MATLAB

for Engineering Applications

Fifth Edition

William J. Palm III

Chapter 09

Numerical Methods for Calculus and Differential Equations

The Integral of $f(x)$ Interpreted as the Area A Under the Curve of $f(x)$ from $x = a$ to $x = b$

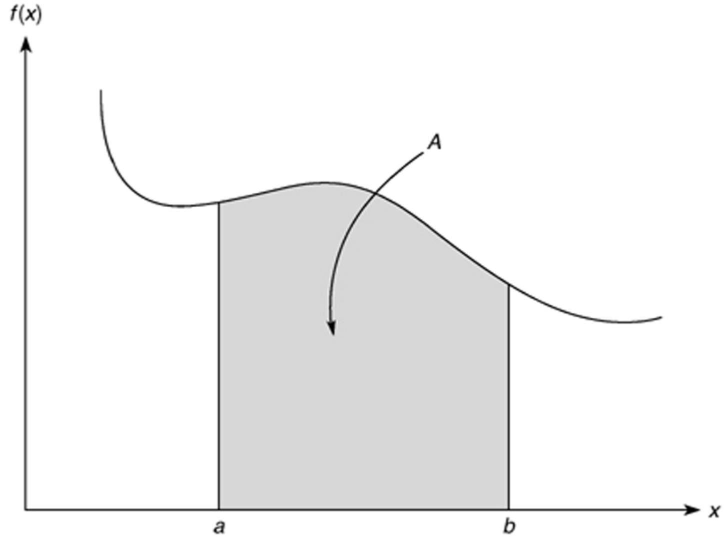
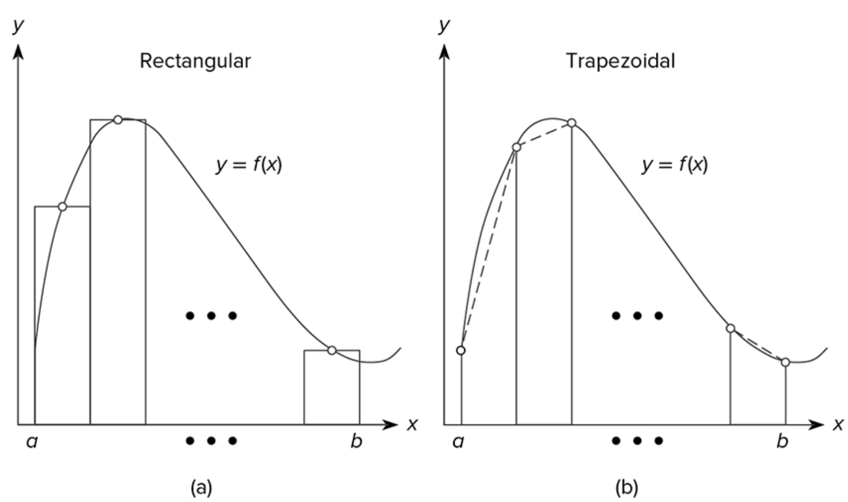


Illustration of (a) Rectangular and (b) Trapezoidal Numerical Integration



Access the text alternative for slide images.

Numerical Integration Functions 1

Command	Description
`integral(fun,a,b)`	Uses an adaptive Simpson's rule to compute the integral of the function whose handle is fun, with a as the lower integration limit and b as the upper limit. The function fun must accept a vector argument.
`trapz(x,y)`	Uses trapezoidal integration to compute the integral of `y` with respect to `x`, where the array `y` contains the function values at the points contained in the array `x`.

Numerical Integration Functions 2

Although the `integral` function is more accurate than `trapz`, it is restricted to computing the integrals of functions and cannot be used when the integrand is specified by a set of points. For such cases, use the `trapz` function.

