USING MATLAB AS A CALCULATOR

Tutorial 1-1: Using MATLAB as a calculator.

```
>> 7+8/2
ans =
    11
>> (7+8)/2
ans =
    7.5000
>> 4+5/3+2
ans =
    7.6667
>> 5^3/2
ans =
   62.5000
>> 27^(1/3)+32^0.2
ans =
     5
>> 27^1/3+32^0.2
ans =
    11

>> 0.7854-(0.7854)^3/(1*2*3)+0.785^5/(1*2*3*4*5)...
-(0.785)^7/(1*2*3*4*5*6*7)

ans =
    0.7071
>>
```

Type and press **Enter**.

8/2 is executed first.

Type and press **Enter**.

7+8 is executed first.

5/3 is executed first.

5^3 is executed first, /2 is executed next.

1/3 is executed first, 27^(1/3) and 32^0.2 are executed next, and + is executed last.

27^1 and 32^0.2 are executed first, /3 is executed next, and + is executed last.

Type three periods ... (and press **Enter**) to continue the expression on the next line.

The last expression is the first 4 terms of the Taylor series for $\sin(\pi/4)$.

Table 1-2: Display formats

Command	Description	Example

`format short`	Fixed-point with 4 decimal digits for: $0.001 \le number \le 1000$ Otherwise display format `short e`.	`>> 290/7`
		`ans =`
		`41.4286`

`format long`	Fixed-point with 14 decimal digits for: $0.001 \le number \le 100$ Otherwise display format `long e`.	`>> 290/7`
		`ans =`
		`41.42857142857143`

`format short e`	Scientific notation with 4 decimal digits.	`>> 290/7`
		`ans =`
		`4.1429e+001`

`format long e`	Scientific notation with 15 decimal digits.	`>> 290/7`
		`ans =`
		`4.142857142857143e+001`

`format short g`	Best of 5-digit fixed or floating point.	`>> 290/7`
		`ans =`
		`41.429`

`format long g`	Best of 15-digit fixed or floating point.	`>> 290/7`
		`ans =`
		`41.4285714285714`

`format bank`	Two decimal digits.	`>> 290/7`
		`ans =`
		`41.43`

`format compact`	Eliminates empty lines to allow more lines with information displayed on the screen.	
`format loose`	Adds empty lines (opposite of `compact`).	

Table 1-3: Elementary math functions

Function	Description	Example

`sqrt(x)`	Square root.	`>> sqrt(81)`
		`ans =`
		`9`

`nthroot(x,n)`	Real *n*th root or a real number *x*. (If *x* is negative *n* must be an odd integer.)	`>> nthroot(80,5)`
		`ans =`
		`2.4022`

`exp(x)`	Exponential (e^x).	`>> exp(5)`
		`ans =`
		`148.4132`

`abs(x)`	Absolute value.	`>> abs(-24)`
		`ans =`
		`24`

`log(x)`	Natural logarithm.
	Base e logarithm (ln).
		`>> log(1000)`
		`ans =`
		`6.9078`

`log10(x)`	Base 10 logarithm.	`>> log10(1000)`
		`ans =`
		`3.0000`

Table 1-4: Trigonometric math functions

Function	Description	Example

`sin(x)`
`sind(x)`
	Sine of angle x (x in radians).
	Sine of angle x (x in degrees).
		`>> sin(pi/6)`
		`ans =`
		`0.5000`

`cos(x)`
`cosd(x)`
	Cosine of angle x (x in radians).
	Cosine of angle x (x in degrees).
		`>> cosd(30)`
		`ans =`
		`0.8660`

`tan(x)`
`tand(x)`
	Tangent of angle x (x in radians).
	Tangent of angle x (x in degrees).
		`>> tan(pi/6)`
		`ans =`
		`0.5774`

`cot(x)`
`cotd(x)`
	Cotangent of angle x (x in radians).
	Cotangent of angle x (x in radians).
		`>> cotd(30)`
		`ans =`
		`1.7321`

