Graphics

PLOT   Linear plot.

PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up. If X is a scalar and Y is a vector, disconnected line objects are created and plotted as discrete points vertically at X.

PLOT(Y) plots the columns of Y versus their index.
If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
In all other uses of PLOT, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with PLOT(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus at each data point; PLOT(X,Y,'bd') plots blue diamond at each data point but does not draw any line.

PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by the (X,Y,S) triples, where the X's and Y's are vectors or matrices and the S's are strings.

For example, PLOT(X,Y,'y-',X,Y,'go') plots the data twice, with a solid yellow line interpolating green circles at the data points.

The PLOT command, if no color is specified, makes automatic use of the colors specified by the axes ColorOrder property. By default, PLOT cycles through the colors in the ColorOrder property. For monochrome systems, PLOT cycles over the axes LineStyleOrder property.

Note that RGB colors in the ColorOrder property may differ from similarly-named colors in the (X,Y,S) triples. For example, the second axes ColorOrder property is medium green with RGB [0 .5 0], while PLOT(X,Y,'g') plots a green line with RGB [0 1 0].

If you do not specify a marker type, PLOT uses no marker.
If you do not specify a line style, PLOT uses a solid line.

PLOT(AX,...) plots into the axes with handle AX.

PLOT returns a column vector of handles to lineseries objects, one handle per plotted line.

The X,Y pairs, or X,Y,S triples, can be followed by parameter/value pairs to specify additional properties of the lines. For example, PLOT(X,Y,'LineWidth',2,'Color',[.6 0 0]) will create a plot with a dark red line width of 2 points.

Example1:

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```

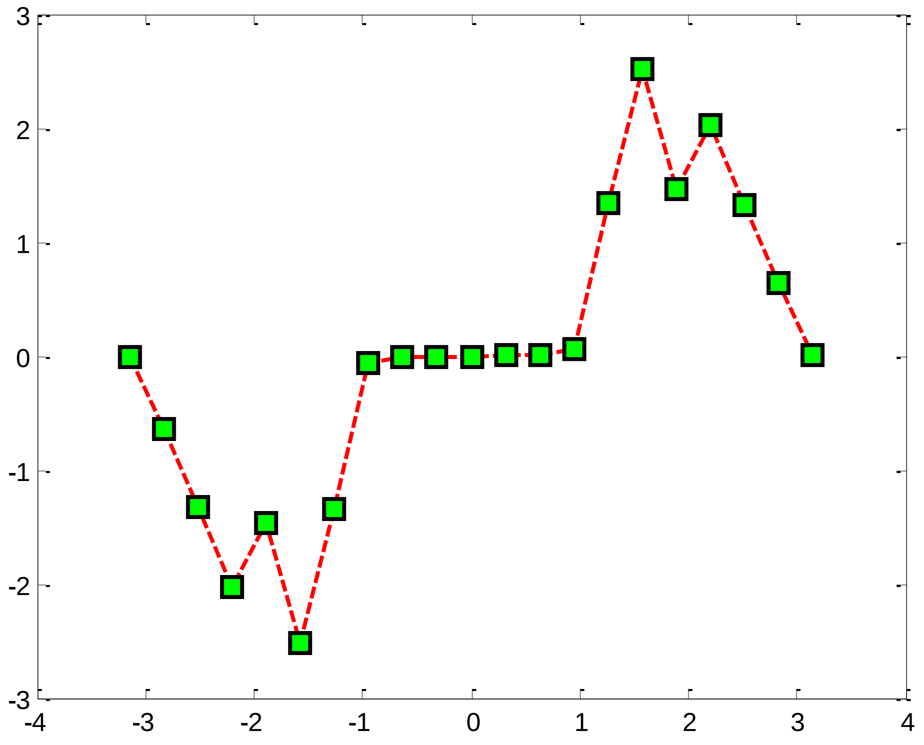


Fig.(1) example1.

Example2:

```
x = -pi:pi/10:pi;
 y = sin(x) ;
       plot(x,y,'--rs','LineWidth',2,...
                       'MarkerEdgeColor','k',...
                       'MarkerFaceColor','g',...
                       'MarkerSize',10)
```

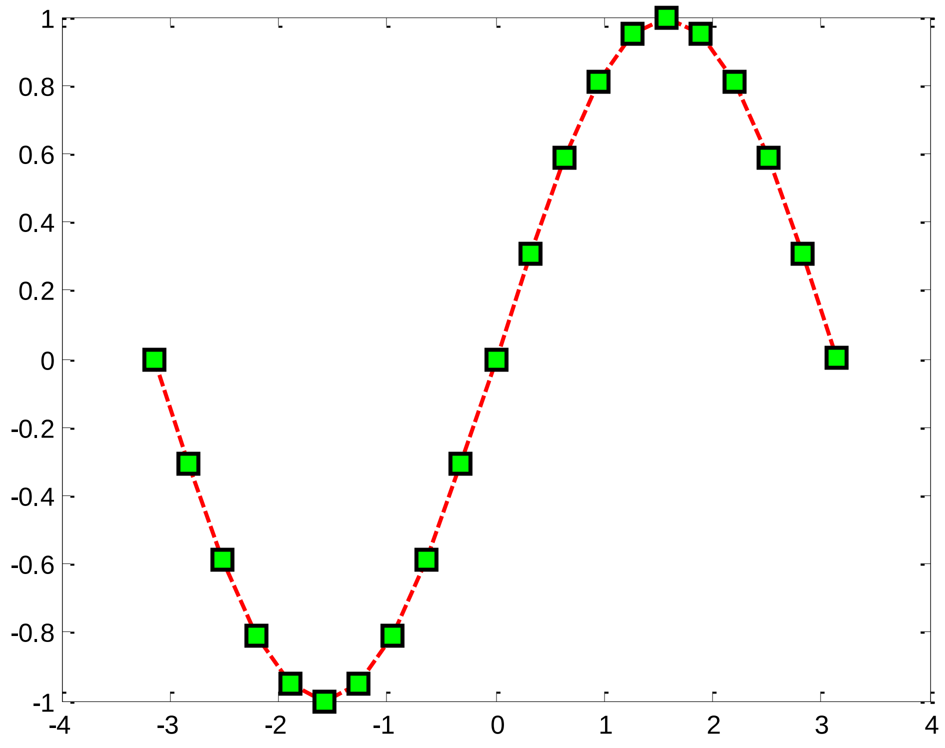


Image processing in MATLAB

- Require MATLAB to be installed with *image processing toolbox*
- Digital image is composed of array of pixels – array of numbers (Matrix)
- Process on image is easily handled by MATLAB

Image Formats Supported

- BMP
- JPEG
- PNG
- TIFF
- PCX
- HDF

etc.

Working formats in MATLAB

- Binary image (0=black, 1=white)
- Intensity image (gray scale image)
- Indexed image (color image)
- RGB image (true color image)
- Multiframe image

How to read image files

imread('file name')
Example :

```
>> imread('image1.jpg');
>> a = imread('image2.jpg');
>> b = imread('image3.bmp');
```

Image file information's.

imfinfo (‘image_file’)

Ex.

```
>> imfinfo (‘test_image.png’)
    ans =
                File name: ………………….
                              FileModDate : ……………..
                              File size : …………………..
                 ………………………………
```

How to write image files

imwrite(var,‘file name’, ‘format’)

Example :

```
>> a = imread(‘image1.png’);
>> imwrite(a, ‘image2.jpg’, ‘jpeg’);
>> imwrite(a, ‘image3.bmp’, ‘bmp’);
```

How to display image

imshow(image_var)

or   figure, imshow(image_var)

Example:

```
>> a = imread(‘image1.png’);
>> a
ans =
     ……………………(numeric)………………………….
>> imshow(a);
>> figure, imshow(a);
```

(diff. window to display image)

Example: Crop for a part of image

```
>> a = imread('image1.png');
>> imshow(a)  % display a
>> for i=1:50
          for j=1:50
           for k=1:3
               b(i,j,k) = a(i,j,k);
end
end
end
                    or
>> b = a(1:50, 1:50, :);
>> figure, imshow(b) % display b on another window
```

Creation of function file

```
file>new
     function y = sum1(n)
     y = 0;
     for i = 1:n
          y = y + i;
     end
Save as  sum1.m
                Calling this function
>>sum1(10)
ans =
```

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.

