

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

1	Introduction	3
1.1	Objectives	3
2	Pre-Lab	3
3	LSIM and gensig	5
4	Varying Step-Size of $1.0 \sin(0.1)$	8
5	ODE23 with Zero Input	17
6	Varying ω in $1.0 \sin(\omega t)$	17
6.1	Bode Plot	23
7	Conclusion	25

List of Figures

1	lsim Simulation of the State Space Model with a domain of 0-100 seconds	7
2	lsim Simulation of the State Space Model with a domain of 0-4 seconds	8
3	Euler Plot for $h = 1$	11
4	Euler Plot for $h = 0.1$	12
5	Euler Plot for $h = 0.05$	13
6	Euler Plot for $h = 0.01$	14
7	Euler Plot for $h = 0.001$	15
8	ODE Plot for Zero Input	17
9	Ode Plot for $\omega = 0.1$	19
10	Ode Plot for $\omega = 1$	21
11	Ode Plot for $\omega = 0.09$	22
12	Ode Plot for $\omega = 0.001$	23
13	Bode Plot for State Space Model	25

List of Tables

1	Cputimes for Various Euler's Step Sizes	16
---	---	----

Listings

1	lsim Simulation of the State Space Model with domain of 0-100 seconds	5
2	lsim Simulation of the State Space Model with domain of 0-4 seconds	5
3	Function for the State Space of the representation of the system	8
4	Function to run Euler recursion	9
5	Run the Euler function for different step sizes h	10
6	Function for the State Space of the representation of the system for ODE23	17
7	Run ODE23 for $\sin(0.1t)$	18
8	Run ODE23 for various $\sin(\omega t)$	19
9	Bode Plot Code for State Space Function	23

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:

$$\omega_n = 6$$

$$G = 1$$

$$\zeta = \frac{3}{12} = \frac{1}{4}$$

$$\sigma = \zeta \omega_n = \frac{6}{4} = 1.5$$

$$\omega_d = \omega_n \sqrt{1 - \zeta^2} \approx 5.2$$

$$T_r = \frac{1.8}{\omega_n} \quad T_p = \frac{\pi}{\omega_d} \quad T_s = \frac{4.6}{\sigma} \quad M_p(\zeta) = e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}$$

$$T_r = 0.3 \quad T_p \approx 0.60384 \quad T_s = 3.066\bar{6} \quad M_p(\zeta) \approx 0.4329$$

The steady state response is given by:

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

The steady state response is given by the following:

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

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

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

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 LSIM and gensig

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

```

1 clear all;
2 clc;
3 % Define the transfer function of the system
4 H = tf(36,[1 3 36]);
5 % Define A,B,C,D
6 A = [0,1; -36,-3];
7 B = [0;36];
8 C = [1 0];
9 D = 0;
10 % Define state-space model
11 sys = ss(A,B,C,D);
12 % Define initial state
13 x0 = [0,0];
14 % Define the input u(t) = 1.0sin(0.1t) using sinusoidal signal
15 tau = 2*pi/0.1; % Period = 2*pi/0.1
16 % 0:Ts:Tf
17 Ts = 0.01; % Time step
18 Tf = 100; % Duration
19 [u,t] = gensig('sin',tau,Tf,Ts);
20 % Simulate the system
21 lsim(sys,u,t,x0);
22 grid on;
23 legend(System response);
24 fig = gcf; % Obtains current graphic in matlab
25 exportgraphics(fig, 'Fig/lsim_run_100s.pdf', 'ContentType','vector');
```

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

```
1 clear all;
2 clc;
3 % Define the transfer function of the system
4 H = tf(36,[1 3 36]);
5 % Define A,B,C,D
6 A = [0,1; -36,-3];
7 B = [0;36];
8 C = [1 0];
9 D = 0;
10 % Define state-space model
11 sys = ss(A,B,C,D);
12 % Define initial state
13 x0 = [0,0];
14 % Define the input u(t) = 1.0sin(0.1t) using sinusoidal signal
15 tau = 2*pi/0.1; % Period = 2*pi/0.1
16 % 0:Ts:Tf
17 Ts = 0.01; % Time step
18 Tf = 4; % Duration
19 [u,t] = gensig('sin',tau,Tf,Ts);
20 % Simulate the system
21 lsim(sys,u,t,x0);
22 grid on;
23 legend(System response);
24 fig = gcf; % Obtains current graphic in matlab
25 exportgraphics(fig, 'Fig/lsim_run_4s.pdf', 'ContentType','vector');
```

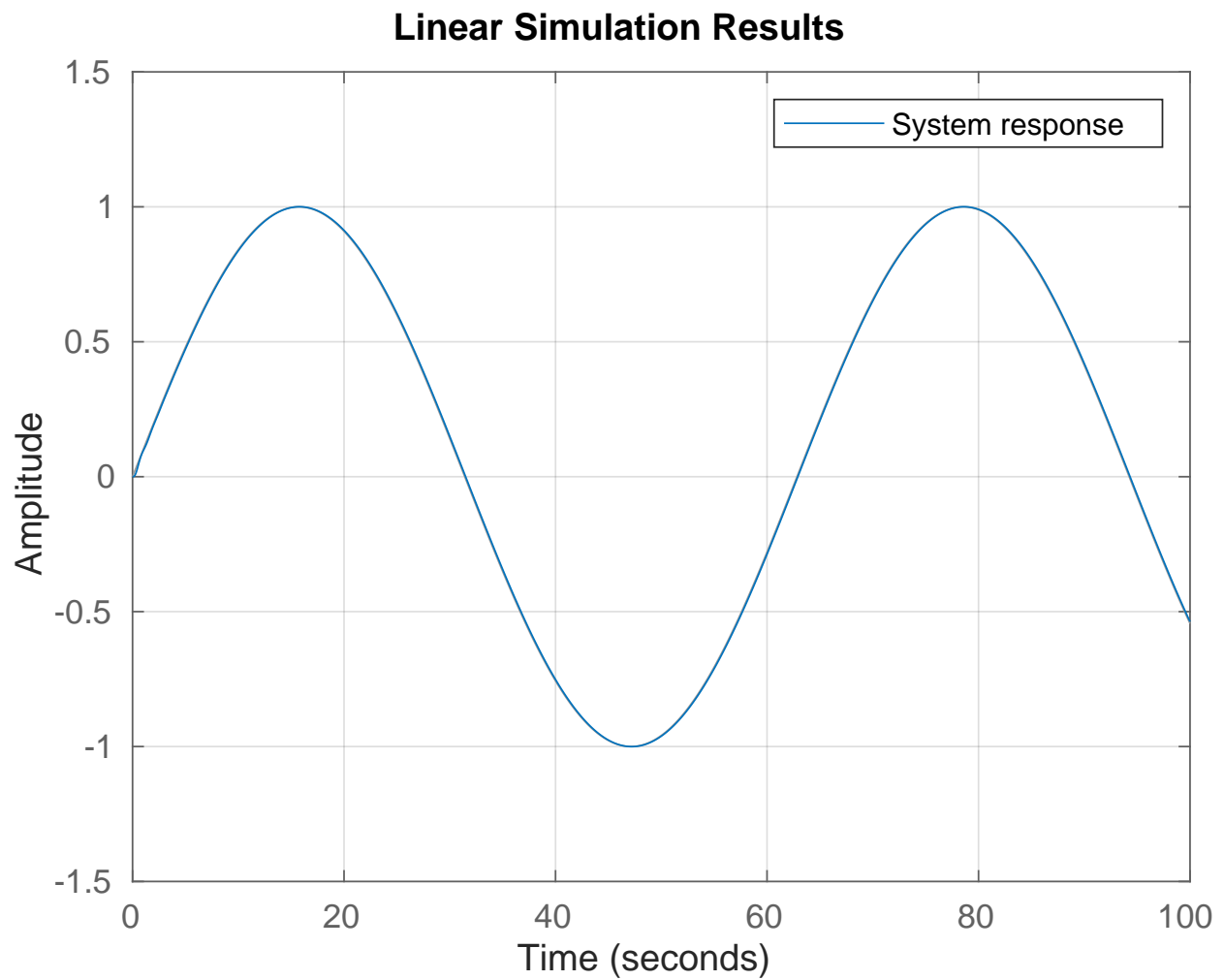


Figure 1: lsim Simulation of the State Space Model with a domain of 0-100 seconds

