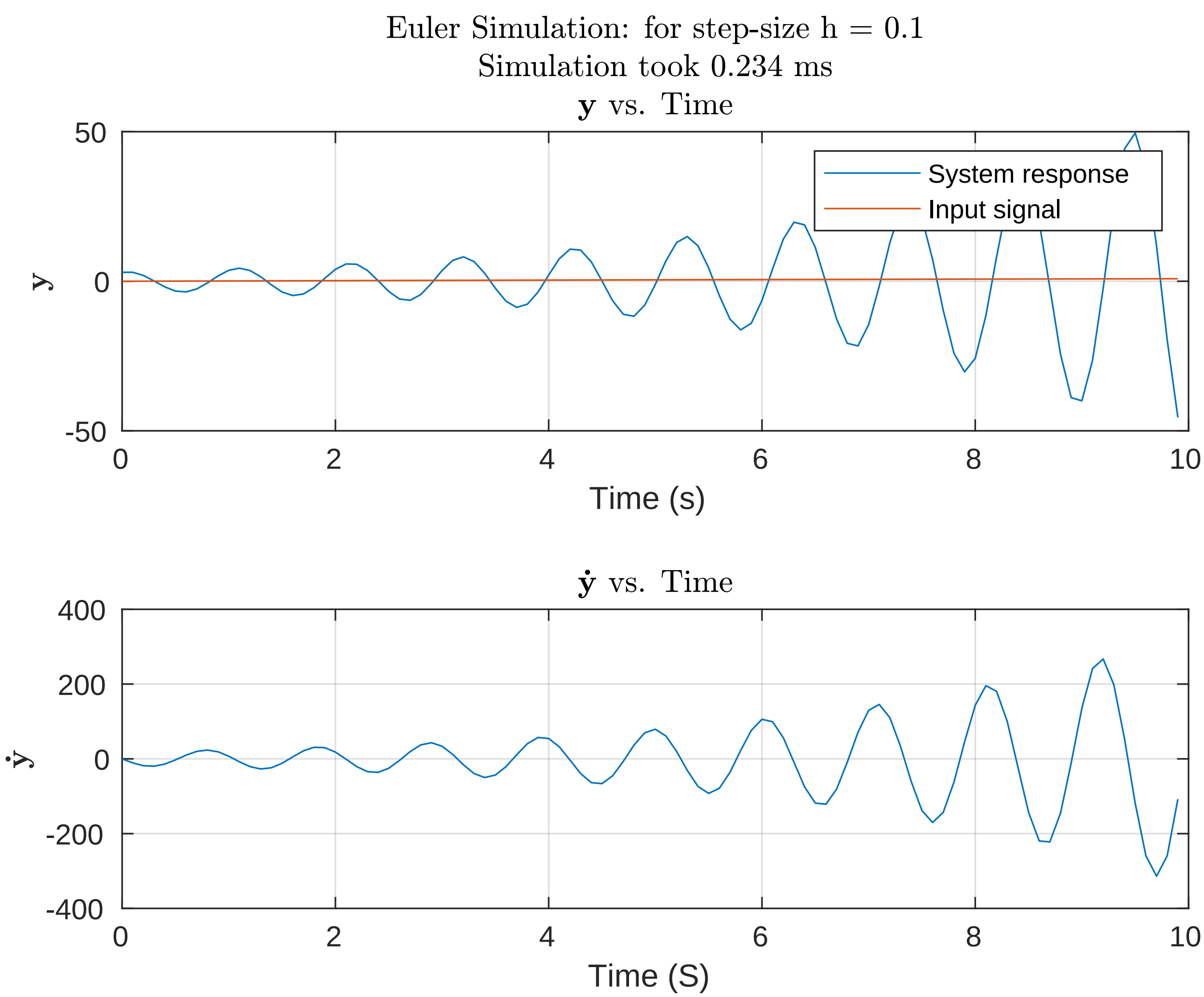

Figure 4: Euler Plot for $h = 0.1$

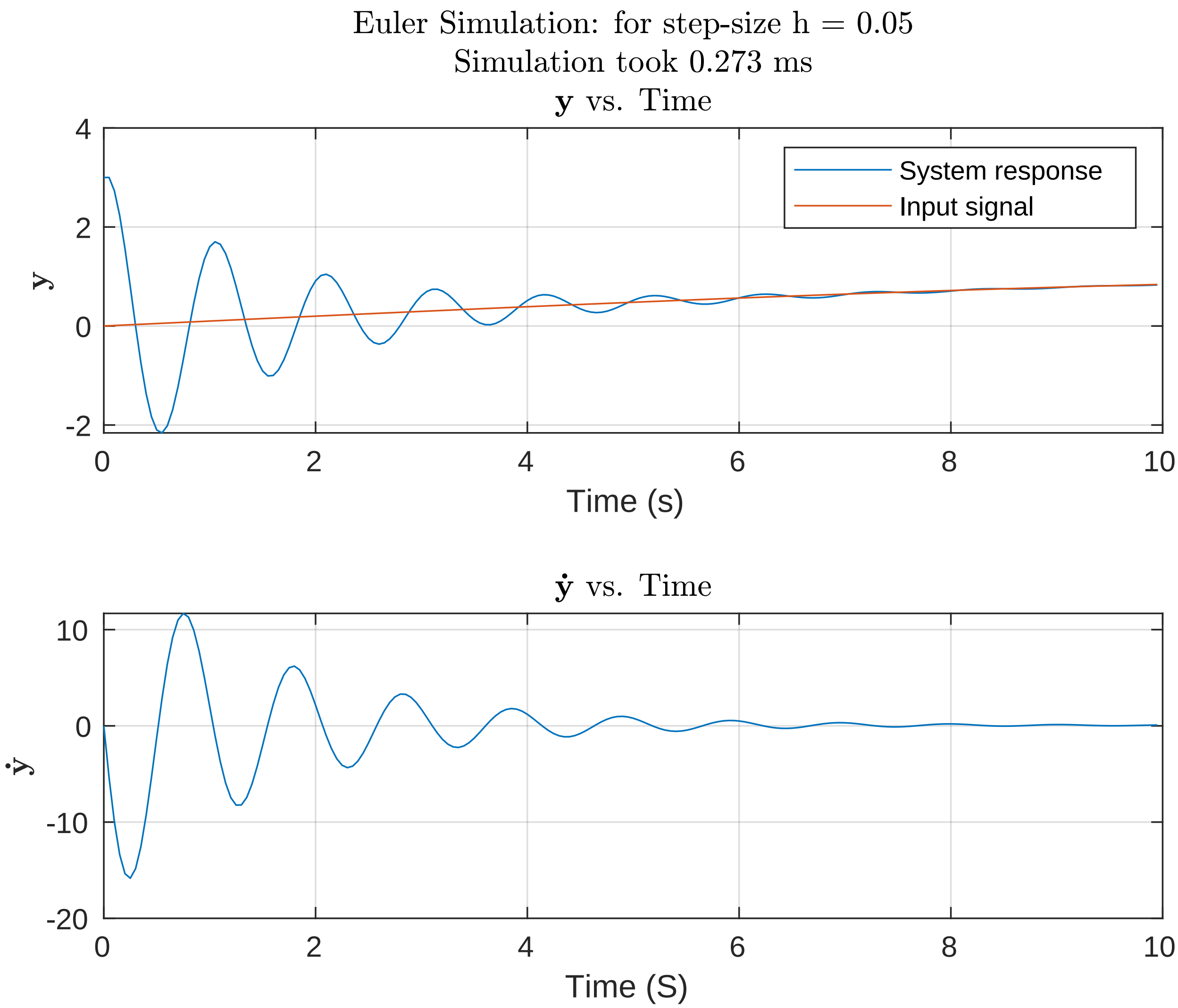

Figure 5: Euler Plot for $h = 0.05$

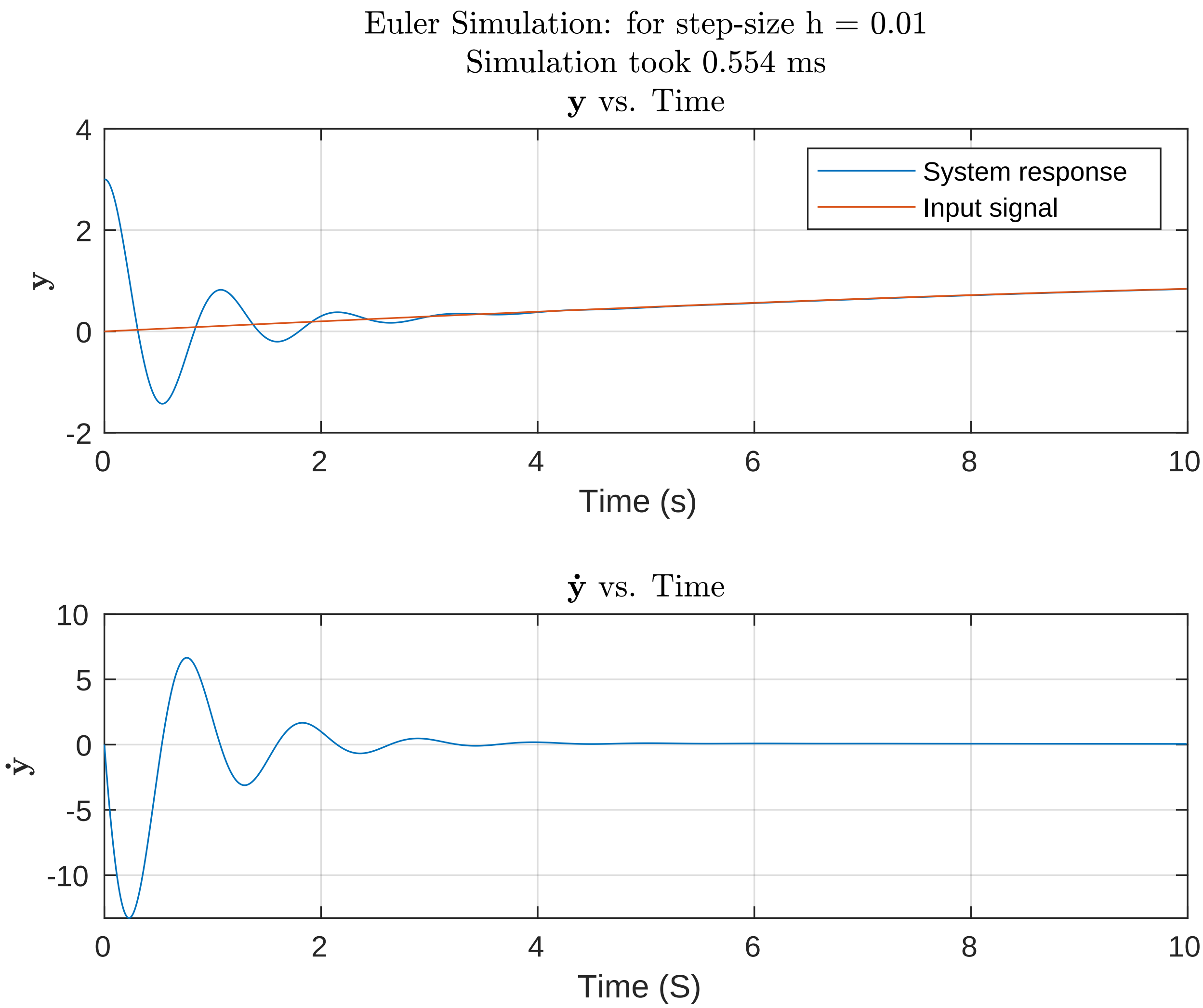

Figure 6: Euler Plot for $h = 0.01$

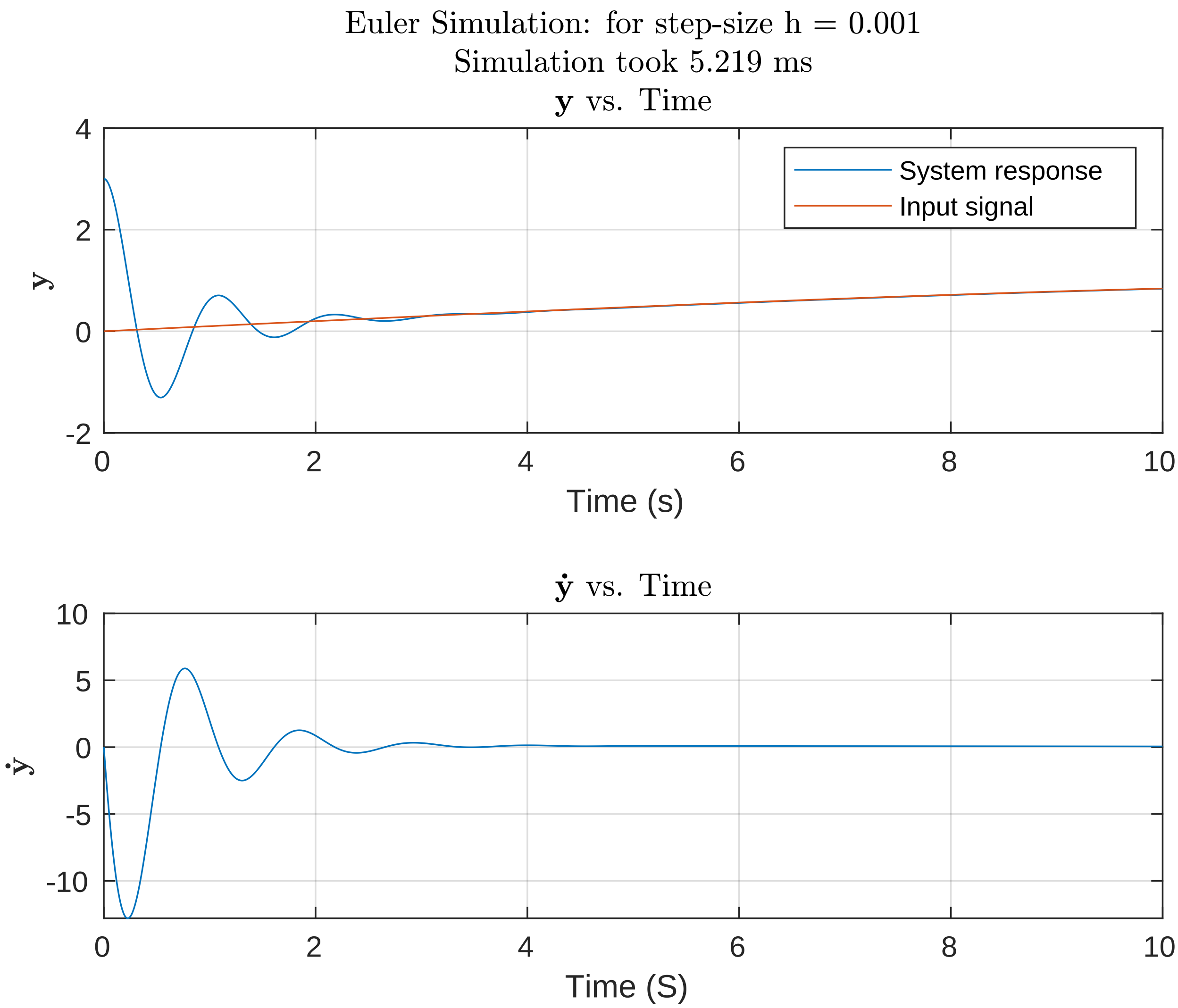

Figure 7: Euler Plot for $h = 0.001$

The curves in Figures 3, 4 do not settle at all. The curve in Figure 5 does settle, but does so at around 7-8 seconds. The curves Figures 6, 7 do settle at around 3 seconds which is close to T_S calculated.

Table 1: Cputimes for Various Euler's Step Sizes

	$h = 1.0$ s		$h = 0.1$ s		$h = 0.01$ s		$h = 0.001$ s	
Time, t	k	x_k	k	x_k	k	x_k	k	x_k
0.0	0	3	0	3	0	3	0	3