Control Flow

- **while – end**

```
x=0;
while (x<5)
        x=x+s;
        disp(x);
End
```

- **if – else – elseif – end**

```
a=6;   % a=7;   % a=10;
if (rem(a,3)==0)
        a=a*3;
elseif (rem(a,2)==0)
        a=a*2;
else
        a=a*10;
end
disp (a);
```

- **switch – case – otherwise – end**

```
x= input (‘The value of x:’);
units= input (‘Enter the unit of x: (Please Enter the unit between ‘ ‘) ’);
switch units
        case (‘inch’,’in’)
                y=x*2.54 ;
        case (‘feet’,’ft’)
                y=x*2.54*12 ;
        case (‘meter’,’m’)
                y=x*100 ;
        case (‘millimeter’,’mm’)
                y=x/10 ;
        otherwise
                disp (‘Unknown unit’);
end
```

- **break**

```
for i=1:10
        if (i>5)
                break;
        else
                a=a*4.5;
        end
end
disp (a);
```

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:

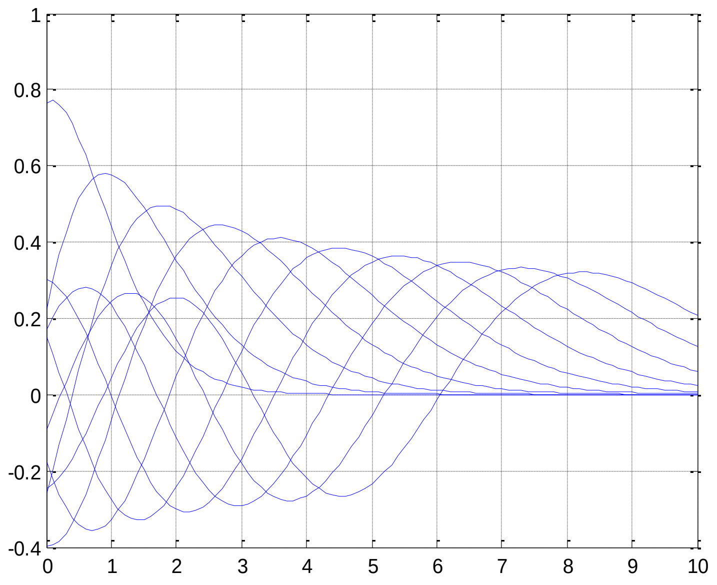
```
format long
z = (0:0.2:1)';
bessely(1,z)

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
```



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

Basic Functions

Function name	**Function mean**
clc	Clear the command window
clear	Clear workspace
who	Show workspace
whos	Show workspace in details
help	Show information about any function
lookfor	Search for a word in MATLAB files
save	Saving workspace
load	Loading saved workspace

Getting help

>> help *function*
>> help *load*
>> help *save*

Reading Data

- We can read a file containing a matrix by using the load function

  load file_name.ext

  File.ext is read into variable file_name, or

  var =load('file name.ext')

  File is read into variable var
- File must be created 1 row/line, each element is separated by *blank* or *tab* or ,
- Each line must contain equal no. of data

Writing Data to Disk

- We can write a file containing a matrix by using the save function

  save *file_name.ext* –ascii *var*

  var is written to *file_name.ext* in ASCII

Examples

save yyy.dat –ascii a   % write a to yyy.dat in ASCII format

```
>> load xxx.txt
>>xxx
………………………………………
>> a = load('xxx.dat');
>> a
……………………………………….
```

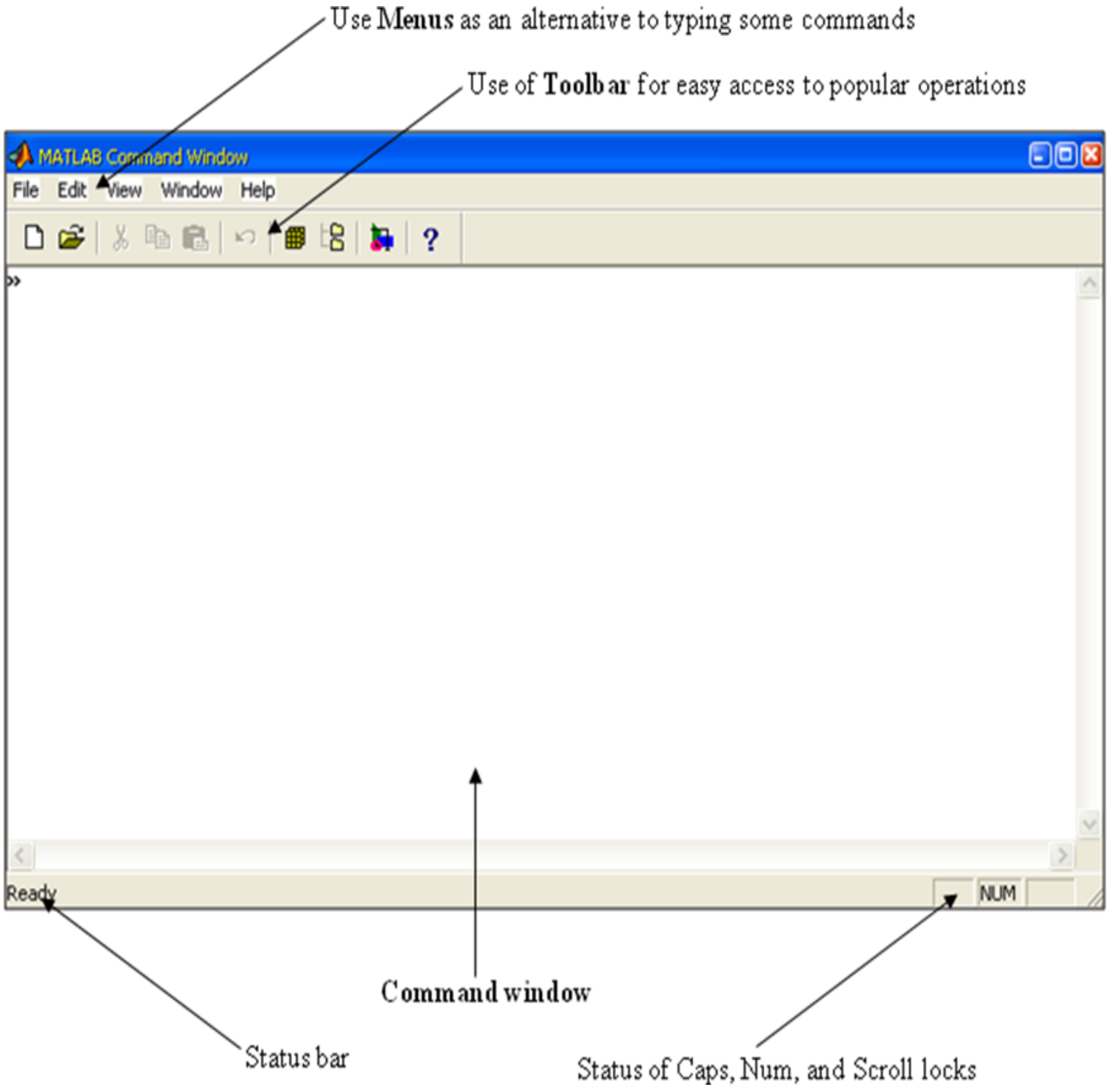
Variables

- MATLAB variables are arrays
- Variable must be one word
  - myvariable accepted
  - my variable not accepted
- Variable length is up to 31 character
- Variables must begin by a character
- MATLAB variables are case sensitive
- There are preserved variable names:

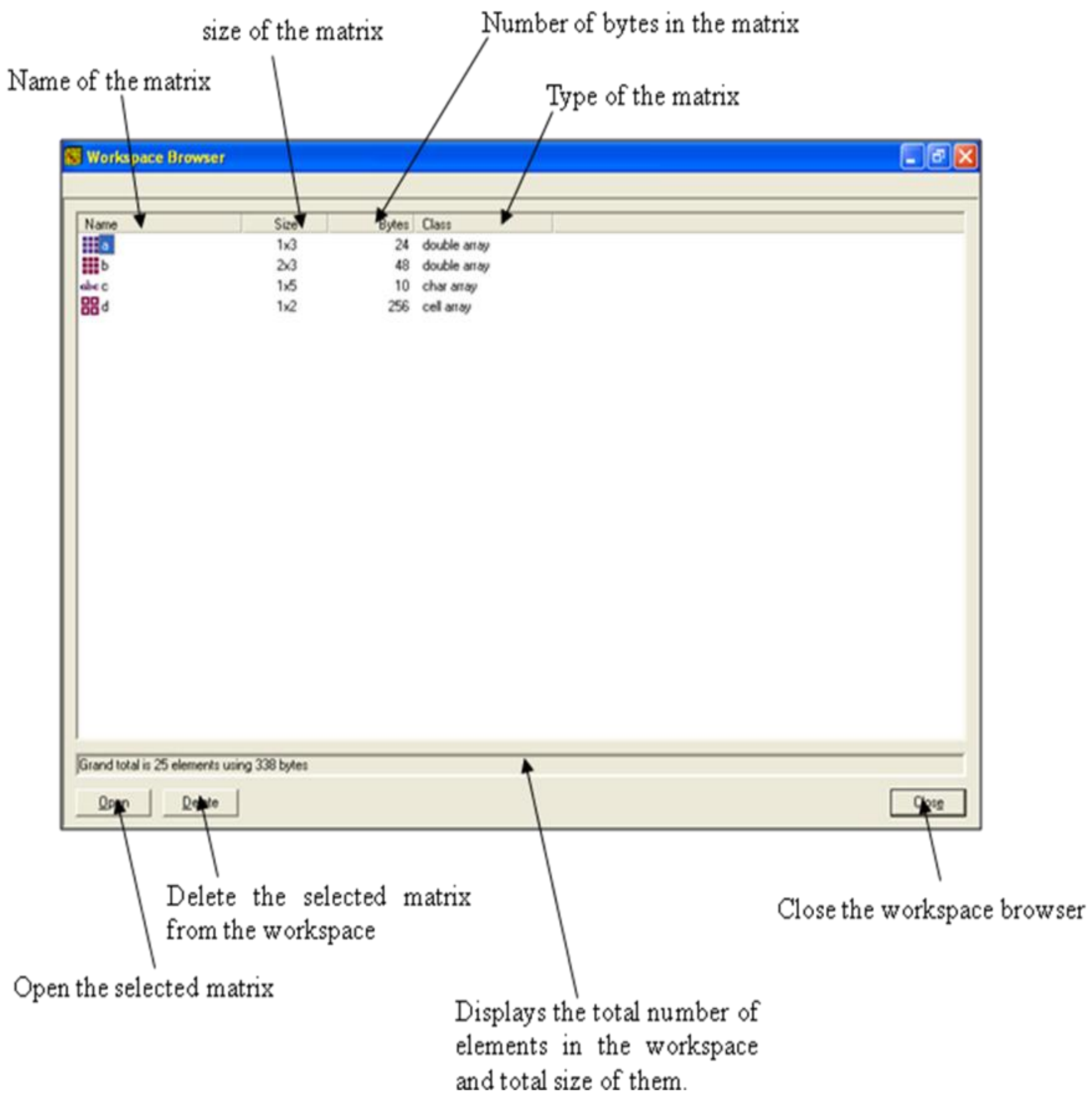
ans	Default function output
pi	Pi = 3.14
eps	Very small value
inf	Very large value = ∞ as the result of (1/0)
NaN	When the result = 0/0
realmin	Minimum real number = $10^{-308} \times 2.2251$
realmax	Maximum real number = $10^{308} \times 1.7977$
nargin	The number of input parameters
nargout	The number of output parameters

