

Computational Methods in Physics

1 Introduction to MATLAB

MATLAB is a software package for high-performance numerical computation and visualization. It provides an interactive environment with hundreds of built-in functions for technical computation, graphics, and animation. Best of all, it also provides easy extensibility with its own high-level programming language. The name MATLAB stands for MATrix LABoratory. The diagram in Fig. 1 shows the main features and capabilities of MATLAB. MATLAB's built-in functions provide excellent tools for linear algebra computations, data analysis, signal processing, optimization, numerical solution of ordinary differential equations (ODEs), quadrature, and many other types of scientific computations. Most of these functions use state-of-the-art algorithms. There are various; functions for 2-D and 3-D graphics, as well as for animation. Also, for those who cannot do without their Fortran or C codes, MATLAB even provides an external interface to run those programs from within MATLAB. The user, however, is not limited to the built-in functions; he can write his own functions in the MATLAB language. Once written, these functions behave just like the built-in functions. MATLAB's language is very easy to learn and to use. There are also several optional "toolboxes" available from the developers of MATLAB. These toolboxes are collections of functions written for special applications such as symbolic computation, image processing, statistics, control system design, and neural networks. The list of toolboxes keeps growing with time. There are now more than 50 such toolboxes. The basic building block of MATLAB is the matrix. The fundamental data type is the array. Vectors, scalars, real matrices, and complex matrices are all automatically handled as special cases of the basic data type. In 2004, MATLAB had around one million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics. MATLAB is widely used in academic and research institutions as well as industrial enterprises.