The inverse trigonometric functions are `asin(x)`, `acos(x)`, `atan(x)`, `acot(x)` for the angle in radians, and `asind(x)`, `acosd(x)`, `atand(x)`, `acotd(x)` for the angle in degrees. The hyperbolic trigonometric functions are `sinh(x)`, `cosh(x)`, `tanh(x)`, and `coth(x)`. The previous table uses `pi` which is equal to π (see Section 1.6.3).

```
>> a=12;
>> B=4;
>> C=(a-B)+40-a/B*10;
>> C
C =
    18
```

The variables a, B, and C are defined but are not displayed since a semicolon is typed at the end of each statement.

The value of the variable C is displayed by typing the name of the variable.

- Several assignments can be typed in the same line. The assignments must be separated with a comma (spaces can be added after the comma). When the **Enter** key is pressed, the assignments are executed from left to right and the variables and their assignments are displayed. A variable is not displayed if a semicolon is typed instead of a comma. For example, the assignments of the variables a, B, and C above can all be done in the same line.

```
>> a=12, B=4; C=(a-B)+40-a/B*10
a =
    12
C =
    18
```

The variable B is not displayed because a semicolon is typed at the end of the assignment.

```
>> x=0.75;
>> E=sin(x)^2+cos(x)^2
E =
     1
>>
```

Command	**Outcome**
`clear`	Removes all variables from the memory.
`clear x y z`	Removes only variables x, y, and z from the memory.
`who`	Displays a list of the variables currently in the memory.
`whos`	Displays a list of the variables currently in the memory and their size together with information about their bytes and class (see Section 4.1).

ONE-DIMENSIONAL ARRAY

```
variable_name = [ type vector elements ]
```

Creating a vector with constant spacing by specifying the first term, the spacing, and the last term:

In a vector with constant spacing the difference between the elements is the same. For example, in the vector: $v = 2\ \ 4\ \ 6\ \ 8\ \ 10$, the spacing between the elements is 2. A vector in which the first term is m, the spacing is q, and the last term is n is created by typing:

```
variable_name = [m:q:n]
```

or

```
variable_name = m:q:n
```

(The brackets are optional.)

```
>> x=[1:2:13]
```

First element 1, spacing 2, last element 13.

```
x =
     1     3     5     7     9    11    13
>> y=[1.5:0.1:2.1]
```

First element 1.5, spacing 0.1, last element 2.1.

```
y =
   1.5000   1.6000   1.7000   1.8000   1.9000   2.0000   2.1000

>> z=[-3:7]
```

First element –3, last term 7.
If spacing is omitted, the default is 1.

```
z =
    -3    -2    -1     0     1     2     3     4     5     6
```

```
>> va=linspace(0,8,6)
va =
      0    1.6000    3.2000    4.8000    6.4000    8.0000
>> vb=linspace(30,10,11)
vb =
    30    28    26    24    22    20    18    16    14    12
10
>> u=linspace(49.5,0.5)

u =
  Columns 1 through 10
   49.5000   49.0051   48.5101   48.0152   47.5202   47.0253
46.5303   46.0354   45.5404   45.0455
.............
Columns 91 through 100
    4.9545    4.4596    3.9646    3.4697    2.9747    2.4798
1.9848    1.4899    0.9949    0.5000
>>
```

6 elements, first element 0, last element 8.

11 elements, first element 30, last element 10.

First element 49.5, last element 0.5.

When the number of elements is omitted, the default is 100.

100 elements are displayed.

TWO-DIMENSIONAL ARRAY

```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ….. ; last row elements]
```

Tutorial 2-2: Creating matrices.

```
>> a=[5  35  43;  4  76  81;  21  32  40]
a =
     5    35    43
     4    76    81
    21    32    40
>> b = [7  2  76  33  8
1  98  6  25  6
5  54  68  9  0]
b =
     7     2    76    33     8
     1    98     6    25     6
     5    54    68     9     0
```

A semicolon is typed before a new line is entered.