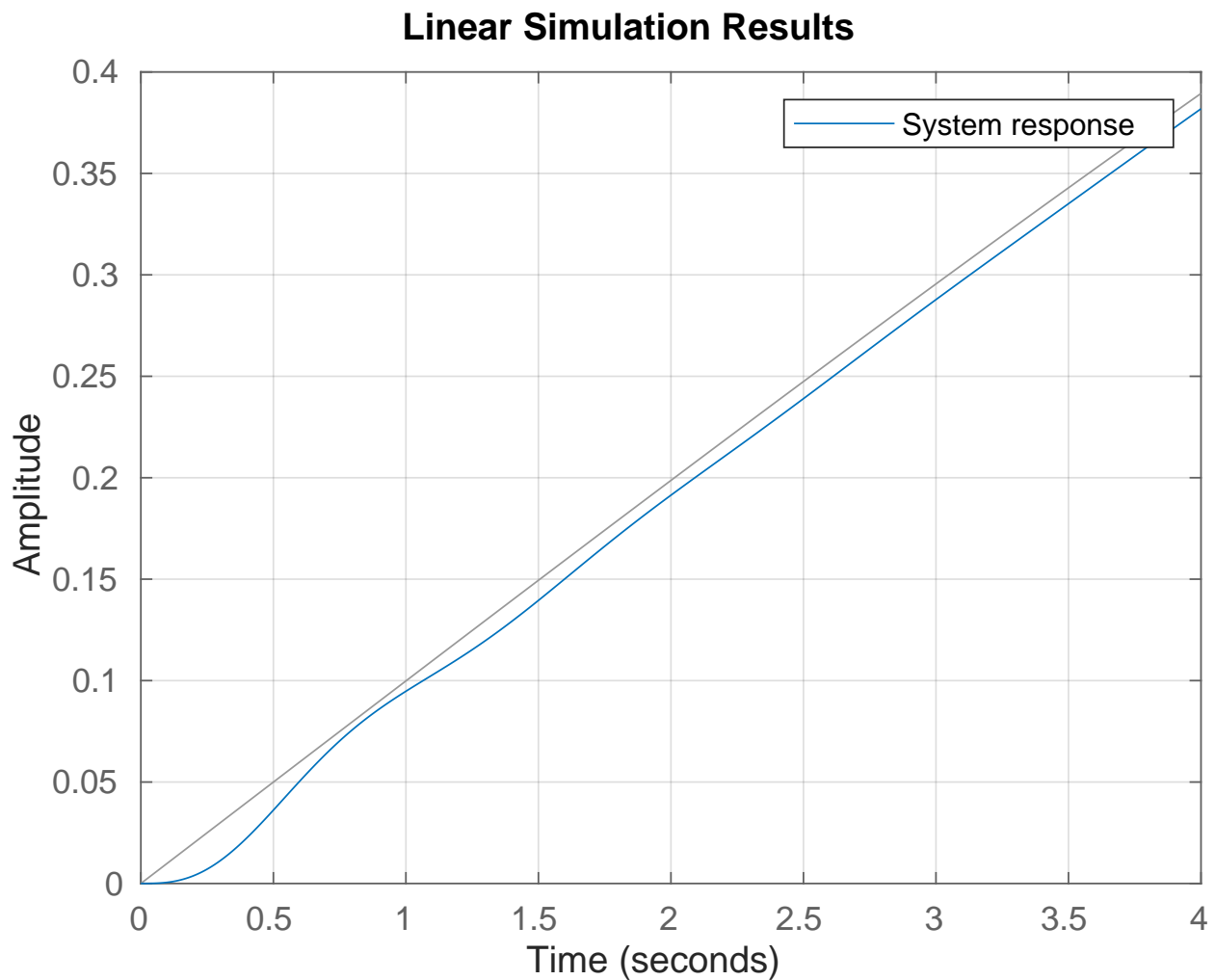


Figure 2: lsim Simulation of the State Space Model with a domain of 0-4 seconds

4 Varying Step-Size of $1.0 \sin(0.1)$

Listing 3: Function for the State Space of the representation of the system

```

1 function [dx] = f(x,u)
2 % x — [2xn] column vector
3 % u — [1xn] vector
4 A = [0,1; -36,-3];
5 B = [0;36];

