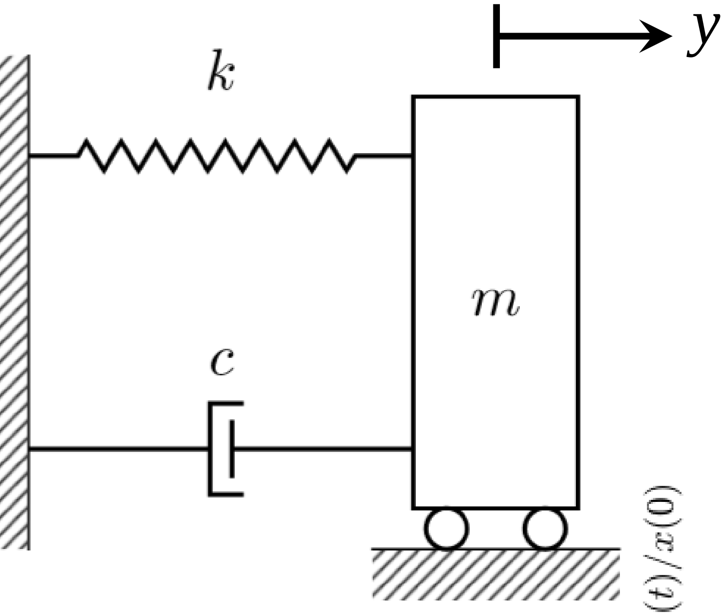
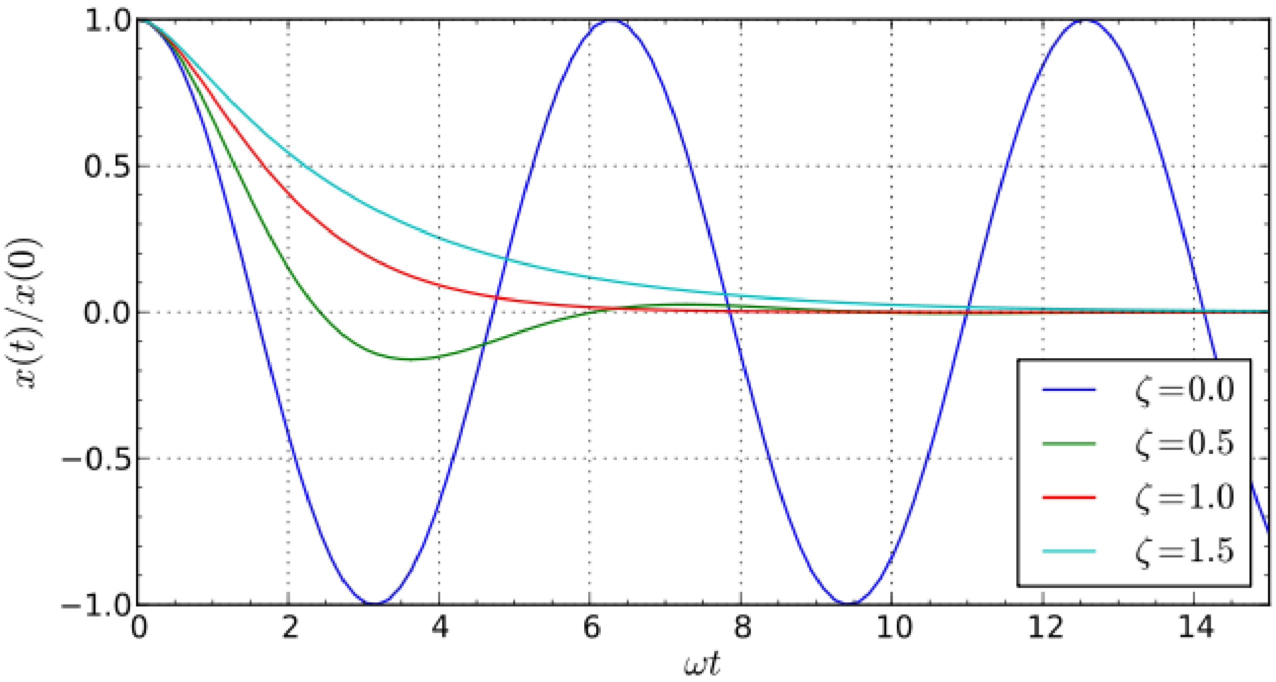
Chapter 9c: Numerical Methods for Calculus and Differential Equations

- Higher-Order Differential Equations
  - Cauchy/State-Variable Form
  - Euler Method
  - MATLAB ODE Solver ode45
  - ode45 with Matrix Method
- Matrix Methods for Linear Equations
- Control System Toolbox

Higher-Order Differential Equations

The methods used to solve first-order differential equations can be used to solve higher-order ordinary differential equations. Consider a **Spring-Mass-Damper** system:





Higher-Order Differential Equations

The mass is *m*, the spring constant is *k*, and the damping coefficient is *c*. Newton's Second Law for this system is:

$$m\ddot{y} + c\dot{y} + ky = 0$$

where the first derivative of position with respect to time is $\dot{y} = \frac{dy}{dt}$ and the second derivative is $\ddot{y} = \frac{d^2y}{dt^2}$

Solve this equation by turning it into a system of two first-order differential equations. First, solve the equation for the second derivative:

$$\ddot{y} = -\frac{c}{m}\dot{y} - \frac{k}{m}y$$

Cauchy/State-Variable Form

Let $x_1 = y$ (Position) and $x_2 = \dot{y}$ (Velocity). Taking the derivative of the first equation gives:

$$\dot{x}_1 = \dot{y} = x_2 \quad \text{or} \quad \dot{x}_1 = x_2$$

Taking the derivative of the second equation gives:

$$\dot{x}_2 = \ddot{y} = -\frac{c}{m}\dot{y} - \frac{k}{m}y \quad \text{or} \quad \dot{x}_2 = -\frac{c}{m}x_2 - \frac{k}{m}x_1$$

This is called the **Cauchy Form** or the **State-Variable Form**:

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

$$\dot{x}_2 = -\frac{c}{m}x_2 - \frac{k}{m}x_1$$

Euler Method

Now use the Euler Method to discretize the system of equations as follows:

$$x_{1,k+1} = x_{1,k} + \Delta t \cdot x_{2,k}$$

$$x_{2,k+1} = x_{2,k} + \Delta t \cdot \left(-\frac{c}{m} x_{2,k} - \frac{k}{m} x_{1,k} \right)$$

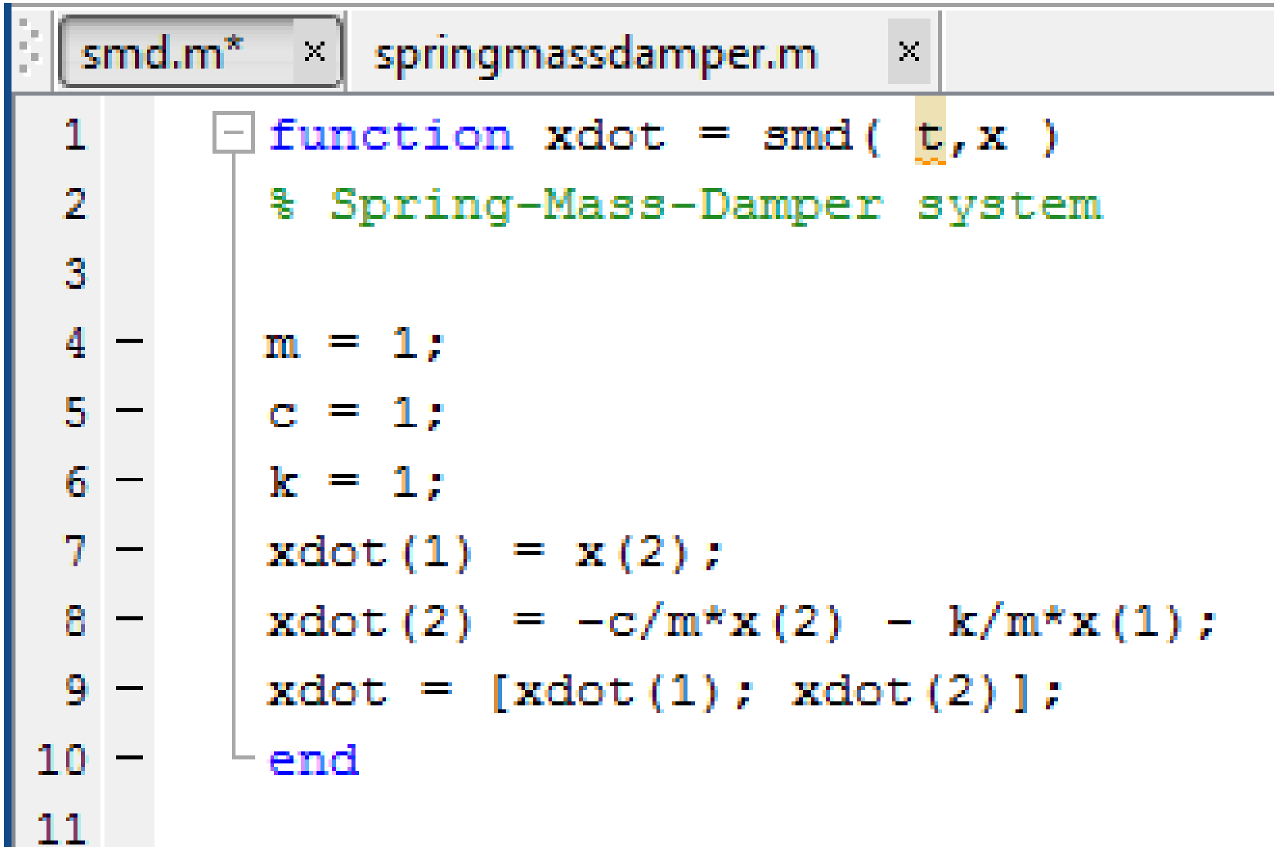
This system of equations is solved using the same **Time-Stepping** technique that was shown previously using the **Euler Method**.

MATLAB ODE Solver ode45

Alternatively, use `ode45` to solve the system:

```
[t, x] = ode45(@xdot, tspan, x0)
```

Function File:



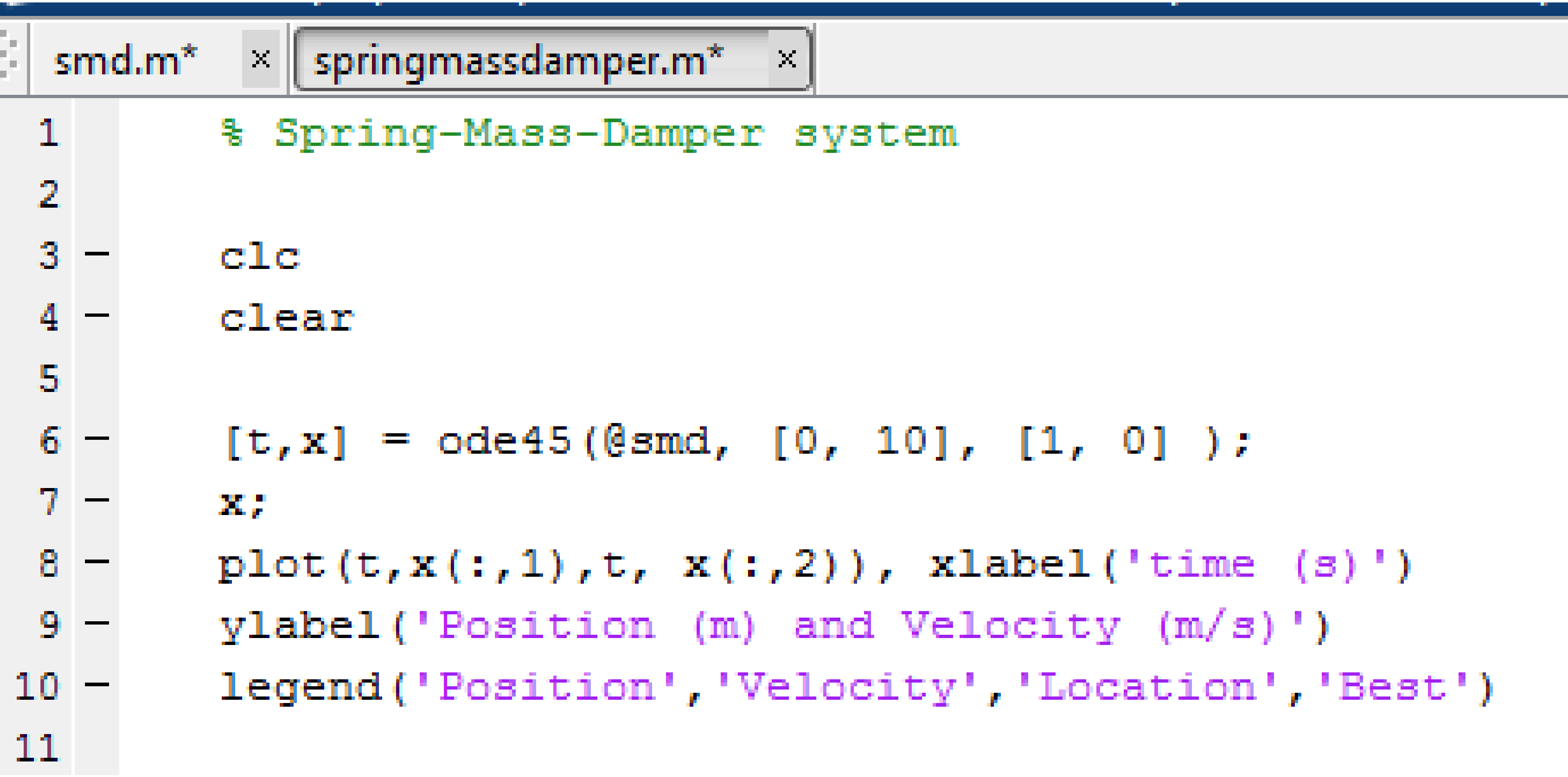
```
function xdot = smd( t,x )
% Spring-Mass-Damper system

m = 1;
c = 1;
k = 1;
xdot(1) = x(2);
xdot(2) = -c/m*x(2) - k/m*x(1);
xdot = [xdot(1); xdot(2)];
end

```

MATLAB ODE Solver ode45

Script File:



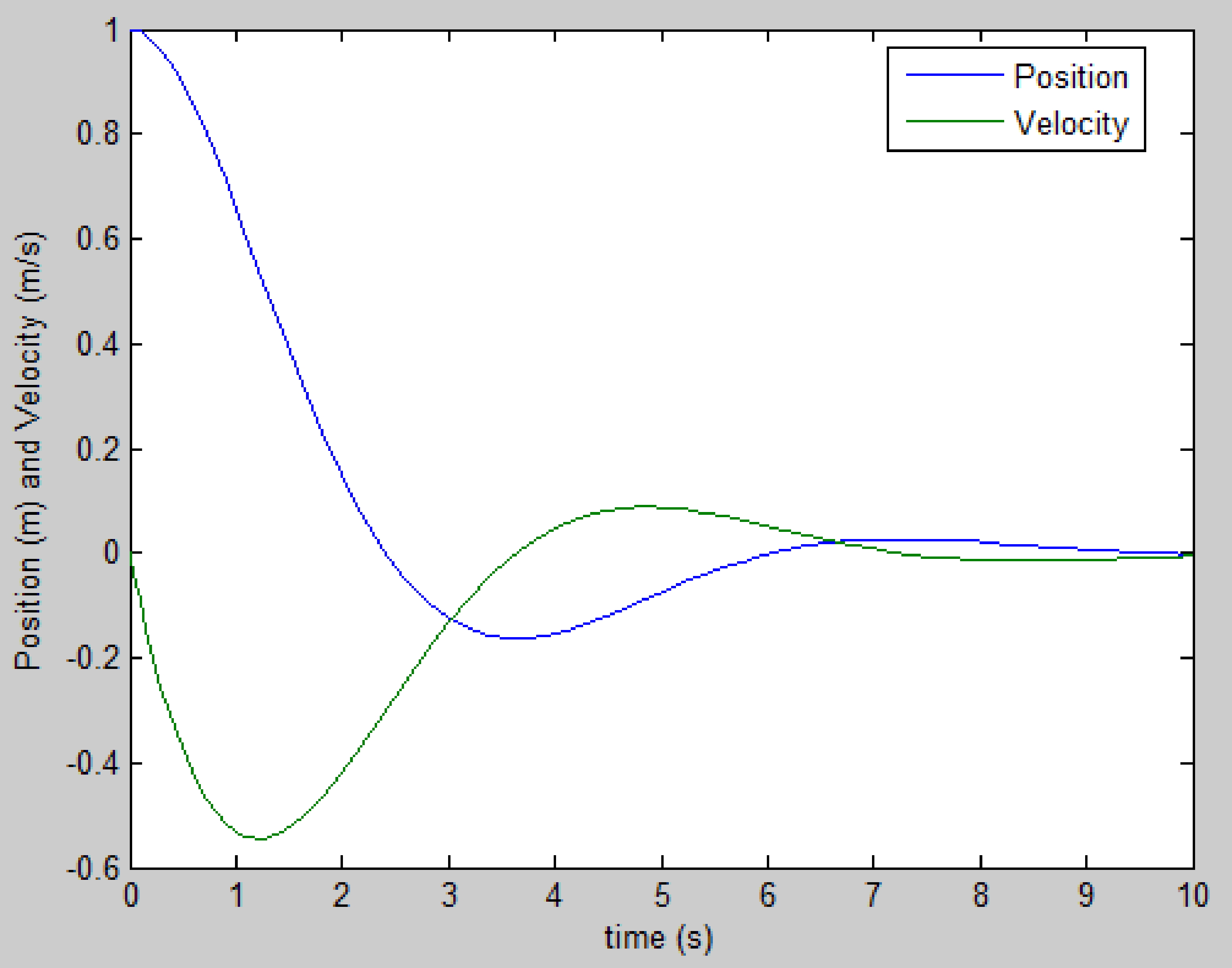
```
% Spring-Mass-Damper system

clc
clear

[t,x] = ode45(@smd, [0, 10], [1, 0] );
x;
plot(t,x(:,1),t, x(:,2)), xlabel('time (s)')
ylabel('Position (m) and Velocity (m/s)')
legend('Position','Velocity','Location','Best')

```

MATLAB ODE Solver ode45



ode45 with Matrix Method

The general **Spring-Mass-Damper** problem, where $u(t)$ is a forcing function, can be solved by casting the equation in **Matrix Form**:

$$m\ddot{y} + c\dot{y} + ky = u(t)$$

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

$$\dot{x}_2 = \frac{1}{m}u(t) - \frac{c}{m}x_2 - \frac{k}{m}x_1$$

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m}\end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m}\end{bmatrix} \cdot u(t)$$

ode45 with Matrix Method

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \cdot u(t)$$

In **Matrix Form**:

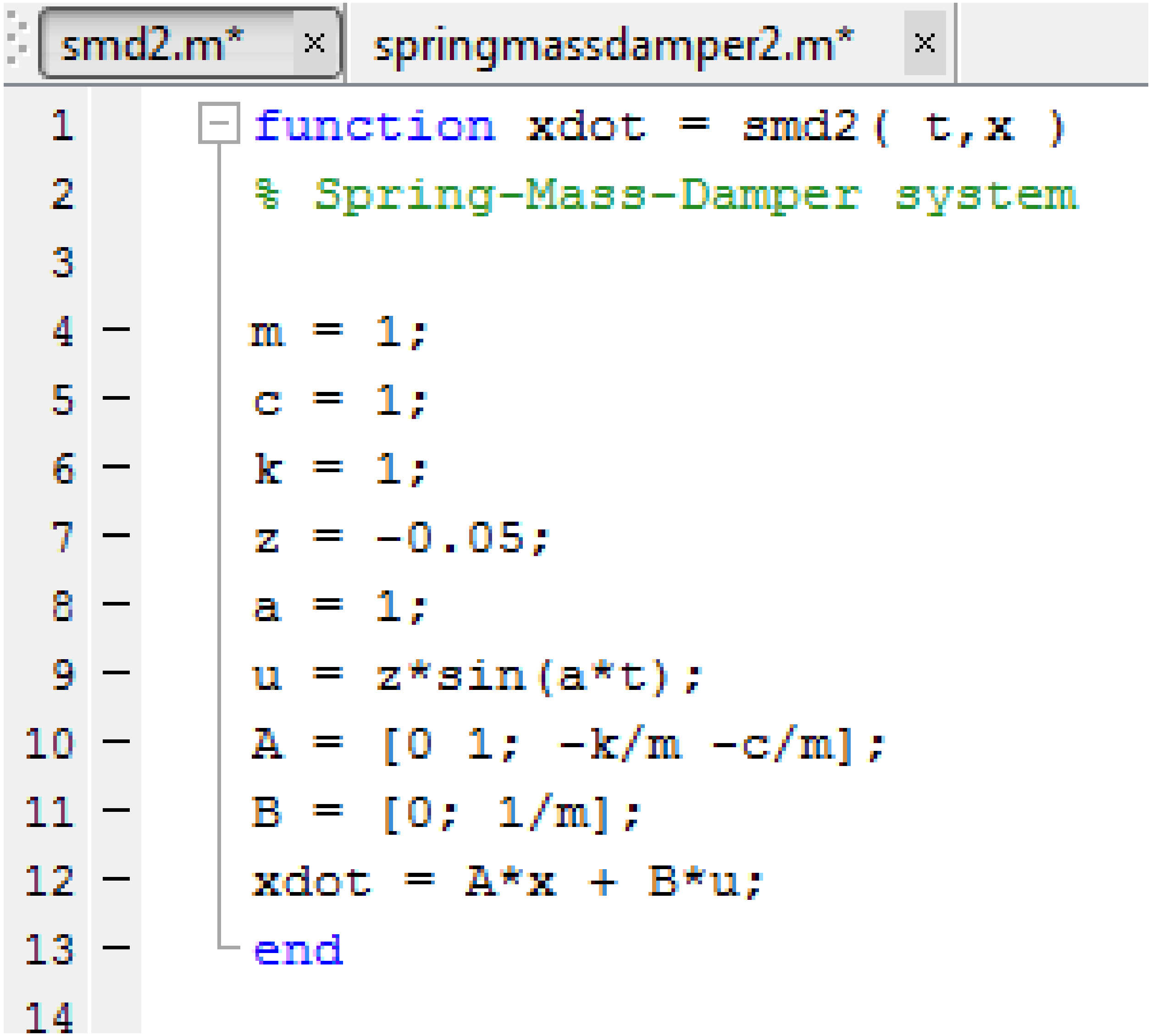
$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B} \cdot u(t)$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

ode45 with Matrix Method

Function File:

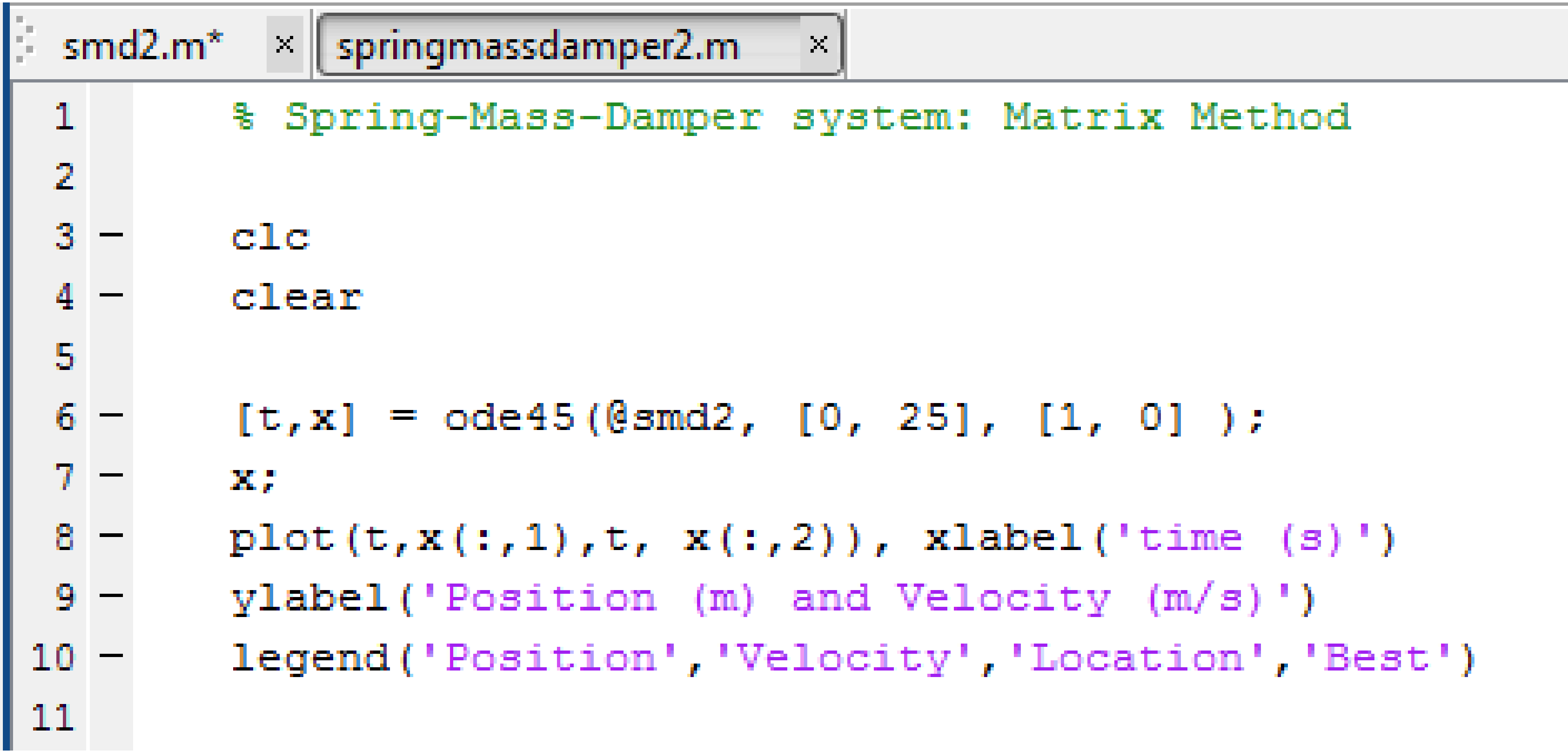


```
function xdot = smd2( t,x )
% Spring-Mass-Damper system

m = 1;
c = 1;
k = 1;
z = -0.05;
a = 1;
u = z*sin(a*t);
A = [0 1; -k/m -c/m];
B = [0; 1/m];
xdot = A*x + B*u;
end
```

ode45 with Matrix Method

Script File:



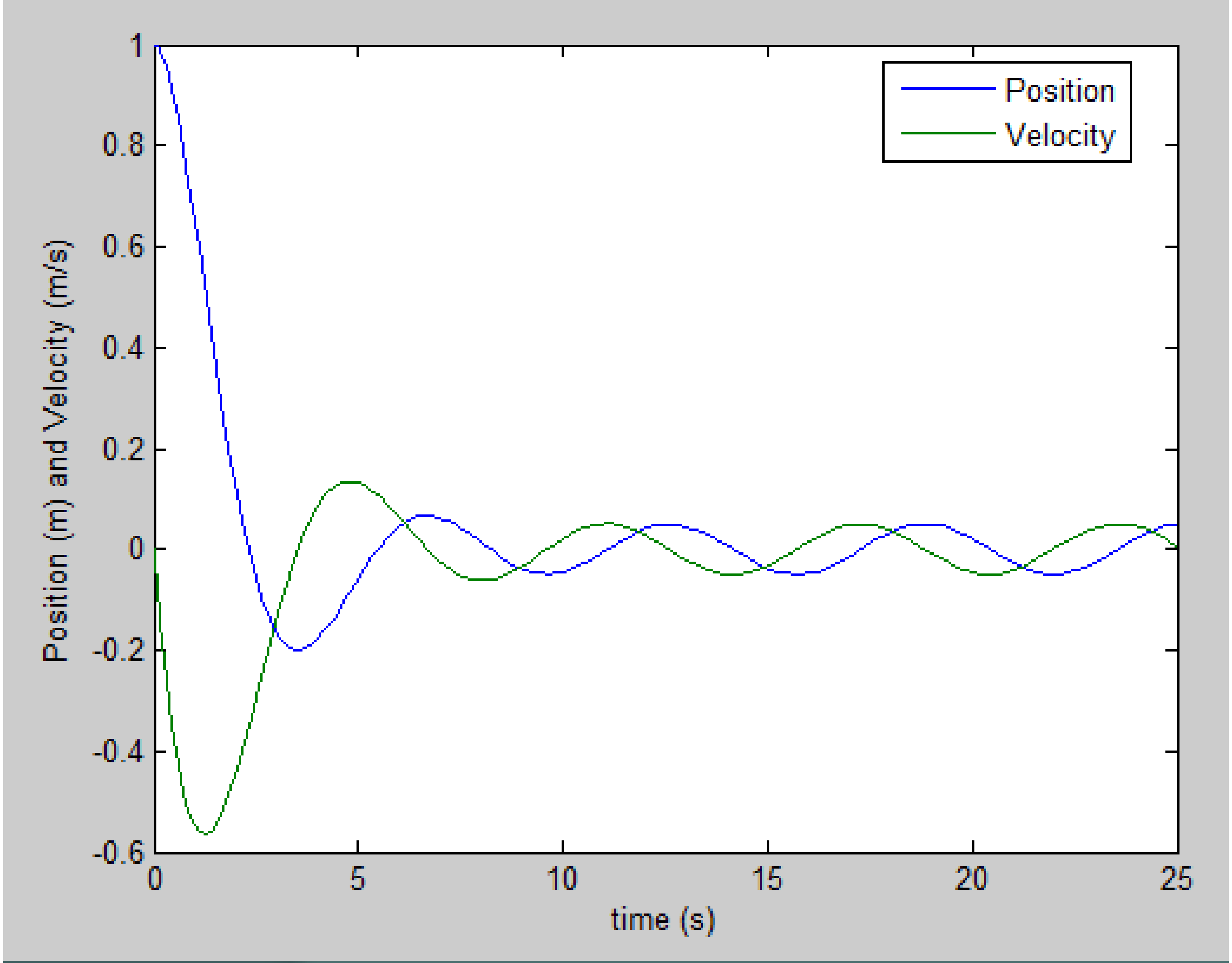
```
% Spring-Mass-Damper system: Matrix Method

clc
clear

[t,x] = ode45(@smd2, [0, 25], [1, 0] );
x;
plot(t,x(:,1),t, x(:,2)), xlabel('time (s)')
ylabel('Position (m) and Velocity (m/s)')
legend('Position','Velocity','Location','Best')

```

ode45 with Matrix Method



Matrix Methods for Linear Equations

Spring-Mass-Damper system in **Reduced Form** or **Transfer Function Form**:

$$m\ddot{y} + c\dot{y} + ky = u(t)$$

SMD in **State-Variable** or **State-Space Form**:

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

$$\dot{x}_2 = \frac{1}{m}u(t) - \frac{c}{m}x_2 - \frac{k}{m}x_1$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \cdot u(t)$$

Matrix Methods for Linear Equations

SMD in **Matrix Form**:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B} \cdot u(t)$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

All three forms describe the same second-order differential equation. When the coefficients are constant, the above representation is called a Linear, Time-Invariant equation, or an **LTI Object** or **LTI System**.

Matrix Methods for Linear Equations

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B} \cdot u(t)$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

In this case, there are only two outputs: x_1 and x_2, which represent the position and the velocity of the mass m. The outputs are given in the following matrix:

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\boldsymbol{u}(t)$$

If the position of the mass is desired, $\mathbf{C} = [1,0]$. If the velocity is desired, $\mathbf{C} = [0,1]$. In all cases, $\mathbf{D} = 0$.

Control System Toolbox

The most general case for a second-order **LTI System** in **Reduced Form** is:

$$a\frac{d^2y}{dx^2} + b\frac{dy}{dx} + cy = d\frac{du}{dx} + eu$$

This system can be input to MATLAB as follows:

```
sys = tf(right,left)
```

where `tf` stands for **Transfer Function**. The right- and left-hand coefficient vectors are:

`right = [d, e]` and `left = [a, b, c]`

Control System Toolbox

Alternatively, the **LTI System** can be input to MATLAB in **State-Space Form** directly:

```
A = [0, 1; -k/m, -c/m]
B = [0; 1/m]
C = [1, 0]  for position of mass
D = 0
sys = ss(A,B,C,D)
```

	Function	**Required Form**	**Initial Conditions**
`initial(sys,x0)`	Free Response (Undriven)	State	Default Zero or Input
`impulse(sys)`	Impulse Response	Transfer or State	Zero
`step(sys)`	Unit-Step	Transfer or State	Zero
`lsim(sys,u,t,x0)`	Arbitrary Input Response	Transfer or State	Default Zero or Input

Initial-Condition Response

`initial(sys,x0)` gives the **Undriven Response** of the system of equations, where $u(t) = 0$, subject to a set of initial conditions:

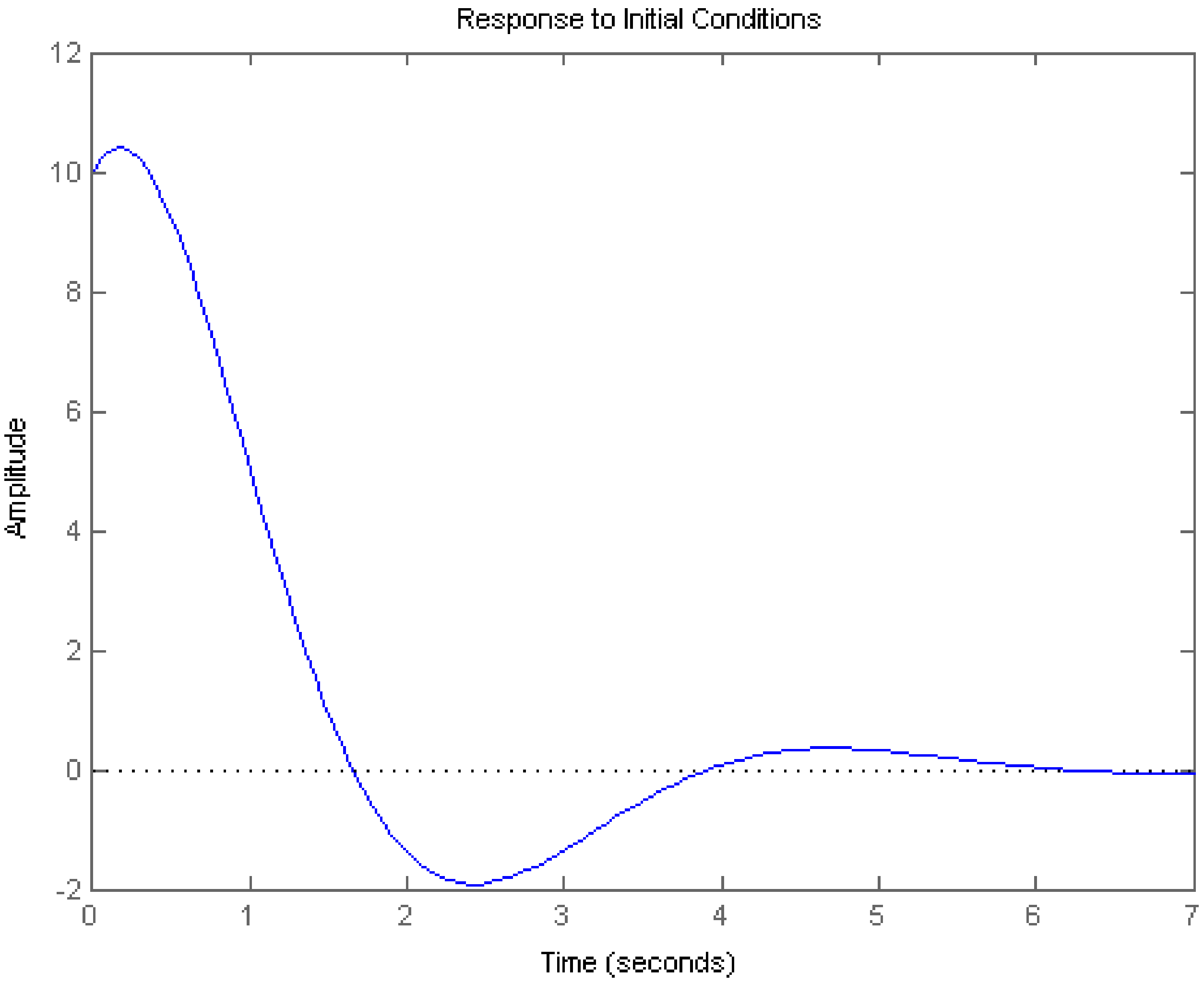
$$2\ddot{y} + 3\dot{y} + 5y = u(t), \qquad y(0) = 10, \qquad \dot{y}(0) = 5$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{5}{2} & -\frac{3}{2} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix}$$