The **Enter** key is pressed before a new line is entered.

```
>> cd=6; e=3; h=4;
```

Three variables are defined.

```
>> Mat=[e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]
Mat =
    3.0000   24.0000    0.5000
   16.0000    1.6330   14.0000
>>
```

Elements are defined by mathematical expressions.

ma(*k*,*p*) refers to the element in row *k* and column *p*.

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
MAT =
     3    11     6     5
     4     7    10     2
    13     9     0     8
>> MAT(3,1)=20
MAT =
     3    11     6     5
     4     7    10     2
    20     9     0     8
>> MAT(2,4)-MAT(1,2)
ans =
    -9
```

Create a 3 × 4 matrix.

Assign a new value to the (3,1) element.

Use elements in a mathematical expression.

$A(:,n)$ Refers to the elements in all the rows of column n of the matrix A.

$A(n,:)$ Refers to the elements in all the columns of row n of the matrix A.

By using A=[1:2:11;2:2:12;3:3:18;4:4:24;….]

Define a matrix A with 5 rows and 6 columns.

```
A =
     1     3     5     7     9    11
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
     5    10    15    20    25    30
>> B=A(:,3)
B =
     5
     6
     9
    12
    15
>> C=A(2,:)
```

Define a column vector B from the elements in all the rows of column 3 in matrix A.

Define a row vector C from the elements in all the columns of row 2 in matrix A.

Table 2-2: Built-in functions for handling arrays

Function	Description	Example

`length(A)`	Returns the number of elements in the vector `A`.	`>> A=[5 9 2 4];`
		`>> length(A)`
		`ans =`
		`4`

`size(A)`	Returns a row vector `[m,n]`, where `m` and `n` are the size $m \times n$ of the array `A`.	`>> A=[6 1 4 0 12; 5 19 6 8 2]`
		`A =`
		`6 1 4 0 12`
		`5 19 6 8 2`
		`>> size(A)`
		`ans =`
		`2 5`

Tutorial 3-1: Multiplication of arrays.

```
>> A=[1 4 2; 5 7 3; 9 1 6; 4 2 8]
A =
     1     4     2
     5     7     3
     9     1     6
     4     2     8
>> B=[6 1; 2 5; 7 3]
B =
     6     1
     2     5
     7     3
>> C=A*B
C =
    28    27
    65    49
    98    32
    84    38

>> D=B*A
??? Error using ==> *
Inner matrix dimensions must agree.
```

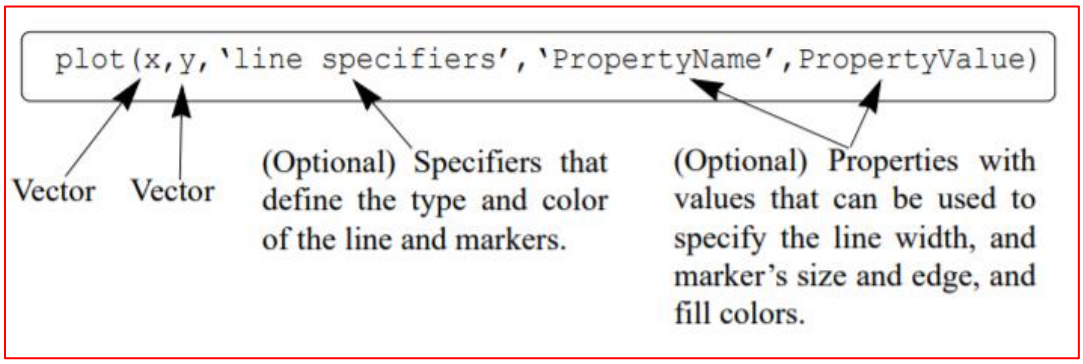
Define a 4 × 3 matrix A.

Define a 3 × 2 matrix B.

Multiply matrix A by matrix B and assign the result to variable C.

Trying to multiply B by A, B*A, gives an error since the number of columns in B is 2, and the number of

PLOTTING



Line Specifiers:

Line specifiers are optional and can be used to define the style and color of the line and the type of markers (if markers are desired). The line style specifiers are:

Line Style	Specifier
solid (default)	-
dashed	--

Line Style	Specifier
dotted	:
dash-dot	-.

Line Color	Specifier
red	r
green	g
blue	b
cyan	c

Line Color	Specifier
magenta	m
yellow	y
black	k
white	w

The marker type specifiers are:

Marker Type	Specifier		Marker Type	Specifier
plus sign	+		square	s
circle	o		diamond	d
asterisk	*		five-pointed star	p
point	.		six-pointed star	h
cross	x		triangle (pointed left)	<
triangle (pointed up)	^		triangle (pointed right)	>
triangle (pointed down)	v			

Notes about using the specifiers:

- The specifiers are typed inside the `plot` command as strings.
- Within the string the specifiers can be typed in any order.
- The specifiers are optional. This means that none, one, two, or all the three can be included in a command.

Some examples:

`plot(x,y)`	A blue solid line connects the points with no markers (default).
`plot(x,y,'r')`	A red solid line connects the points.
`plot(x,y,'--y')`	A yellow dashed line connects the points.
`plot(x,y,'*')`	The points are marked with * (no line between the points).
`plot(x,y,'g:d')`	A green dotted line connects the points that are marked with diamond markers.