Numerical Integration Functions 3

Using the `trapz` function. Compute the integral

$$\int_0^{\pi} \sin x \, dx$$

First use 10 panels with equal widths of $\pi/10$. The script file is

```
x = linspace(0,pi,10);
y = sin(x);
trapz(x,y)
```

The answer is 1.9797, which gives a relative error of 100 (2 − 1.9797)/2) = 1%.

Numerical Integration Functions 4

MATLAB function `integral` implements an adaptive version of Simpson's rule. To compute the integral of sin(x) from 0 to π, type

```
>>A = integral(@sin,0,pi)
```

The answer given by MATLAB is 2.0000, which is correct.

Numerical Integration Functions 5

To integrate $\cos(x^2)$ from 0 to $\sqrt{(2\pi)}$, create the function:

```
function c2 = cossq(x)
  % cosine squared function.
  c2 = cos(x.^2);
end
```

Note that we must use array exponentiation.

The `integral` function is called as follows:

```
>>integral(@cossq,0,sqrt(2*pi))
```

The result is 0.6119.

Polynomial Integration

`q = polyint(p,C)` returns a polynomial `q` representing the integral of polynomial `p` with a user-specified scalar constant of integration `C`.

Compute the indefinite integral of $p(x) = 6x^2 - 7x + 10$

```
q = polyint([6,-7,10])
q =
2.0000   -3.5000   10.0000         0
```

which corresponds to $q = 2x^3 - 3.5x^2 + 10x + C$ after adding the constant of integration

Double Integrals

`A = integral2(fun, a, b, c, d)` computes the integral of $f(x,y)$ from $x = a$ to b, and $y = c$ to d. Here is an example using an anonymous function to represent $f(x,y) = xy^2$.

```
>>fun = @(x,y)x.*y.^2;
>>A = integral2(fun,1,3,0,1)
```

The answer is $A = 1.3333$.

Triple Integrals

`A = integral3(fun, a, b, c, d, e, f)` computes the triple integral of $f(x,y, z)$ from $x = a$ to b, $y = c$ to d, and $z = e$ to f. Here is an example using an anonymous function to represent $f(x,y,z) = (xy - y^2)/z$.

```
>>fun = @(x,y,z)(x.*y -y.^2)./z;
>>A = integral3(fun,1,3,0,2,1,2)
```

The answer is $A = 1.8484$. Note that the function must accept a vector x, a scalar y, and a scalar z.

- Use MATLAB to evaluate the following double integral:

$$\int_1^2 \int_0^1 (x^2 + xy^3)\,dx\,dy$$

```
fun = @(x,y)(x.^2+x.*y.^3);
A = integral2(fun, 0, 1, 1, 2)          % ans → A = 2.2083
```

- Use MATLAB to evaluate the following triple integral:

$$\int_0^1 \int_1^2 \int_2^3 xyz\,dx\,dy\,dz$$

```
fun = @(x,y,z)(x.*y.*z);
V = integral3(fun, 2, 3, 1, 2, 0, 1)     % ans → V = 1.8750
```


Numerical Differentiation

MATLAB provides the `diff` function to use for computing derivative estimates. Its syntax is `d = diff(x)`, where `x` is a vector of values, and the result is a vector `d` containing the differences between adjacent elements in `x`.

That is, if `x` has n elements, `d` will have $n - 1$ elements, where,

$$d = [x(2) - x(1), x(3) - x(2), \ldots, x(n) - x(n-1)].$$

For example, if `x = [5, 7, 12, -20]`, then `diff(x)` returns the vector `[2, 5, -32]`.

Polynomial Differentiation Functions

Command	Description
`b = polyder(p)`	Returns a vector `b` containing the coefficients of the derivative of the polynomial represented by the vector `p`.
`b = polyder(p1,p2)`	Returns a vector `b` containing the coefficients of the polynomial that is the derivative of the product of the polynomials represented by `p1` and `p2`.
`[num, den] = polyder(p2,p1)`	Returns the vectors `num` and `den` containing the coefficients of the numerator and denominator polynomials of the derivative of the quotient p_2/p_1, where `p1` and `p2` are polynomials.