The system must be cast into **State-Variable** or **State-Space** (ss) form.

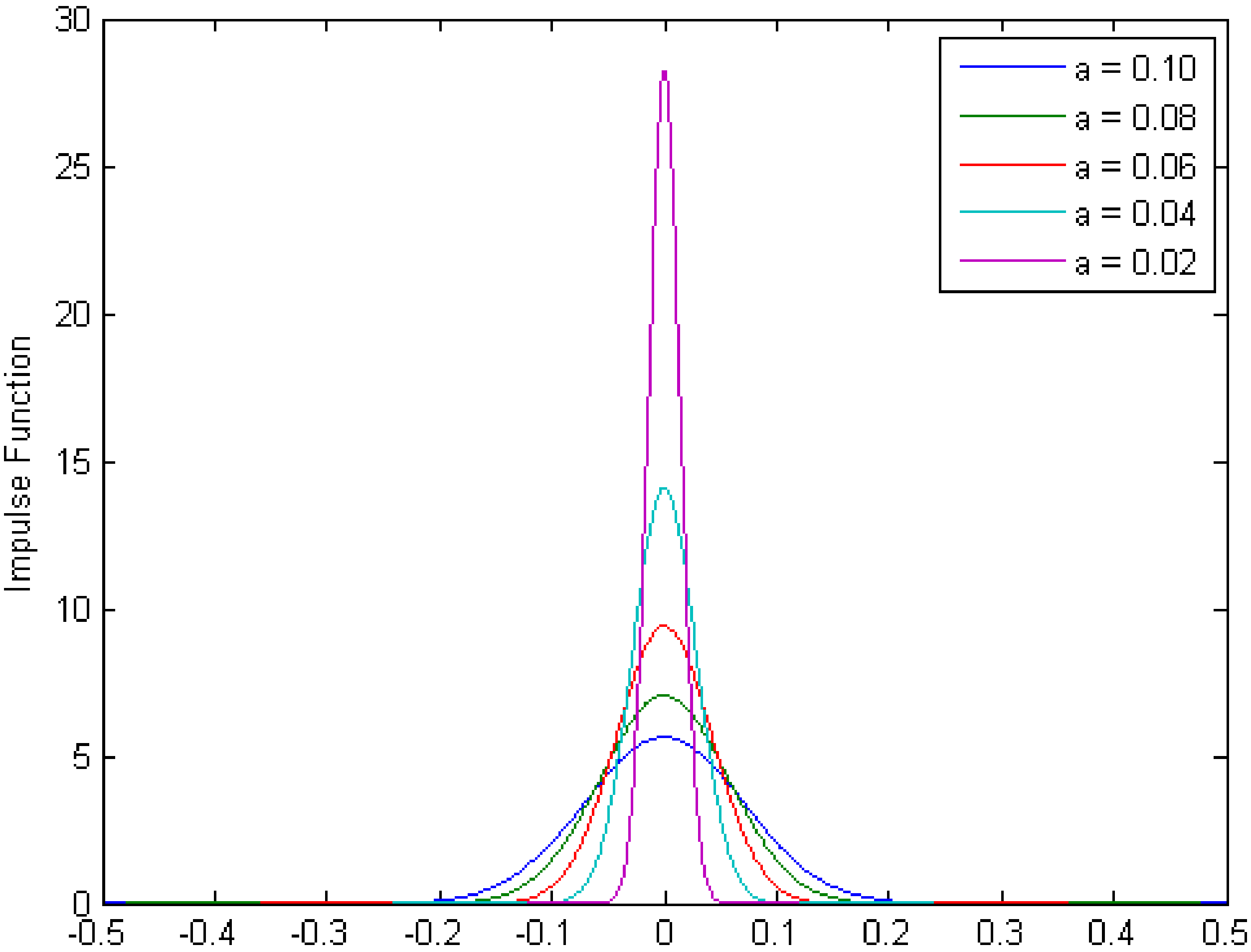
Initial-Condition Response

```
A = [0 1; -5/2 -3/2];
B = [0; 1/2];
C = [1 0];
D = 0;
sys_ss = ss(A,B,C,D);
x0 = [10 5];
initial(sys_ss, x0)
```



Impulse Response

`impulse(sys)` gives the response of the system of equations to an **Impulse Function**, where the initial conditions are set to zero.

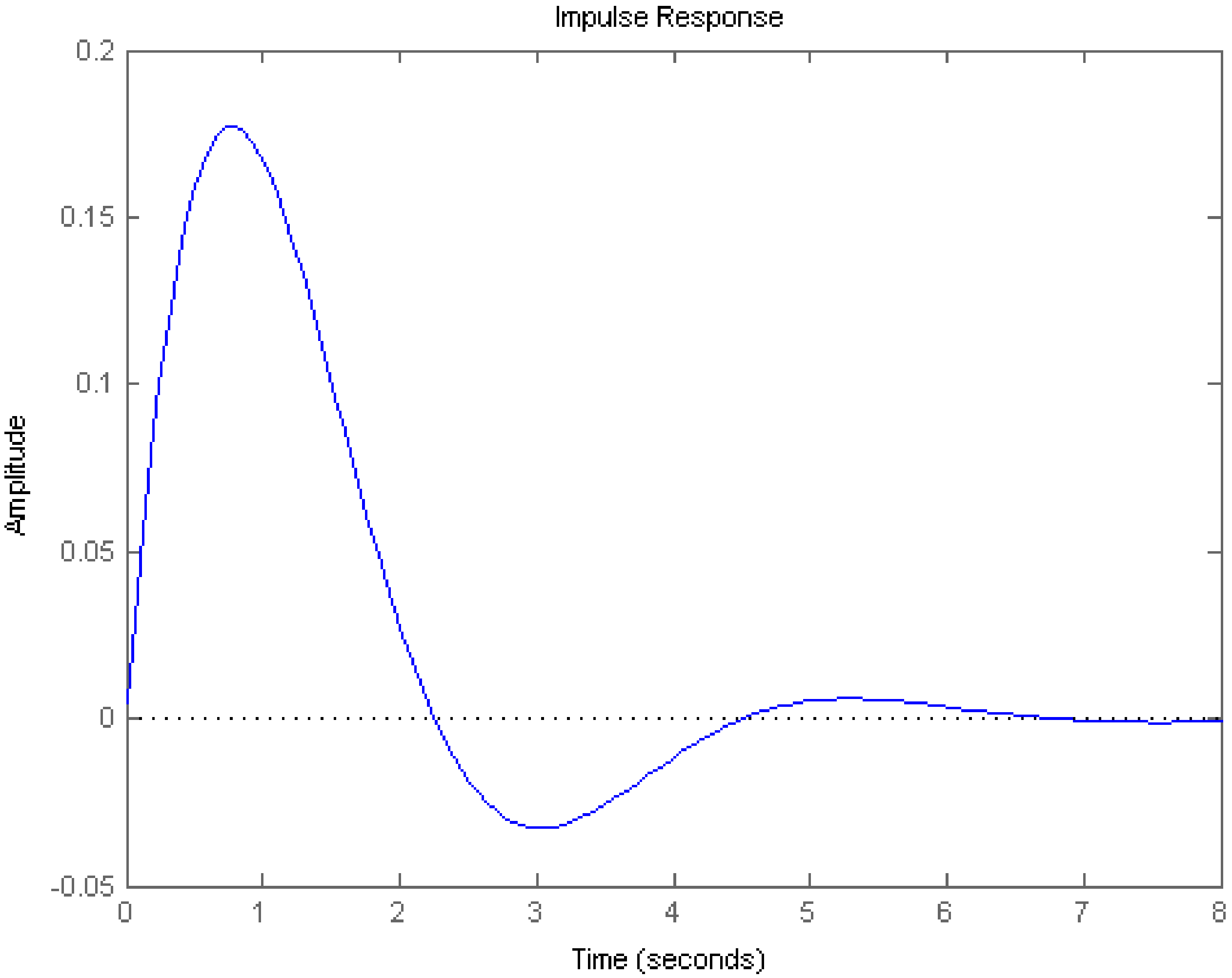


Impulse Function:

$$u(t) = \frac{1}{a\sqrt{\pi}} e^{-x^2/a^2} \quad \text{as } a \rightarrow 0$$

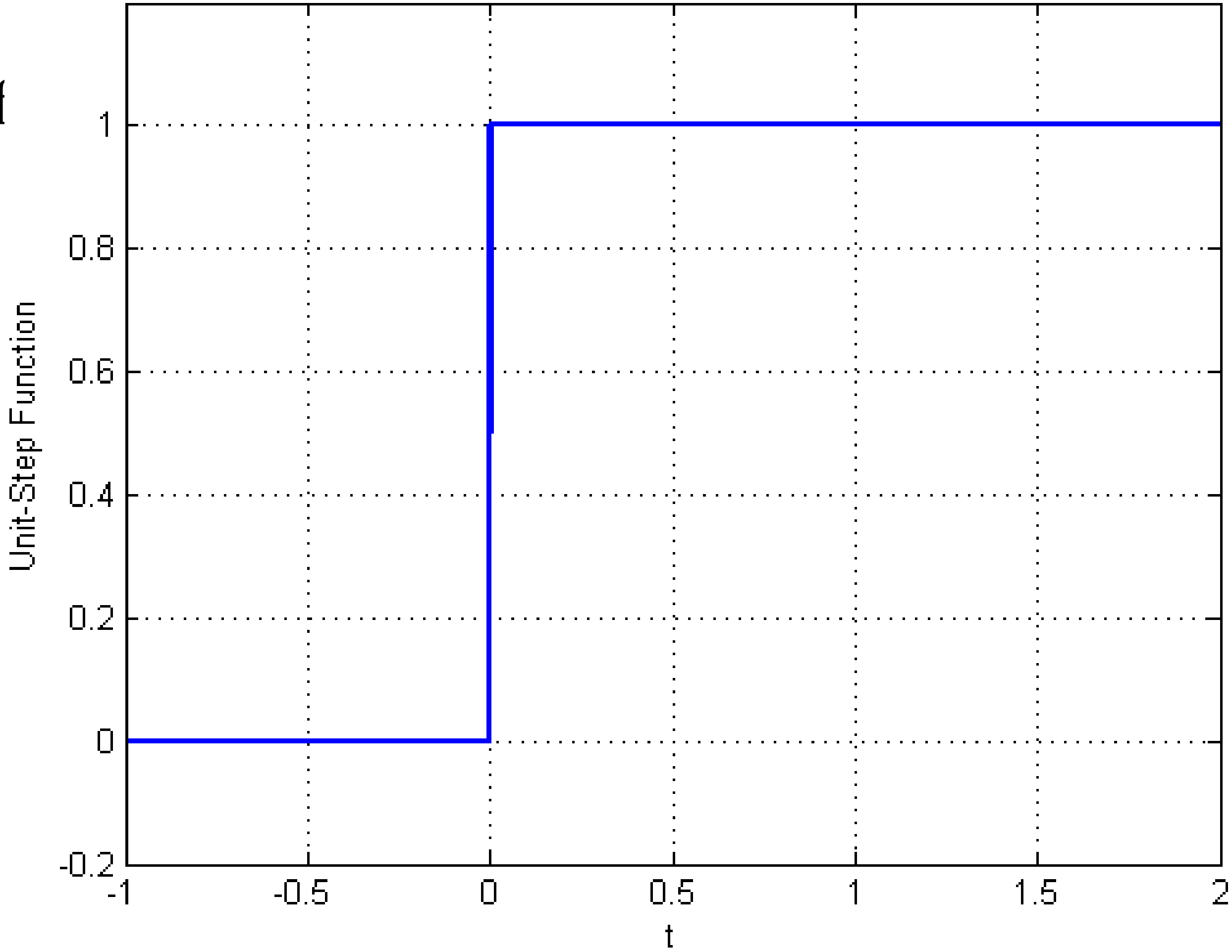
Impulse Response

```
A = [0 1; -5/2 -3/2];
B = [0; 1/2];
C = [1 0];
D = 0;
sys_ss = ss(A,B,C,D);
impulse(sys_ss)
```



Unit-Step Response

`step(sys)` gives the response of the system of equations to a **Unit-Step Function**, where the initial conditions are set to zero.

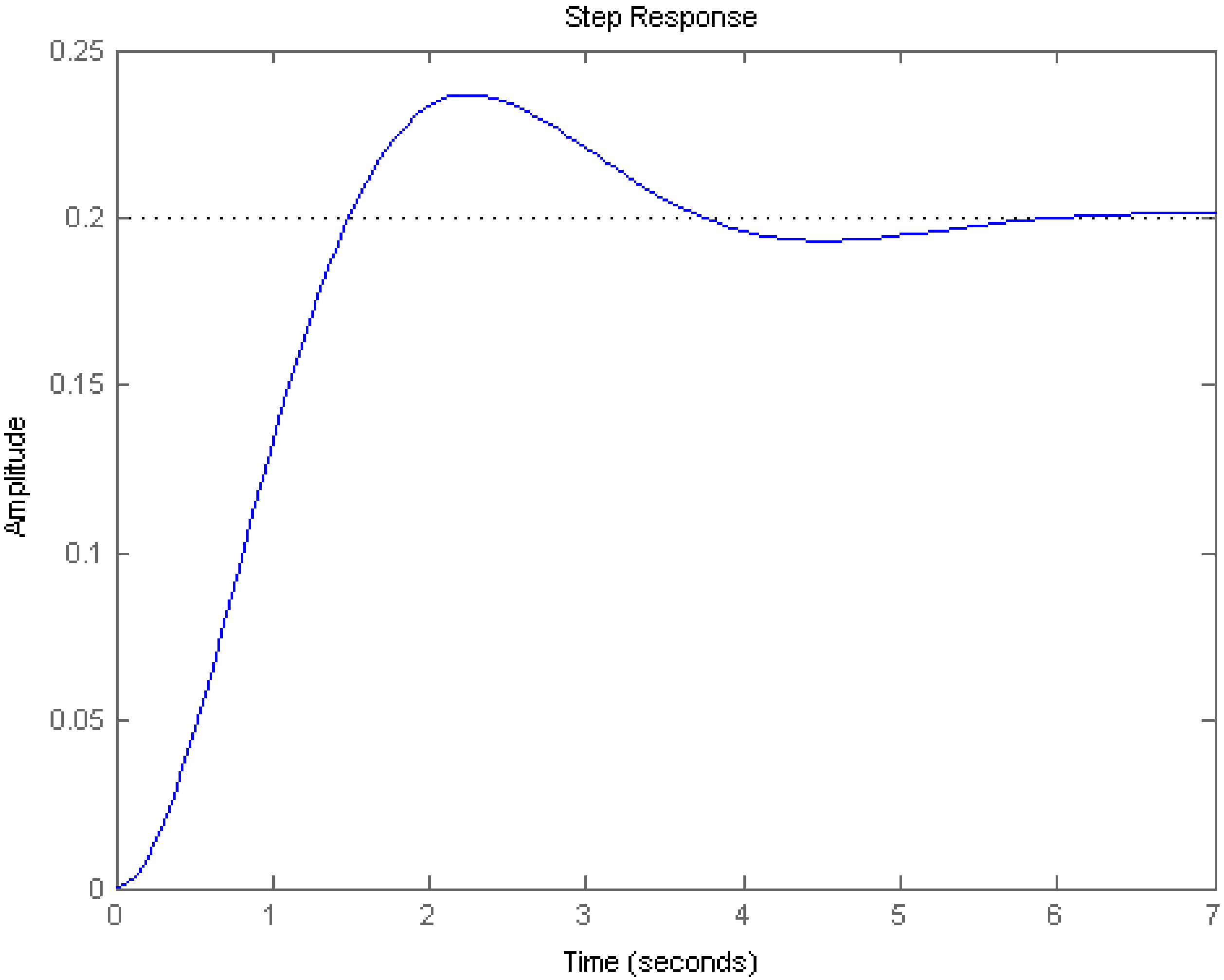


Unit−Step Function:

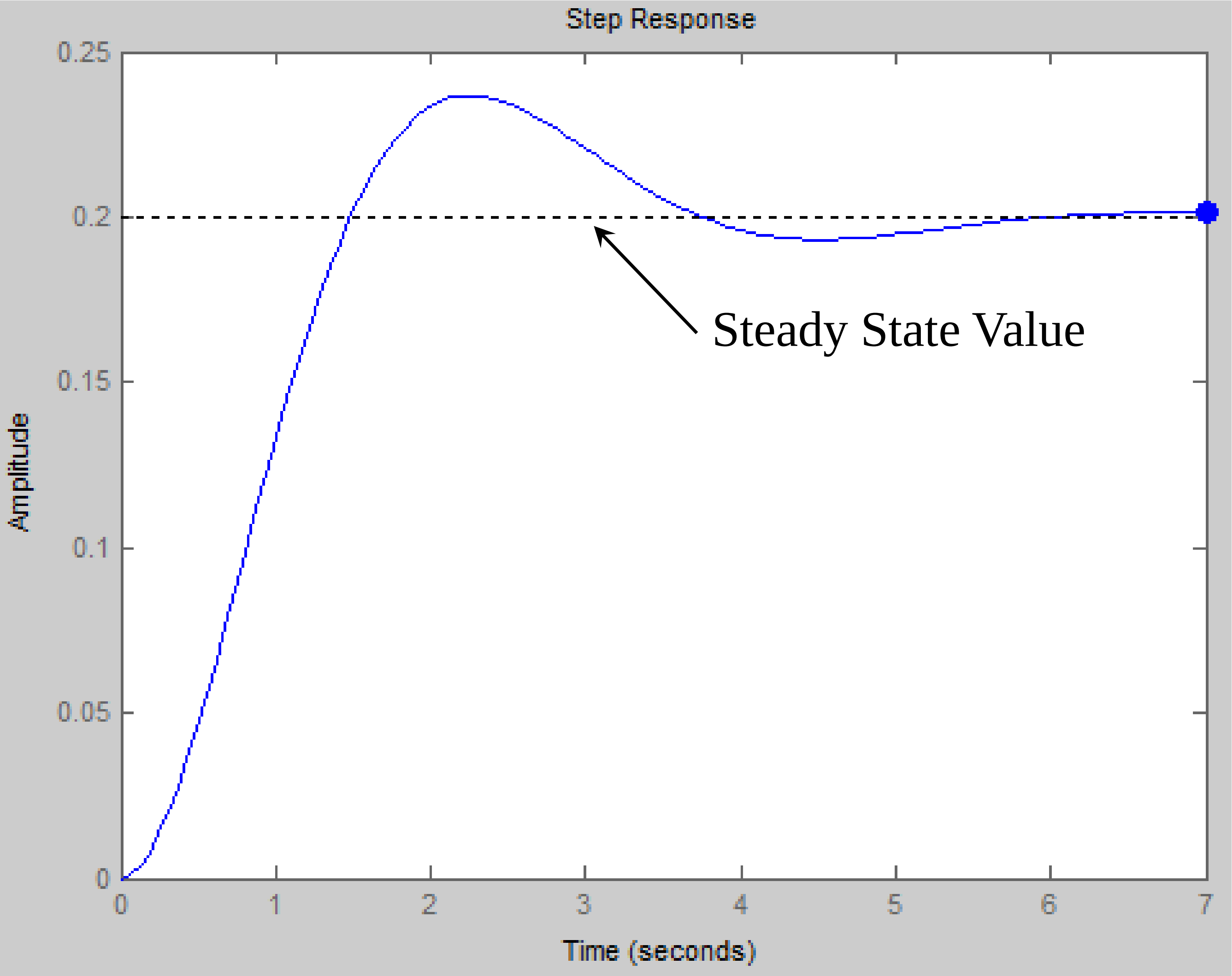
$$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$

Unit-Step Response

```
A = [0 1; -5/2 -3/2];
B = [0; 1/2];
C = [1 0];
D = 0;
sys_ss = ss(A,B,C,D);
step(sys_ss)
```