MATLAB Environments

- ## Command window

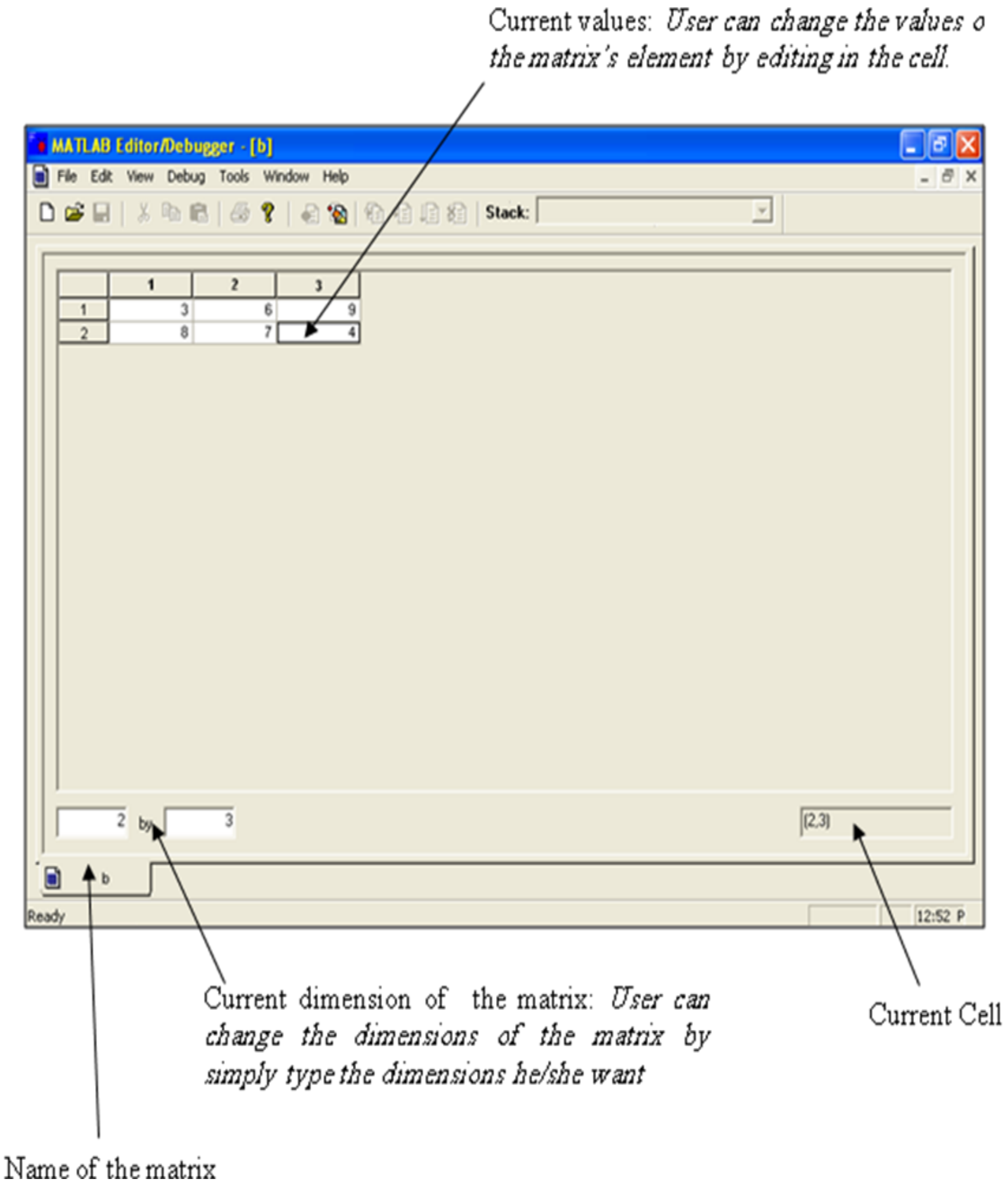

Use Menus as an alternative to typing some commands
Use of Toolbar for easy access to popular operations
MATLAB Command Window
File Edit View Window Help
>>
Ready
NUM
Command window
Status bar
Status of Caps, Num, and Scroll locks

- **Workspace**


size of the matrix
Number of bytes in the matrix
Name of the matrix
Type of the matrix
Workspace Browser
Open
Delete
Close
Delete the selected matrix from the workspace
Close the workspace browser
Open the selected matrix
Displays the total number of elements in the workspace and total size of them.

Name	Size	Bytes	Class
a	1x3	24	double array
b	2x3	48	double array
c	1x5	10	char array
d	1x2	256	cell array

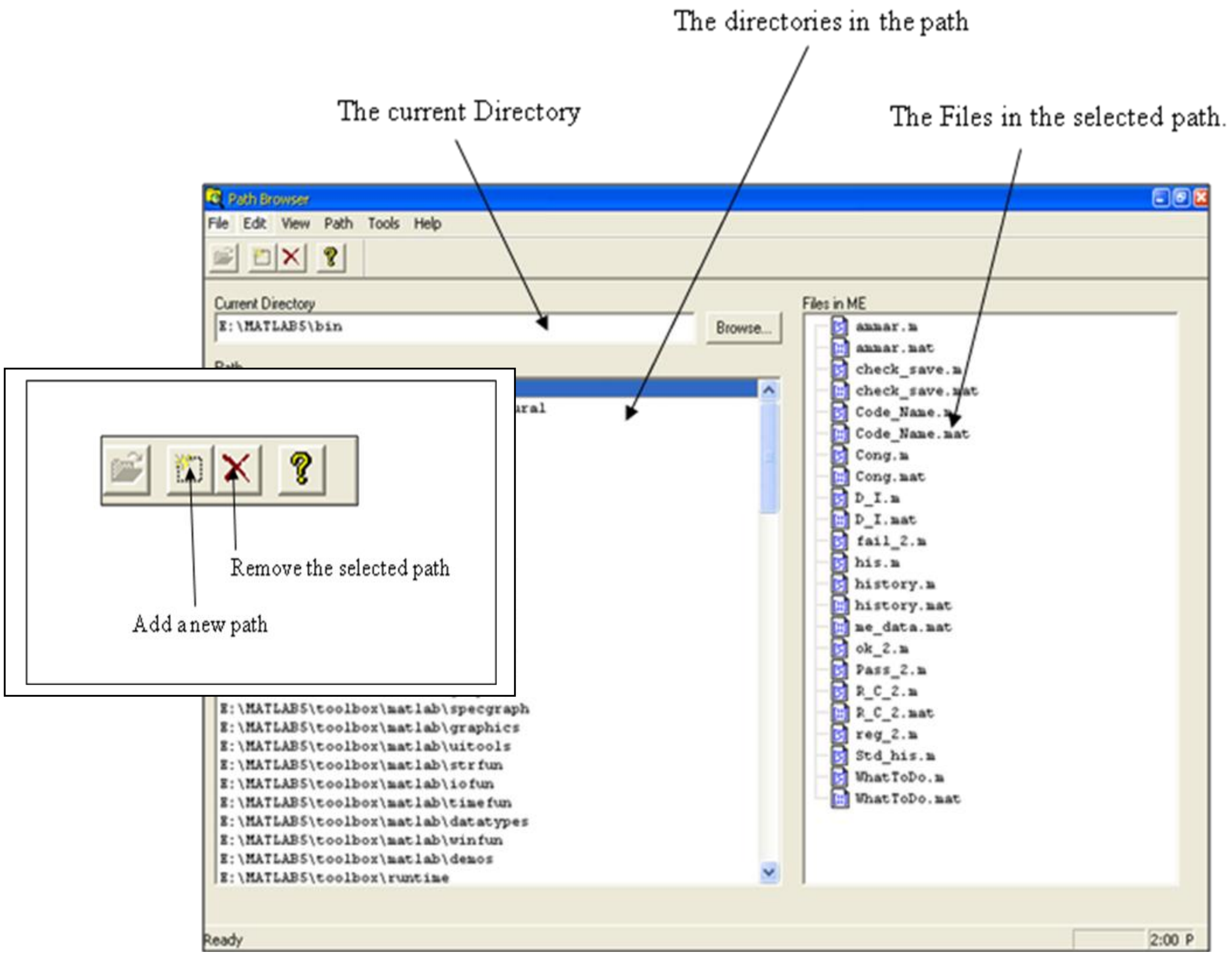
Grand total is 25 elements using 338 bytes