```
plot(x,y,'-mo','LineWidth',2,'markersize',12,
      'MarkerEdgeColor','g','markerfacecolor','y')
```

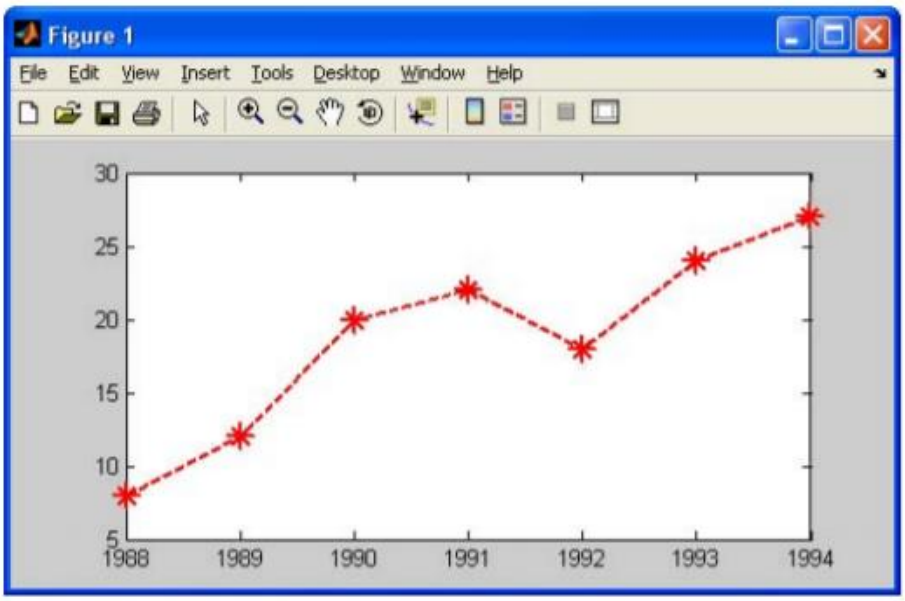
creates a plot that connects the points with a magenta solid line and circles as markers at the points. The line width is two points and the size of the circle markers is 12 points. The markers have a green edge line and yellow filling.

```
>> yr=[1988:1:1994];
>> sle=[8   12   20   22   18   24   27];
>> plot(yr,sle,'--r*','linewidth',2,'markersize',12)
>>
```

Line Specifiers: dashed red line and asterisk marker.

Property Name and Property Value: the line width is 2 points and the markers size is 12 point.

Once the `plot` command is executed the Figure Window with the plot, as shown in Figure 5-3, opens. The plot appears on the screen in red.



