MATLAB Windows

We have already described the MATLAB Command Window and the Help Browser, and have mentioned in passing the Command History window, Current Directory browser, Workspace browser, and Launch Pad. These, and several other windows you will encounter as you work with MATLAB, will allow you to: control files and folders that you and MATLAB will need to access; write and edit the small MATLAB programs (that is, M-files) that you will utilize to run MATLAB most effectively; keep track of the variables and functions that you define as you use MATLAB; and design graphical models to solve problems and simulate processes. Some of these windows launch separately, and some are embedded in the Desktop. You can dock some of those that launch separately inside the Desktop (through the **View**: **Dock** menu button), or you can separate windows inside your MATLAB Desktop out to your computer desktop by clicking on the curved arrow in the upper right.

Typing in the Command Window

Click in the Command Window to make it active. When a window becomes active, its title bar darkens. It is also likely that your cursor will change from outline form to solid, or from light to dark, or it may simply appear. Now you can begin entering commands [1].

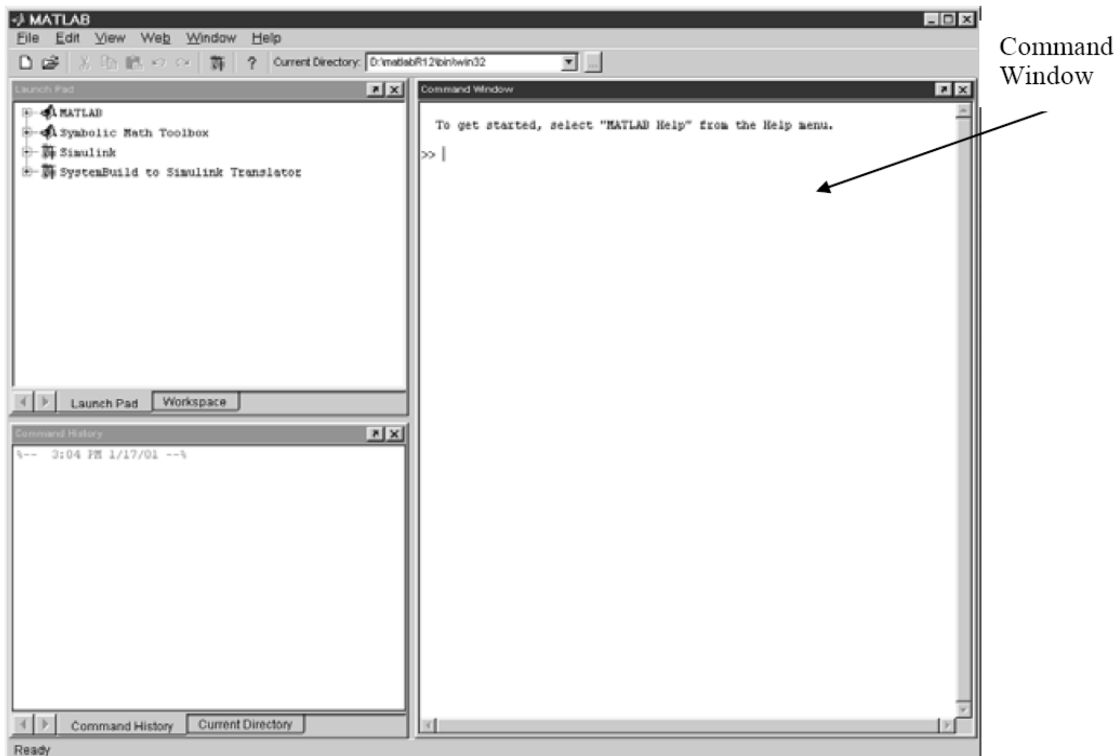


Figure 1-1: A MATLAB Desktop

Input/output of Data from MATLAB Command Window

MATLAB remembers all input data in a session (anything entered through direct keyboard input or running a script file) until the command 'clear()' is given or you exit MATLAB. One of the many features of MATLAB is that it enables us to deal with the vectors/matrices in the same way as scalars. For instance, to input the matrices/ vectors,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & -2 & 3 & -4 \end{bmatrix}$$

type in the MATLAB Command window as below:

```
>>A = [1 2 3;4 5 6]
   A=1 2 3
      4 5 6
>>B = [3;-2;1]; %put the semicolon at the end of the statement to suppress the result printout onto the screen
>>C = [1 -2 3 -4]
```

At the end of the statement, press <Enter> if you want to check the result of executing the statement immediately. Otherwise, type a semicolon ";" before pressing <Enter> so that your window will not be overloaded by a long display of results [2].

Arithmetic Operations

+ Addition

\- Subtraction

* Multiplication

.* Array multiplication

\ Left division

.\ Array left division

/ Right division

./ Array right division

^ Matrix or scalar raised to a power

.^ Array raised to a power

' Complex conjugate transpose

.' Real transpose

Introduction to Vectors in Matlab

This is the basic introduction to Matlab. Creation of vectors is included with a few basic operations. Matlab is a software package that makes it easier for you to enter matrices and vectors, and manipulate them. Almost all of Matlabs basic commands revolve around the use of vectors. To simplify the creation of vectors, you can define a vector by specifying the first entry, an increment, and the last entry. Matlab will automatically figure out how many entries you need and their values. For example, to create a vector whose entries are 0, 2, 4, 6, and 8, you can type in the following line:

```
>> 0:2:8

ans =

   0   2   4   6   8
```

Matlab also keeps track of the last result. In the previous example, a variable "*ans*" is created. To look at the transpose of the previous result, enter the following:

```
>> ans'

ans =

   0
   2
   4
   6
   8
```

To be able to keep track of the vectors you create, you can give them names. For example, a row vector v can be created:

```
>> v = [0:2:8]

v =

   0   2   4   6   8

>> v

v =

   0   2   4   6   8

>> v'

ans =

   0
   2
   4
   6
   8
```

Note that in the previous example, if you end the line with a semi-colon, the result is not displayed. This will come in handy later when you want to use Matlab to work with very large systems of equations. Matlab will allow you to look at specific parts of the vector. If you want to only look at the first three entries in a vector you can use the same notation you used to create the vector:

1. Solving Matrix Equations Using Matrix Division

If A is a square, nonsingular matrix, then the solution of the equation Ax=b is $x = A^{-1}b$. Matlab implements this operation with the backslash operator:

```
>> A = rand (3, 3)
A =
0.2190   0.6793   0.5194
0.0470   0.9347   0.8310
0.6789   0.3835   0.0346

>> b = rand (3, 1)
b =
0.0535
0.5297
0.6711

>> x = A\b
x =
 -159.3380
 314.8625
 -344.5078

>> A*x-b
ans =
1.0e-13 *
-0.2602
-0.1732
-0.0322
```

A\b is (mathematically) equivalent to multiplying b on the left by A^{-1} (however, Matlab does not compute the inverse matrix; instead it solves the linear system directly).

2. Vectorized functions and operators

Matlab has many commands to create special matrices; the following command creates a row vector whose components increase arithmetically:

```
>> t = 1:5
t =
1    2    3    4    5
```

The components can change by non-unit steps:

```
>> x = 0:.1:1
x =
Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
Columns 8 through 11
 0.7000    0.8000    0.9000    1.0000

>> linspace(0,1,11)
```

```
>> eye(2,3)
        ans =
            1    0    0
            0    1    0

>> diag(2,3)
        ans =
            0    0    0    2
            0    0    0    0
            0    0    0    0
            0    0    0    0

>> diag(2)
        ans =
            2
```

5. formatting display

```
format
format short 3.1416
format short e 3.1416e+00
format long 3.14159265358979
format long e 3.141592653589793e+00
```

6. Conditionals and loops

Matlab has a standard if-elseif-else conditional; for example:

```
>> t = rand(1);
>>      if t > 0.75
                s = 0;
        elseif t < 0.25
                s = 1;
        else
                s = 1-2*(t-0.25);
        end
>> s
        s =
        0
>> t
        t =
        0.7622
```

The logical operators in Matlab are <, >, <=, >=, == (logical equals), and ~= (not equal). These are binary operators which return the values 0 and 1 (for scalar arguments):

```
>> 5>3
ans =
        1
>> 5<3
ans =
```

```
        0
>> 5==3
ans =
        0
```

Thus the general form of the if statement is

```
        if expr1
                statements
        elseif expr2
                statements
         .
         .
         .
        else
                statements
        end
```

Matlab provides two types of loops, a for-loop and a while-loop. A for-loop repeats the statements in the loop as the loop index takes on the values in a given row vector:

Syntax:

```
        FOR J= 1: N,
                FOR I= 1: N,
                        A(I,J) = 1/(I+J-1);
                END
        END
```

Example

```
>> for i=1:4
        disp(i^2)
    end
        1
        4
        9
        16
```

Syntax:

```
        WHILE expression
                statements
        END
```

Example

The while-loop repeats as long as the given expression is true (nonzero):

```
        >> x=1;
        >> while 1+x > 1
                x = x/2;
            end
        >> x
        x =
        1.1102e-16
```

Graphics

MATLAB can produce planar plots, images, and 3-D mesh surface plots.

Planar plots

The **plot** command creates linear x-y plots; if *x* and *y* are vectors of the same length, the command **plot(*x,y*)** opens a graphics window and draws an x-y plot of the elements of *x* versus the elements of *y*. You can, for example, draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
x = -4:.01:4;
y = sin(x);
plot(x,y)
```

As a second example, we will draw the graph of a exponential function:

```
x = -1.5:.01:1.5;
y = exp(-x.^2);
plot(x,y)
```

Plots of parametrically defined curves can also be made, for example.