	1	2	3
1	3	6	9
2	8	7	4

Current values: *User can change the values o the matrix's element by editing in the cell.*
MATLAB Editor/Debugger - [b]
File Edit View Debug Tools Window Help
Stack:
2 by 3
(2,3)
b
Ready
12:52 P
Current dimension of the matrix: *User can change the dimensions of the matrix by simply type the dimensions he/she want*
Current Cell
Name of the matrix

- **Path browser**

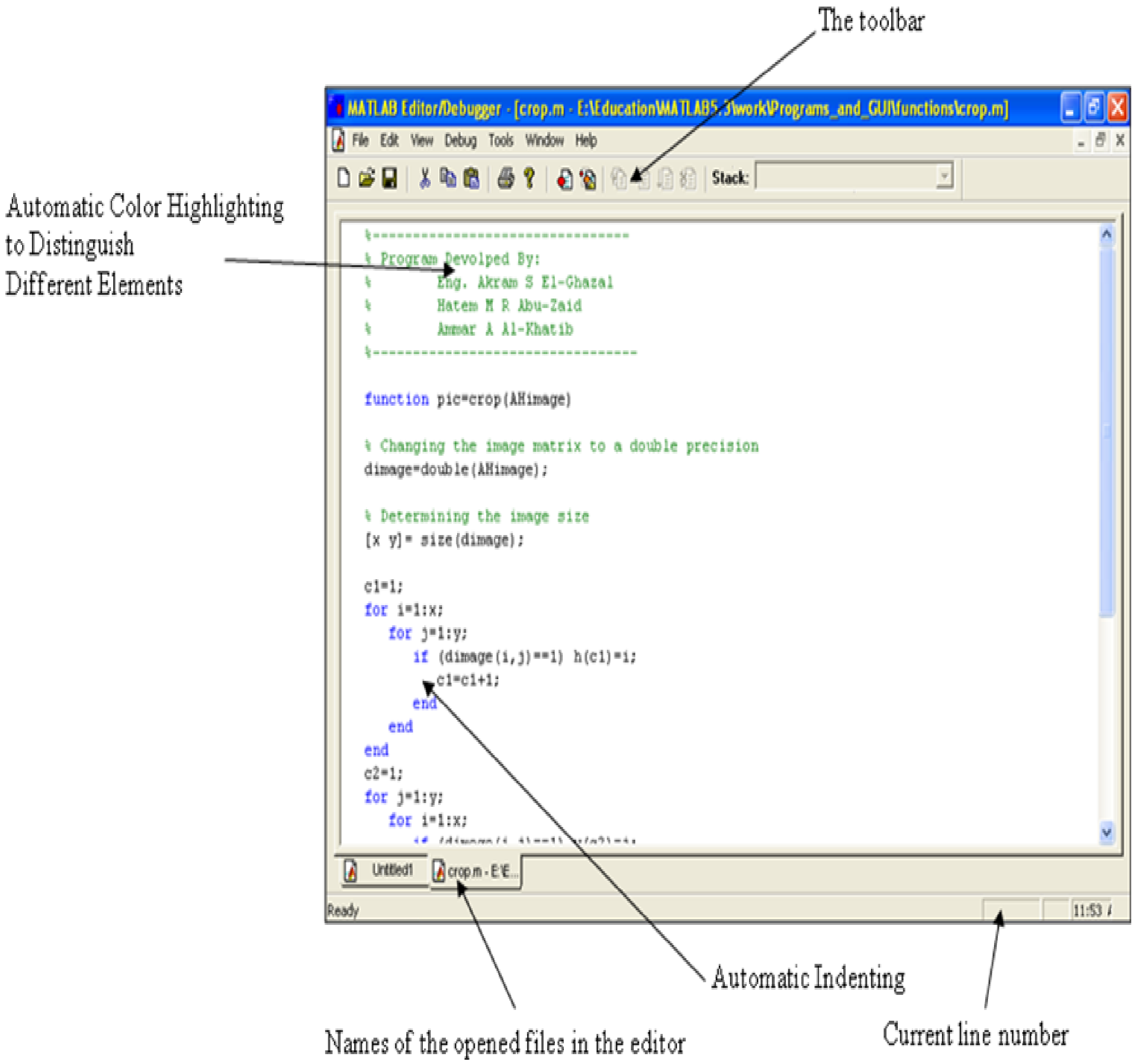


```
E:\MATLAB5\bin
```

```
E:\MATLAB5\toolbox\matlab\specgraph
E:\MATLAB5\toolbox\matlab\graphics
E:\MATLAB5\toolbox\matlab\uitools
E:\MATLAB5\toolbox\matlab\strfun
E:\MATLAB5\toolbox\matlab\iofun
E:\MATLAB5\toolbox\matlab\timefun
E:\MATLAB5\toolbox\matlab\datatypes
E:\MATLAB5\toolbox\matlab\winfun
E:\MATLAB5\toolbox\matlab\demos
E:\MATLAB5\toolbox\runtime
```

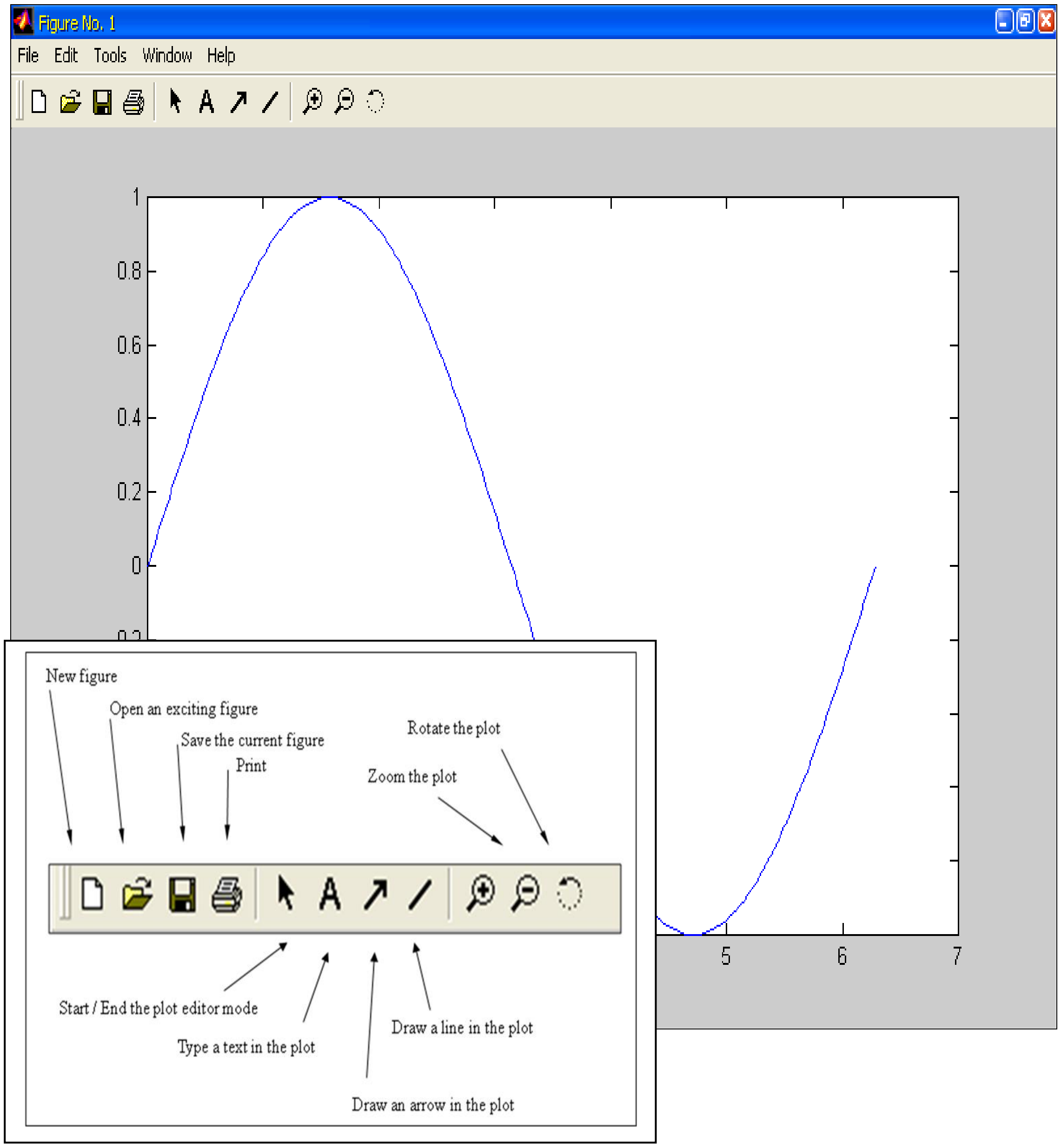
```
ammar.m
ammar.mat
check_save.m
check_save.mat
Code_Name.m
Code_Name.mat
Cong.m
Cong.mat
D_I.m
D_I.mat
fail_2.m
his.m
history.m
history.mat
me_data.mat
ok_2.m
Pass_2.m
R_C_2.m
R_C_2.mat
reg_2.m
Std_his.m
WhatToDo.m
WhatToDo.mat
```