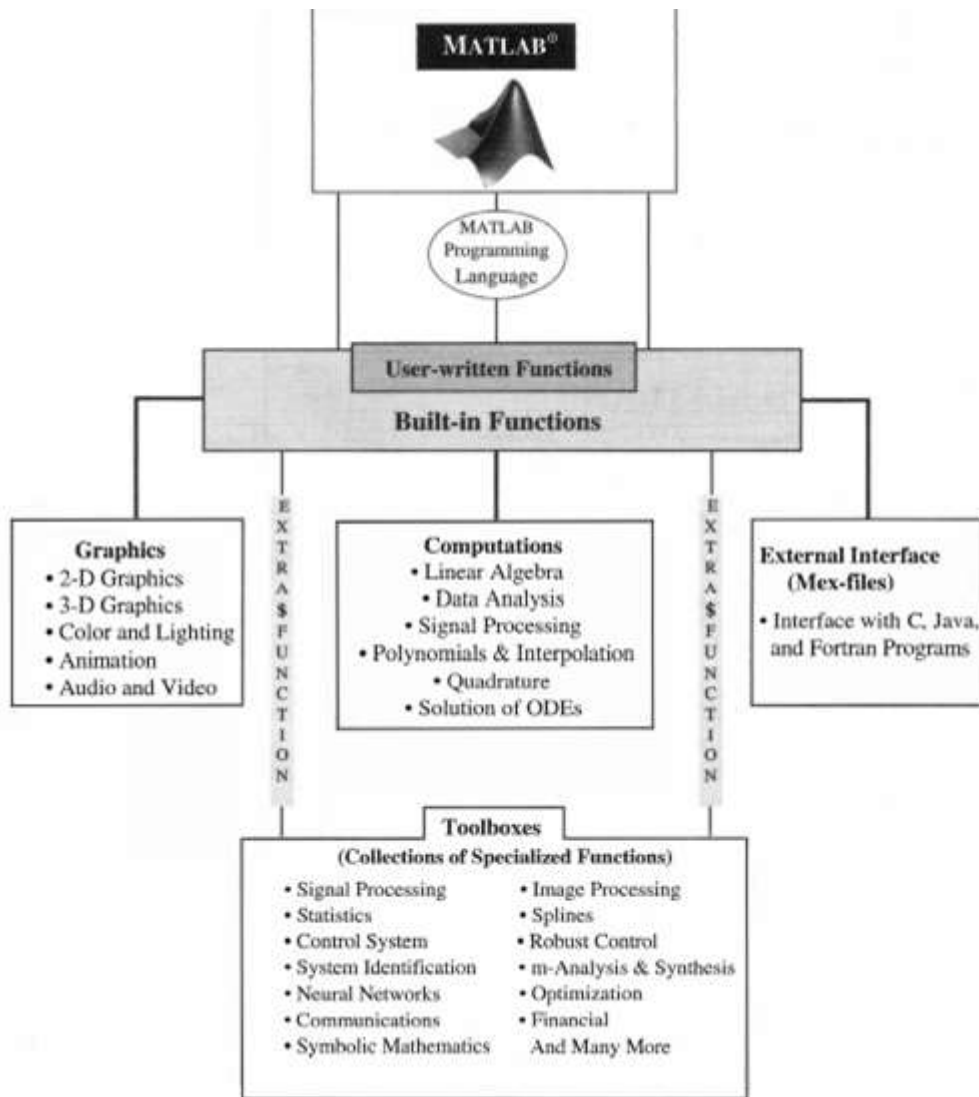



Fig.1 A schematic diagram of MATLAB's main features

1.1 How to run, interrupt, and terminate MATLAB?

To run MATLAB in MS Windows platforms, use MATLAB shortcut on the Windows Start Menu or double click MATLAB icon  on your Windows desktop. To run MATLAB on Linux® platforms, type *matlab* at the operating system prompt.

When you start MATLAB, there are four embedded windows: Command Window, Command History, working Directory, and Workspace (see Fig.2). You can start using MATLAB by issuing your commands on Command Window.

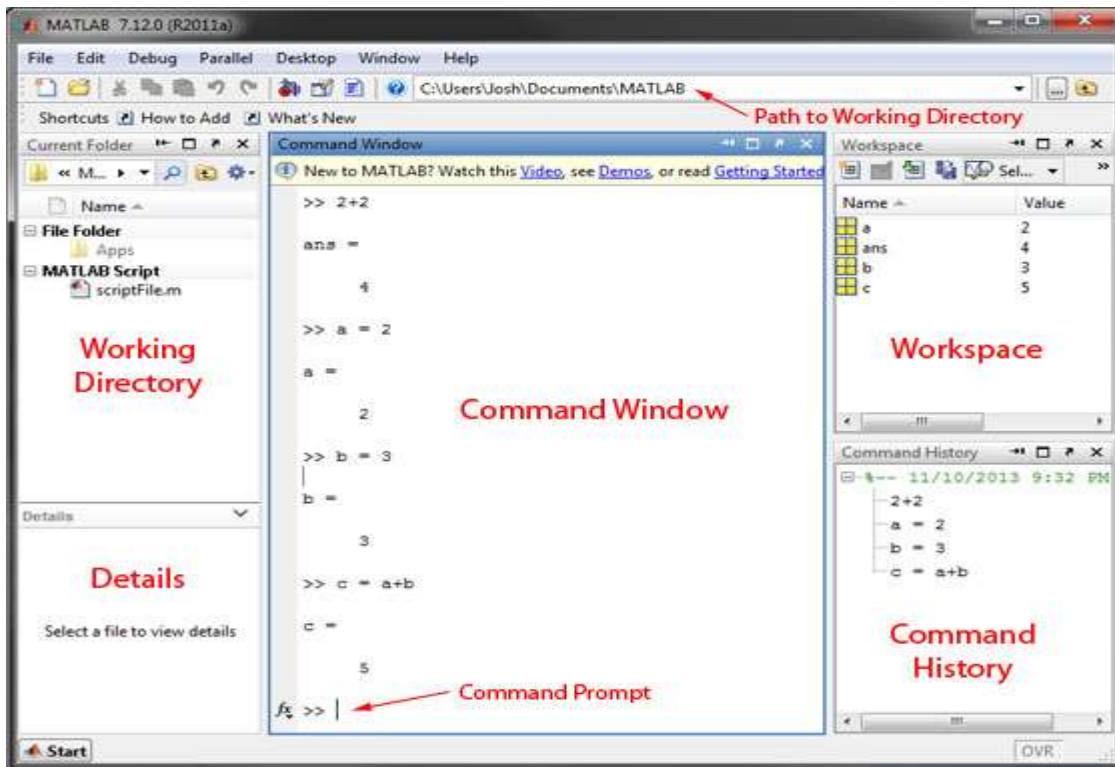


Fig.2 Layout of MATLAB

Typing *quit* or *exit* terminates MATLAB. Another way of terminating MATLAB is to select File>Exit menu.