```
t=0:.001:2*pi;
x=cos(3*t);
 y=sin(2*t);
plot(x,y);
grid;
```

The command **grid** will place grid lines on the current graph. The graphs can be given titles, axes labeled and text placed within the graph with the following commands which take a string as an argument.

title	graph title
xlabel	x-axis label
ylabel	y-axis label
gtext	interactively-positioned text
text	position text at specific coordinates

For example, the command title('Best Least Squares Fit') gives the graph a title. The command **gtext('The Spot')** allows a mouse or the arrow keys to position a crosshair on the graph, at which the text will be placed when any key is pressed.

By default, the axes are auto-scaled. This can be overridden by the command **axis**. If c = [*xmin,xmax,ymin,ymax*] is a 4-element vector, then **axis(*c*)** sets the axis scaling to the prescribed limits. By itself, **axis** freezes the current scaling for subsequent graphs; entering **axis** again returns to auto-scaling. The command **axis('square')** ensures that the same scale is used on both axes. Two ways to make multiple plots on a single graph are illustrated by

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,y2,y3)
```

By forming a matrix *Y* containing the functional values as columns

```
x=0:.01:2*pi;
Y=[sin(x)', sin(2*x)', sin(4*x)'];
plot(x,Y)
```

Another way is with the **hold** command. The command **hold** freezes the current graphics screen so that subsequent plots are superimposed on it. Entering **hold** again releases the "hold". The commands **hold on** and **hold off** are also available.

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1) ;
hold
plot(y1,x)
```

One can override the default linetypes and pointtypes. For example,

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
 y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
```

Renders a dashed line and dotted line for the first two graphs while for the third the symbol + is placed at each node. The line- and mark-type are:

- **Line types**: dashed(--), dotted(:), dash dot(-.), and the default solid(-)
- **Mark types**: point(.), plus(+), star(*), circle(o), x-mark(x), square(s), diamond(d), up-triangle(v), down-triangle(^), left-triangle(<), right-triangle(>), pentagram(p), hexagram(h)
- **Colors**: yellow(y), magenta(m), cyan(c), red(r), green(g), white(w), black(k), and the default blue(b)

The command **subplot** can be used to partition the screen so that up to four plots can be viewed simultaneously.

Application Examples

Example1:

```
L=0:0.1:60;
m=0.25;
Po=0.01.*exp(-m.*L/10);
plot(L,Po)
grid on
xlabel('Distance(Km)')
ylabel('power(w)')
```

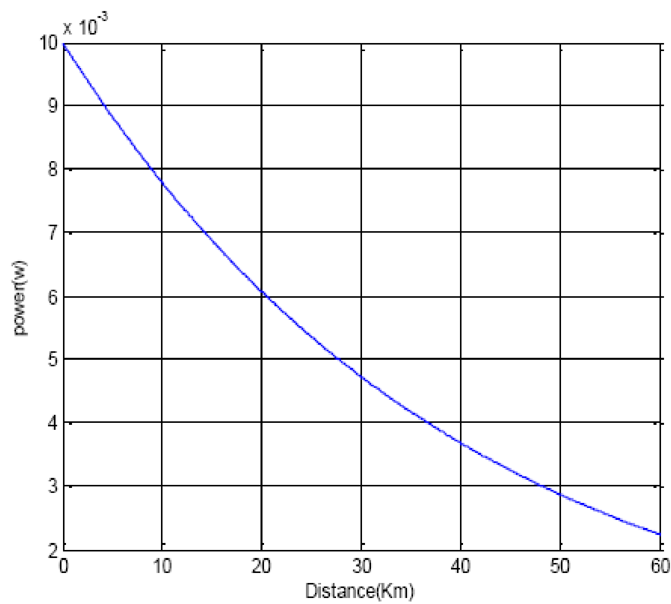
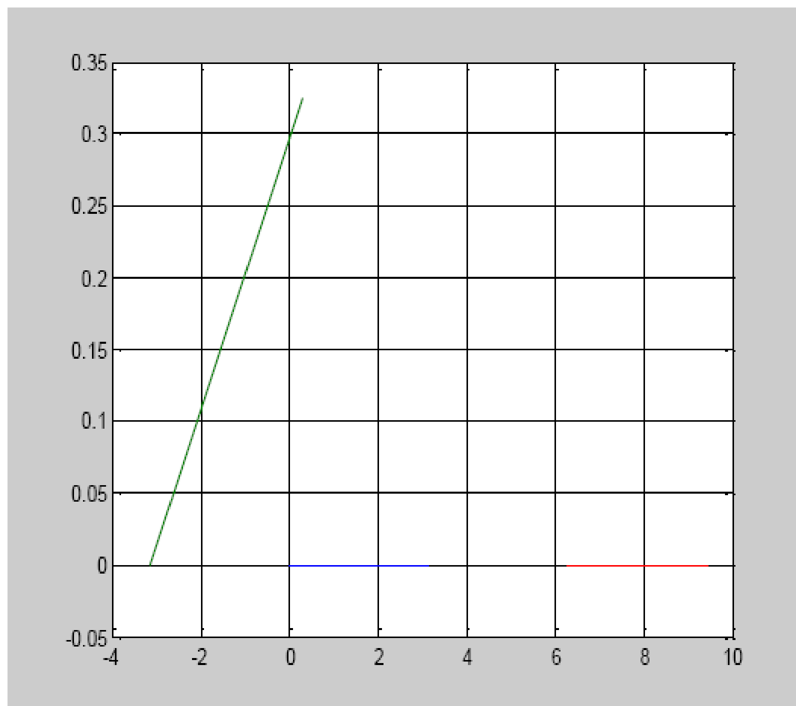


Figure1-2 plotting of example1

Example2:

```
x=[0 pi/10 2*pi; pi -pi 3*pi]
y=tan(x)
plot(x,y)
grid on
```



7. Scripts and functions

A script is simply a collection of Matlab commands in an m-file (a text file whose name ends in the extension ``.m"). Upon typing the name of the file (without the extension), those commands are executed as if they had been entered at the keyboard. The m-file must be located in one of the directories in which Matlab automatically looks for m-files; a list of these directories can be obtained by the command path. One of the directories in which Matlab always looks is the current working directory; the command cd identifies the current working directory, and cd newdir changes the working directory to newdir.

For example, suppose that

```
x = 0:2*pi/N:2*pi;
y = sin(w*x);
plot(x,y)
```

Then the sequence of commands

```
>> N=100;w=5;
>> plotsin
```

Produces Figure 3.

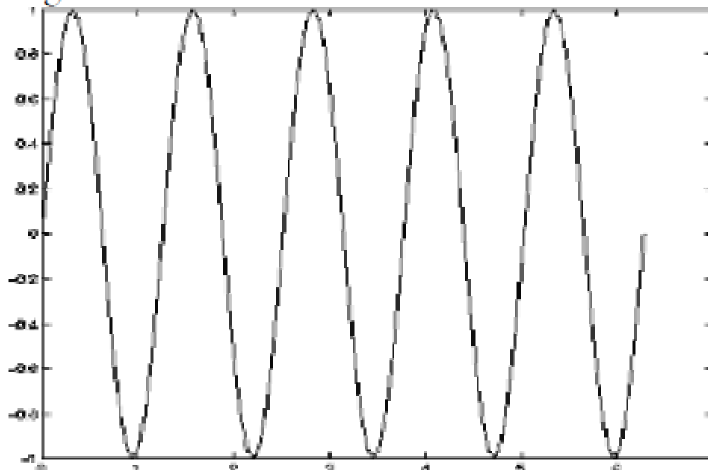


Figure 3: Effect of an m-file

As this example shows, the commands in the script can refer to the variables already defined in Matlab, which are said to be in the global workspace (notice the reference to N and w in plotsin.m). Much more powerful than scripts are functions, which allow the user to create new Matlab commands. A function is defined in an m-file that begins with a line of the following form:

function [output1,output2,...] = command_ name(input1,input2,...)

The rest of the m-file consists of ordinary Matlab commands computing the values of the outputs and performing other desired actions. It is important to note that when a function is invoked, Matlab creates a local workspace. The commands in the function cannot refer to variables from the global (interactive) workspace unless they are passed as inputs. By the same token, variables created as the function executes are erased when the execution of the function ends, unless they are passed back as outputs.

Here is a simple example of a function; it computes the function $f(x) = \min(x^2)$. The following commands should be stored in the file fcn.m (the name of the function within Matlab is the name of the m-file, without the extension):

```
function y = fcn(x)
y = sin(x.^2);
```

(Note that I used the vectorized operator. ^ so that the function fcn is also vectorized.) With this function defined, I can now use fcn just as the built-in function sin:

```
>> x = (-pi:2*pi/100:pi)';
>> y = sin(x);
>> z = fcn(x);
>> plot(x,y,x,z)
>> grid
```

The graph is shown in Figure 4. Notice how plot can be used to graph two (or more) functions together. The computer will display the curves with different line types--different colors on a color monitor, or different styles (e.g. solid versus dashed) on a black-and-white monitor. See help plot for more information.

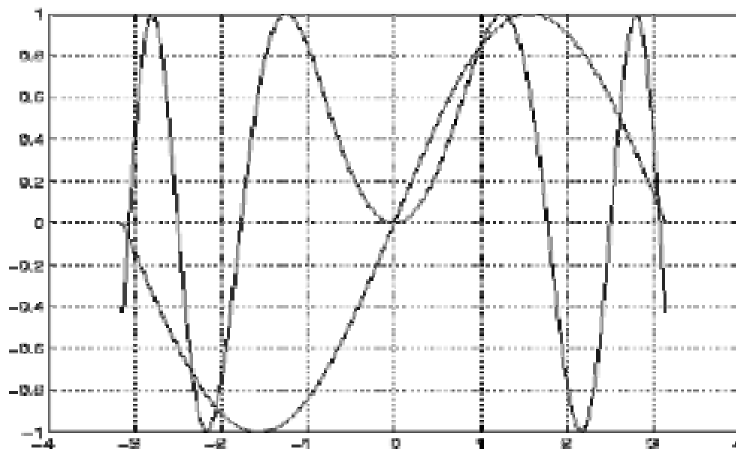


Figure 4: Two curves graphed together

Notice from Figure 4 that $f(x) = \min(x^2)$ has a root between 1 and 2 (of course, this root is).