- **Editor window**


The toolbar
Automatic Color Highlighting to Distinguish Different Elements
MATLAB Editor/Debugger - [crop.m - E:\Education\MATLAB5.3\work\Programs_and_GUI\functions\crop.m]
File Edit View Debug Tools Window Help
Stack:
Untitled1
crop.m - E:\E...
Ready
11:53
Automatic Indenting
Names of the opened files in the editor
Current line number

```
%--------------------------------
% Program Devolped By:
%        Eng. Akram S El-Ghazal
%        Hatem M R Abu-Zaid
%        Ammar A Al-Khatib
%--------------------------------

function pic=crop(AHimage)

% Changing the image matrix to a double precision
dimage=double(AHimage);

% Determining the image size
[x y]= size(dimage);

c1=1;
for i=1:x;
   for j=1:y;
      if (dimage(i,j)==1) h(c1)=i;
         c1=c1+1;
      end
   end
end
c2=1;
for j=1:y;
   for i=1:x;
```

- **Figure window**



Math functions

Basic functions	**triangular functions**	**Approximation functions**	**complex functions**	**conversion functions**
abs	cos	fix	abs	dec2hex
sqrt	sin	round	angle	dec2bin
mean	tan	floor	conj	dec2base
power	acos	ceil	imag	hex2dec
log	asin	rem	real	bin2dec
log10	atan		floor	base2dec
exp	csc		ceil	rad2deg
max	sec		rem	deg2rad
min	cot			cart2sph
sort	acsc			cart2pol
	asec			pol2cart
	acot			sph2cart
	cosh			
	sinh			
	tanh			

Relational and Logical Operators

Relational Operators		
Operator name	**Symbol**	**Comment**
eq	= =	equal
ne	~=	not equal
lt	<	less than
gt	>	greater than
le	<=	less than or equal
ge	>=	greater than or equal
Logical Operators		
and	&	logical and
or	|	logical or
not	~	logical not
xor		logical exclusive or

Matrix and Array

- Matrix = rectangular array of numbers
- 1-by-1 → scalar
- 1-by-n or n-by-1 → vector
- m-by-n → two-dim matrix
- n_1-by-n_2-by-n_3 … → n-dim matrix

1. Solving Matrix Equations

Example1:

```
x=[1 2 3; 4 5 6; 7 8 9];
v=[3;-6;1];
y=inv(x)*v
```

Example2:

```
x = 0:25;
y = [exp(-.07*x).*cos(x);exp(.05*x).*cos(x)]';
```

Example3:

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

Example4:

```
x=[10^3 10^4 10^5 10^6 10^7 10^8]
y=x.^2
```

Example5:

```
x=-10:0.01:10;
y=cos((pi.*x*(2000/(5000+1000*cos(0.0002)))*sin(0.0002
))/0.0005*10^-3).^2
```

IF Conditionally execute statements.The general form of the IF statement is

```
IF expression
  statements
ELSEIF expression
  statements
ELSE
  statements
END
```

The statements are executed if the real part of the expression has all non-zero elements. The ELSE and ELSEIF parts are optional. Zero or more ELSEIF parts can be used as well as nested IF's. The expression is usually of the form expr rop expr where rop is ==, <, >, <=, >=, or ~=.

Example

```
if I == J
  A(I,J) = 2;
elseif abs(I-J) == 1
  A(I,J) = -1;
else
  A(I,J) = 0;
end
```

ELSE is used with IF. The statements after the ELSE are executed if all the preceding IF and ELSEIF expressions are false. The general form of the IF statement is

```
IF expression
  statements
ELSEIF expression
  statements
ELSE
  statements
END
```

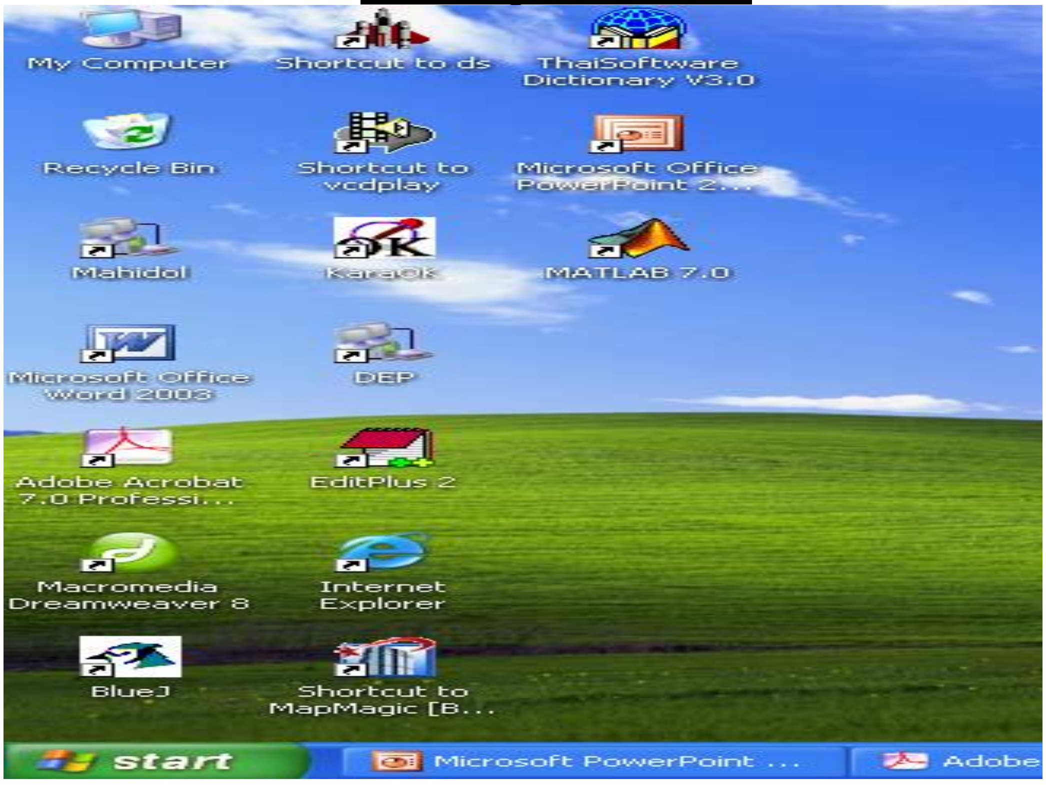
MATLAB Windows

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.