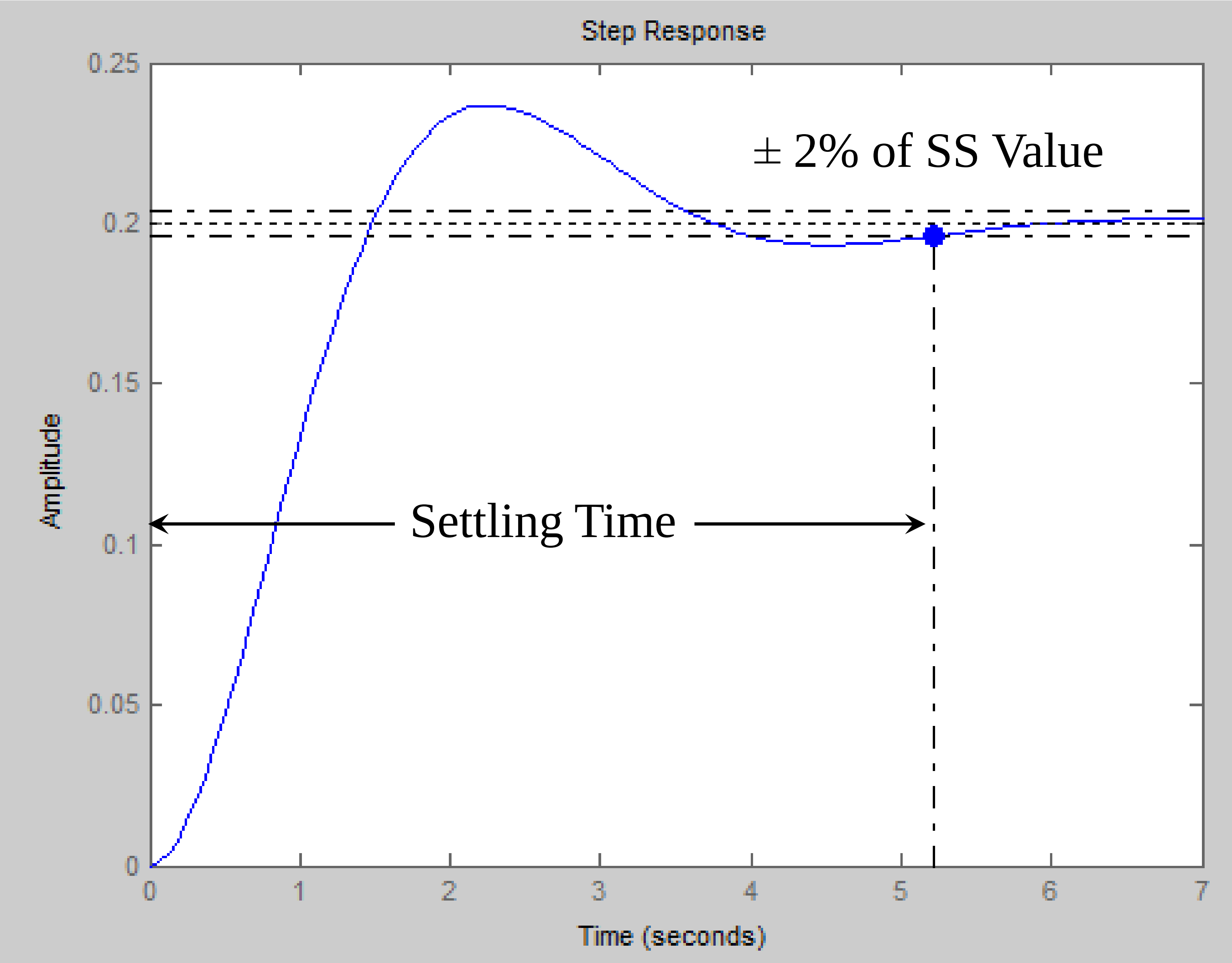


Unit-Step Response

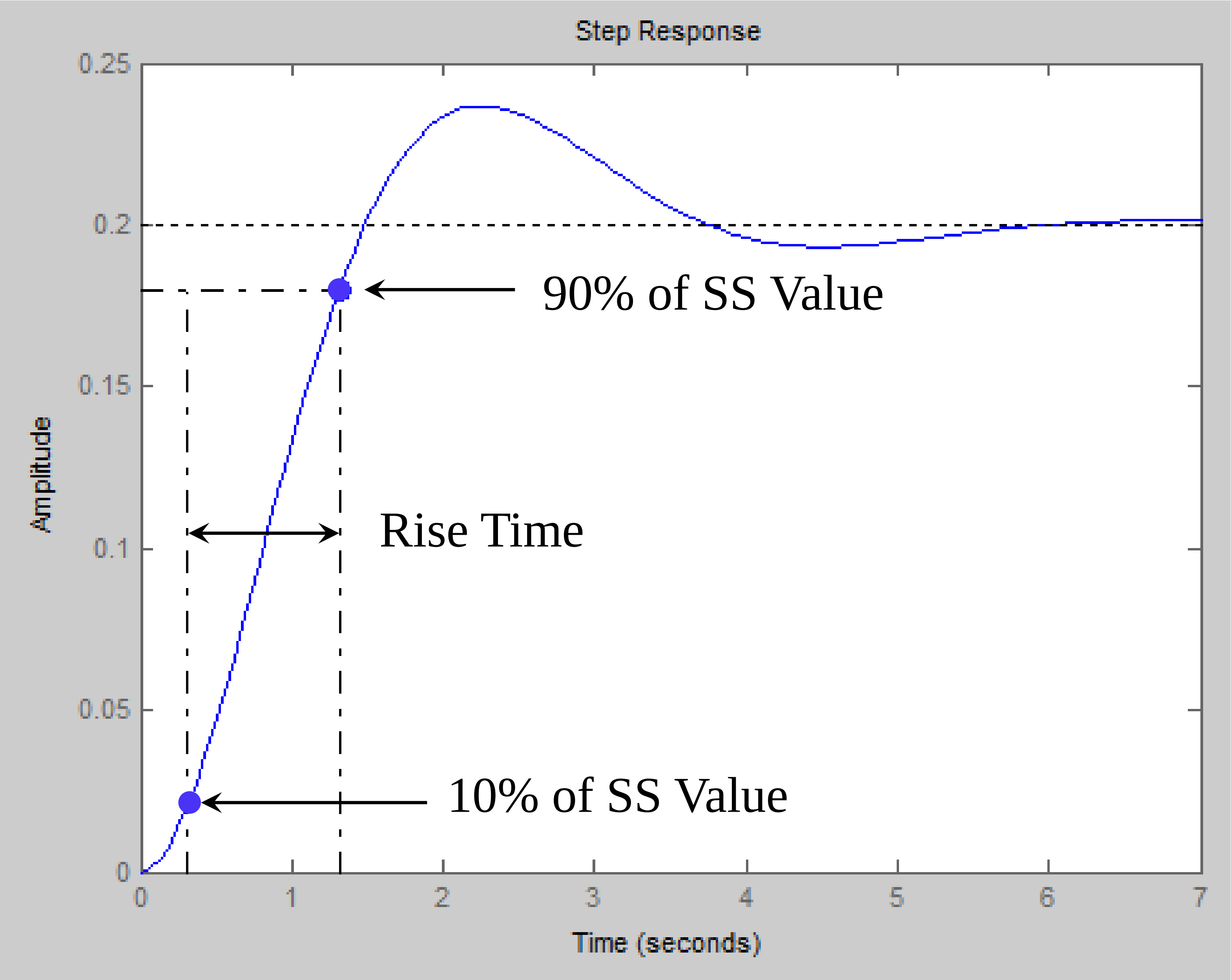


Steady State Value

Unit-Step Response



Unit-Step Response

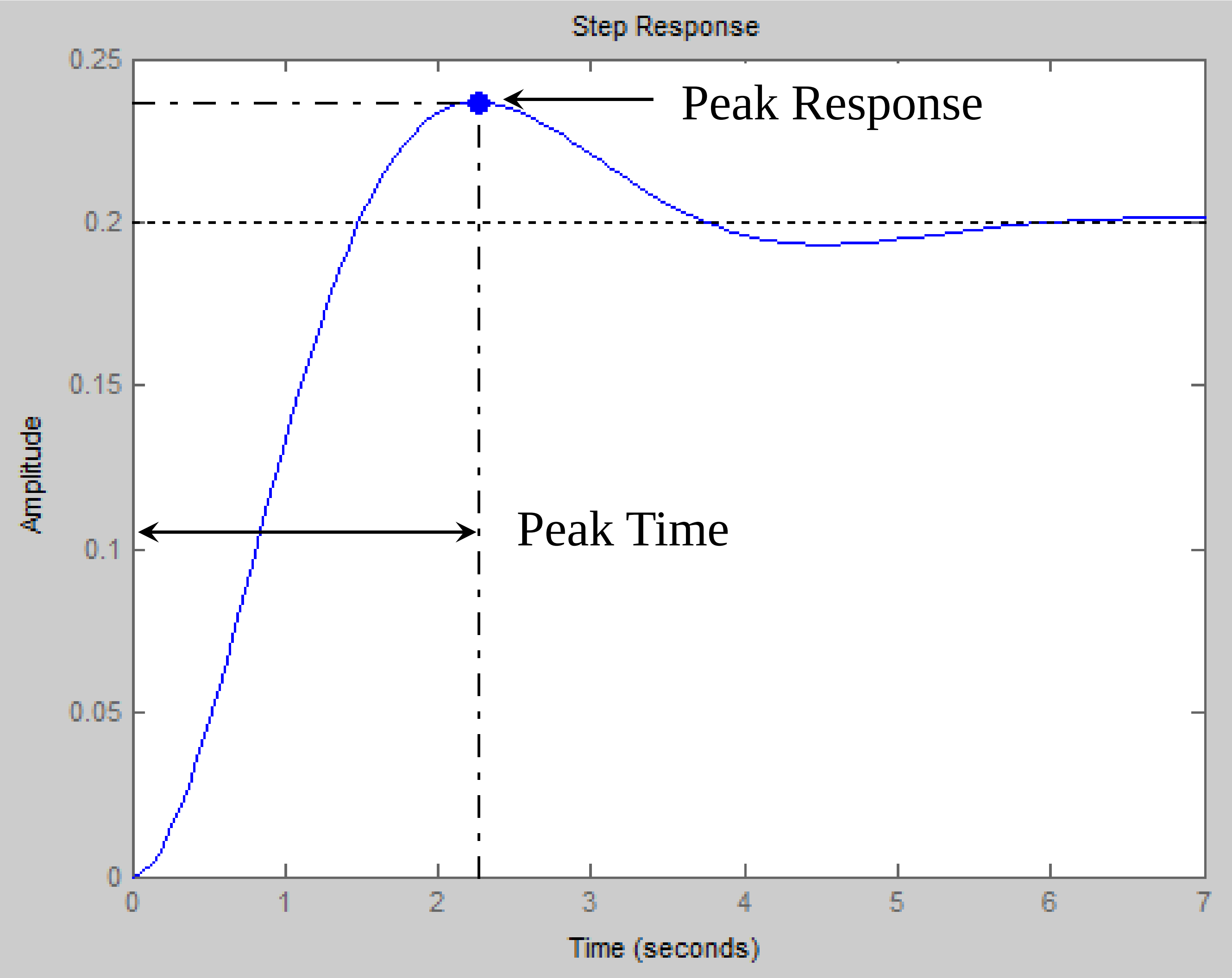


90% of SS Value

Rise Time

10% of SS Value

Unit-Step Response

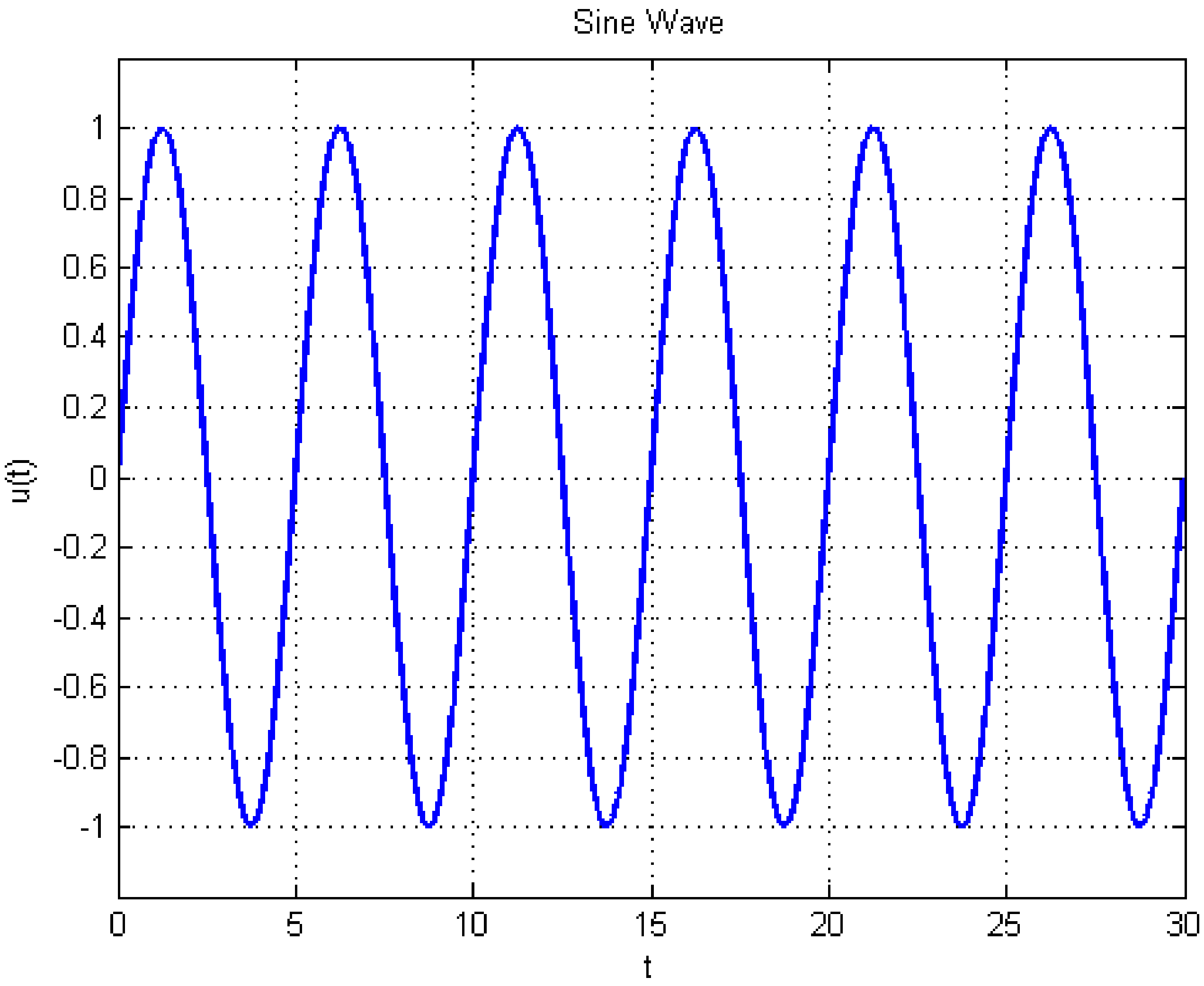


Peak Response

Peak Time

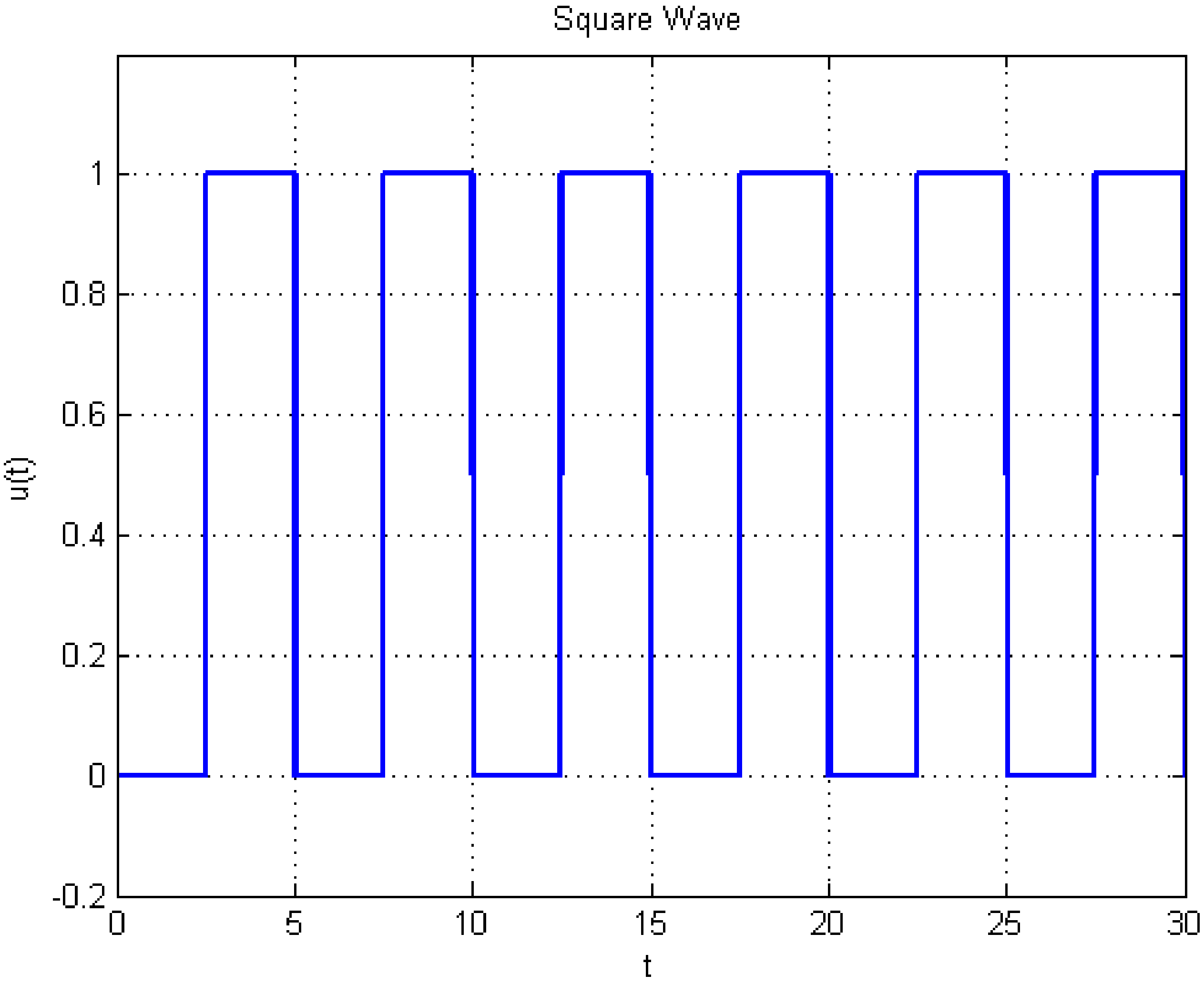
Signal Generator: Sine Wave

```
[u,t] = gensig('sin',5,30,0.01)
plot(t, u, 'LineWidth',2)
xlabel('t'), ylabel('u(t)')