8. Putting several graphs in one window

The subplot command creates several plots in a single window. To be precise, subplot (m,n,i) creates mn plots, arranged in an array with m rows and n columns. It also sets the next plot command to go to the ith coordinate system (counting across the rows). Here is an example (see Figure 5):

```
>> t = (0:.1:2*pi)';
>> subplot(2,2,1)
>> plot(t,sin(t))
>> subplot(2,2,2)
```

```
>> plot(t,cos(t))
>> subplot(2,2,3)
>> plot(t,exp(t))
>> subplot(2,2,4)
>> plot(t,1./(1+t.^2))
```

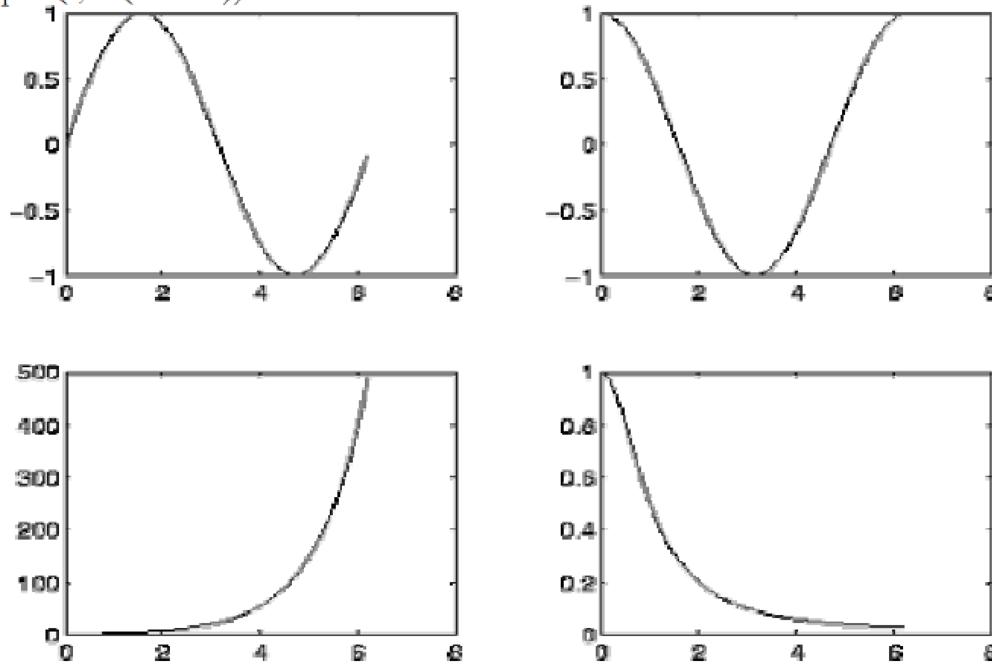


Figure 5: Using the subplot command

9. 3D plots

In order to create a graph of a surface in 3-space (or a contour plot of a surface), it is necessary to evaluate the function on a regular rectangular grid. This can be done using the meshgrid command. First, create 1D vectors describing the grids in the x- and y-directions:

```
>> x = (0:2*pi/20:2*pi)';
>> y = (0:4*pi/40:4*pi)';
```

Next, ``spread'' these grids into two dimensions using meshgrid:

```
>> [X,Y] = meshgrid(x,y);
```

The effect of meshgrid is to create a vector X with the x-grid along each row, and a vector Y with the y-grid along each column. Then, using vectorized functions and/or operators, it is easy to evaluate a function z = f(x,y) of two variables on the rectangular grid:

```
>> z = cos(X).*cos(2*Y);
```

Having created the matrix containing the samples of the function, the surface can be graphed using either the mesh or the surf commands (see Figures 6 and 7, respectively):

```
>> mesh(x,y,z)
>> surf(x,y,z)
```

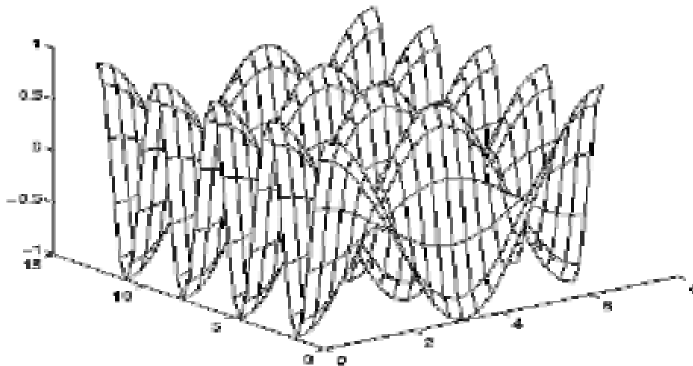


Figure 6: Using the mesh command

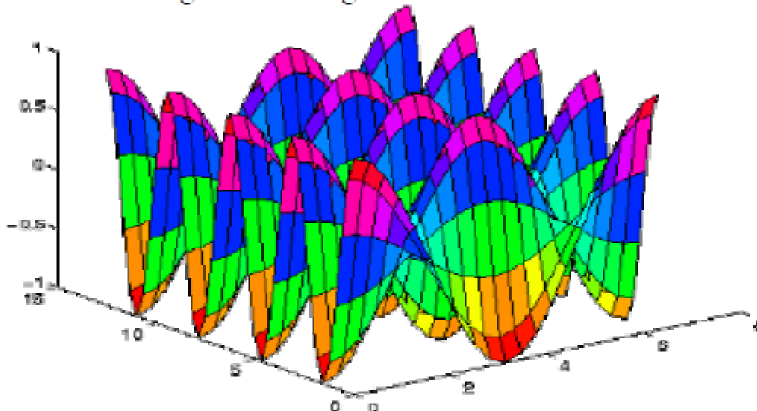


Figure 7: Using the surf command

In addition, a contour plot can be created (see Figure 8):

```
>> contour(x,y,z)
```

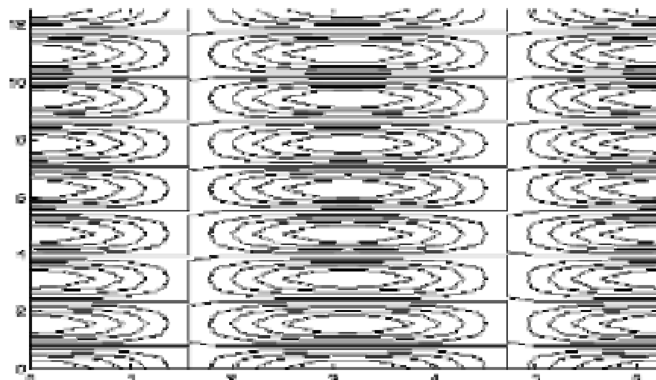


Figure 8: Using the contour command

Example

```
>> subplot(2,2,1)
>> mesh(x,y,z)
>> subplot(2,2,2)
>> surf(x,y,z)
>> subplot(2,2,3)
>> surfc(x,y,z)
>> subplot(2,2,4)
>> contour(x,y,z)
```

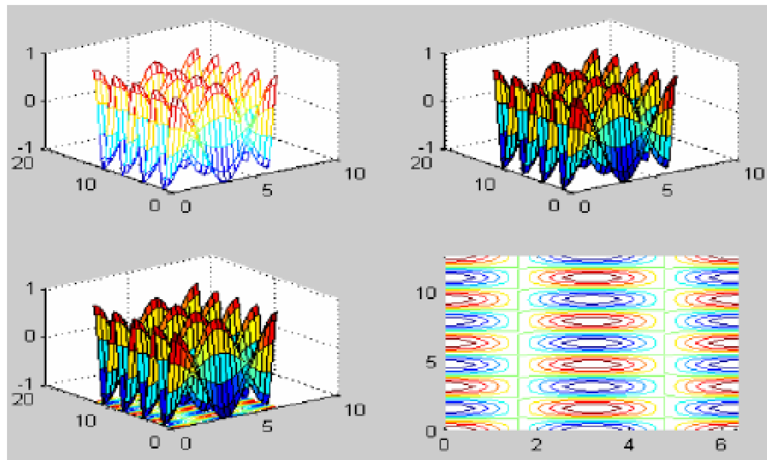


Figure 9: surface plots with mesh, surf, meshc, and contour

Example

```
>> subplot(2,2,1)
>> hist(x)
>> subplot(2,2,2)
>> bar(x)
>> subplot(2,2,3)
>> pie(x)
>> subplot(2,2,4)
>> waterfall(x)
```