```
%  A script file that creates a plot of
%  the function: 3.5.^(-0.5*x).*cos(6x)
x=[-2:0.01:4];
y=3.5.^(-0.5*x).*cos(6*x);

plot(x,y)
```

Create vector x with the domain of the function.

Create vector y with the function value at each x.

Plot y as a function of x.

Once the script file is executed, the plot is created in the Figure Window, as shown in Figure 5-4. Since the plot is made up of segments of straight lines that connect the points, to obtain an accurate plot of a function, the spacing between the elements of the vector x must be appropriate. Smaller spacing is needed for a func-

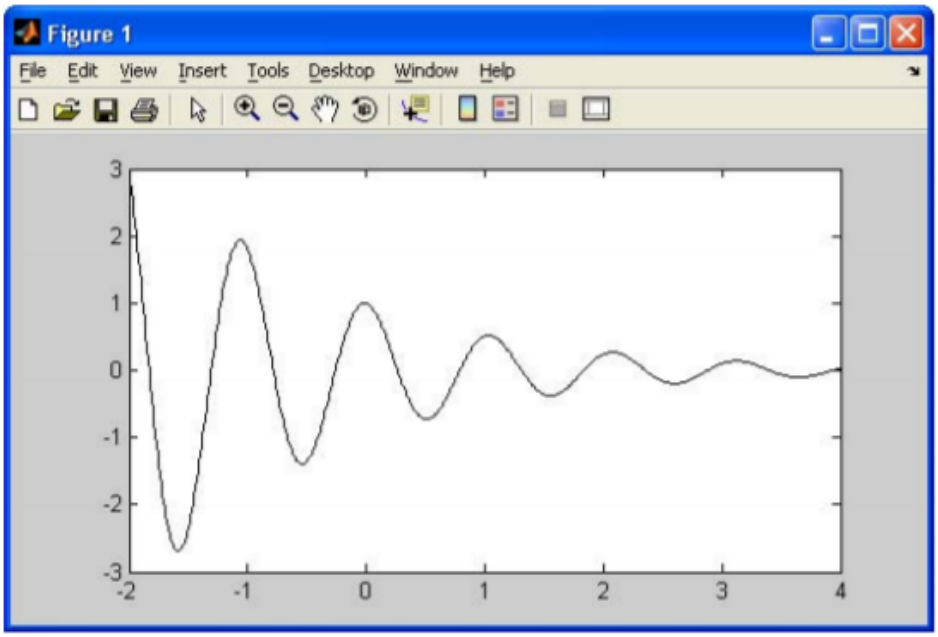


Figure 5-4: The Figure Window with a plot of the function: $y = 3.5^{-0.5x}\cos(6x)$.

```
>>x = [0:0.02:6];
>>y = 5*sin(x);
>>plot(x,y),xlabel('x'),ylabel('y')
```

Command	Description
`axis([xmin xmax ymin ymax])`	Sets the minimum and maximum limits of the *x*- and *y*-axes.
`fplot(function,[xmin xmax])`	Performs intelligent plotting of functions, where `function` is a function handle that describes the function to be plotted and `[xmin xmax]` specifies the minimum and maximum values of the independent variable. The range of the dependent variable can also be specified. In this case the syntax is `fplot(function, [xmin xmax ymin ymax])`.
`grid`	Displays gridlines at the tick marks corresponding to the tick labels.
`plot(x,y)`	Generates a plot of the array `y` versus the array `x` on rectilinear axes.
`plot(y)`	Plots the values of `y` versus their indices if `y` is a vector. Plots the imaginary parts of `y` versus the real parts if `y` is a vector having complex values.
`print`	Prints the plot in the Figure window.
`title('text')`	Puts text in a title at the top of a plot.
`xlabel('text')`	Adds a text label to the *x*-axis (the abscissa).
`ylabel('text')`	Adds a text label to the *y*-axis (the ordinate).