Computing Gradients

Typing

```
[df_dx, df_dy] = gradient(f,dx,dy)
```

computes the gradient of the function *f(x,y)*, where df_dx and df_dy represent the partial derivatives, and dx, dy represent the spacing.

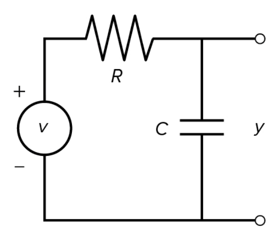
Solving First-Order Differential Equations

The MATLAB ode solver, `ode45`. To solve the equation $dy/dt = f(t,y)$ the syntax is

```
[t,y] = ode45(@ydot,tspan,y0)
```

where `@ydot` is the handle of the function file whose inputs must be *t* and *y*, and whose output must be a column vector representing dy/dt; that is, $f(t,y)$. The number of rows in this column vector must equal the order of the equation. The array `tspan` contains the starting and ending values of the independent variable *t*, and optionally any intermediate values. The array `y0` contains the initial values of *y*. If the equation is first order, then `y0` is a scalar.

Response of an RC circuit



The circuit model for zero input voltage *v* is

$$dy/dt + 10y = 0$$

First solve this for dy/dt:

$$dy/dt = -10y$$

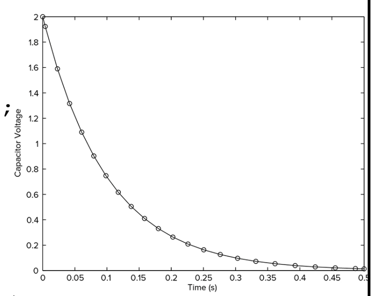
Next define the following function file. Note that the order of the input arguments must be *t* and *y*.

```
function ydot = RC_circuit(t,y)
   ydot = -10*y;
end
```


Response of an RC circuit

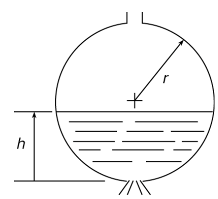
The function is called as follows, and the solution plotted along with the analytical solution `y_true`. The initial condition is $y(0)=2$.

```
[t, y]=ode45(@RC_circuit, [0, 0.5], 2);
y_true = 2*exp(-10*t);
plot(t,y,'o',t,y_true)
xlabel('Time(s)')
ylabel('Capacitor Voltage')
```



Note that we need not generate the array `t` to evaluate `y_true`, because `t` is generated by the `ode45` function.

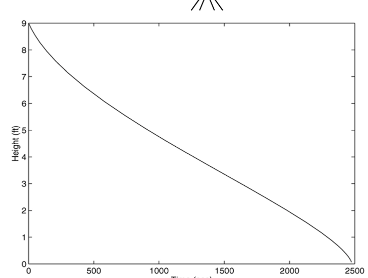
Nonlinear Example



Draining of a spherical tank.

The equation for the height is

$$\frac{dh}{dt} = -\frac{0.0344\sqrt{h}}{10h - h^2}$$



First create the following function file.

```
function hdot = height(t,h)
  Hdot = -(0.0344*sqrt(h))/(10*h-h^2);
end
```

This file is called as follows. The initial height is 9 ft.

```
[t,h] = ode45(@height, [0, 2475], 9);
plot(t,h),xlabel('Time(sec)',ylabel('Height'(ft)')
```

Extension to Higher-Order Equations

To use the ODE solvers to solve an equation higher than order 2, you must first write the equation as a set of first-order equations.

For example, consider the equation

$$5\ddot{y} + 7\dot{y} + 4y = f(t)$$

Define $x_1 = y$ and $x_2 = dy/dt$. Then the above equation can be expressed as two equations:

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

$$\dot{x}_2 = \frac{1}{5}f(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

This form is sometimes called the *Cauchy form* or the *state-variable form*.