2.0	2	3	20	0.9790	200	0.289	2000	0.2508
4.0	4	6.5940	40	-3.6305	400	0.3764	4000	0.3775
6.0	6	-3784.1155	60	-14.03887	600	0.5561	6000	0.5574
8.0	8	3752170.88274	80	-30.2920	800	0.7109	8000	0.7116
10.0	10	-12125600.5695	100	-45.5193	1000	0.8366	10000	0.8371
CPUTIME (ms)		0.054		0.234		0.554		5.219

After running the sine input with different step-sizes, the computational time and accuracy are shown in Table 1. The following conclusions can be made'

- There is a direct relationship between step-size and computational cost. Reducing the step-size by 10x leads to a roughly 10x increase in computation time. This suggests that the chosen numerical method might not be very efficient for small step-sizes, and alternative methods should be considered if high accuracy is required at small scales.

- The simulation exhibits stability issues for large step-sizes (h > 0.01). This indicates that the chosen step-size might be too large to capture the dynamics of the system accurately. Stable simulations are crucial for obtaining reliable results, so it's important to address this instability

- For smaller step-sizes (h < 0.01), the simulation becomes stable and the accuracy improves as the step-size decreases. This suggests that the chosen numerical method is more accurate for smaller step-sizes, but at the cost of increased computational effort.