```

```

6 dx = A*x + B*u;
7 end

```

Listing 4: Function to run Euler recursion

```

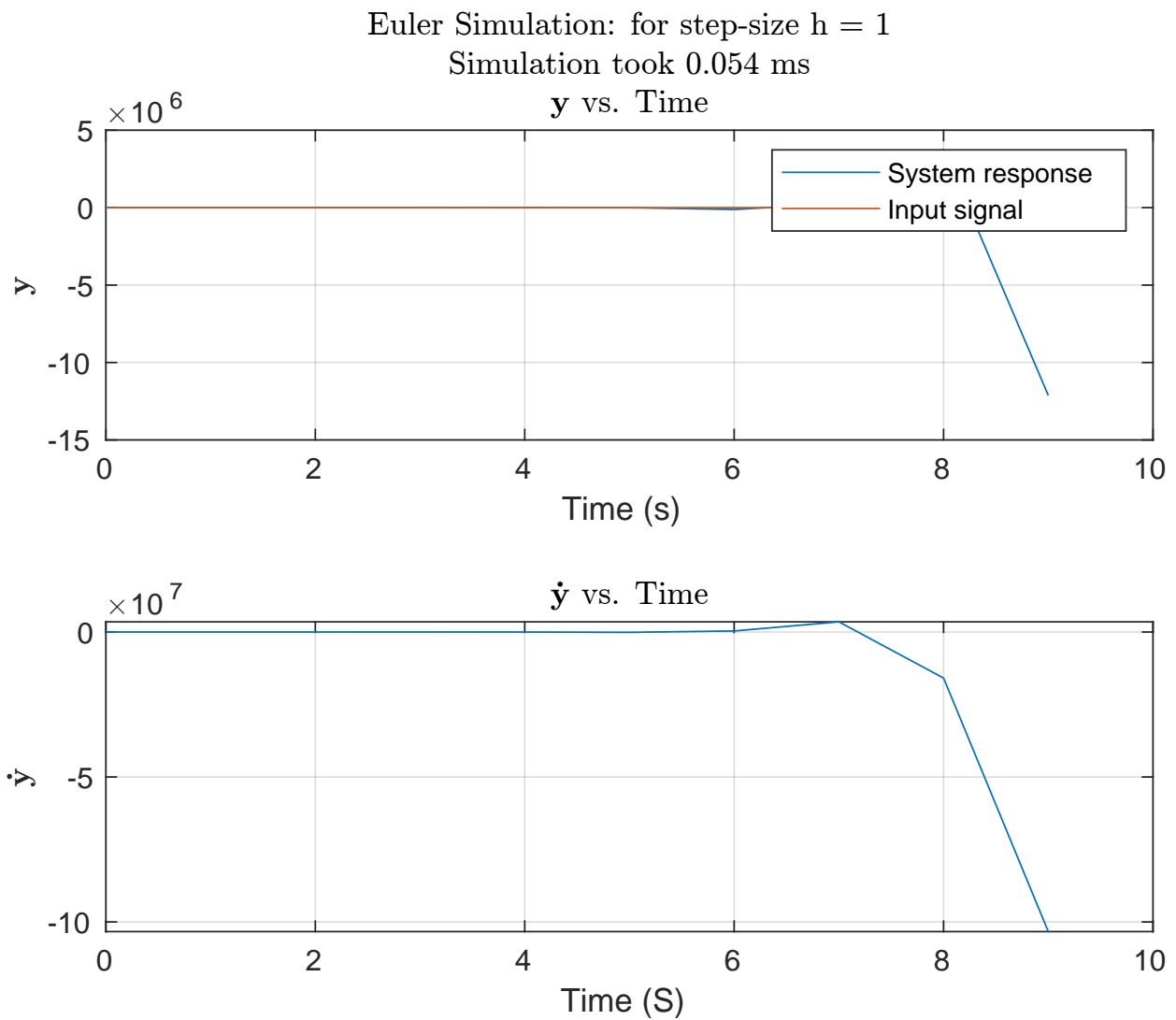
1 function sim_t = Euler(t,x0,h,u, filename, filename_csv)
2 % For the given initial condition x0 and step size
3 % h this function uses Euler integration to
4 % numerically solve the differential equation
5 % of the transfer function.
6 % Function output: sim_t, Euler Simulation time cost
7 tic; % start the clock
8 N = length(u); % The iteration steps based on the length of input signal
9 % Initialize x
10 x = zeros(length(x0),N); %The dimension of x in terms of dimension of x0
11 x(:,1) = x0; % IC
12 for i=1:N
13     [dx] = f(x(:,i),u(i));
14     x(:,i+1) = x(:,i) + dx*h;
15 end
16 sim_t = toc; % end the clock
17 sim_t = sim_t*1000;
18 display(append('Euler Simulation Took = ', string(sim_t) , 's'));
19 figure
20 subplot(2, 1, 1);
21 plot(t,x(1,1:i));
22 hold on;
23 plot(t,u);
24 legend('System response', 'Input signal');
25 plot_title = {[append('Euler Simulation: for step-size h = ', string(h))] [
    append('Simulation took ', string(sim_t), ' ms')] ['$\bf y$ vs. Time']};
26 grid on;
27 title(plot_title, 'Interpreter', 'latex');
28 xlabel('Time (s)');
29 ylabel('$ \bf y$', 'Interpreter', 'latex');
30 subplot(2, 1, 2);
31 plot (t,x(2,1:i));
32 title ('y prime vs. Time', 'Interpreter', 'latex')
33 xlabel('Time (S)');
34 ylabel('y prime', 'Interpreter', 'latex');
35 grid on;

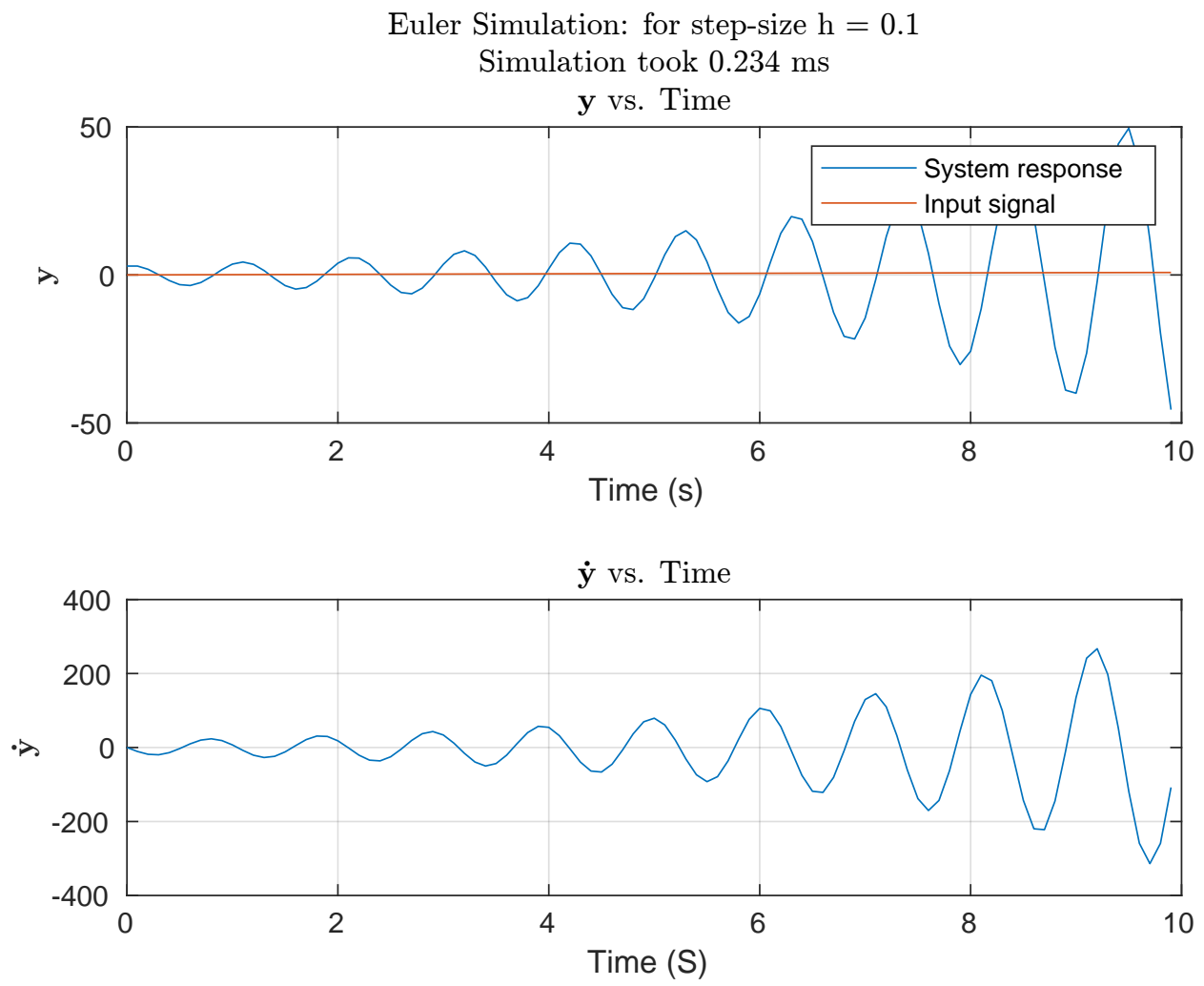
```