Extension to Higher-Order Equations

Suppose that $f(t) = \sin t$. Then the required file is

```
function xdot = example_1(t,x)
   xdot(1) = x(2);
   xdot(2) = (1/5)*(sin(t)-4*x(1)-7*x(2));
   xdot = [xdot(1); xdot(2)];
end
```

Note that:

`xdot(1)` represents dx_1/dt

`xdot(2)` represents dx_2/dt

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

$$\dot{x}_2 = \frac{1}{5}f(t) - \frac{4}{5}x_1 - \frac{7}{5}x_2$$

`x(1)` represents x_1 and `x(2)` represents x_2

Extension to Higher-Order Equations

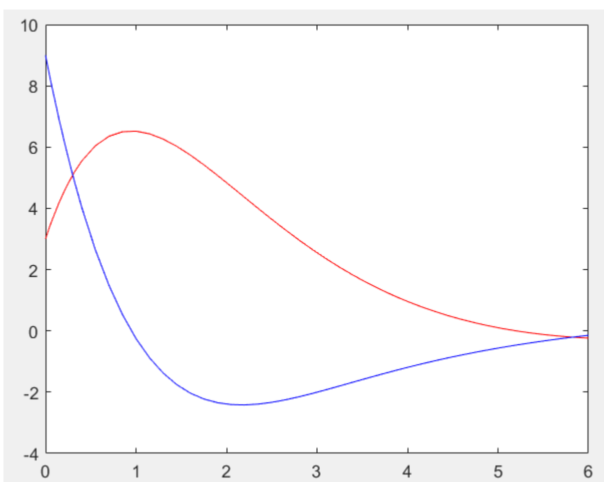
Suppose we want to solve the equation for $0 \leq t \leq 6$ with the initial conditions $x_1(0) = 3$, $x_2(0) = 9$ and $f(t) = \sin t$. Then the initial condition for the *vector* **x** is `[3, 9]`. To use `ode45`, you type `[t, x] = ode45(@example_1, [0, 6], [3, 9]);`

Each row in the vector `x` corresponds to a time returned in the column vector `t`. If you type `plot(t,x)`, you will obtain a plot of both x_1 and x_2 versus t.

Note that `x` is a matrix with two columns; the first column contains the values of x_1 at the various times generated by the solver. The second column contains the values of x_2.

Thus, to plot only x_1, type `plot(t,x(:,1))`.

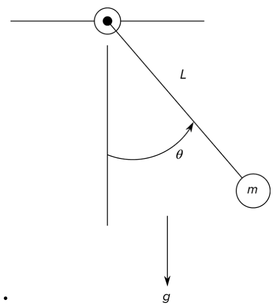
Extension to Higher-Order Equations



Pendulum Example - Nonlinear Model

The model is nonlinear and is

$$\ddot{\theta} + \frac{g}{L}\sin\theta = 0$$



It must be rewritten as follows to use ode45.

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

$$\dot{x}_2 = -\frac{g}{L}\sin x_1$$

Nonlinear Model

Create the following function file. Note how we can express `xdot` as a vector in one line, instead of two.

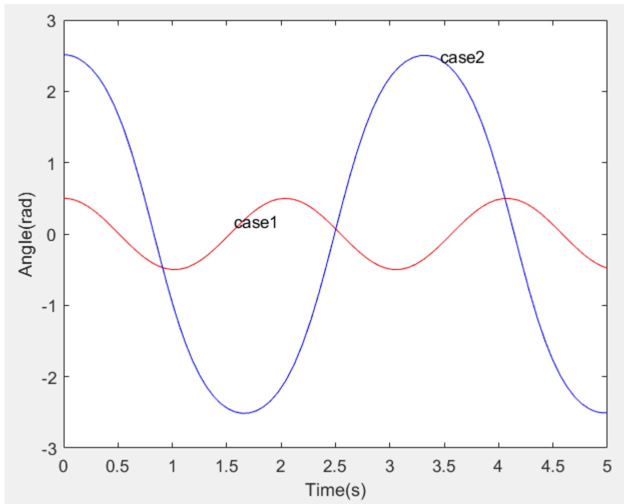
pendulum.m

```
function xdot=pendulum(t,x)
g=9.81;L=1;
xdot=[x(2);-(g/L)*sin(x(1))];
end
```