1.2 MATLAB Basics

- In MATLAB, every expression, or variable, has a type associated with it. By default, numbers are stored as the type double (short for double-precision). The type char is used to store either single characters (e.g., 'x') or strings, which are sequences of characters (e.g., 'cat'). Both characters and strings are enclosed in single quotes. The type logical is used to store true/false values.
- MATLAB supports many types of values, which are called classes. A class is essentially a combination of a type and the operations that can be performed on values of that type. Three different classes of MATLAB data are widely used: floating point numbers, strings, and symbolic expressions.
- MATLAB uses double-precision floating point arithmetic accurate to approximately 15 digits, however, only 5 digits are displayed, by default. To display more digits, type *format long*. Then all subsequent numerical output will have 15 digits displayed. Type *format short* to return to 5-digit display.
- Built-in constants in MATLAB are: *pi*, *i*, *inf*, *NaN* (not a number).

1.2.1 MATLAB windows

On almost all systems, MATLAB works through three basic windows, which are shown in Fig. 3 and discussed here.

1.2.1.1 MATLAB desktop: This is where MATLAB puts you when you launch it (see Fig. 3). The MATLAB desktop, by default, consists of the following subwindows.

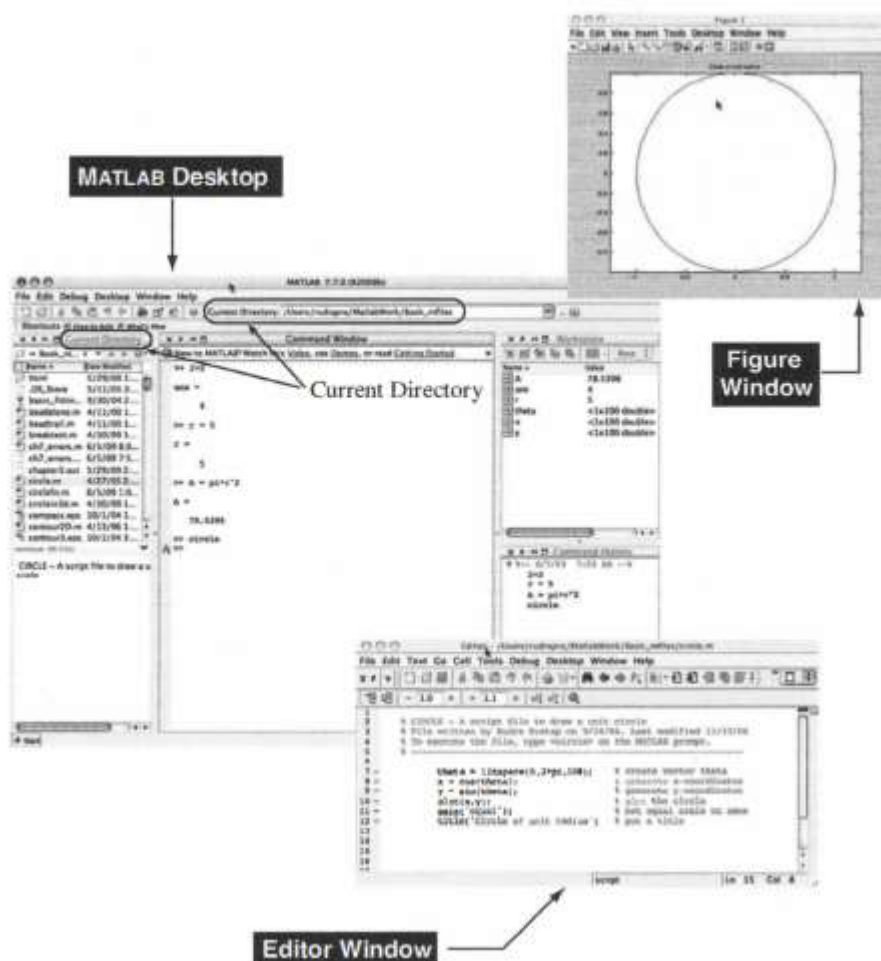
Command window: This is the main window. It is characterized by the MATLAB command prompt (`>`). When you launch the application program, MATLAB puts you in this window. All commands, including those for running user-written programs, are typed in this window at the **MATLAB prompt**. In MATLAB, this window is a part of the MATLAB window (see Fig. 2) that contains other smaller windows or panes.