- There is a trade-off between the accuracy of the simulation and the computational cost. Smaller step-sizes lead to more accurate results but require more computation time. The optimal step-size depends on the specific requirements of the study. If high accuracy is essential, even if it means longer computation times, then smaller step-sizes should be used. However, if computational efficiency is a priority, then a larger step-size might be acceptable if the accuracy remains within acceptable limits.

5 ODE23 with Zero Input

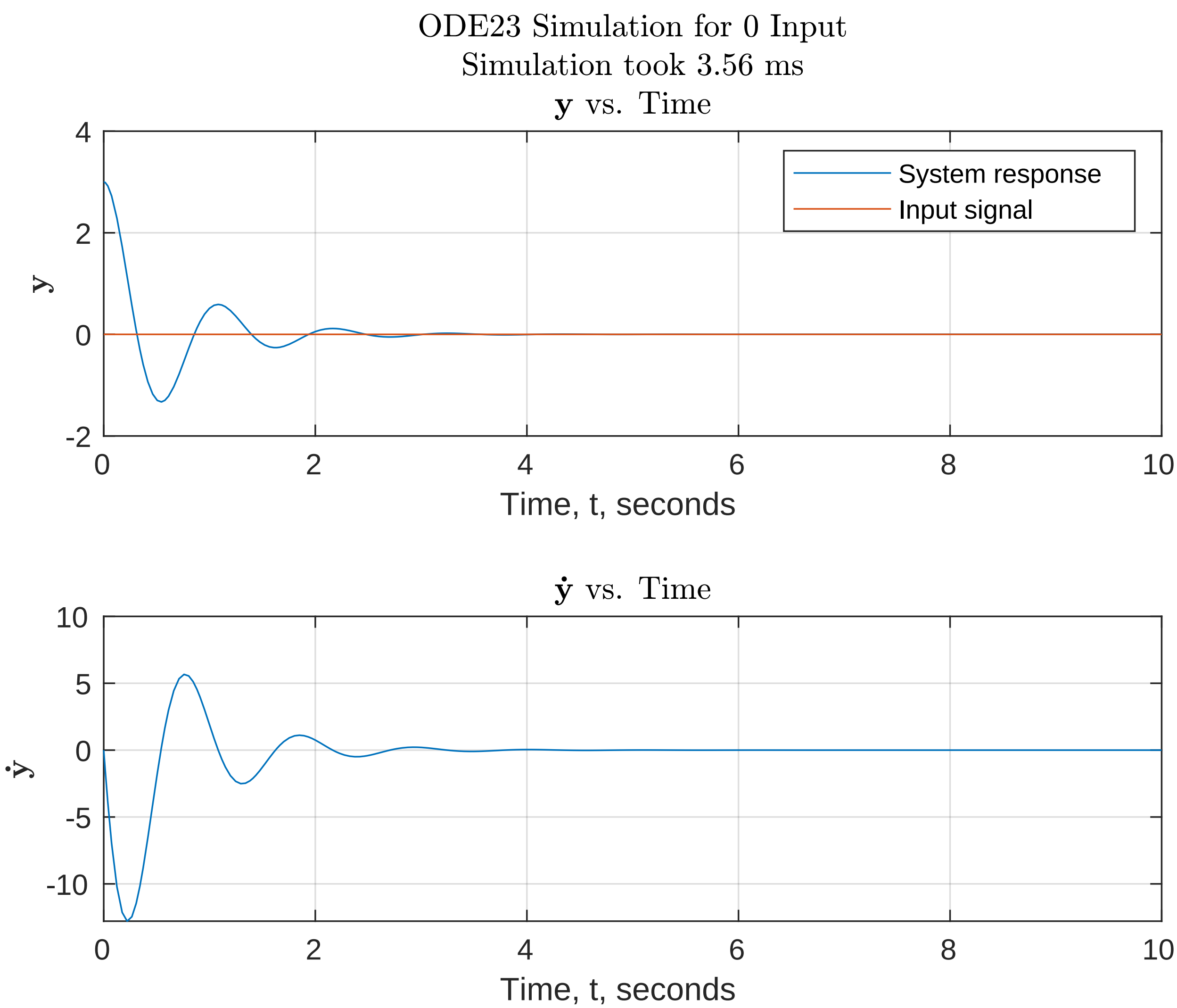

Figure 8: ODE Plot for Zero Input

6 Varying ω in 1.0 sin(ωt) using ODE23

Listing 6: Function for the State Space of the representation of the system for ODE23

```
function [dx, u] = f(t,x,omega)
% x -- [2xn] column vector
% u -- [1xn] vector
A = [0,1; -36,-3];
B = [0;36];
u = sin(omega*t);
dx = A*x + B*u;
end
```

Listing 7: Run ODE23 for sin(0.1t)

```
tspan = [0 10]; % Interval of integration
x0 = [3 0]; % Initial condition
omega = 0.1;
tic
[t_out,y] = ode23(@(t,x) f_ode(t,x,omega), tspan, x0);
tm = toc;
tm = tm*1000;
h = length(t_out) \ (tspabn(2) - tspan(1));
t = tspan(1):h:tspan(2);
u = sin(omega*t);
display(append('ODE23 Simulation Took ', string(tm), 's'));
figure;
subplot(2, 1, 1);
plot(t_out, y(:,1));
hold on;
plot(t,u);
legend('System response', 'Input signal');
plot_title = {[append('ODE23 Simulation: for omega = ', string(omega))] [
    append('Simulation took ', string(sim_t), 's')] ['$ \bf y$ vs. Time']};
title(plot_title, 'Interpreter', 'latex');
xlabel('Time, t, seconds');
ylabel('$ \bf y$', 'Interpreter', 'latex');
grid on;
subplot(2, 1, 2);
```

```
plot(t_out, y(:,2));
title ('y prime vs. Time', 'Interpreter', 'latex');
xlabel('Time, t, seconds');
ylabel('y prime', 'Interpreter', 'latex');
grid on;
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/ode_sin_input_01.pdf','ContentType','vector');
```