The file is called as follows. The vectors ta and xa contain the results for the case where $\theta(0)$=0.5. The vectors tb and xb contain the results for $\theta(0)$=0.8pi. In both cases, the initial angular velocity is zero $\dot{\theta}(0)$.

```
% Pendulum Example - Non-linear Model
[ta,xa] = ode45(@pendulum, [0,5],[0.5,0]);
[tb,xb] = ode45(@pendulum, [0,5],[0.8*pi 0]);
plot(ta,xa(:,1),'r', tb,xb(:,1),'b')
xlabel('Time(s)')
ylabel('Angle(rad)')
gtext('case1')
gtext('case2')
```

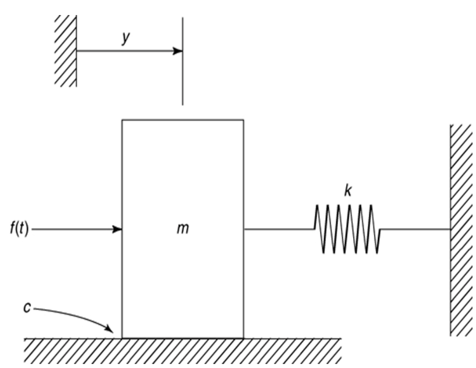
The Pendulum Angle as a Function of Time for Two Starting Positions



A Mass and Spring

A mass and spring with viscous surface friction. Its equation of motion is

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



Access the text alternative for slide images.

Special Methods for Linear Differential Equations

The equation of motion can be put into the following state variable form.

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

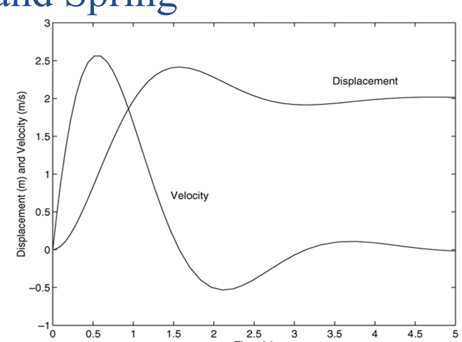
$$\dot{x}_2 = \frac{1}{m}u(t) - \frac{k}{m}x_1 - \frac{c}{m}x_2$$

These can be put into matrix form as shown below.

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

A Mass and Spring

```
function xdot = msd(t,x)
    u = 10; m = 1; c = 2; k = 5;
    A = [0, 1;-k/m, -c/m];
    B = [0; 1/m];
    xdot = A*x + B*u;
end
```



For $0 \le t \le 5$, $x_1(0) = 0$ and $x_2(0) = 0$ The equations can be solved, and the solution plotted.

```
[t, x] = ode45(@msd, [0, 5], [0, 0]);
plot(t, x(:,1), t, x(:,2))
```


Control Systems

A **transfer function** represents the relationship between the output signal of a control system and the input signal, for all possible input values.

Step response is the response to a system when the input is a step signal.

An **impulse response** means the output/behaviour of a system/process when we provide it with an impulse signal.