Working Directory pane: This pane is located on the left of the Command Window in the default MATLAB desktop layout. This is where all your files from the current directory are listed. You can do file navigation here. Make sure that this is the directory where you want to work so that MATLAB has access to your files and where it can save your new files. If you change the working directory (by navigating through your file system), make sure that the selected directory is also reflected in the little window above the Command Window marked working Directory.

Workspace: This subwindow list all the variables that you have generated so far and shows their type and size.

Command history: All the commands typed on MATLAB prompt in the command window get recorded in the command history. You can retrieve your previous data just double click on the command history.

1.2.1.2 Figure window: The output of all graphics commands typed in the command window are flushed to the graphics or figure window, a separate gray window with (default) white background color. The user can create as many figure windows as the system memory will allow.



1.2.1.3 Editor window: This is where you write, edit, create, and save your own programs in files called M-files. You can use any text editor to carry out these tasks. On most systems, MATLAB provides its own built-in editor. However, you can use your own editor by typing the standard file-editing command that you normally

use on your system. From within MATLAB, the command is typed at the MATLAB prompt following the exclamation character (!). The exclamation character prompts MATLAB to return the control temporarily to the local operating system, which executes the command following the character. After the editing is completed, the control is returned to MATLAB.

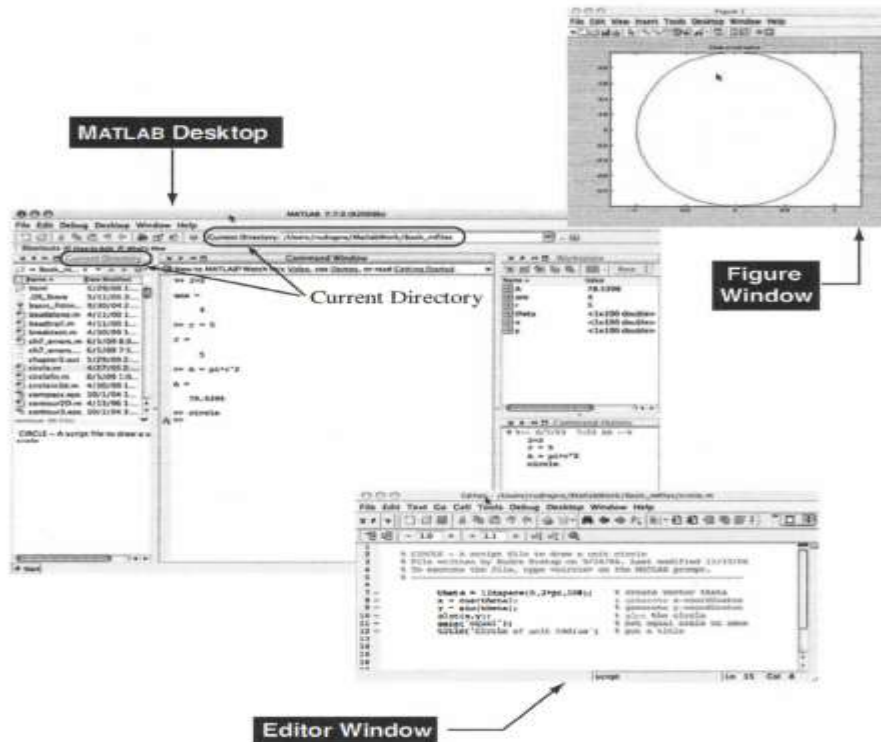


Fig.3 Windows in MATLAB

2 Vectors and Matrices in MATLAB

It is possible to deal with vectors (1 dimensional arrays), matrices (2 dimensional arrays), and higher dimensional arrays easily in MATLAB. A vector in MATLAB is equivalent to what is called a one-dimensional array in other languages. A matrix is equivalent to a two-dimensional array. You can enter a vector of any length in MATLAB by typing a list of numbers, separated by commas or spaces, inside square brackets. For example,

```
>> u = [1,6,13,4] % Vector u is defined separated by commas
u =
1 6 13 4

>> v = [2 -3 4 -5 6 -7] % Vector v is defined separated by spaces
v =
2 -3 4 -5 6 -7

>> newv = [u v] % Concatenating vectors
newv =
```

```

1 6 13 4 2 -3 4 -5 6 -7

>> u(3)% Extracting the elements of a vector, i.e., u(3)
ans =
13

>> u=[1:5]%Generate a vector of equally-spaced elements with colon operator
u =
1 2 3 4 5

>> u=[1:2:10] % Increment by 2
u =
1 3 5 7 9

>> transu = u' % Get the tranpose of u