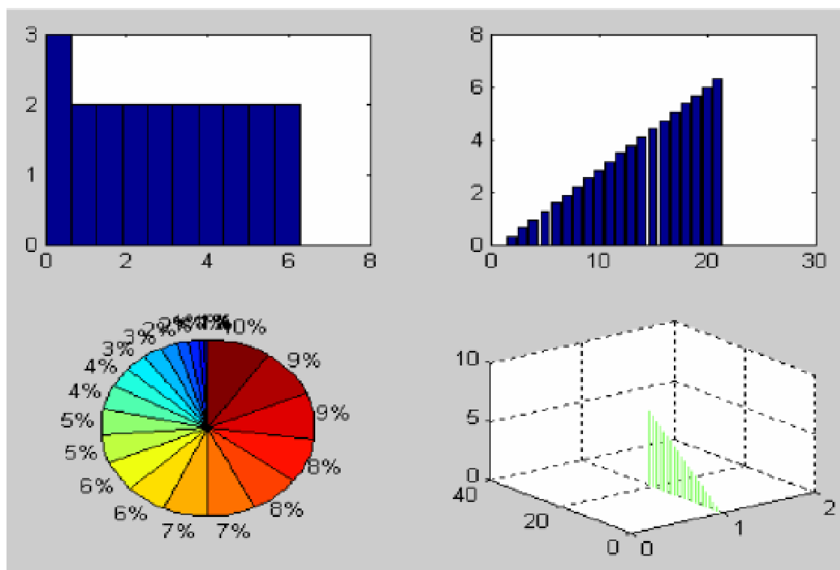


Figure 10: plotting by using hist, bar, pie, and waterfall

STEM Function

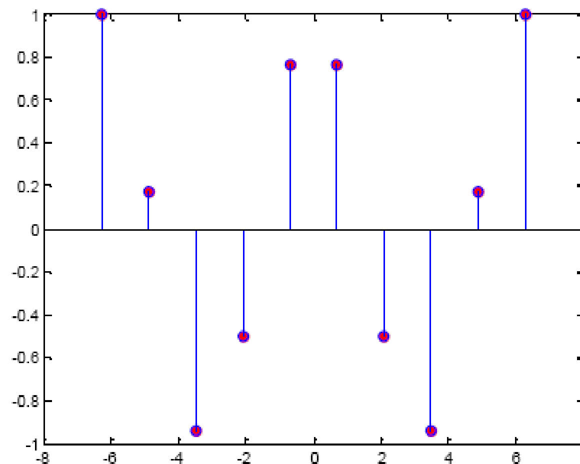
STEM Discrete sequence or "stem" plot. STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. If Y is a matrix then each column is plotted as a separate series STEM(X,Y) plots the data sequence Y at the values specified in X. STEM(...,'filled') produces a stem plot with filled markers. STEM(...,'LINESPEC') uses the linetype specified for the stems and markers. See PLOT for possibilities. STEM(AX,...) plots into axes with handle AX. Use GCA to get the handle to the current axes or to create one if none exist. H = STEM(...) returns a vector of stem series handles in H, one handle per column of data in Y.

Examples

Single Series of Data

This example creates a stem plot representing the cosine of 10 values linearly spaced between 0 and 2π. Note that the line style of the baseline is set by first getting its handle from the stemseries object's BaseLine property.

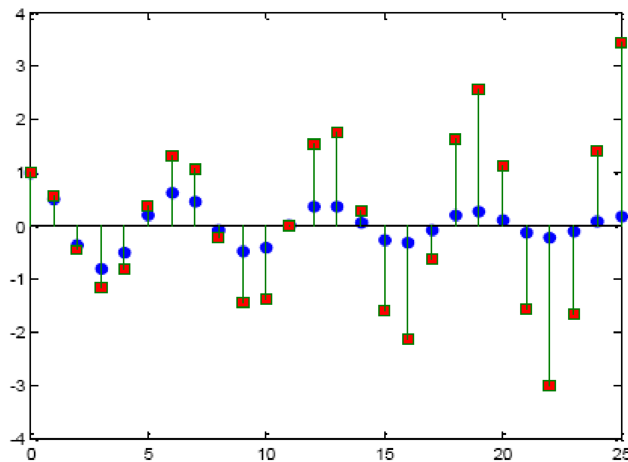
```
t = linspace(-2*pi,2*pi,10);
h = stem(t,cos(t),'fill','--');
set(get(h,'BaseLine'),'LineStyle',':')
set(h,'MarkerFaceColor','red')
```



Two Series of Data on One Graph

The following example creates a stem plot from a two-column matrix. In this case, the stem function creates two stemseries objects, one of each column of data. Both objects' handles are returned in the output argument h.

- h(1) is the handle to the stemseries object plotting the expression exp(-.07*x).*cos(x).
- h(2) is the handle to the stemseries object plotting the expression exp(.05*x).*cos(x).
- x = 0:25;
- y = [exp(-.07*x).*cos(x);exp(.05*x).*cos(x)]';
- h = stem(x,y);
- set(h(1),'MarkerFaceColor','blue')
- set(h(2),'MarkerFaceColor','red','Marker','square')



Three-Dimensional Stem Plots

stem3 displays 3-D stem plots extending from the *xy*-plane. With only one vector argument, the stems are plotted in one row at x = 1 or y = 1, depending on whether the argument is a column or row vector. stem3 is intended to display data that you cannot visualize in a 2-D view.

Example — 3-D Stem Plot of an FFT

Fast Fourier transforms are calculated at points around the unit circle on the complex plane. It is interesting to visualize the plot around the unit circle. Calculating the unit circle

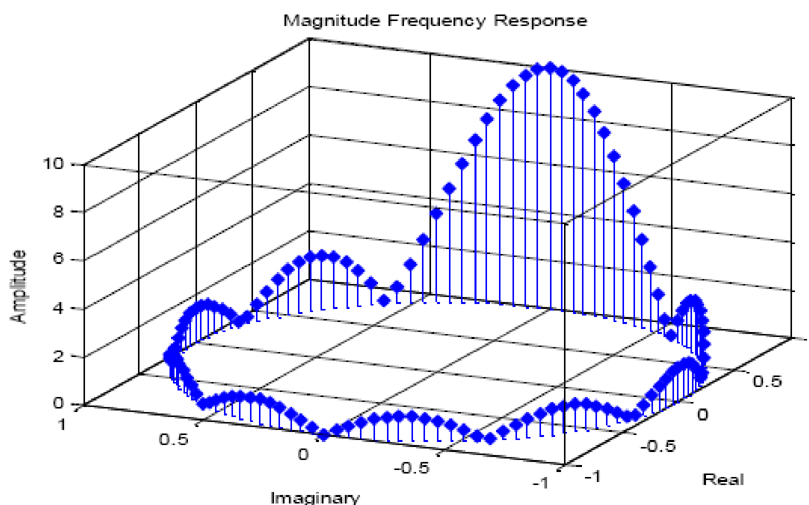
```
th = (0:127)/128*2*pi;
x = cos(th);
y = sin(th);
```

and the magnitude frequency response of a step function. The command

```
f = abs(fft(ones(10,1),128));
```

displays the data using a 3-D stem plot, terminating the stems with filled diamond markers:

```
stem3(x,y,f,'d','fill')
view([-65 30])
```



Fourier Transform

A theorem of mathematics says, roughly, that any function can be represented as a sum of sinusoids of different amplitudes and frequencies. The Fourier transform is the mathematical technique of finding the amplitudes and frequencies of those sinusoids. The Discrete Fourier Transform (DFT) is an algorithm that calculates the Fourier transform for numerical data. The Fast Fourier Transform is an efficient implementation of the DFT. The following functions are available in mat lab to do Fourier transforms and related operations:

fft	One-dimensional fast Fourier transform
fft2	Two-dimensional fast Fourier transform
fftn	N-dimensional fast Fourier transform
fftshift	Move zeroth lag to centre of transform
ifft	Inverse one-dimensional fast Fourier transform
ifft2	Inverse two-dimensional fast Fourier transform
ifftn	inverse N-dimensional fast Fourier transform
abs	Absolute value (complex magnitude)
angle	Phase angle
cplxpair	Sort complex numbers into complex conjugate pairs
nextpow2	Next power of two
unwrap	Correct phase angles

The FFT of the column vector

```
y = [2 0 1 0 2 1 1 0]';
```

is

```
>> Y = fft(y)
Y  =
   7.0000
  -0.7071+ 0.7071i
   2.0000- 1.0000i
   0.7071+ 0.7071i
   5.0000
   0.7071- 0.7071i
   2.0000+ 1.0000i
  -0.7071- 0.7071i
```

The first value of **Y** is the sum of the elements of **y**, and is the amplitude of the "zero-frequency", or constant, component of the Fourier series. Terms 2 to 4 are the (complex) amplitudes of the positive frequency Fourier components. Term 5 is the amplitude of the component at the Nyquist frequency, which is half the sampling frequency. The last three terms are the negative frequency components, which, for real signals, are complex conjugates of the positive frequency components.

The **fftshift** function rearranges a Fourier transform so that the negative and positive frequencies lie either side of the zero frequency.

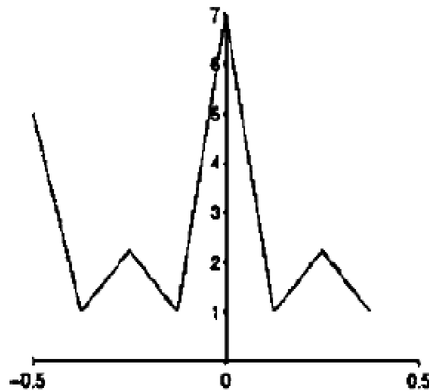
Companion M-Files Feature 4 *The function* ***fftfreq*** *gives you a two-sided frequency vector for use with* ***fft*** *and* ***fftshift****. For example, the frequency vector corresponding to an 8-point FFT assuming a Nyquist frequency of 0.5 is*

```
>> fftfreq(.5,8)'
ans  =
```

```
  -0.5000
  -0.3750
  -0.2500
  -0.1250
        0
   0.1250
   0.2500
   0.3750
```

We combine **fftshift** and **fftfreq** to plot the two-sided FFT:

```
plot(fftfreq(.5,8),fftshift(abs(Y)))
axis([-.5 .5 0 7])
zeroaxes
```



Let us do a slightly more realistic example. We simulate some data recorded at a sampling frequency of 1 kHz, corresponding to a time step `dt = 1/1000` of a second. The Nyquist frequency is, therefore, 500 Hz. Suppose there is a 100 Hz sinusoid contaminated by noise. We simulate the data, calculate the FFT, and plot the results as follows:

[3].

```
dt = 1/1000;
t = dt:dt:200*dt;
sine = sin(2*pi*100*t);
y = sine + randn(size(t));
Y = fft(y);
f = fftfreq(500,length(Y));
```

fourier

Fourier integral transform

Syntax

F = fourier(f)
F = fourier(f,v)
F = fourier(f,u,v)

Description

F = fourier(f) is the Fourier transform of the symbolic scalar f with default independent variable x. The default return is a function of w. The Fourier transform is applied to a function of x and returns a function of w.