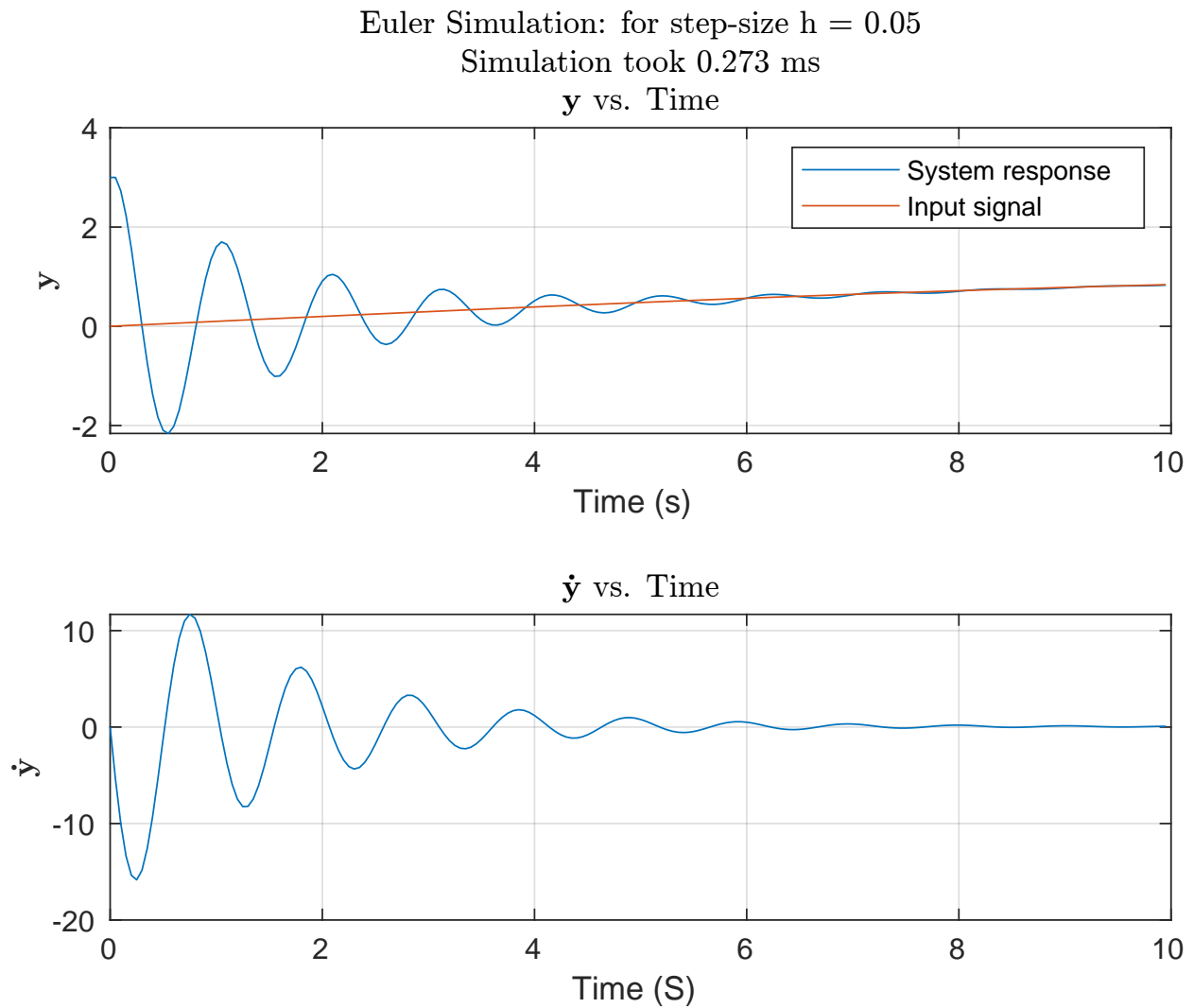
```
36 writematrix(x(1,1:i),filename_csv);
37 fig = gcf; % Obtains current graphic in matlab
38 exportgraphics(fig, filename, 'ContentType', 'vector');
39 end
```