transu=
1
3
5
7
9

```

To type a matrix you must: begin with a square bracket, separate elements in a row with commas or spaces, use a semicolon to separate rows, end the matrix with another square bracket :

```

>>A = [1 2 3; 4 5 6; 7 8 9]
A =
1 2 3
4 5 6
7 8 9

>>A(2,:) % To publish 2nd row of the a matrix
ans =
4 5 6

>>A(:,3) % To publish 3rd column of the a matrix
ans =
3
6

```

MATLAB has several commands that generate special matrices. The commands `zeros(n,m)` and `ones(n,m)` produce $n \times m$ matrices of zeros and ones, respectively. Also, `eye(n)` represents the $n \times n$ identity matrix, `rand(n,m)` generates a matrix with all entries random numbers in $[0,1]$.

The `length(vector)` and `size(matrix)` functions in MATLAB are used to find array dimensions. The length function returns the number of elements in a vector. The size function returns the number of rows and columns in a matrix.

```
>> length (u)
ans=
5

>> size(A)
ans=
3 3
```

Operations like `+`, `-`, `*`, `/`, and `^` can be carried out in a matrix sense (according to the rules of matrix algebra) or elementwise. When elementwise operation is carried out in MATLAB, a period precedes the operator: `./`, `.-`, `.*`, `./`, and `.^`.

2.1 MATLAB Functions

In MATLAB you will use built-in functions as well as functions that you create yourself. MATLAB has many built-in functions, typing `help elfun` and/or `help specfun` calls up full lists of elementary and special functions. These include `sqrt`, `cos`, `sin`, `tan`, `log`, and `exp`.

For the user-defined functions, you can use `inline ('function', 'independent variable')` command:

```
>> f = inline('x^2 + 2*x + 1', 'x')
f =
Inline function:
f(x) = x^2 + 2*x + 1
>> f(4) % Once the function is defined, you can evaluate it
ans =
25
```

2.2 Symbolic Computation in MATLAB

You can carry out algebraic or symbolic calculations in MATLAB, such as simplifying polynomials, differentiation with `diff` function, integration with `int` function or solving algebraic equations. To find out about the `int` function, for example, from the Command Window:

```
>> help sym/int
```

To perform symbolic computations, use `syms` to declare the variables you plan to use as symbolic variables. Some of the important functions are: `simplify`, `subs`, `solve`, `diff` and `int`.

```
>> syms x y
>> (x-y)*(x+y)*(x^2+2*x+1)
ans =
(x + y)*(x - y)*(x^2 + 2*x + 1)
```

```

>> f=simplify((x-y)*(x+y)*(x^2+2*x+1))% to simplify the expression
f =
(x^2 - y^2)*(x + 1)^2
>> subs(f, x, 2) % to substitute x=2 in f
ans =
36 - 9*y^2
>> solve(f) % to solve f=0 with respect to x
ans =
y
-1
-1
-y
>> f1=diff(f,x) % to differentiate f with respect to x
f1 =
(2*x + 2)*(x^2 - y^2) + 2*x*(x + 1)^2
>> f2=int(f,x) % to integrate f with respect to x
f2 =
x^4/2 - x*y^2 - x^3*(y^2/3 - 1/3) - x^2*y^2 + x^5/5