If f = f(w), fourier returns a function of t.

Laplace transform

laplace(F)
laplace(F, t)
laplace(F, w, z)

L = laplace(F) is the Laplace transform of the scalar symbol F with default independent variable t. The default return is a function of s. The Laplace transform is applied to a function of t and returns a function of s.

Example

```
syms a t;
f1=t^4;
f2=1/sqrt(t);
f3=exp(a*t)
L1 = laplace(f1)
L2 = laplace(f2)
L3= laplace(f3)
```

Bessel Functions

BESSEL Bessel functions of various kinds.
Bessel functions are solutions to Bessel's differential equation of order NU:

$$x^2 * y'' + x * y' + (x^2 - nu^2) * y = 0$$

There are several functions available to produce solutions to Bessel's equations. These are:

```
BESSELJ(NU,Z)    Bessel function of the first kind
BESSELY(NU,Z)    Bessel function of the second kind
BESSELI(NU,Z)    Modified Bessel function of the first kind
BESSELK(NU,Z)    Modified Bessel function of the second kind
BESSELH(NU,K,Z)  Hankel function
```

Examples

Example 1

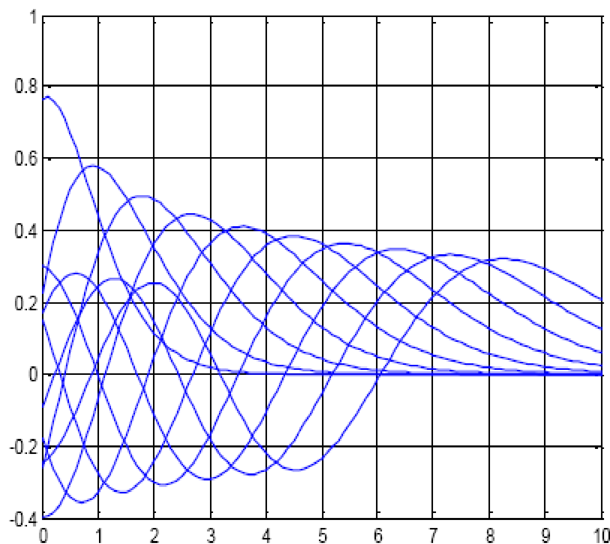
```
format long
z = (0:0.2:1)';
bessely(1,z)

ans =
              -Inf
  -3.32382498811185
  -1.78087204427005
  -1.26039134717739
  -0.97814417668336
  -0.78121282130029
```

Example2

```
format long
z = (0:0.2:1)';
besseli(1,z)
```

Examples 3

```
x=0:0.1:10;
for n=1:10
   y=besselj(x,n);
   plot(x,y)
   grid on
   hold on
end
```



Integration

INT Integrate

INT(S) is the indefinite integral of S with respect to its symbolic variable as defined by SYMVAR. S is a SYM (matrix or scalar). If S is a constant, the integral is with respect to 'x'. INT(S,v) is the indefinite integral of S with respect to v. v is aalar SYM.

INT(S,a,b) is the definite integral of S with respect to its symbolic variable from a to b. a and b are each double or symbolic scalars. INT(S,v,a,b) is the definite integral of S with respect to v from a to b.

Examples:

```
syms x x1 alpha u t;
A = [cos(x*t),sin(x*t);-sin(x*t),cos(x*t)];
int(1/(1+x^2))                returns   atan(x)
int(sin(alpha*u),alpha)       returns   -cos(alpha*u)/u
int(besselj(1,x),x)           returns   -besselj(0,x)
int(x1*log(1+x1),0,1)         returns   1/4
int(4*x*t,x,2,sin(t))         returns   -2*t*cos(t)^2 - 6*t
int([exp(t),exp(alpha*t)])    returns [ exp(t), exp(alpha*t)/alpha]
int(A,t)                      returns   [sin(t*x)/x, -cos(t*x)/x]
                                        [cos(t*x)/x,  sin(t*x)/x]
```

DIFF Difference and approximate derivative.

DIFF(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].

DIFF(X), for a matrix X, is the matrix of row differences,

[X(2:n,:) - X(1:n-1,:)].

DIFF(X), for an N-D array X, is the difference along the first non-singleton dimension of X.

DIFF(X,N) is the N-th order difference along the first non-singleton dimension (denote it by DIM).

If N >= size(X,DIM), DIFF takes successive differences along the next non-singleton dimension.

DIFF(X,N,DIM) is the Nth difference function along dimension DIM.

If N >= size(X,DIM), DIFF returns an empty array.

Examples:

```
h = .001; x = 0:h:pi;
diff(sin(x.^2))/h is an approximation to 2*cos(x.^2).*x
diff((1:10).^2) is 3:2:19
If X = [3 7 5 0 9 2]
then diff(X,1,1) is [-3 2 -3], diff(X,1,2) is [4 -2 9 -7],
diff(X,2,2) is the 2nd order difference along the dimension 2,
and diff(X,3,2) is the empty matrix.
```

Simulink

Introduction

Simulink is a software package that enables you to model, simulate, and analyze systems whose outputs change over time. Such systems are often referred to as dynamic systems. The Simulink software can be used to explore the behavior of a wide range of real-world dynamic systems, including electrical circuits, shock absorbers, braking systems, and many other electrical, mechanical, and thermodynamic systems. This section explains how Simulink works.

Simulating a dynamic system is a two-step process. First, a user creates a block diagram, using the Simulink model editor, that graphically depicts time-dependent mathematical relationships among the system's inputs, states, and outputs. The user then commands the Simulink software to simulate the system represented by the model from a specified start time to a specified stop time.

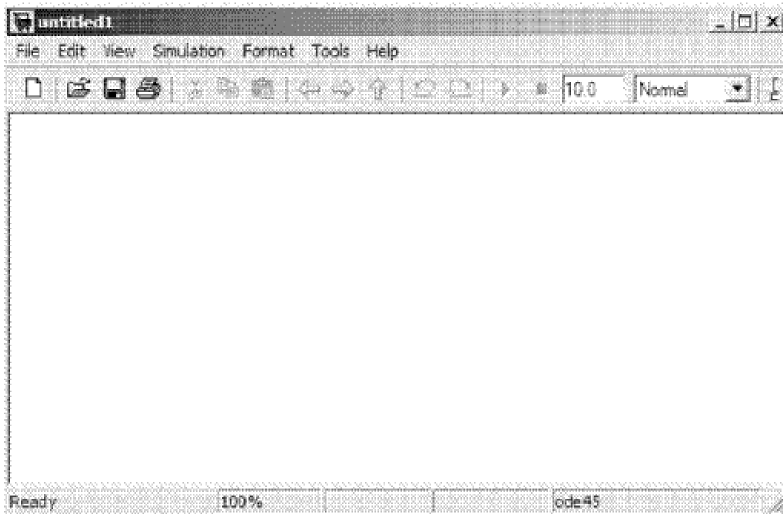
Block Diagram Semantics

A classic block diagram model of a dynamic system graphically consists of blocks and lines (signals). The history of these block diagram models is derived from engineering areas such as Feedback Control Theory and Signal Processing. A block within a block diagram defines a dynamic system in itself. The relationships between each elementary dynamic system in a block diagram are illustrated by the use of signals connecting the blocks. Collectively the blocks and lines in a block diagram describe an overall dynamic system.

The Simulink product extends these classic block diagram models by introducing the notion of two classes of blocks, nonvirtual blocks and virtual blocks. Nonvirtual blocks represent elementary systems. Virtual blocks exist for graphical and organizational convenience only: they have no effect on the system of equations described by the block diagram model. You can use virtual blocks to improve the readability of your models.

Creating an Empty Model

To create an empty model, click the **New** button on the Library Browser's toolbar, or choose **New** from the library window's **File** menu and select **Model**. An empty model is created in memory and it is displayed in a new model editor window.



Creating a Model Template

When you create a model, Simulink uses defaults for many configuration parameters. For example, by default new models have a white canvas, the `ode45` solver, and a visible toolbar. If these or other defaults do not meet your needs, you can use the Simulink software model construction commands described in Model Construction to write a function that creates a model with the defaults you prefer. For example, the following function creates a model that has a green canvas and a hidden toolbar, and uses the `ode3` solver:

```
function new_model(modelname)
% NEW_MODEL Create a new, empty Simulink model
%    NEW_MODEL('MODELNAME') creates a new model with
%    the name 'MODELNAME'. Without the 'MODELNAME'
%    argument, the new model is named 'my_untitled'.

if nargin == 0
     modelname = 'my_untitled';
end

% create and open the model
open_system(new_system(modelname));

% set default screen color
set_param(modelname, 'ScreenColor', 'green');

% set default solver
set_param(modelname, 'Solver', 'ode3');

% set default toolbar visibility
set_param(modelname, 'Toolbar', 'off');

% save the model
save_system(modelname);
```

Selecting Objects

On this page…

Selecting an Object
Selecting Multiple Objects

Selecting an Object

To select an object, click it. Small black square handles appear at the corners of a selected block and near the end points of a selected line. For example, the figure below shows a selected Sine Wave block and a selected line.

When you select an object by clicking it, any other selected objects are deselected.

Back to Top

Selecting Multiple Objects

You can select more than one object either by selecting objects one at a time, by selecting objects located near each other using a bounding box, or by selecting the entire model.

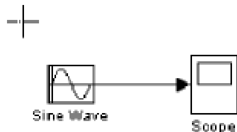
Selecting Multiple Objects One at a Time

To select more than one object by selecting each object individually, hold down the **Shift** key and click each object to be selected. To deselect a selected object, click the object again while holding down the **Shift** key.