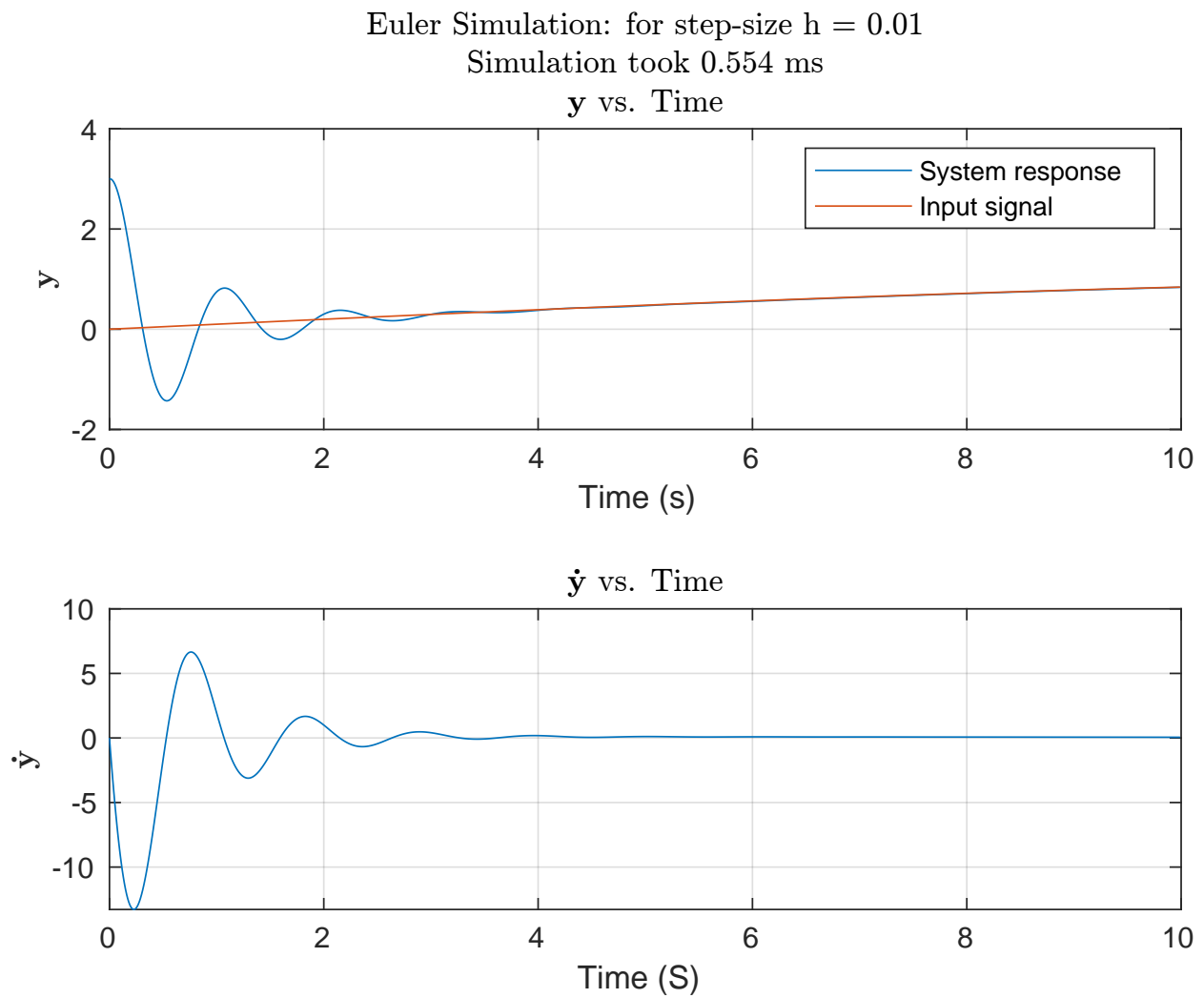
Listing 5: Run the Euler function for different step sizes h

```
1 % Define the input  $u(t) = 1.0\sin(0.1t)$  using sinusoidal signal
2 tau = 2*pi/0.1; % Period = 2*pi/0.1
3 % 0:Ts:Tf
4 Ts = 1; % Time step
5 Tf = 10; % Duration
6 % Define initial state
7 x0 = [3,0];
8 % Define step-size
9 for h = [1, 0.1, 0.05, 0.01, 0.001]
10     % [u,t] = gensig('sin',tau,Tf,h);
11     N = (Tf/h);
12     t = h*(0:N-1);
13     u = sin(0.1*t);
14     filename = append('Fig/Euler_plot_h_', string(h), '.pdf');
15     filename_csv = append('Euler_plot_h_', string(h), '.csv');
16     sim_t = Euler(t,x0,h,u, filename, filename_csv);
17 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 $h = 0.01$

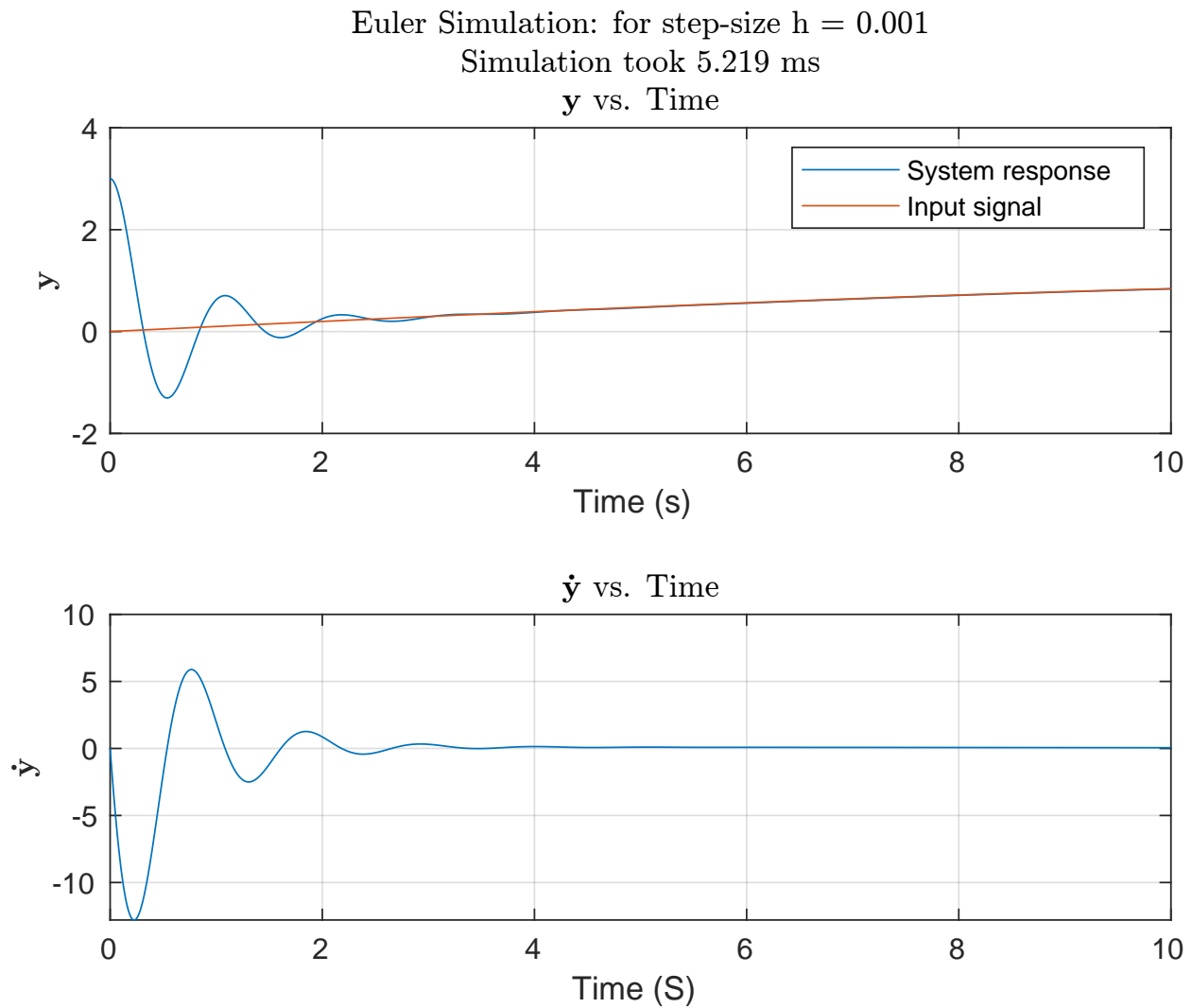


Figure 7: Euler Plot for $h = 0.001$

Figures 3, 4 do not settle at all. Figure 5 does settle, but does so at around 7-8 seconds. 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

5 ODE23 with Zero Input

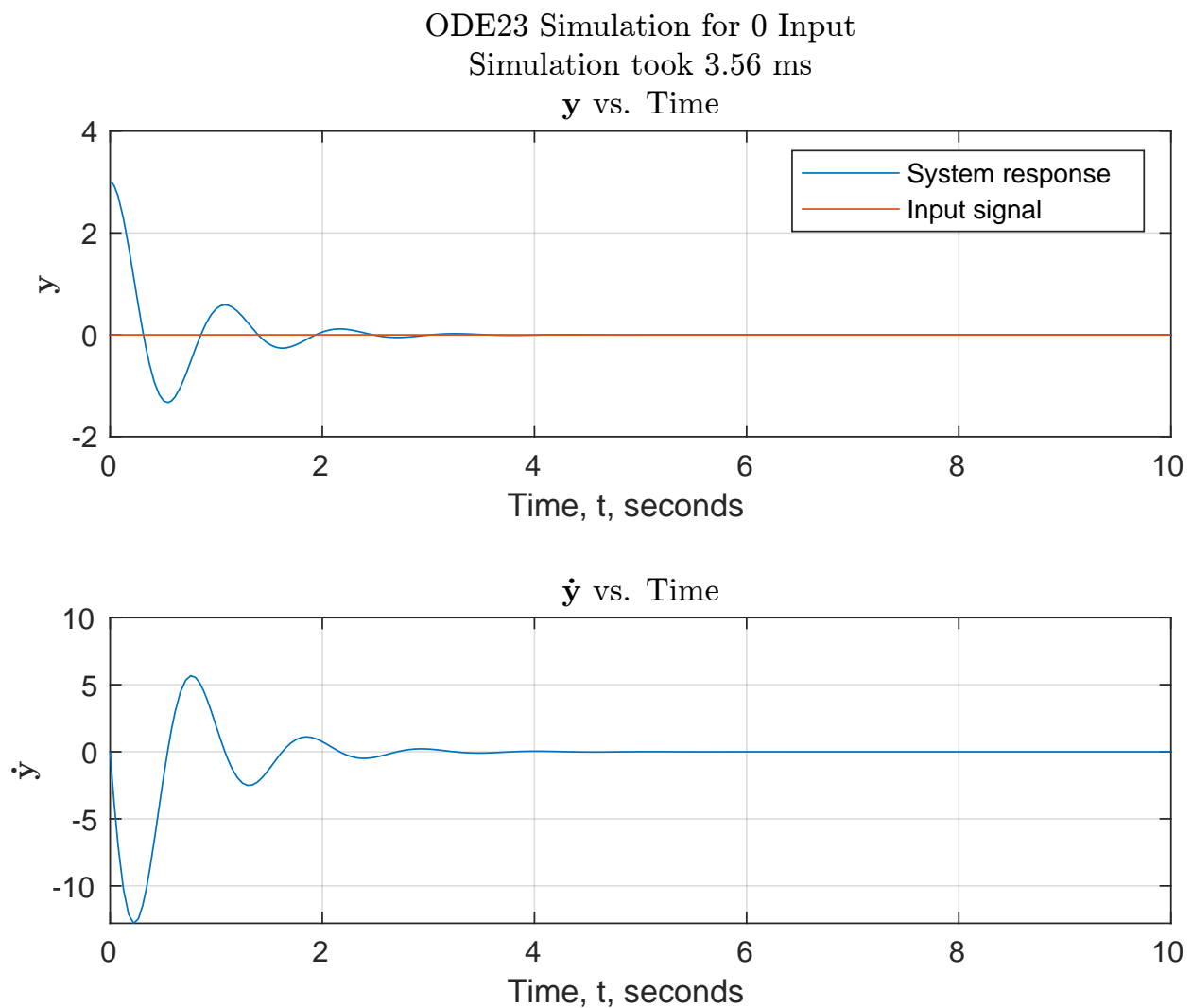


Figure 8: ODE Plot for Zero Input

6 Varying ω in $1.0 \sin(\omega t)$

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

```
1 function [dx, u] = f(t,x,omega)
```

```

2 % x — [2xn] column vector
3 % u — [1xn] vector
4 A = [0,1; -36,-3];
5 B = [0;36];
6 u = sin(omega*t);
7 dx = A*x + B*u;
8 end