axis([0 30 -1.2 1.2])
grid on
title('Sine Wave')
```



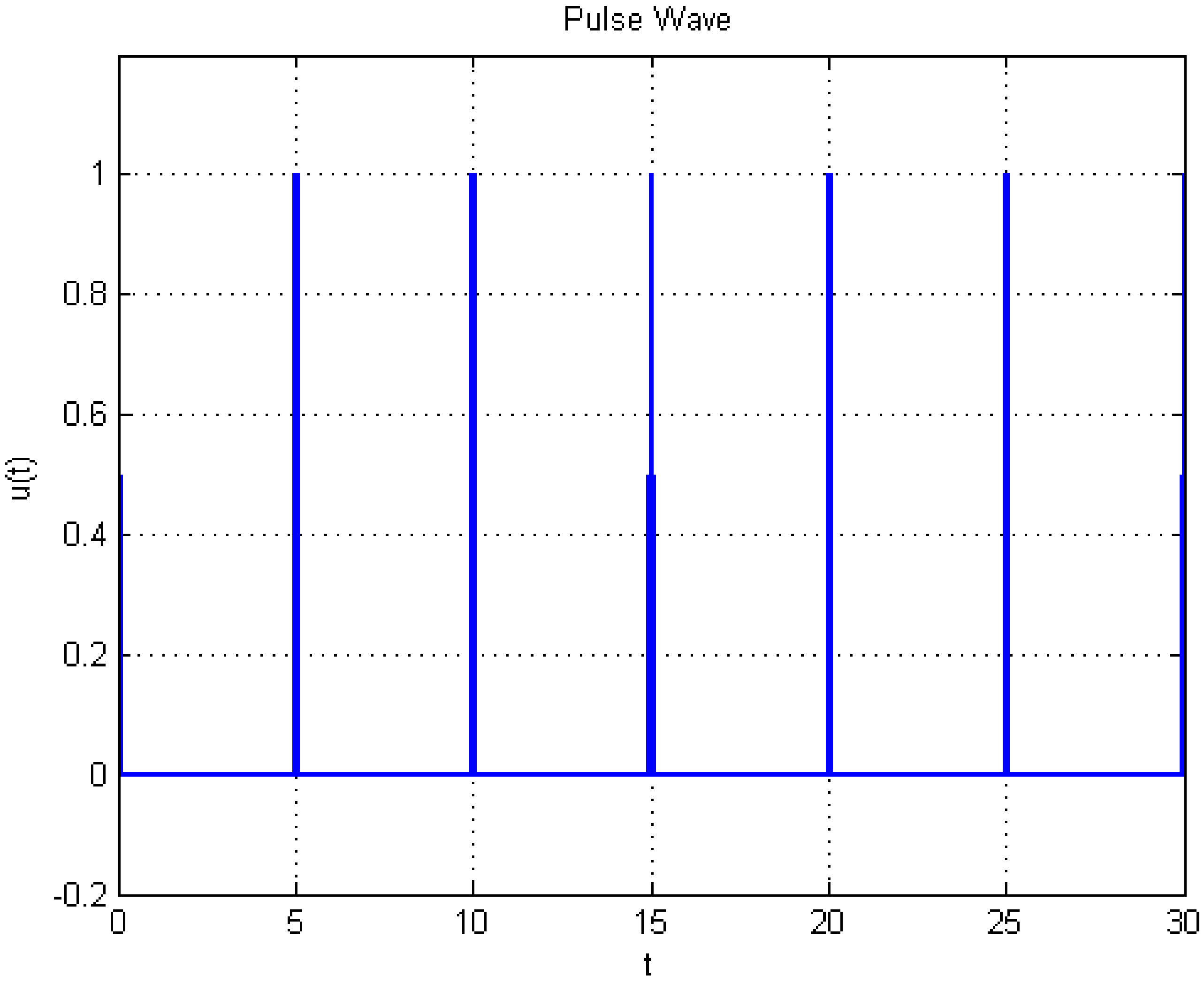
Signal Generator: Square Wave

```
[u,t] = gensig('square',5,30,0.01)
plot(t, u, 'LineWidth',2)
xlabel('t'), ylabel('u(t)')
axis([0 30 -1.2 1.2])
grid on
title('Square Wave')
```



Signal Generator: Pulse Wave

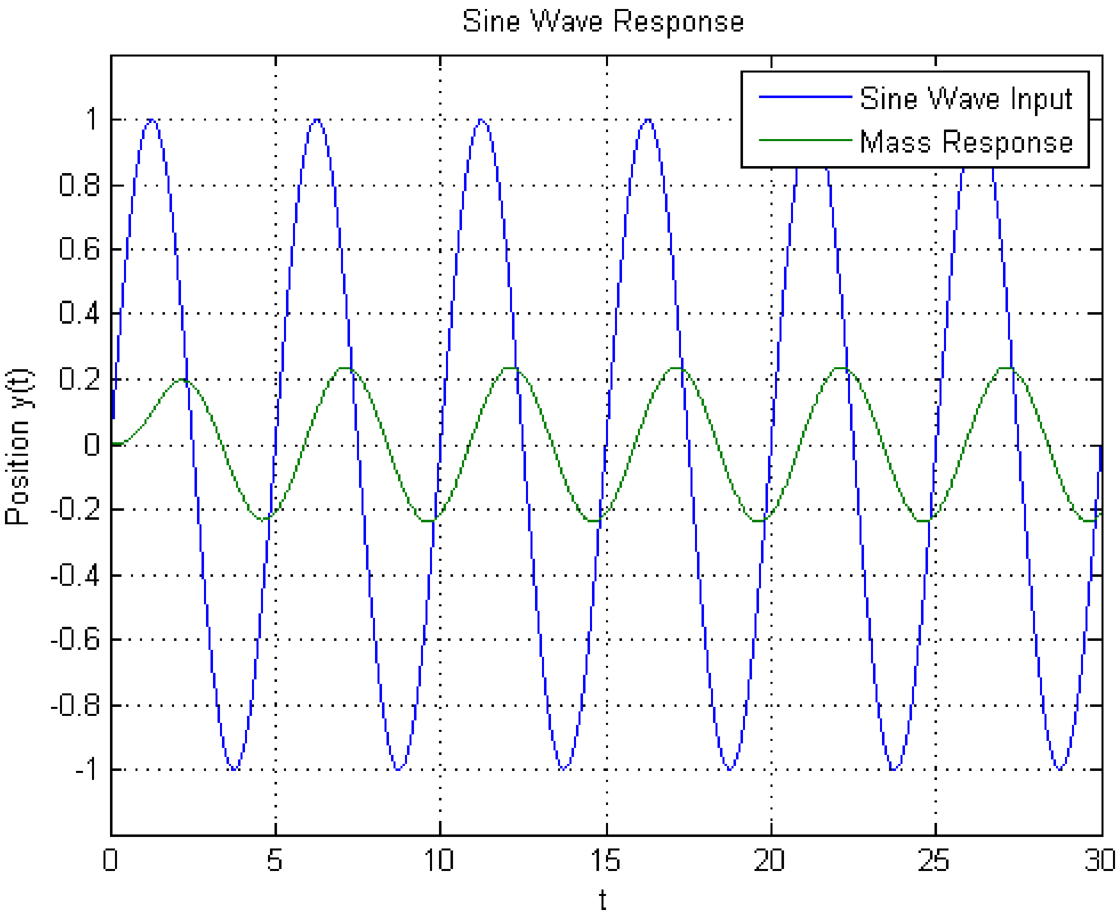
```
[u,t] = gensig(pulse',5,30,0.01)
plot(t, u, 'LineWidth',2)
xlabel('t'), ylabel('u(t)')
axis([0 30 -1.2 1.2])
grid on
title(‘Pulse Wave')
```