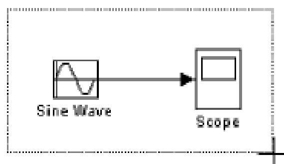
Selecting Multiple Objects Using a Bounding Box

An easy way to select more than one object in the same area of the window is to draw a bounding box around the objects:

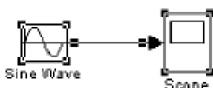
1. Define the starting corner of a bounding box by positioning the pointer at one corner of the box, then pressing and holding down the mouse button. Notice the shape of the cursor.



2. Drag the pointer to the opposite corner of the box. A dotted rectangle encloses the selected blocks and lines.



3. Release the mouse button. All blocks and lines at least partially enclosed by the bounding box are selected.

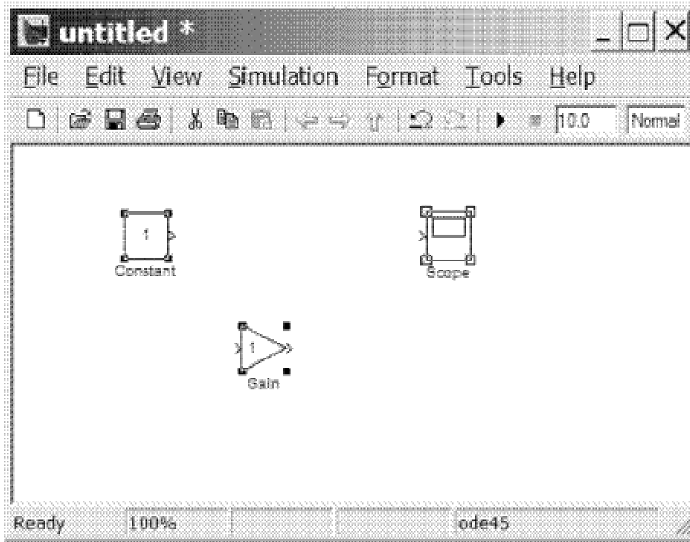


Selecting All Objects

Aligning, Distributing, and Resizing Groups of Blocks Automatically

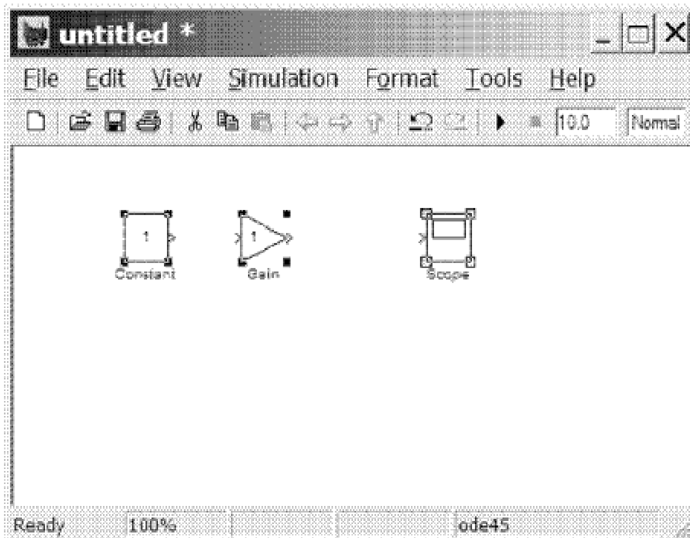
The model editor's **Format** menu includes commands that let you quickly align, distribute, and resize groups of blocks. To align (or distribute or resize) a group of blocks:

1. Select the blocks that you want to align.



One of the selected blocks displays empty selection handles. The model editor uses this block as the reference for aligning the other selected blocks. If you want another block to serve as the alignment reference, click that block.

2. Select one of the alignment options from the editor's **Format** > **Align Blocks** menu (or distribution options from the **Format** > **Distribute Blocks** or resize options from the **Format** > **Resize Blocks** menu). For example, selecting **Align Top Edges** aligns the top edges of the selected blocks with the top edge of the reference block.



Provide feedback about this page

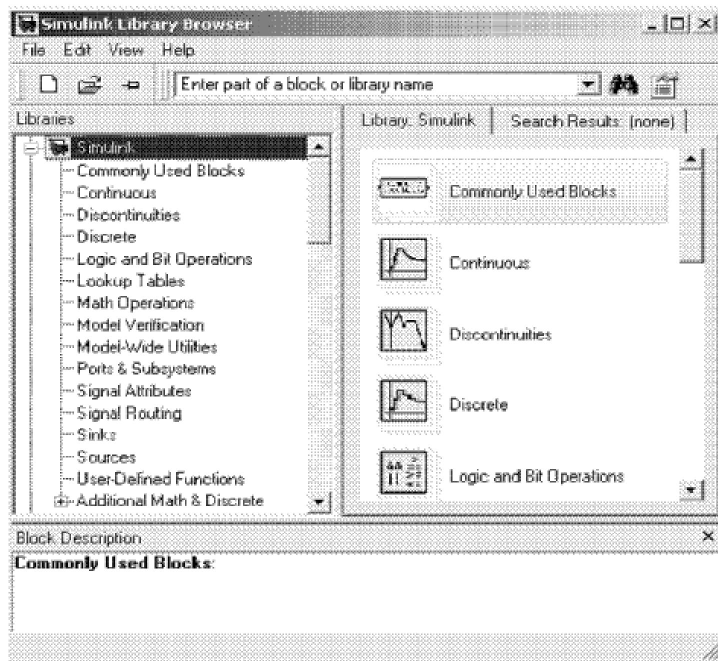
Starting Simulink Software

To start the Simulink software, you must first start the MATLAB® technical computing environment. Consult your MATLAB documentation for more information. You can then start the Simulink software in two ways:

- On the toolbar, click the Simulink icon.
- Enter the `simulink` command at the MATLAB prompt.

The Library Browser appears. It displays a tree-structured view of the Simulink block libraries installed on your system. You build models by copying blocks from the Library Browser into a model window (see Editing Blocks).

The Simulink library window displays icons representing the pre-installed block libraries. You can create models by copying blocks from the library into a model window.



> **Note** On computers running the Windows® operating system, you can display the Simulink library window by right-clicking the Simulink node in the Library Browser window.

Provide feedback about this page

Simulink Basics Opening a Model

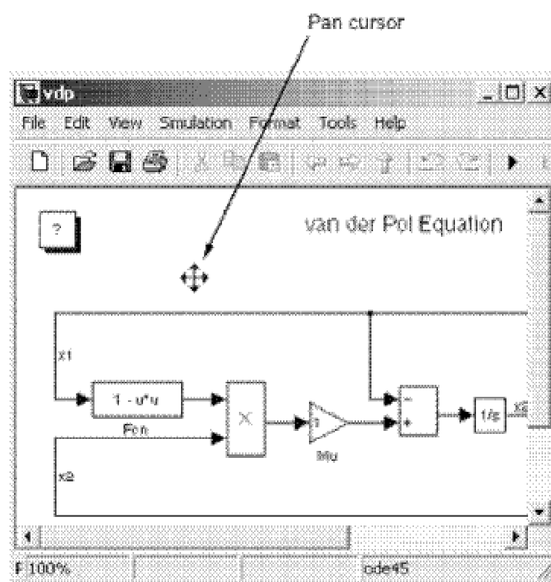
© 1984-2009 The MathWorks, Inc. · Terms of Use · Patents · Trademarks · Acknowledgments

Panning Block Diagrams

You can use your keyboard alone (see Model Viewing Shortcuts) or in combination with your mouse to pan model diagrams that are too large to fit in the Model Editor's window. To use the keyboard and the mouse, position the mouse over the diagram, hold down the **p** or **q** key on the keyboard, then hold down the left mouse button.

> **Note** You must press and hold down the key first and then the mouse button. The reverse does not work.

A pan cursor appears.



Moving the mouse now pans the model diagram in the editor window.

Provide feedback about this page

Zooming Block Diagrams Viewing Command History

© 1984-2009 The MathWorks, Inc. • Terms of Use • Patents • Trademarks • Acknowledgments