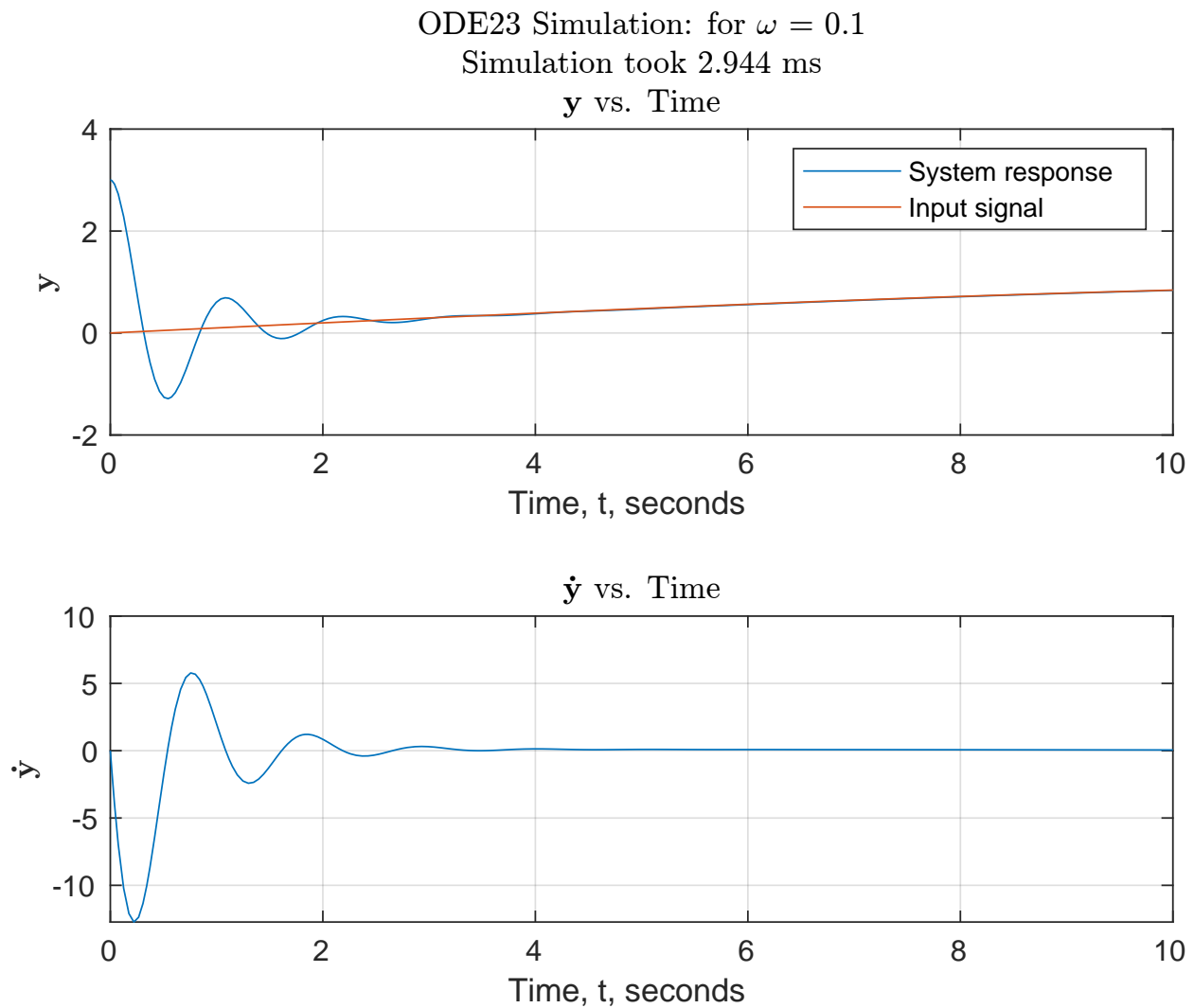
```

Listing 7: Run ODE23 for sin(0.1t)

```

1 tspan = [0 10]; % Interval of integration
2 x0 = [3 0]; % Initial condition
3 omega = 0.1;
4 tic
5 [t_out,y] = ode23(@(t,x) f_ode(t,x,omega), tspan, x0);
6 tm = toc;
7 tm = tm*1000;
8 h = length(t_out) \ (tspabn(2) - tspan(1));
9 t = tspan(1):h:tspan(2);
10 u = sin(omega*t);
11 display(append('ODE23 Simulation Took ', string(tm), 's'));
12 figure;
13 subplot(2, 1, 1);
14 plot(t_out, y(:,1));
15 hold on;
16 plot(t,u);
17 legend('System response', 'Input signal');
18 plot_title = {[append('ODE23 Simulation: for omega = ', string(omega))] [
    append('Simulation took ', string(sim_t), 's')] ['$ \bf y$ vs. Time']};
19 title(plot_title, 'Interpreter', 'latex');
20 xlabel('Time, t, seconds');
21 ylabel('$ \bf y$', 'Interpreter', 'latex');
22 grid on;
23 subplot(2, 1, 2);
24 plot(t_out, y(:,2));
25 title('y prime vs. Time', 'Interpreter', 'latex');
26 xlabel('Time, t, seconds');
27 ylabel('y prime', 'Interpreter', 'latex');
28 grid on;
29 fig = gcf; % Obtains current graphic in matlab
30 exportgraphics(fig, 'Fig/ode_sin_input_01.pdf','ContentType','vector');

```

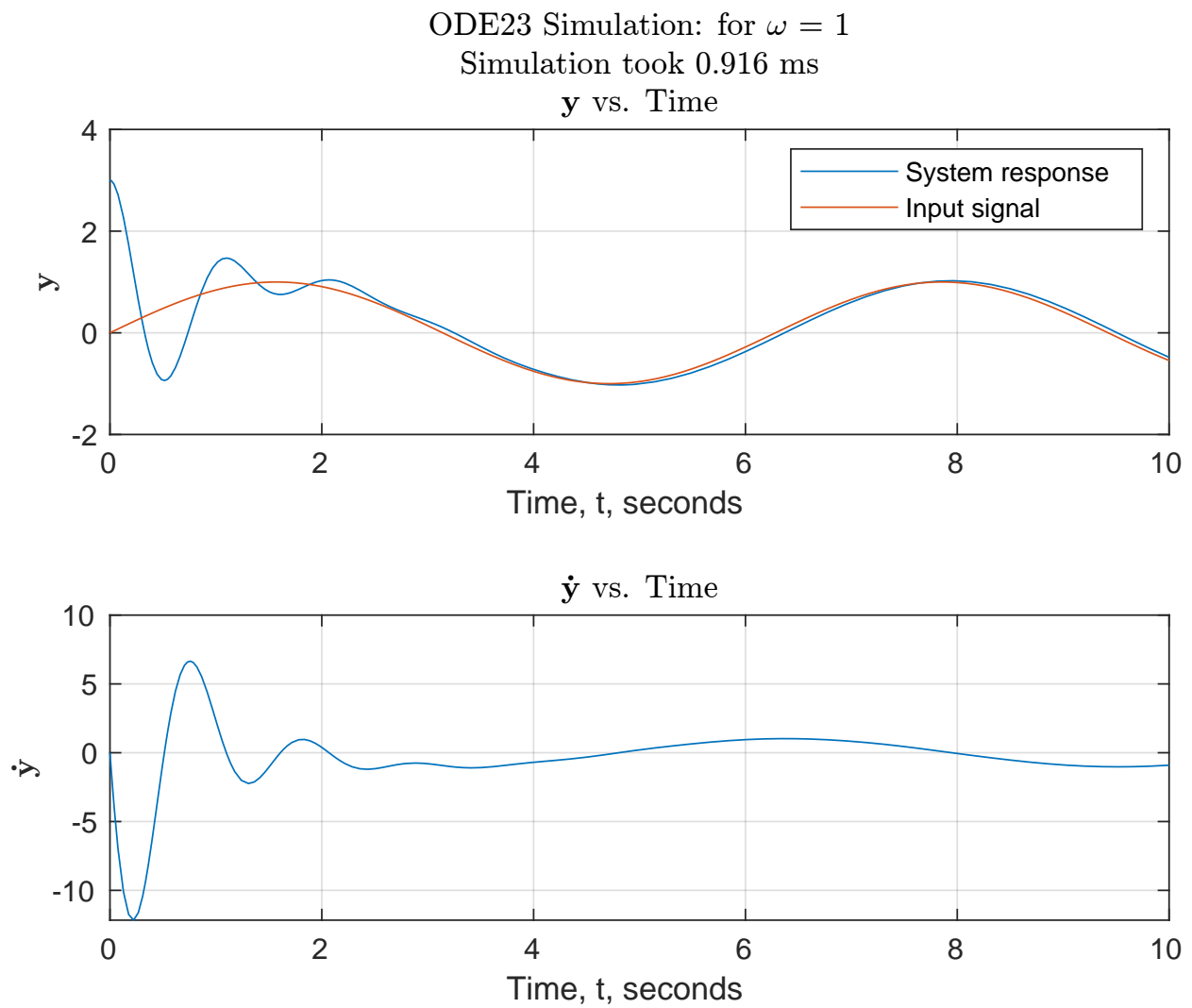
**Figure 9:** Ode Plot for $\omega = 0.1$ **Listing 8:** Run ODE23 for various $\sin(\omega t)$

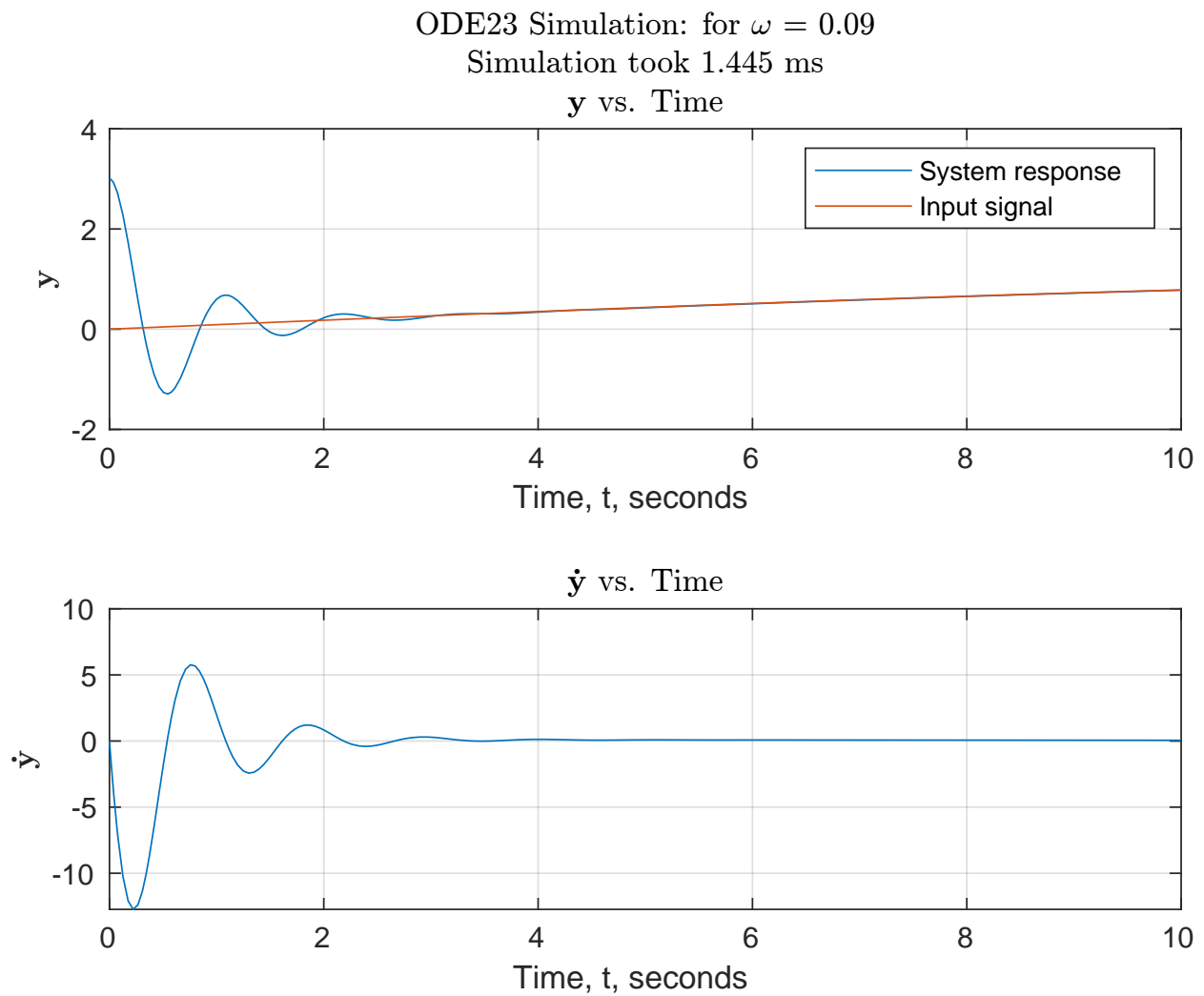
```