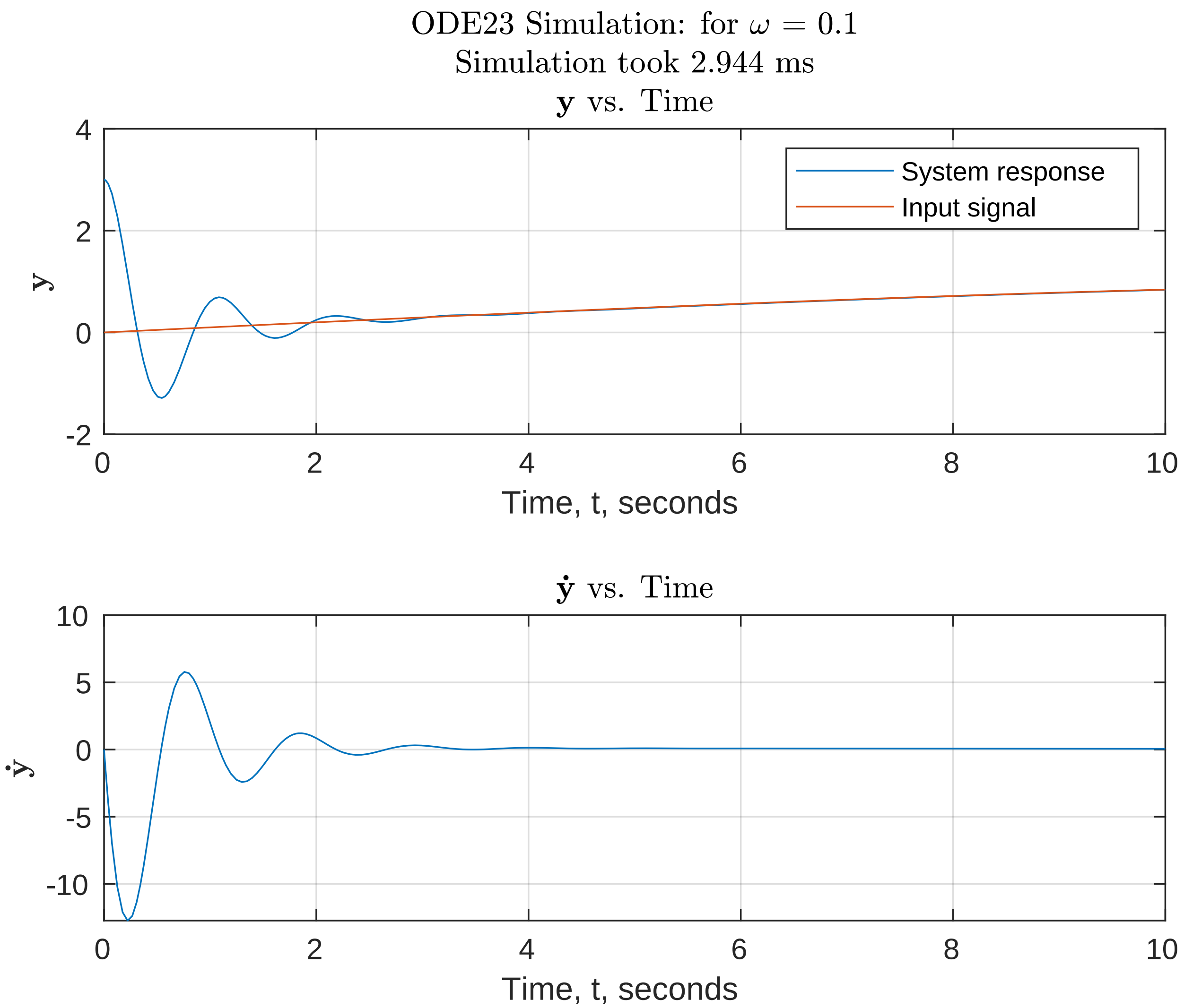

Figure 9: Ode Plot for $\omega = 0.1$

Magnitude and Phase shift of the system response in Figure 9 are near identical to the input signal.

Listing 8: Run ODE23 for various sin(ωt)

```
for i = [1, 5.65, 16]
    tspan = [0 10]; % Interval of integration
    x0 = [3 0]; % Initial condition
    omega = i;
    tic
    [t_out,y] = ode23(@(t,x) f_ode(t,x,omega), tspan, x0);
    tm = toc;
    tm = tm*1000;
    h = length(t_out) \ (tspan(2) - tspan(1));
    t = tspan(1):h:tspan(2);
    u = sin(omega*t);
    display(append('ODE23 Simulation Took ', string(tm), 's'));
    figure;
    subplot(2, 1, 1);
    plot(t_out, y(:,1));
    hold on;
    plot(t,u);
    legend('System response', 'Input signal');
    plot_title = {[append('ODE23 Simulation: for omega = ', string(omega))]
        [append('Simulation took ', string(sim_t), ' s')] ['$ \bf y$ vs. Time
        ']};
    title(plot_title, 'Interpreter', 'latex');
    xlabel('Time, t, seconds');
    ylabel('$ \bf y$', 'Interpreter', 'latex');
    grid on;
    subplot(2, 1, 2)
    plot(t_out, y(:,2))
    title ('y prime vs. Time', 'Interpreter', 'latex');
    xlabel('Time, t, seconds');
    ylabel('y prime', 'Interpreter', 'latex');
    grid on;
    filename = append('Fig/ode_sin_input_', string(omega), '.pdf');
    fig = gcf; % Obtains current graphic in matlab
    exportgraphics(fig, filename,'ContentType','vector');
end
```