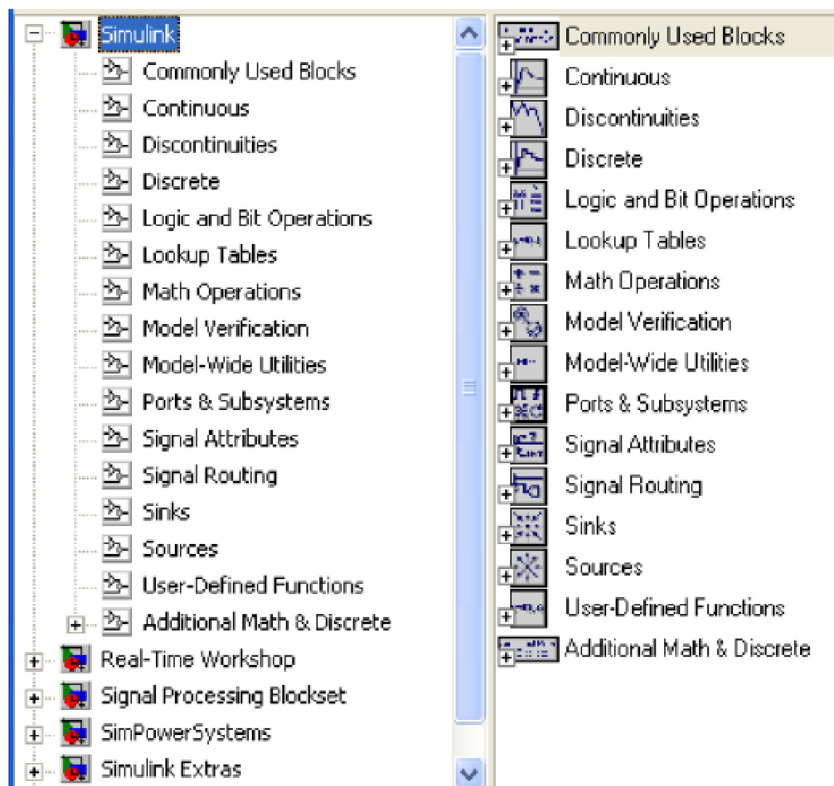


Figure 1.4. The Simulink Library Browser

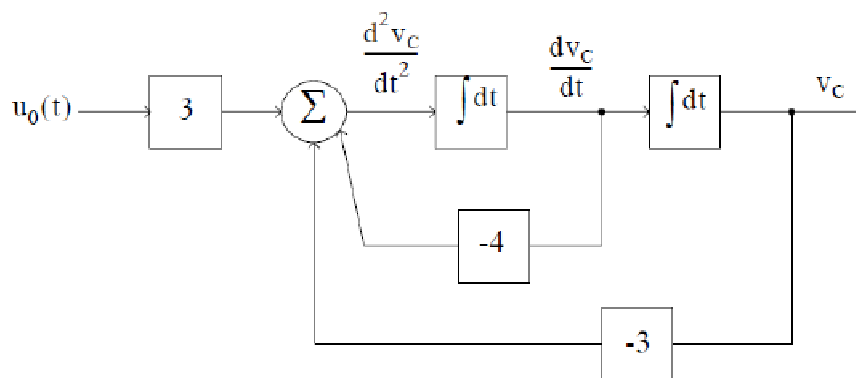
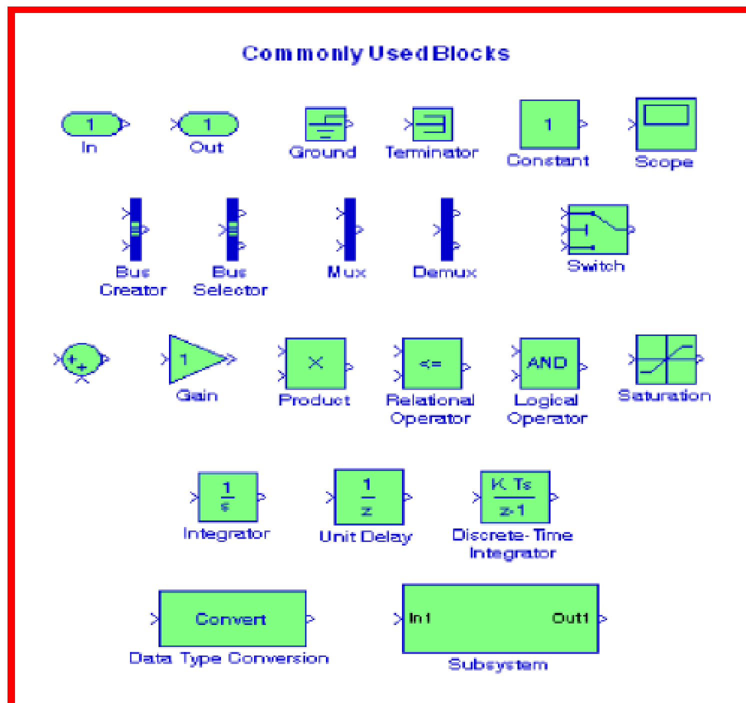


Figure 1.5. Block diagram for equation (1.26)

The Commonly Used Blocks Library

This is the first library in the Simulink group of libraries and contains the blocks shown below. In this chapter, we will describe the function of each block included in this library and we will perform simulation examples to illustrate their application [5].

.



Example 2.4

The model is shown in Figure 2.9 performs the multiplication $(3+j4)\times(4+j3)\times(5-j8)$. After the Start simulation command is executed, it may be necessary to stretch the Display block horizontally to read the result.

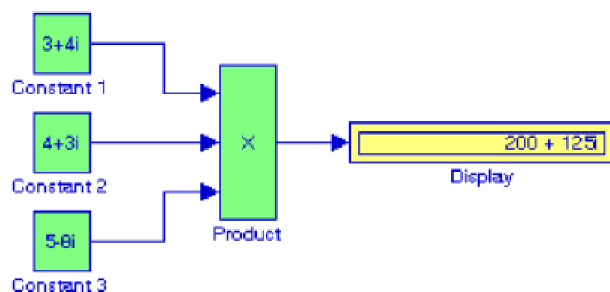


Figure 2.9. Model for Example 2.4

Example 2.5

The model is shown in Figure 2.10 performs the division $(3 + j4)/(4 + j3)$.

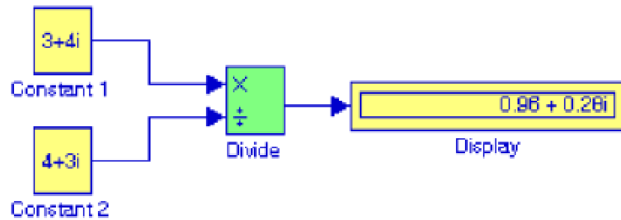


Figure 2.10. Model for Example 2.5

Example 2.6

The model of Figure 2.11 simulates the combined functions $\sin 2t$, $\frac{d}{dt}\sin 2t$, and $\int \sin 2t dt$ into a bus and displays all three on a single scope.

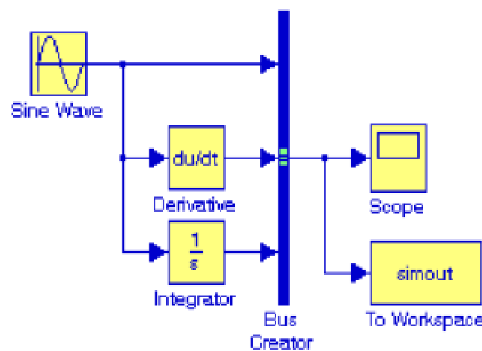


Figure 2.11. Model for Example 2.6

Example 2.14

The model of Figure 2.28 simulates the differential equation

$$\frac{d^2 v_C}{dt^2} + 4\frac{dv_C}{dt} + 3v_C = 3u_0(t)$$

subject to the initial conditions $v_C(0) = 0.5$ and $v'_C(0) = 0$.

The Constant 1 and Constant 2 blocks represent the initial conditions.

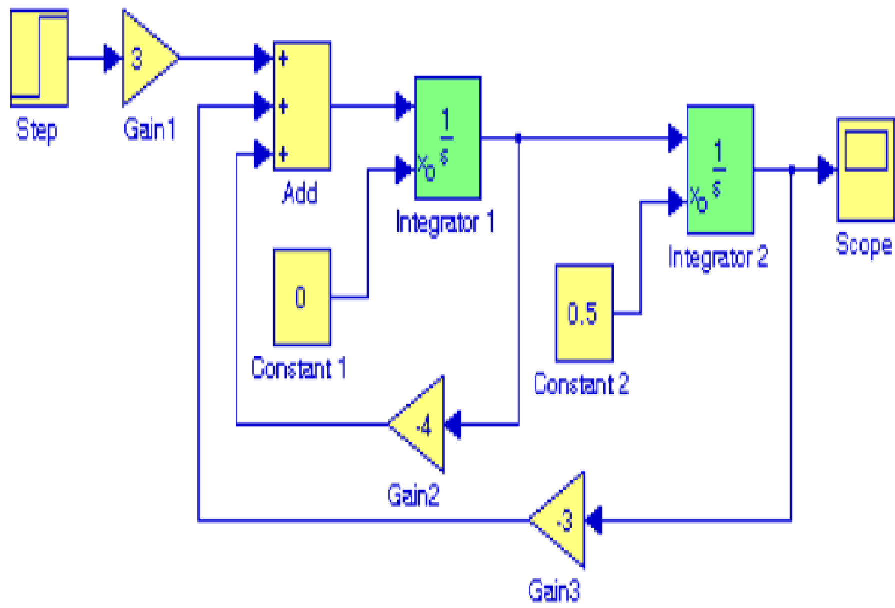
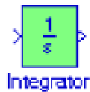


Figure 2.28. Model for Example 2.14

3.1 The Continuous–Time Linear Systems Sub–Library

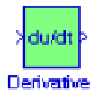
The **Continuous–Time Linear Systems Sub–Library** contains the blocks described in Subsections 3.1.1 through 3.1.5 below.

3.1.1 The Integrator Block



The **Integrator** block is described in Section 2.14, Chapter 2, Page 2–20.

3.1.2 The Derivative Block



The **Derivative block** approximates the derivative of its input. The initial output for the block is zero. The accuracy of the results depends on the size of the time steps taken in the simulation. Smaller steps allow a smoother and more accurate output curve from this block. Unlike blocks that have continuous states, the solver does not take smaller steps when the input changes rapidly. Let us consider the following simple example.

Example 3.1

We will create a model that will compute and display the waveform of the derivative of the function $y = \cos x$.

The model is shown in Figure 3.1, and the input and output waveforms are shown in Figure 3.2.

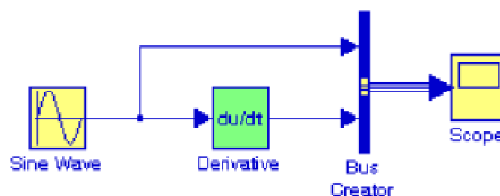


Figure 3.1. Model for Example 3.1

To convert the sine function in the Sine Wave block to a cosine function, in the Source Block Parameters dialog box we specify Phase $= \pi/2$. As we know, the derivative of the cosine function is the negative of the sine function and this is shown in Figure 3.2. [5]

Applications

The Series RLC Circuit with DC Excitation

Let us consider the series RLC circuit of Figure 1.1 where the initial conditions are $i_L(0) = I_0$, $v_C(0) = V_0$, and $u_0(t)$ is the unit step function.* We want to find an expression for the current $i(t)$ for $t > 0$.