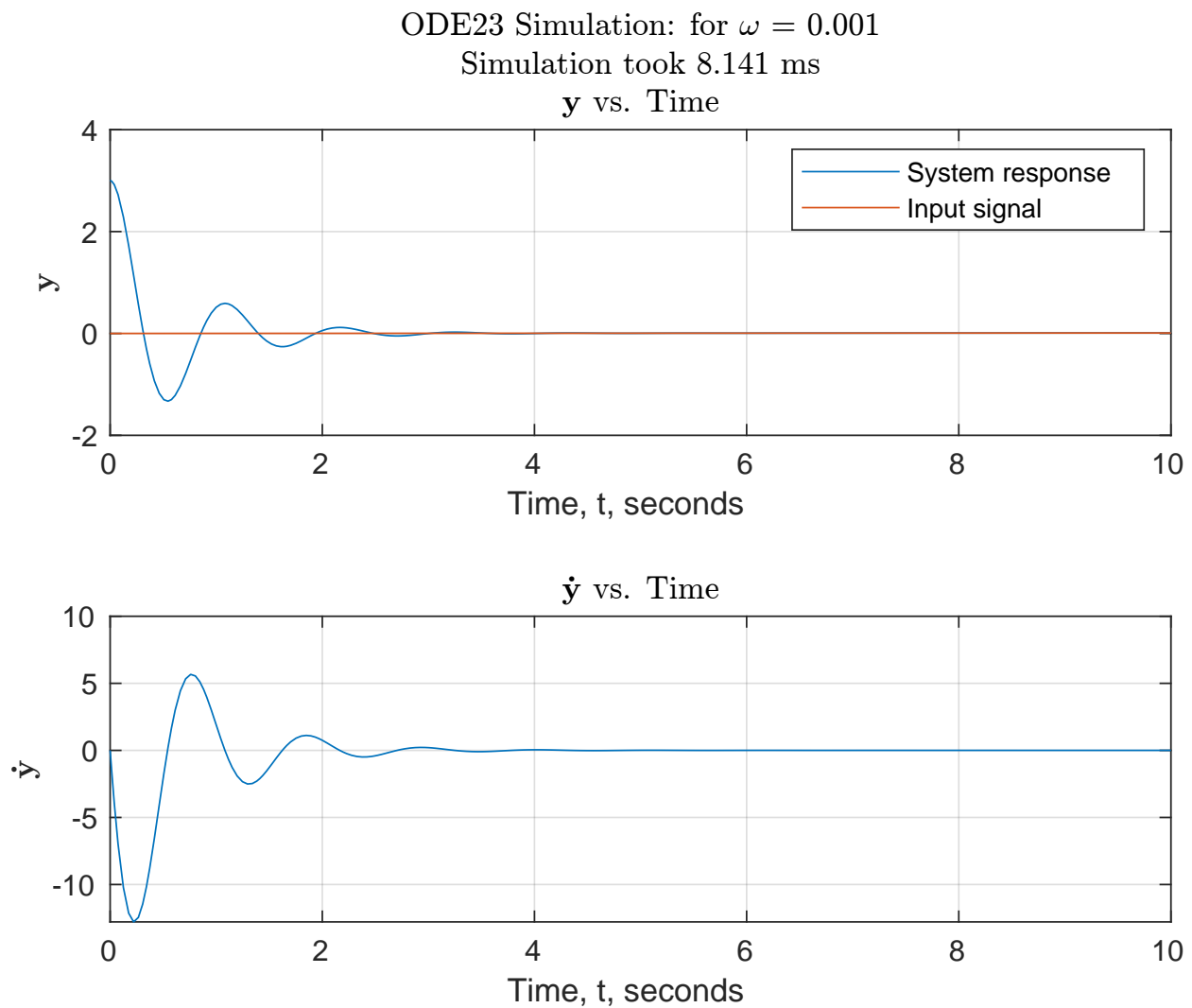
1 for i = [0.001, 0.09, 1]
2     tspan = [0 10]; % Interval of integration
3     x0 = [3 0]; % Initial condition
4     omega = i;
5     tic
6     [t_out,y] = ode23(@(t,x) f_ode(t,x,omega), tspan, x0);

```