ODE Solvers in the Control System Toolbox The Step Function

Transfer function form: It is created by typing `tf(right, left)` where the vector `right` contains the coefficients on the right side of the equation and the vector `left` contains the coefficients on the left side. Consider the equation:

$$5\ddot{y} + 7\dot{y} + 5y = 5\dot{f} + f(t)$$

You create the transfer function model form named `sys1` by typing

```
sys = tf([5, 1],[5, 7, 5]);
```

You can plot the unit step response for zero initial conditions by typing `step(sys)`.

```
[x,t] = step(sys);
plot(t,x)
```


LTI Object Functions

Command	Description
`sys = ss(A, B, C, D)`	Creates an LTI (Linear Time-Invariant) object in state-space form, where the matrices `A, B, C,` and `D` correspond to those in the model $dx/dt = \mathbf{Ax} + \mathbf{Bu}$, $\mathbf{y} = \mathbf{Cx} + \mathbf{Du}$.
`[A, B, C, D] = ssdata(sys)`	Extracts the matrices `A, B, C,` and `D` corresponding to those in the model $dx/dt = \mathbf{Ax} + \mathbf{Bu}$, $\mathbf{y} = \mathbf{Cx} + \mathbf{Du}$.
`sys = tf(right,left)`	Creates an LTI object in transfer-function form, where the vector `right` is the vector of coefficients of the right-hand side of the equation, arranged in descending derivative order, and `left` is the vector of coefficients of the left-hand side of the equation, also arranged in descending derivative order.
`[right, left] = tfdata(sys)`	Extracts the coefficients on the right- and left-hand sides of the reduced-form model.

Basic Syntax of the LTI ODE Solvers

Command	Description
`impulse(sys)`	Computes and plots the unit-impulse response of the LTI object `sys`.
`initial(sys,x0)`	Computes and plots the free response of the LTI object `sys` given in state-model form, for the initial conditions specified in the vector `x0`.
`lsim(sys,u,t)`	Computes and plots the response of the LTI object `sys` to the input specified by the vector `u`, at the times specified by the vector `t`.
`step(sys)`	Computes and plots the unit-step response of the LTI object `sys`.

The State Variable Form Can Be Created with the `ss` Function

Consider the model

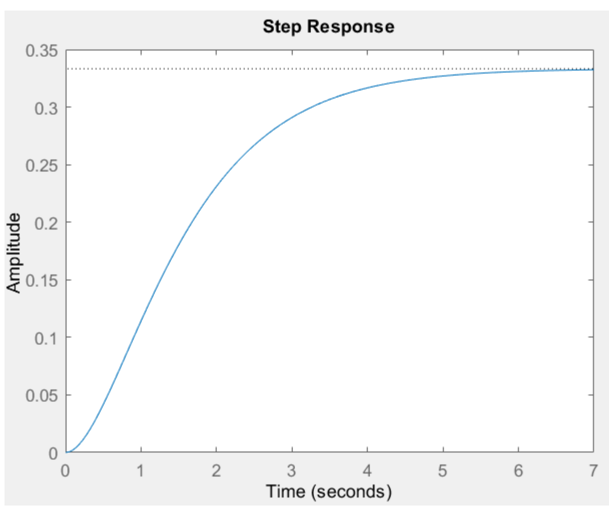
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

y = x1

Create the state space model `sys` for $m = 2$, $c = 5$ and $k = 3$, and plot the unit step response of the first variable by typing

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

[A, B, C, D] = ssdata(sys) % if we know the system we can get the
State Space data
```

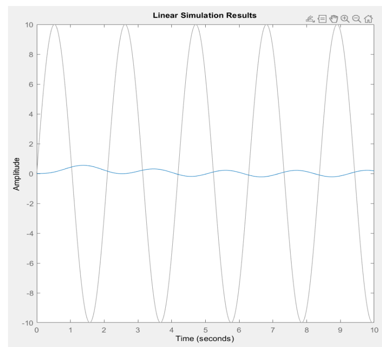



The `impulse` and `lsim` Funcions

The `impulse` function computes and plots the impulse response. The `lsim` function computes and plots the solution for a user-defined input function. Both can be used with either the transfer function or the state variable forms. Here is an example for the following equation with $f(t) = 10 \sin 3t$ and $y(0) = 2$ for $t = 0$ to $t = 10$.

$$5\ddot{y} + 7\dot{y} + 4y = f(t)$$