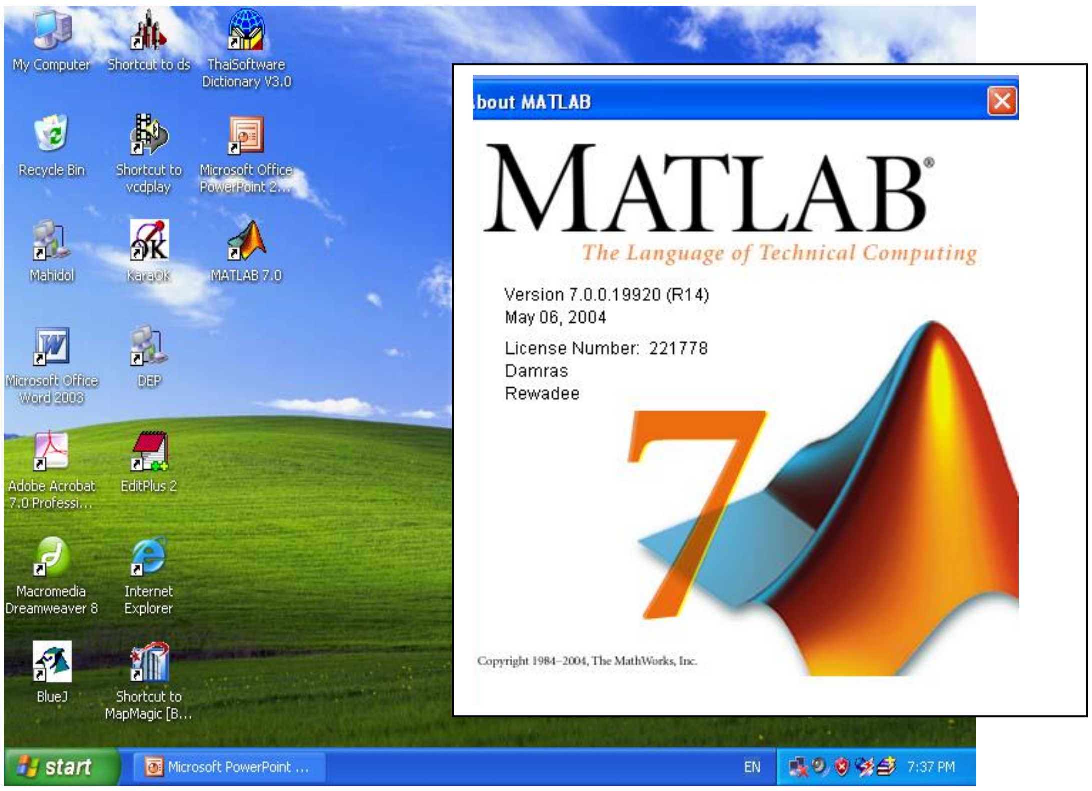
Typing in the Command Window

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].

Starting MATLAB



- **Double click on the MATLAB shortcut**



bout MATLAB

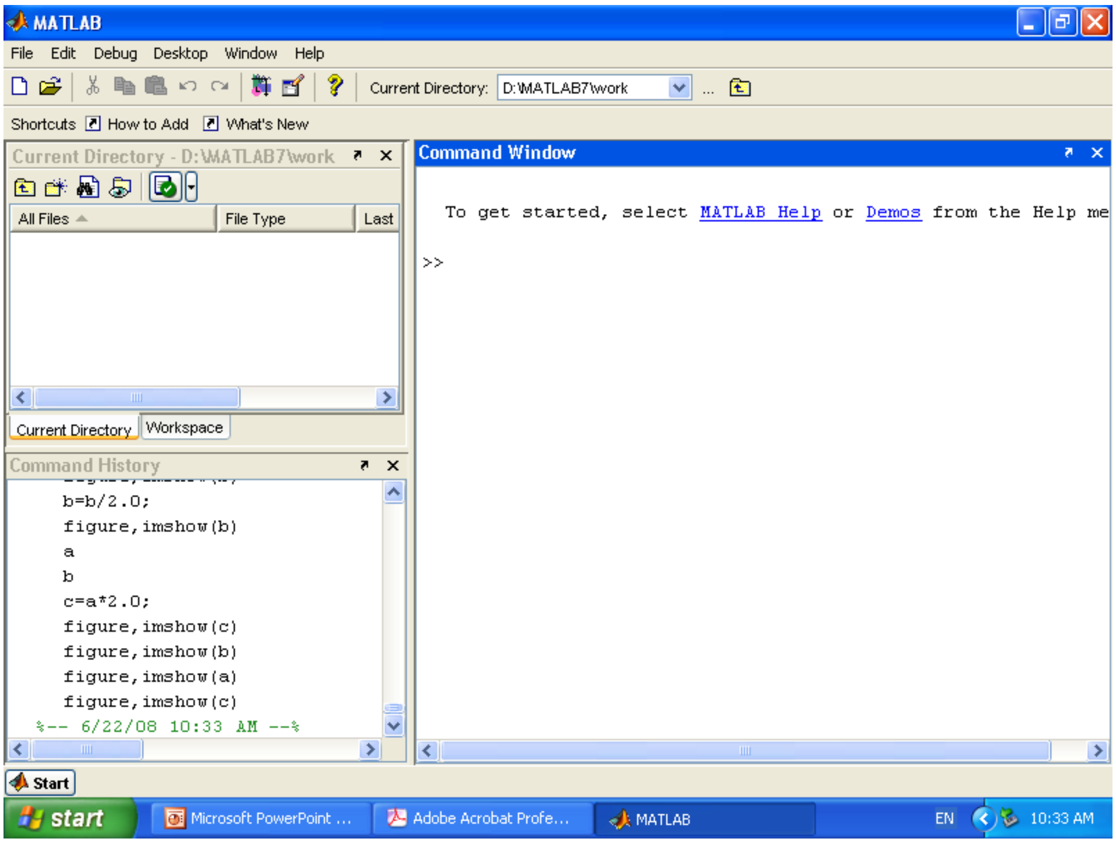
MATLAB®

The Language of Technical Computing

Version 7.0.0.19920 (R14)
May 06, 2004

License Number: 221778
Damras
Rewadee

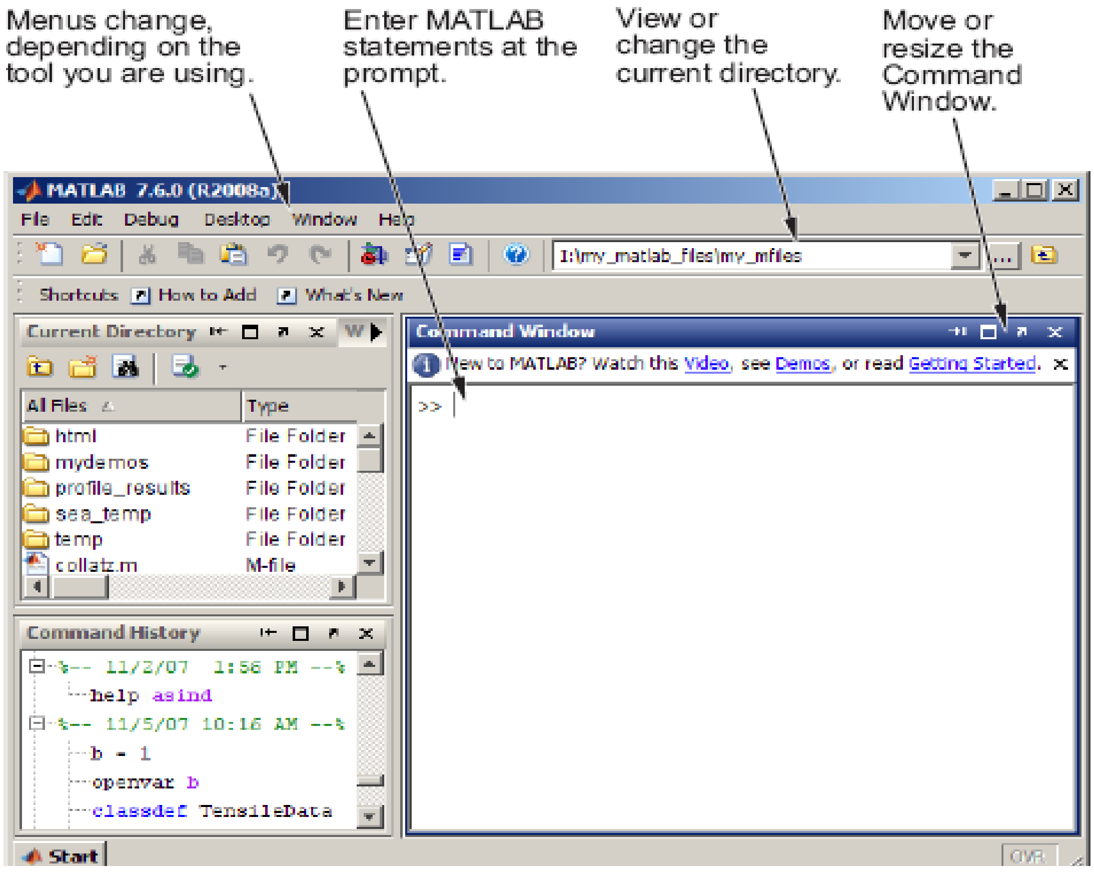
Copyright 1984–2004, The MathWorks, Inc.



```
  To get started, select MATLAB Help or Demos from the Help me

>>
```

```
    b=b/2.0;
    figure,imshow(b)
    a
    b
    c=a*2.0;
    figure,imshow(c)
    figure,imshow(b)
    figure,imshow(a)
    figure,imshow(c)
%-- 6/22/08 10:33 AM --%
```