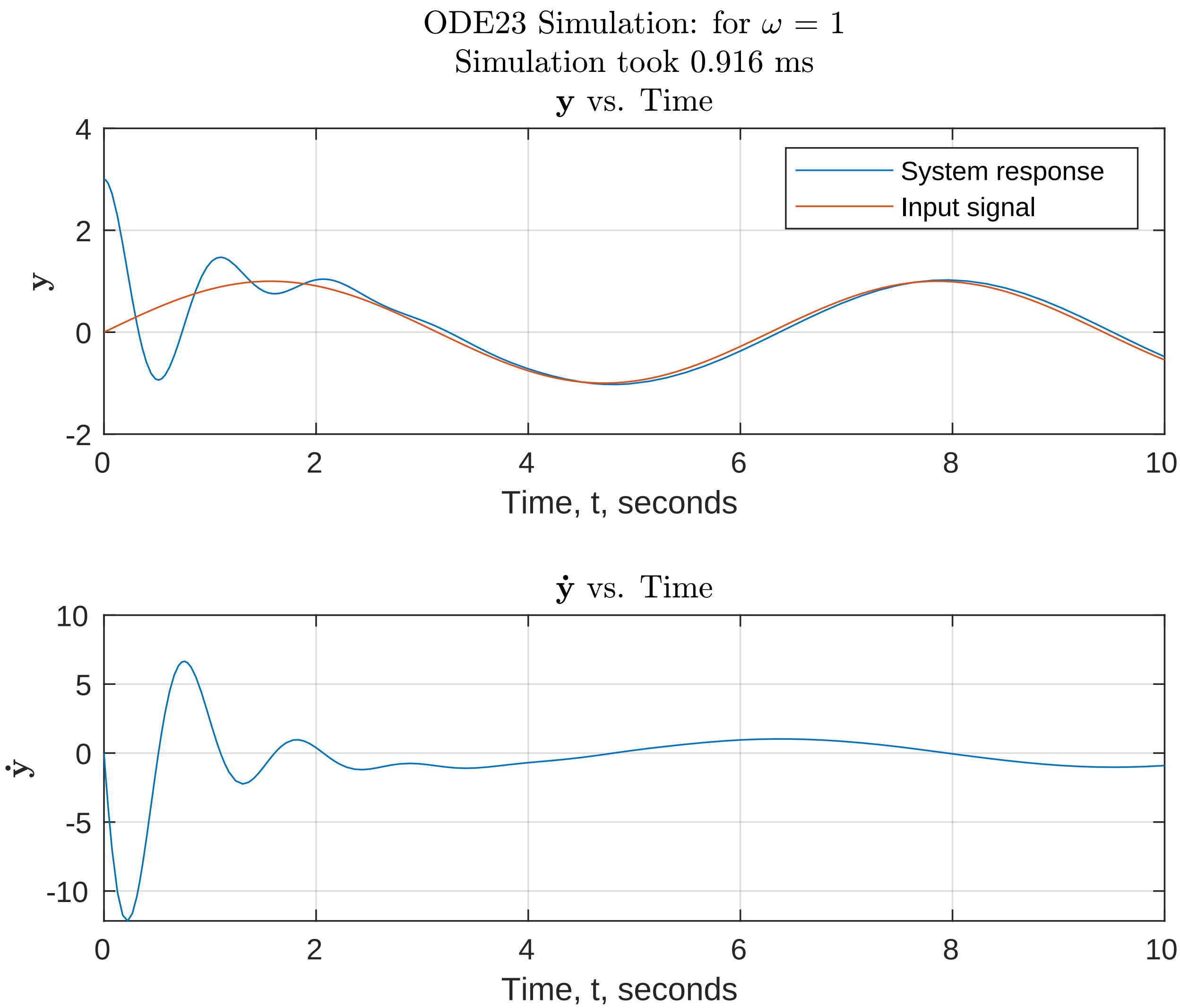

Figure 10: Ode Plot for $\omega = 1$

Magnitude and Phase shift of the system response in Figure 10 are near identical to the input signal.