```
sys = tf(1,[5,7,4]);
t = linspace(0,10,500);
f = 10*sin(3*t);
lsim(sys,f,t,2)
```

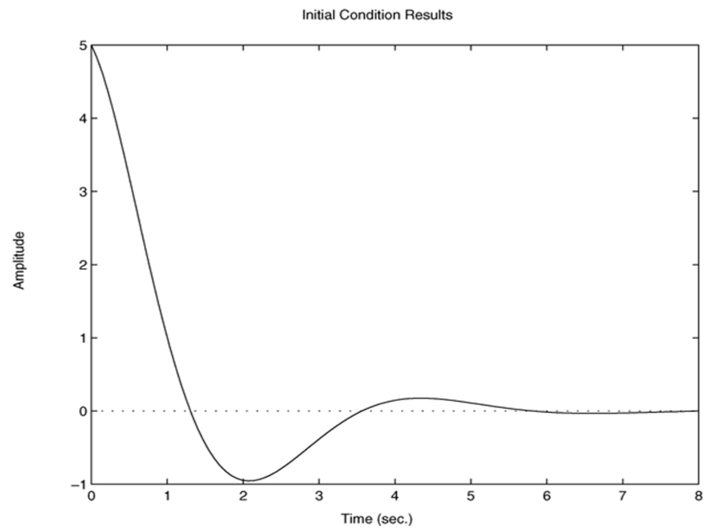


The `initial` Function

The command `initial(sys,x0)` computes and plots the free response of the LTI object `sys` given in state-model form, for the initial conditions specified in the vector `x0`. Initial conditions $x_1(0) = 5$, and $x_2(0) = -2$. For example,

```
m = 2; c = 3; k = 5;
A = [0, 1;-k/m, -c/m];
B = [0; 1/m];
C = [1,0]; D = 0;
sys = ss(A,B,C,D);
initial(sys, [5, -2])
```

Free Response of the Model for $x_1(0) = 5$ and $x_2(0) = -2$



Predefined Input Functions

The `gensig` function makes it easy to construct periodic input functions. The syntax is

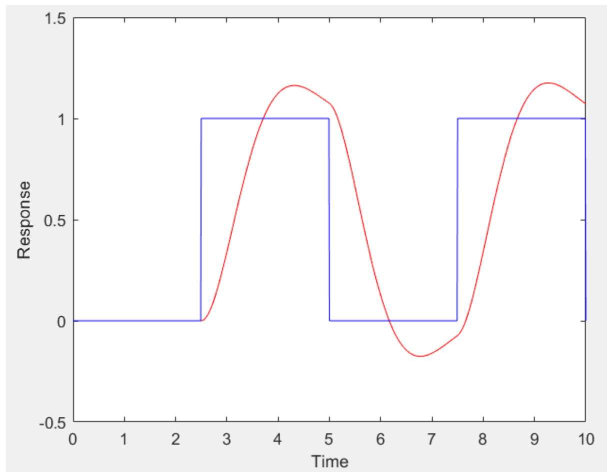
```
[u,t] = gensig(type, period)

[u,t] = gensig(type, period, tF, dt)
```

where `type` can be 'sin', 'square', or 'pulse' and `period` is the desired period of the input. The vector `t` contains the times and the vector `u` contains the input values at those times. tF specifies the time duration of the input, and dt specifies the spacing between time instants.

Square-Wave Response of the Model $\dot{x} + 2x + 4x = 4f$

```
sys = tf(4,[1,2,4]);
[u, t] = gensig('square', 5, 10, 0.01);
[y, t] = lsim(sys, u, t);
plot(t,y,'r',t,u,'b');
axis([0 10 -0.5 1.5]);
xlabel('Time');
ylabel('Response');
```



McGraw Hill

Because learning changes everything.®

www.mheducation.com