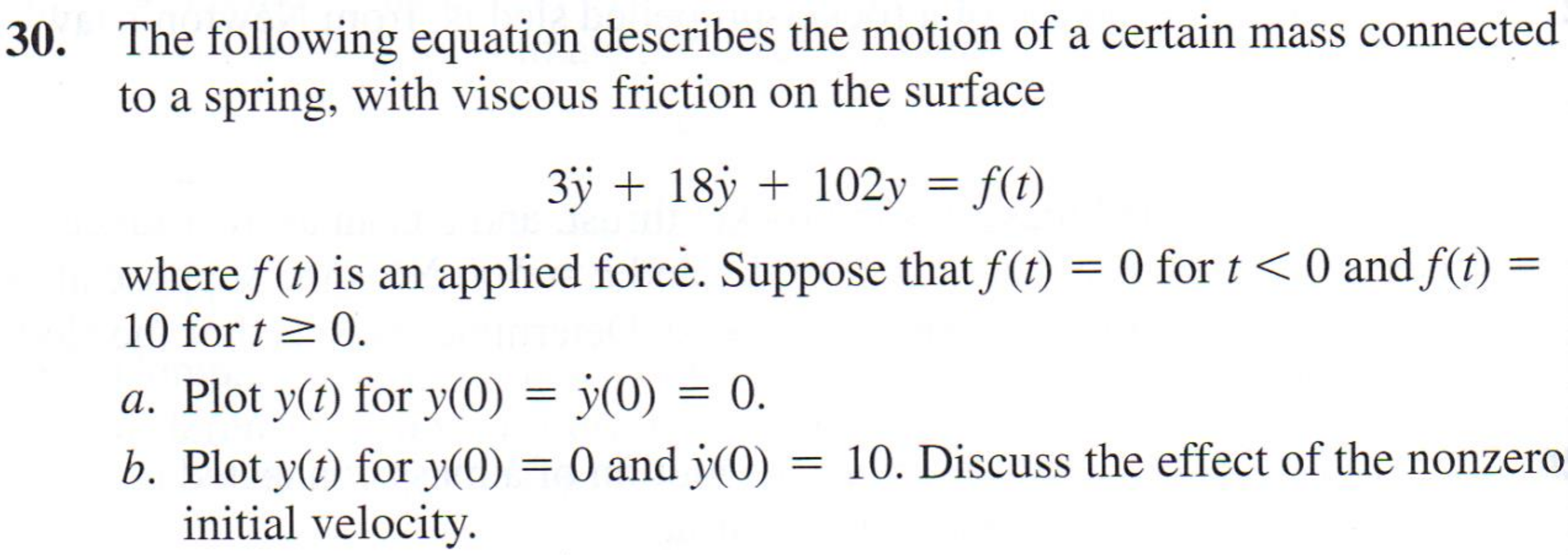
Arbitrary Input Response

`lsim(sys, u,t)` gives the response of the system of equations to an **Arbitrary Input Function**, where the initial conditions are set to zero.

```
A = [0 1; -5/2 -3/2];
B = [0; 1/2];
C = [1 0];
D = 0;
sys_ss = ss(A,B,C,D);
[u,t] = gensig('sin',5,30,0.01);
[y, t] = lsim(sys_ss, u,t);
plot(t, u, t, y),xlabel('t')
```



Problem 9.30:



30. The following equation describes the motion of a certain mass connected to a spring, with viscous friction on the surface

$$3\ddot{y} + 18\dot{y} + 102y = f(t)$$

where $f(t)$ is an applied force. Suppose that $f(t) = 0$ for $t < 0$ and $f(t) = 10$ for $t \geq 0$.

a. Plot $y(t)$ for $y(0) = \dot{y}(0) = 0$.

b. Plot $y(t)$ for $y(0) = 0$ and $\dot{y}(0) = 10$. Discuss the effect of the nonzero initial velocity.

Solve using the **Euler Method**. This is a second-order ordinary differential equation. Rewrite the equation by solving for the second derivative.

$$\ddot{y} = -\frac{18}{3}\dot{y} - \frac{102}{3}y + \frac{10}{3} = -6\dot{y} - 34y + \frac{10}{3}$$

Problem 9.30:

Let $x_1 = y$ and $x_2 = \dot{y}$. Taking the derivative of the first equation gives

$$\dot{x_1} = \dot{y} = x_2 \quad \text{or} \quad \dot{x_1} = x_2$$

Taking the derivative of the second equation gives

$$\dot{x}_2 = \ddot{y} = -6\dot{y} - 34y + \frac{10}{3} = -6x_2 - 34x_1 + \frac{10}{3}$$

or

$$\dot{x}_2 = -6x_2 - 34x_1 + \frac{10}{3}$$

The original second-order ordinary differential equation is now converted into two first-order ordinary differential equations that are coupled.

$$\dot{x_1} = x_2, \qquad \dot{x}_2 = -6x_2 - 34x_1 + \frac{10}{3}, \qquad x_1(0) = 0, \qquad x_2(0) = 0$$

The system of equations can be discretized as follows:

$$x_{1,k+1} = x_{1,k} + \Delta t \cdot x_{2,k}$$

$$x_{2,k+1} = x_{2,k} + \Delta t \cdot \left(-6x_{2,k} - 34x_{1,k} + \frac{10}{3}\right)$$

Problem 9.30:

The initial conditions are $y(0) = x_1(0) = 0$ and $\dot{y}(0) = x_2(0) = 0$. Let $\Delta t = 0.01$ seconds.

For $k = 1$:

$$x_1(2) = x_1(1) + \Delta t \cdot x_2(1) = (0.0) + (0.01)(0.0) = 0.0$$

$$x_2(2) = x_2(1) + \Delta t[-6x_2(1) - 34x_1(1) + 10/3]$$

$$x_2(2) = (0.0) + (0.01)[-(6)(0.0) - (34)(0.0) + 10/3] = 0.0\overline{3}$$

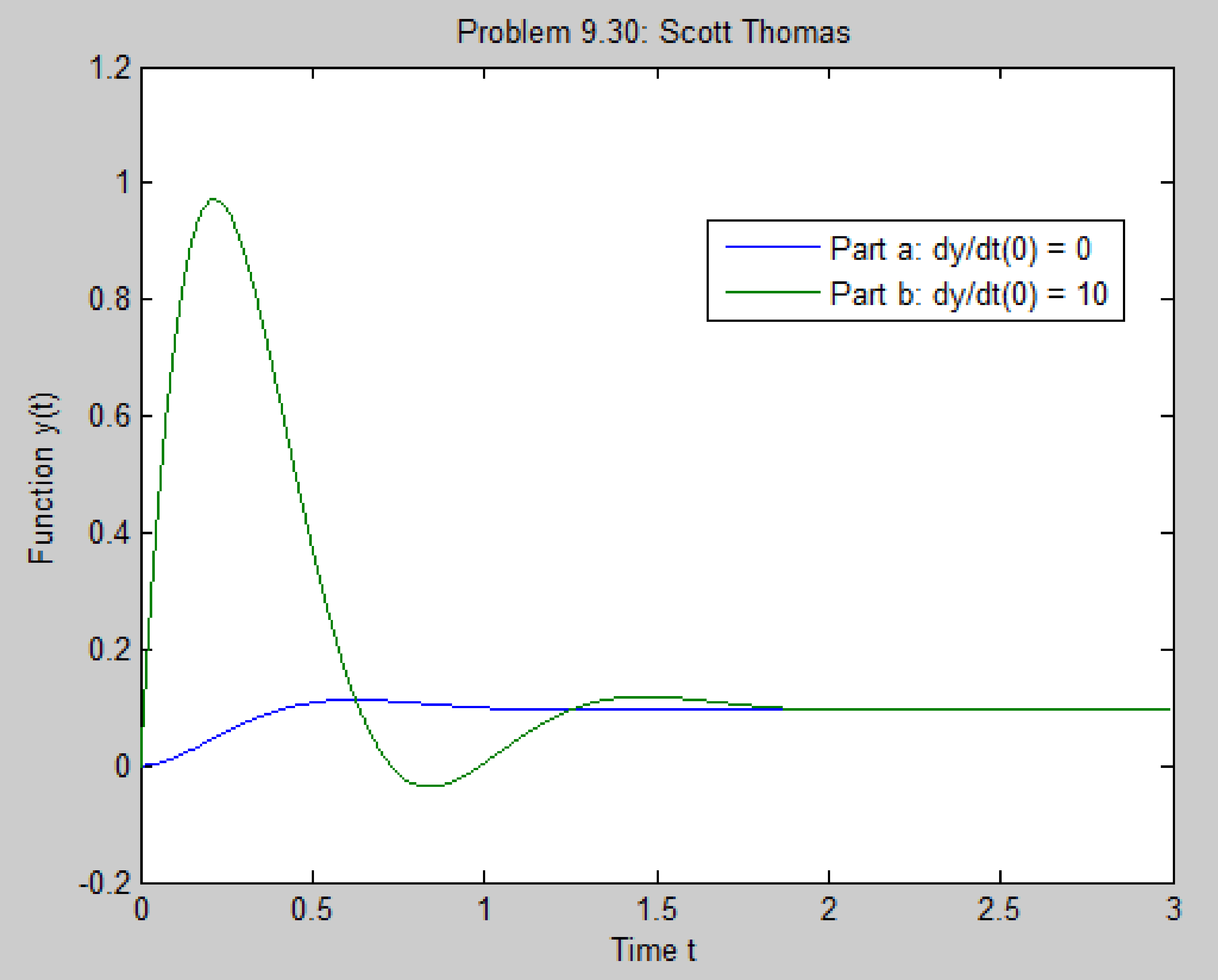
For $k = 2$:

$$x_1(3) = x_1(2) + \Delta t \cdot x_2(2) = (0.0) + (0.01)(0.0\overline{3}) = 0.000\overline{3}$$

$$x_2(3) = x_2(2) + \Delta t[-6x_2(2) - 34x_1(2) + 10/3]$$

$$x_2(3) = (0.0\overline{3}) + (0.01)[-(6)(0.0\overline{3}) - (34)(0.0) + 10/3] = 0.064\overline{6}$$

Problem 9.30:


Problem 9.30: Scott Thomas
Part a: dy/dt(0) = 0
Part b: dy/dt(0) = 10
Function y(t)
Time t
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
0.5
1
1.5
2
2.5
3