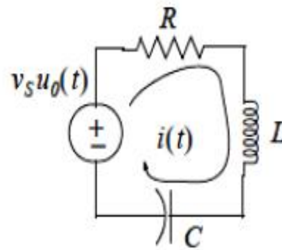


Figure 1.1. Series RLC Circuit

For this circuit

$$Ri + L\frac{di}{dt} + \frac{1}{C}\int_0^t i\,dt + V_0 = v_S \quad t > 0 \tag{1.1}$$

and by differentiation

$$R\frac{di}{dt} + L\frac{d^2 i}{dt^2} + \frac{i}{C} = \frac{dv_S}{dt}, \quad t > 0$$

To find the forced response, we must first specify the nature of the excitation v_S, that is, DC or AC. If v_S is DC (v_S=constant), the right side of (1.1) will be zero and thus the forced response component $i_f = 0$. If v_S is AC ($v_S = V\cos(\omega t + \theta)$, the right side of (1.1) will be another sinusoid and therefore $i_f = I\cos(\omega t + \varphi)$. Since in this section we are concerned with DC excitations, the right side will be zero and thus the total response will be just the natural response.

The natural response is found from the homogeneous equation of (1.1), that is,

$$R\frac{di}{dt} + L\frac{d^2 i}{dt^2} + \frac{i}{C} = 0 \tag{1.2}$$

The characteristic equation of (1.2) is

$$Ls^2 + Rs + \frac{1}{C} = 0$$

or

$$s^2 + \frac{R}{L}s + \frac{1}{LC} = 0$$

from which

$$s_1, s_2 = -\frac{R}{2L} \pm \sqrt{\frac{R^2}{4L^2} - \frac{1}{LC}} \quad (1.3)$$

We will use the following notations:

$$\underbrace{\alpha_S = \frac{R}{2L}}_{\alpha \text{ or Damping Coefficient}} \quad \underbrace{\omega_0 = \frac{1}{\sqrt{LC}}}_{\text{Resonant Frequency}} \quad \underbrace{\beta_S = \sqrt{\alpha_S^2 - \omega_0^2}}_{\text{Beta Coefficient}} \quad \underbrace{\omega_{nS} = \sqrt{\omega_0^2 - \alpha_S^2}}_{\text{Damped Natural Frequency}} \quad (1.4)$$

where the subscript s stands for series circuit. Then, we can express (1.3) as

$$s_1, s_2 = -\alpha_S \pm \sqrt{\alpha_S^2 - \omega_0^2} = -\alpha_S \pm \beta_S \quad if \quad \alpha_S^2 > \omega_0^2 \quad (1.5)$$

or

$$s_1, s_2 = -\alpha_S \pm \sqrt{\omega_0^2 - \alpha_S^2} = -\alpha_S \pm \omega_{nS} \quad if \quad \omega_0^2 > \alpha_S^2 \quad (1.6)$$

Case I: If $\alpha_S^2 > \omega_0^2$, the roots s_1 and s_2 are real, negative, and unequal. This results in the *over-damped* natural response and has the form

$$i_n(t) = k_1 e^{s_1 t} + k_2 e^{s_2 t} \quad (1.7)$$

Case II: If $\alpha_S^2 = \omega_0^2$, the roots s_1 and s_2 are real, negative, and equal. This results in the *critically damped* natural response and has the form

$$i_n(t) = e^{-\alpha_S t}(k_1 + k_2 t) \quad (1.8)$$

Case III: If $\omega_0^2 > \alpha_S^2$, the roots s_1 and s_2 are complex conjugates. This is known as the *underdamped or oscillatory* natural response and has the form

$$i_n(t) = e^{-\alpha_S t}(k_1 cos\omega_{nS} t + k_2 sin\omega_{nS} t) = k_3 e^{-\alpha_S t}(cos\omega_{nS} t + \varphi) \quad (1.9)$$

A typical overdamped response is shown in Figure 1.2 where it is assumed that $i_n(0) = 0$. This plot was created with the following MATLAB code:

```
t=0: 0.01: 6; ft=8.4.*(exp(−t)-exp(−6.*t)); plot(t,ft); grid; xlabel('t');...
ylabel('f(t)'); title('Overdamped Response for 4.8.*(exp(−t)−exp(−6.*t))')
```

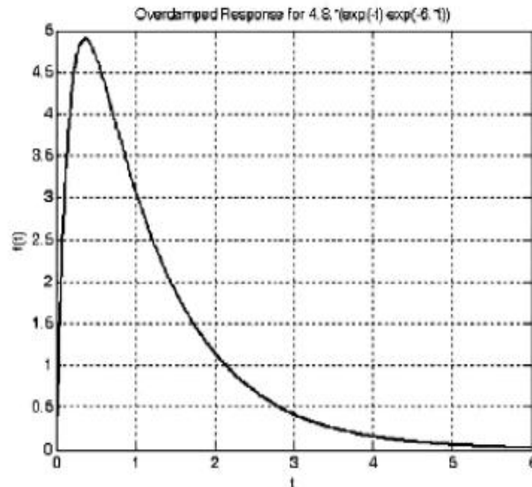


Figure 1.2. Typical overdamped response

A typical critically damped response is shown in Figure 1.3 where it is assumed that $i_n(0) = 0$. This plot was created with the following MATLAB code:

```
t=0: 0.01: 6; ft=420.*t.*(exp(−2.45.*t)); plot(t,ft); grid; xlabel('t');...
ylabel('f(t)'); title('Critically Damped Response for 420.*t.*(exp(−2.45.*t))')
```

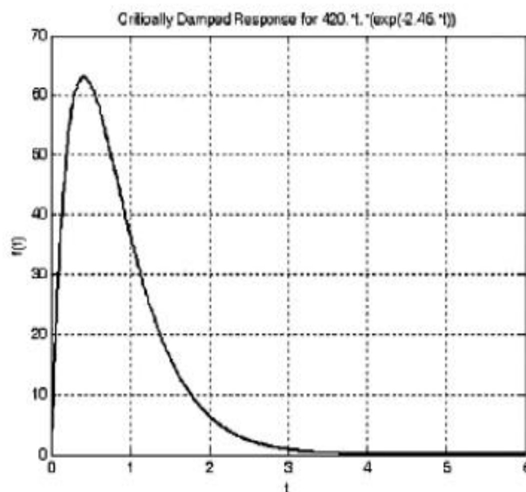


Figure 1.3. Typical critically damped response

A typical underdamped response is shown in Figure 1.4 where it is assumed that $i_n(0) = 0$. This plot was created with the following MATLAB code:

```
t=0: 0.01: 10; ft=210.*sqrt(2).*(exp(–0.5.*t)).*sin(sqrt(2).*t); plot(t,ft); grid; xlabel('t');...
ylabel('f(t)'); title('Underdamped Response for 210.*sqrt(2).*(exp(–0.5.*t)).*sin(sqrt(2).*t)')
```

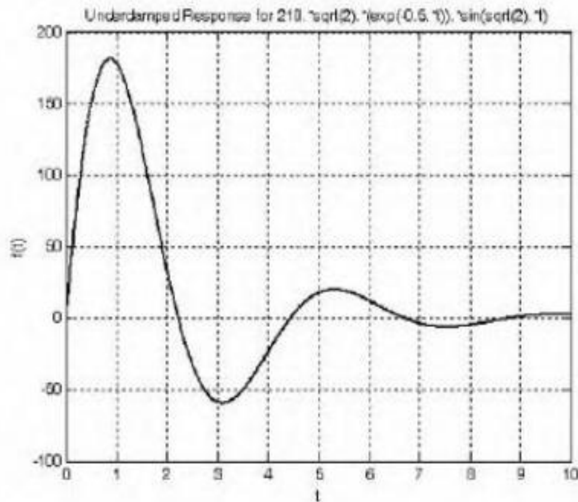


Figure 1.4. Typical underdamped response

Check with MATLAB:

```
syms t;                                     % Define symbolic variable t
R=0.5; L=10^(-3); C=100*10^(-3)/6;% Circuit constants
y0=115*exp(-200*t)-110*exp(-300*t);  % Let solution i(t)=y0
y1=diff(y0);                                % Compute the first derivative of y0, i.e., di/dt
y2=diff(y0,2);                              % Compute the second derivative of y0, i.e, di2/dt2
                                            % Substitute the solution i(t), i.e., equ (1.17)
                                            % into differential equation of (1.11) to verify
                                            % that correct solution was obtained.
                                            % We must also verify that the initial
                                            % conditions are satisfied
y=y2+500*y1+60000*y0;
i0=115*exp(-200*0)-110*exp(-300*0);
vC0=-R*i0-L*(-23000*exp(-200*0)+33000*exp(-300*0))+15;
fprintf(' \n');...
disp('Solution was entered as y0 = '); disp(y0);...
disp('1st derivative of solution is y1 = '); disp(y1);...
disp('2nd derivative of solution is y2 = '); disp(y2);...
disp('Differential equation is satisfied since y = y2+y1+y0 = '); disp(y);...
disp('1st initial condition is satisfied since at t = 0, i0 = '); disp(i0);...
disp('2nd initial condition is also satisfied since vC+vL+vR=15 and vC0 = ');...
disp(vC0);...
fprintf(' \n')
```

[4].

References

[1] B. Hunt, R. Lipsman, J. Rosenberg, K. Coombes, J Osborn and G. Stuck, **"A Guide to MATLAB for Beginners and Experienced Users",** John Wiley & Sons,2001.

[2] Won Young Yang, Wenwu Cao, Tae-Sang Chung and John Morris, **"APPLIED NUMERICAL METHODS USING MATLAB",** John Wiley & Sons, 2005.

[3] Andrew Knight, " **Basics of MATLAB and Beyond"**, CRC Press LLC, 2000.

[4] Steven T. Karris, **"Circuit Analysis II with MATLAB and Applications"**, Orchard Publications, 2003.

[5] Steven T. Karris**, "Introduction to Simulink ® with Engineering Applications",** Orchard Publications, 2006.

[6] K.D. M¨oller, **"Learning by Computing, with Examples Using Mathcad®, Matlab®, Mathematica®, and Maple®"**, Springer Science+Business Media, LLC**,** 2007.