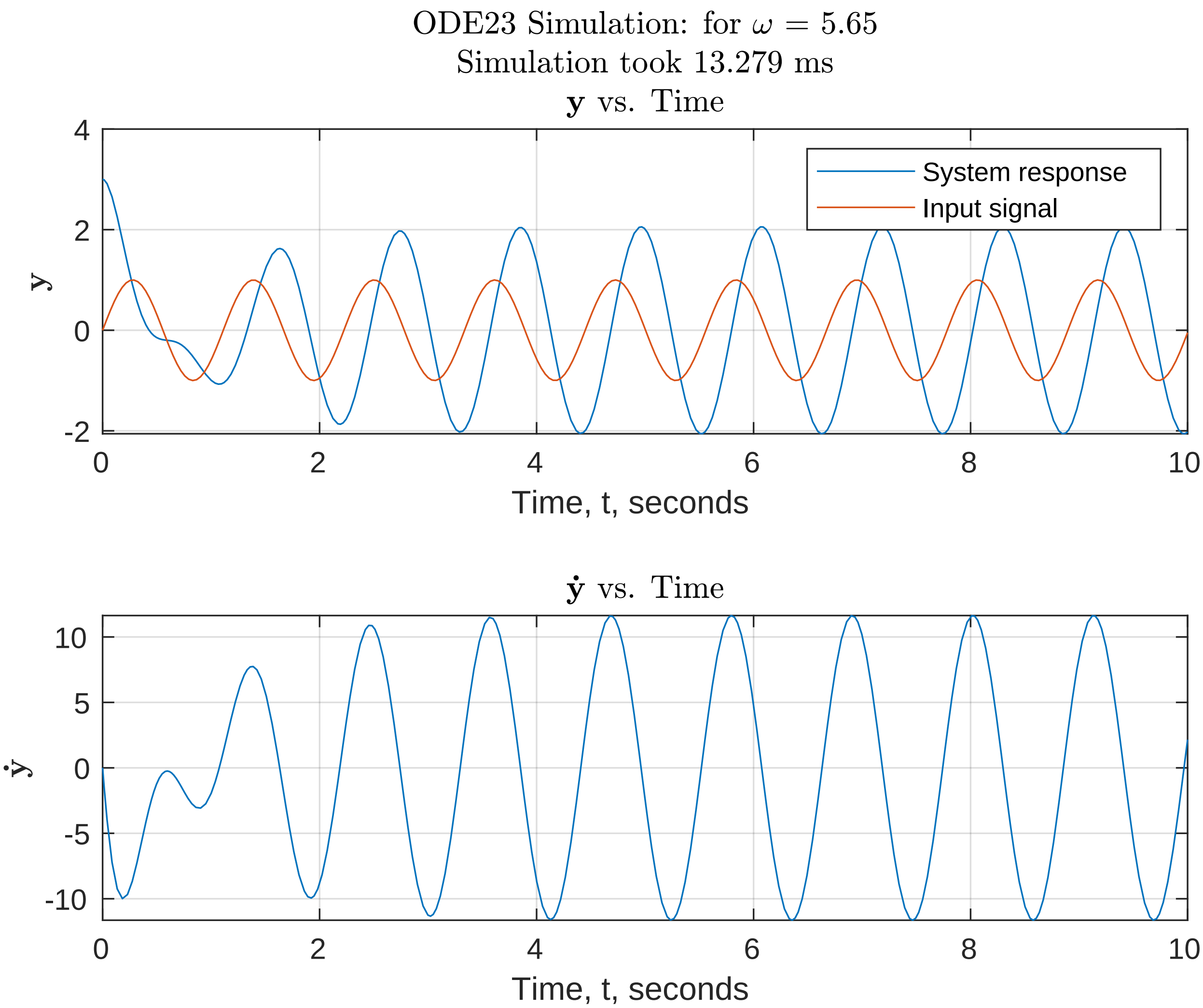

Figure 11: Ode Plot for $\omega = 5.65$

The magnitude of system response Figure 11 is ≈ 2,while the phase shift is ≈ -1.3 rad.

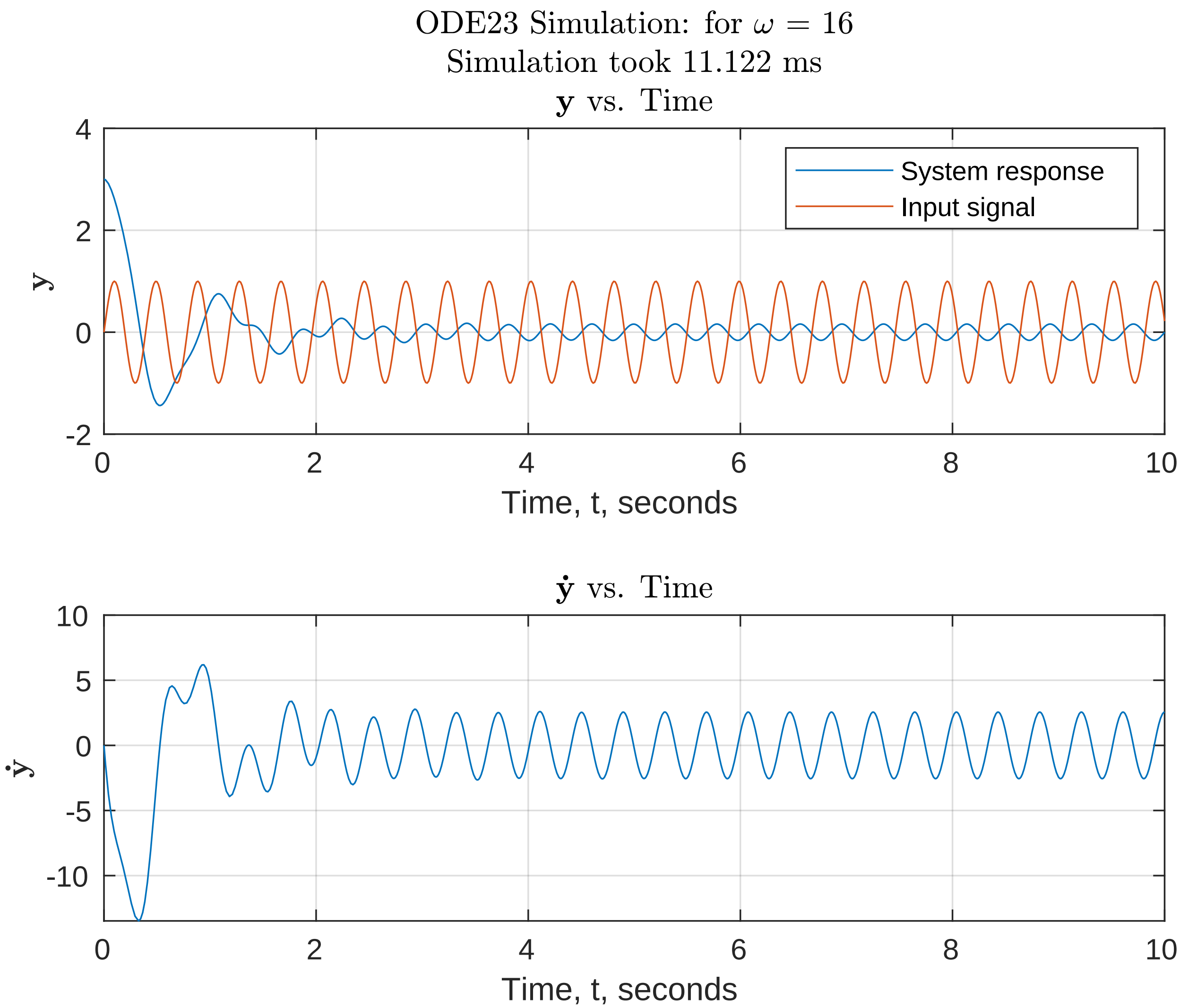

Figure 12: Ode Plot for $\omega = 16$

The magnitude of system response of Figure 12 is ≈ 1.6, while the phase shift is ≈ -3.3 rad.