Figure 1-1: A MATLAB Desktop

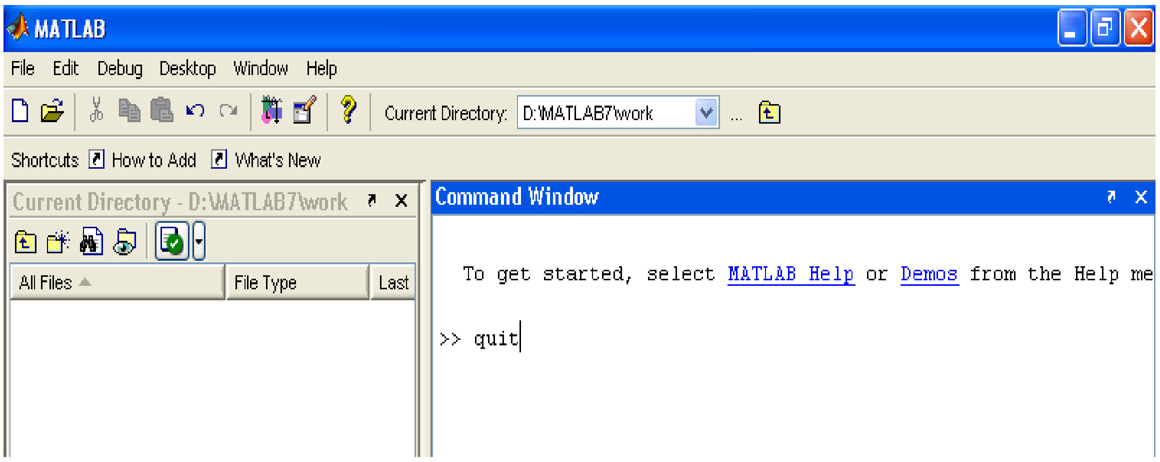

Menus change, depending on the tool you are using.
Enter MATLAB statements at the prompt.
View or change the current directory.
Move or resize the Command Window.
MATLAB 7.6.0 (R2008a)
File Edit Debug Desktop Window Help
I:\my_matlab_files\my_mfiles
Shortcuts How to Add What's New
Current Directory
All Files Type
html File Folder
mydemos File Folder
profile_results File Folder
sea_temp File Folder
temp File Folder
collatz.m M-file
Command History
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Start
OVR

```
%-- 11/2/07  1:56 PM --%
help asind
%-- 11/5/07 10:16 AM --%
b = 1
openvar b
classdef TensileData
```

```
>>
```

Quitting MATLAB

- **Select File > Exit MATLAB in desktop**
- **Or type quit at prompt '>>' in command window**


MATLAB
File Edit Debug Desktop Window Help
Current Directory: D:\MATLAB7\work
Shortcuts How to Add What's New
Current Directory - D:\MATLAB7\work
All Files File Type Last
Command Window

```
To get started, select MATLAB Help or Demos from the Help me

>> quit
```

Overview

- MATLAB = Matrix Laboratory.
- To provide easy access to software developed by LINPACK and EISPACK projects.
- Incorporate LAPACK and BLAS lib. for matrix computation.
- Features add-on applications-specific called *toolboxes.*

The MATLAB System

- Desktop tools &development environment – *tools and facilities to help users use and become more productive*
- Mathematical function lib.-*large collection of computational algorithms*
- The language- *high-level language with control flow statements*
- Graphics – *facilities for data visualization*
- External interfaces – *can be used with other programming language like C or FORTRAN.*

Reference

[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.

STEM Function

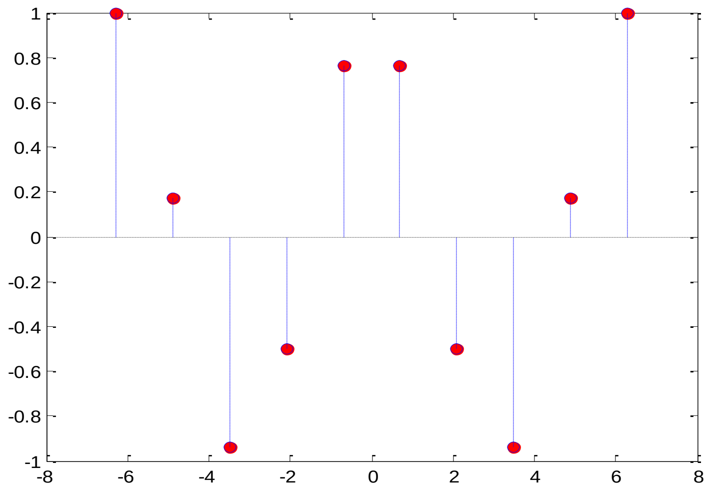
STEM Discrete sequence or "stem" plot. STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. If Y is a matrix then each column is plotted as a separate series STEM(X,Y) plots the data sequence Y at the values specified in X. STEM(...,'filled') produces 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.

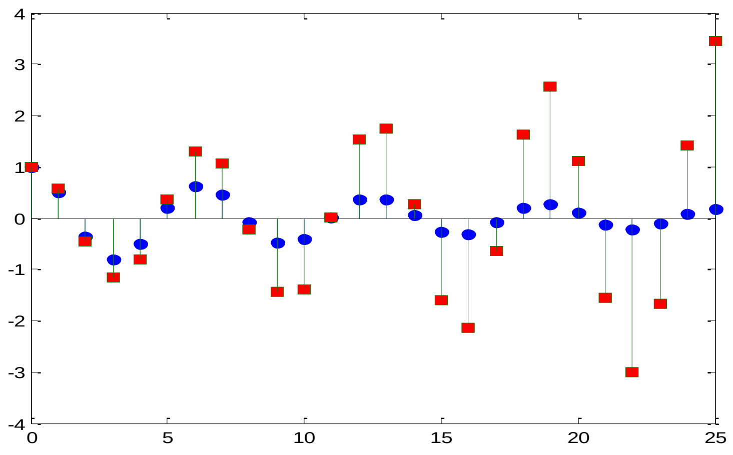
```
t = linspace(-2*pi,2*pi,10);
h = stem(t,cos(t),'fill','--');
set(get(h,'BaseLine'),'LineStyle',':')
set(h,'MarkerFaceColor','red')
```



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

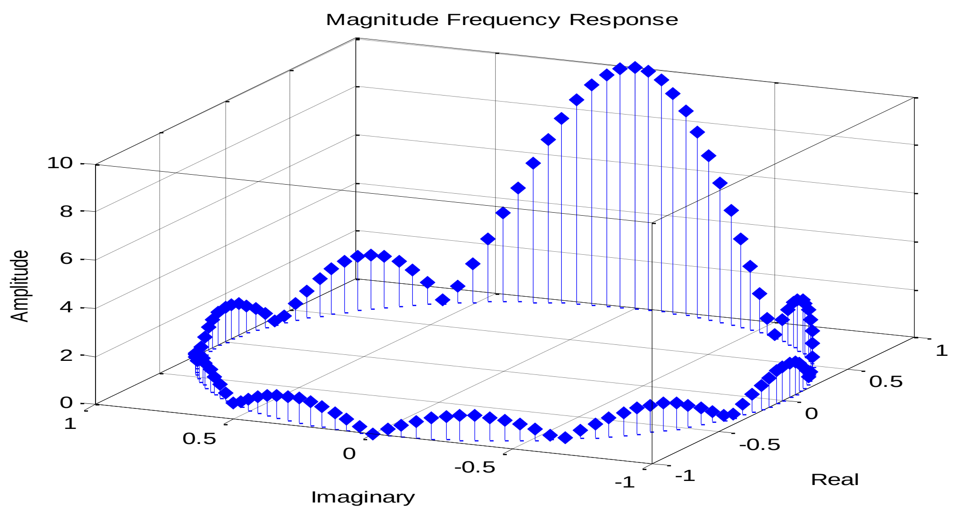
```
th = (0:127)/128*2*pi;
x = cos(th);
y = sin(th);
```

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])
```



[3].

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:

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)
```

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.

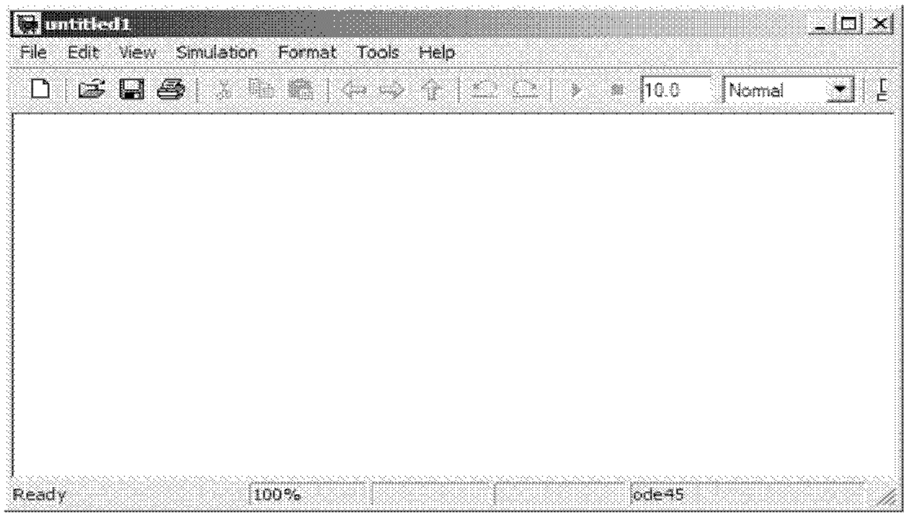
Block Diagram Semantics

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:

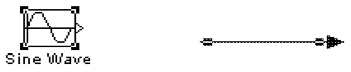
```
function new_model(modelname)
% NEW_MODEL Create a new, empty Simulink model
%    NEW_MODEL('MODELNAME') creates a new model with
%    the name 'MODELNAME'. Without the 'MODELNAME'
%    argument, the new model is named 'my_untitled'.

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

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.

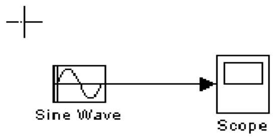
Selecting Multiple Objects One at a Time

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.

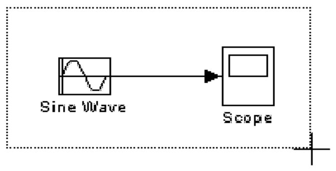
Selecting Multiple Objects Using a Bounding Box

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:

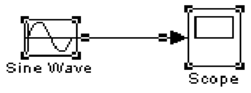
1. Define the starting corner of a bounding box by positioning the pointer at one corner of the box, then pressing and holding down the mouse button. Notice the shape of the cursor.



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

Connecting Blocks

On this page…

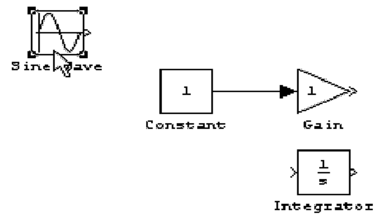
Automatically Connecting Blocks

You can command the Simulink software to connect blocks automatically. This eliminates the need for you to draw the connecting lines yourself. When connecting blocks, the lines are routed around intervening blocks to avoid cluttering the diagram.

Connecting Two Blocks

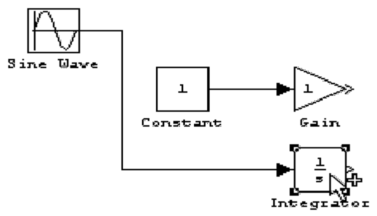
To autoconnect two blocks:

1. Select the source block.

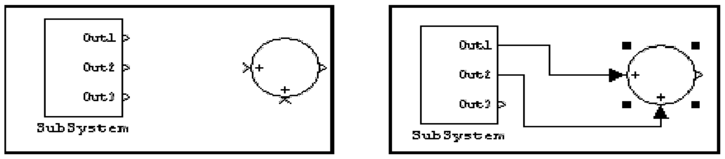


2. Hold down **Ctrl** and left-click the destination block.

   The source block is connected to the destination block, and the lines are routed around intervening blocks if necessary.



When connecting two blocks, the Simulink software draws as many connections as possible between the two blocks as illustrated in the following example.



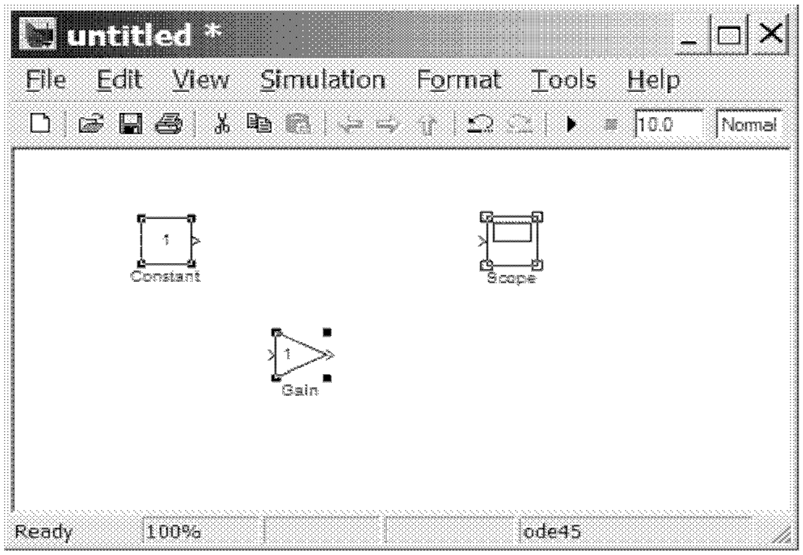
Before autoconnect After autoconnect

Connecting Groups of Blocks

Aligning, Distributing, and Resizing Groups of Blocks Automatically

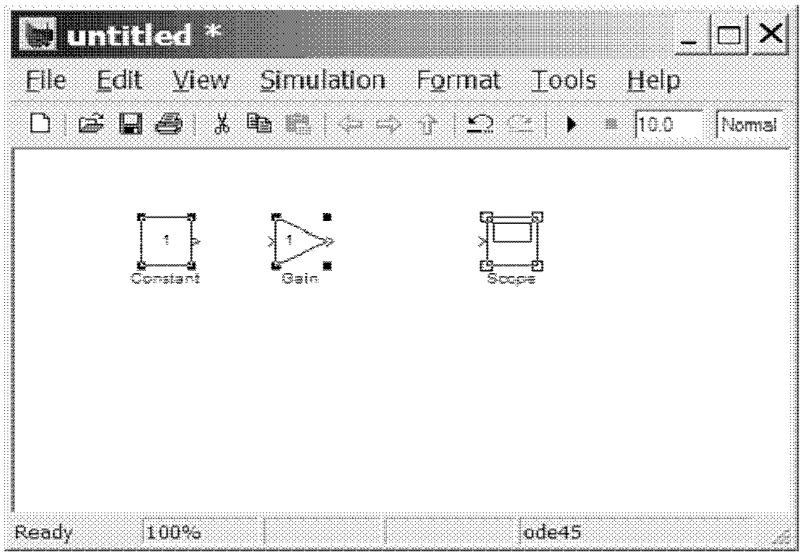
The model editor's **Format** menu includes commands that let you quickly align, distribute, and resize groups of blocks. To align (or distribute or resize) a group of blocks:

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

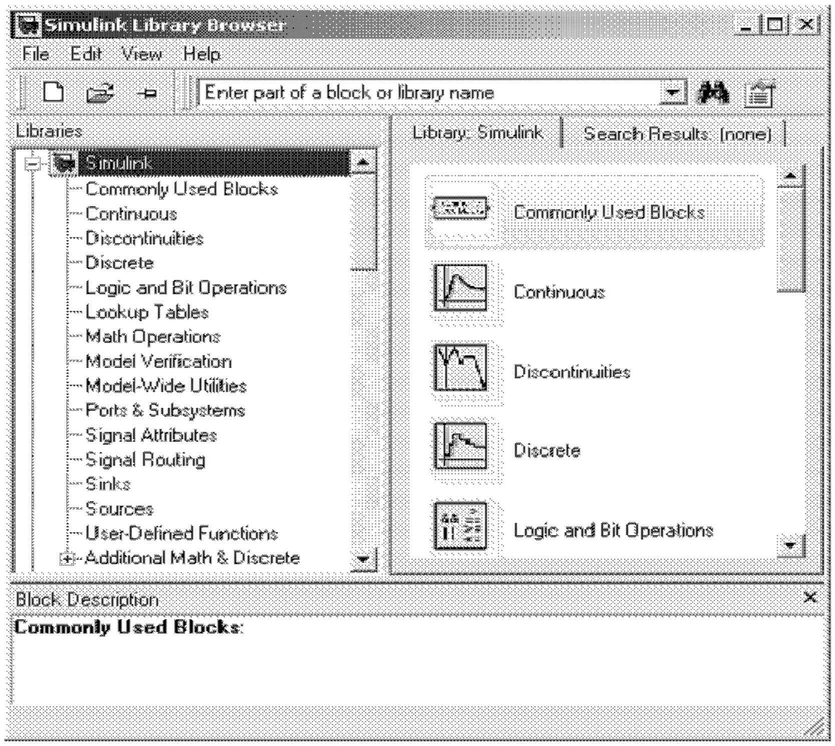
Starting Simulink Software

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

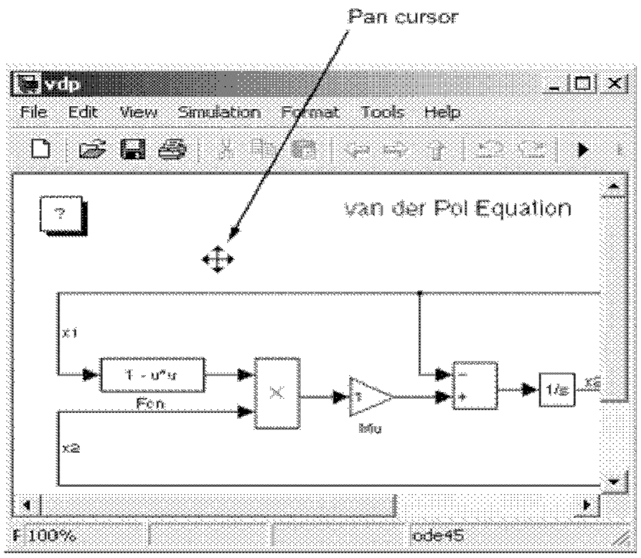
Simulink Basics Opening a Model

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

Reference
Mat lab 2009b Help