```
7   tm = toc;
8   tm = tm*1000;
9   h = length(t_out) \ (tspan(2) - tspan(1));
10  t = tspan(1):h:tspan(2);
11  u = sin(omega*t);
12  display(append('ODE23 Simulation Took ', string(tm), 's'));
13  figure;
14  subplot(2, 1, 1);
15  plot(t_out, y(:,1));
16  hold on;
17  plot(t,u);
18  legend('System response', 'Input signal');
19  plot_title = {[append('ODE23 Simulation: for omega = ', string(omega))]
    [append('Simulation took ', string(sim_t), ' s')] ['$ \bf y$ vs. Time
    ']};
20  title(plot_title, 'Interpreter', 'latex');
21  xlabel('Time, t, seconds');
22  ylabel('$ \bf y$', 'Interpreter', 'latex');
23  grid on;
24  subplot(2, 1, 2)
25  plot(t_out, y(:,2))
26  title ('y prime vs. Time', 'Interpreter', 'latex');
27  xlabel('Time, t, seconds');
28  ylabel('y prime', 'Interpreter', 'latex');
29  grid on;
30  filename = append('Fig/ode_sin_input_', string(omega), '.pdf');
31  fig = gcf; % Obtains current graphic in matlab
32  exportgraphics(fig, filename, 'ContentType', 'vector');
33  end
```

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

**Figure 11:** Ode Plot for $\omega = 0.09$

Figure 12: Ode Plot for $\omega = 0.001$

6.1 Bode Plot

Listing 9: Bode Plot Code for State Space Function

```

1 % Define the state space function
2 H = tf(36,[1 3 36]);
3 opts = bodeoptions('cstprefs');
4 opts.MagUnits = 'abs';
5 opts.PhaseUnits = 'rad';
6 figure;
```



```
7 bode(H, opts);
8 fig = gcf; % Obtains current graphic in matlab
9 exportgraphics(fig, 'Fig/bode_plot.pdf', 'ContentType','vector');% Define A,
    B,C,D
10
11 A = [0,1; -36,-3];
12 B = [0;36];
13 C = [1 0];
14 D = 0;
15
16 % Define state-space model
17 sys = ss(A,B,C,D);
18
19 omega = [1, 0.09, 0.001];
20 [mag,phase] = bode(sys, omega);
21 phase = deg2rad(phase);
22 fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(1)
    , mag(1), phase(1));
23 fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(2)
    , mag(2), phase(2));
24 fprintf('At omega = %g, the amplitude is %g and the phase is %g\n', omega(3)
    , mag(3), phase(3));
```

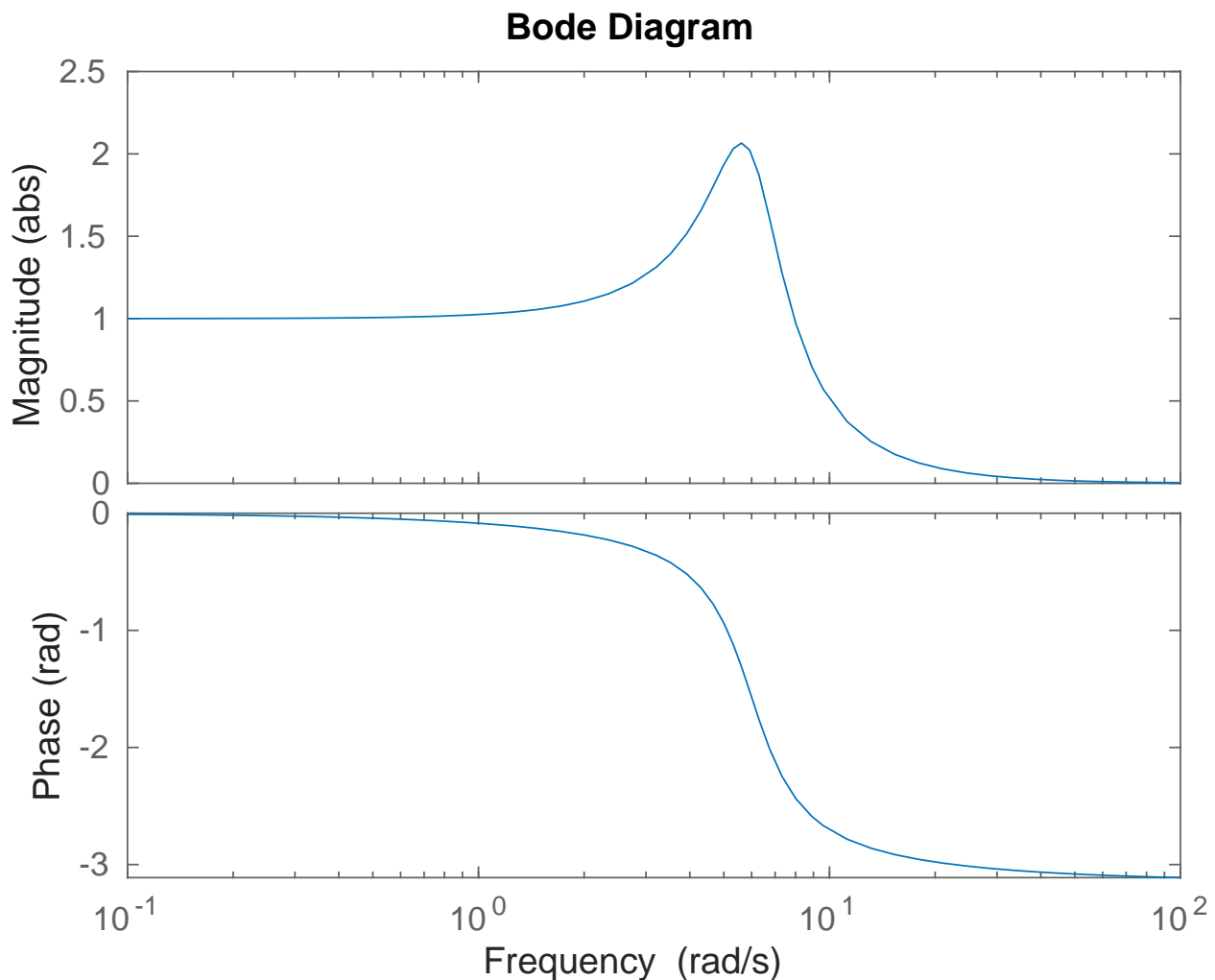


Figure 13: Bode Plot for State Space Model

- At $\omega = 1$, the amplitude is 1.02481 and the phase is -0.0855053
- At $\omega = 0.09$, the amplitude is 1.0002 and the phase is -0.00750155
- At $\omega = 0.001$, the amplitude is 1 and the phase is -8.33333e-05

7 Conclusion

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

- Students solidified our understanding of how numerical methods approximate the solution of differential equations
- Students explored different MATLAB tools and techniques for implementing these methods, such as ode23 and custom functions
- Students learned to critically evaluate the accuracy and stability of numerical solutions, considering factors like step size and tolerance
- Students became aware of potential pitfalls like stiffness and convergence issues, highlighting the importance of choosing appropriate methods and parameter values