6.1 Bode Plot

Listing 9: Bode Plot Code for State Space Function

```
% Define the state space function
H = tf(36,[1 3 36]);
opts = bodeoptions('cstprefs');
opts.MagUnits = 'abs';
opts.PhaseUnits = 'rad';
figure;
bode(H, opts);
fig = gcf; % Obtains current graphic in matlab
exportgraphics(fig, 'Fig/bode_plot.pdf', 'ContentType','vector');% Define A,B,C,D

A = [0,1; -36,-3];
B = [0;36];
C = [1 0];
D = 0;

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

omega = [0.1, 1, 5.65, 16];
[mag,phase] = bode(sys, omega);
phase = deg2rad(phase);
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(1), mag(1), phase(1));
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(2), mag(2), phase(2));
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(3), mag(3), phase(3));
fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(4), mag(4), phase(4));
```

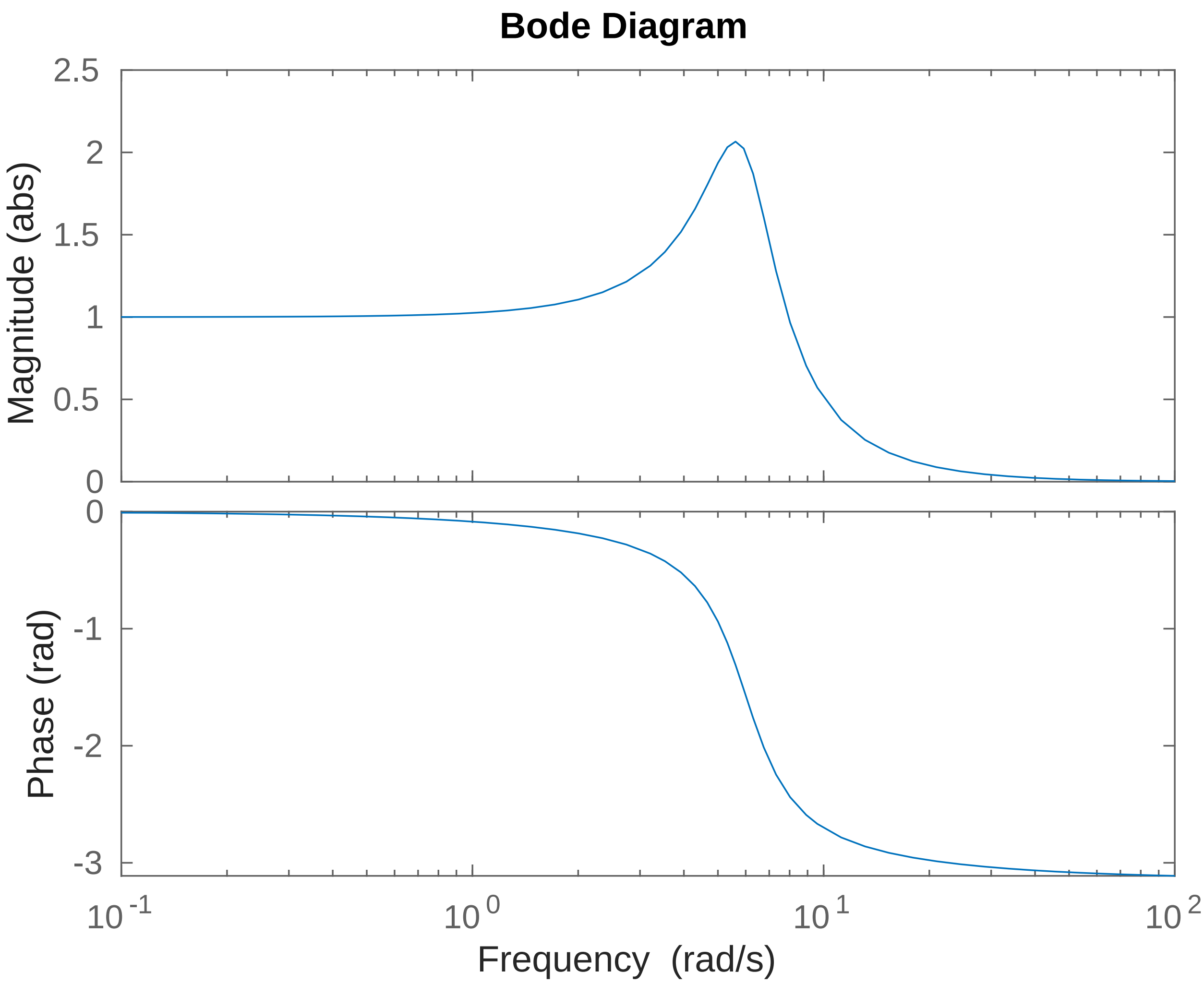

Figure 13: Bode Plot for State Space Model

- At $\omega = 0.1$, the amplitude is 1.000 and the phase is -0.008
- At $\omega = 1$, the amplitude is 1.025 and the phase is -0.0855
- At $\omega = 5.65$, the amplitude is 2.065 and the phase is -1.334
- At $\omega = 16$, the amplitude is 0.160 and the phase is -2.927

The steady state values obtained from the simulations are near identical to the frequency free-response calculations.

7 Conclusion

This MATLAB lab successfully explored the intricacies and challenges of numerically solving differential equations. By constructing a dynamic system simulation, the students gained practical experience in applying various numerical methods and interpreting the results.

- Step Size
  - Smaller step sizes (h < 0.01) enhance accuracy and stabilize the simulation, but at a 10x computational cost increase per 10x step size reduction.
  - Avoid large step sizes (h > 0.01) to ensure stable and reliable simulations.
- Frequency
  - Both Euler integration and sde23 methods accurately predict the expected settling time of $T_s = 3.066\overline{6}$, solidifying the validity of the simulation in a steady-state context.
  - This method effectively predicts the magnitude and phase of the steady-state response, further strengthening confidence in the overall simulation's accuracy.