```

2.3 Input/Output

It is possible to write programmes that accept input from the user and produce informative output. Statements that are called input/output statements are used for these tasks with MATLAB functions *input* and *fprintf*.

```

>> rad = input('Enter the radius: ')
Enter the radius: 5
rad =
    5
>> name=input('Enter your name: ') % to enter characters
Enter your name: 'basak'
Name=
    basak
>> fprintf('The value of six square is %d\n',36)
The value of six square is 36
>> fprintf('Six square is %3d and the square root of 2 is %6.2f\n',36,1.47)
Six square is  36 and the square root of 2 is   1.47

```

The character '\n' at the end of the string is a special character called the newline character; when it is printed the output moves down to the next line. The %d in fprintf is sometimes called a placeholder; it specifies where the value of the expression that is after the string is to be printed. The character in the placeholder is called the conversion character, and it specifies the type of value that is being printed. A list of the simple placeholders:

```

%d integers (it actually stands for decimal integer)
%f floats
%c single characters
%s strings

```

3 MATLAB M-Files

For complicated problems, the simple editing tools provided by the Command Window are insufficient. A much better approach is to create an M-file which are ordinary text files containing MATLAB commands with .m extension. You can create and modify them using any text editor or word processor that is capable of saving files as plain ASCII text. There are two different kinds of M-files: script M-files and function M-files.

3.1 Script M-files

The simplest MATLAB programs are called scripts which are stored in M-files. Script M-files execute a series of MATLAB statements without any input and output arguments. These scriptfiles are interpreted by MATLAB interpreter line by line, rather than compiled. Therefore, the correct terminology is that these are scripts, and not programs. The contents of a script can be displayed in the Command Window using the `type` command. The script can be executed, or run, by simply entering the name of the file (without the `.m` extension).

To create a script, click File, then New, then M-file. A new window will appear called the Editor. To create a new script, simply type the sequence of statements (notice that line numbers will appear on the left). When finished, save the file using File and then Save. Make sure that the extension `.m` is on the filename (this should be the default). The rules for filenames are the same as for variables (they must start with a letter, after that there can be letters, digits, or the underscore, etc.). By default, scripts will be saved in the Work Directory. If you want to save the file in a different directory, the Current Directory can be changed.

3.2 Function M-files

Function M-files accept input arguments and produce output. Note that it is appropriate to use inline functions for defining simple functions that can be expressed in one line. Function M-files are useful for defining functions that require several commands to compute the output. The first line in a function M-file is called the function definition line; it defines the function name, as well as the number and order of input and output arguments:

```
function [output_parameters] = function_name (input_parameter)
```

A function is distinguished by the *function* keyword. MATLAB cannot execute a function unless it knows where to find its M-file. Make sure that you introduce the path of your M-file from MATLAB menu File>Set Path.

functions that require several commands to compute the output. The first line in a function M-file is called the function definition line; it defines the function name, as well as the number and order of input and output arguments:

```
function [output_parameters] = function_name (input_parameter)
```

A function is distinguished by the *function* keyword. MATLAB cannot execute a function unless it knows where to find its M-file. Make sure that you introduce the path of your M-file from MATLAB menu File>Set Path.

4. Plotting in MATLAB

There are several plot functions in MATLAB beginning with “ez” that plot symbolic expressions. For example, the function *ezplot* will draw a 2-D plot in the x-range from $-2p$ to $2p$, with the expression as the title (in pretty form).

Function *ezplot* expects a string or a symbolic expression representing the function to be plotted. The string form notation is *ezplot ('function', interval)* where specifying interval is optional. For example, to graph $x^2 + 2x + 1$ on the interval -2 to 2 using the string form:

```
>> ezplot ('x^2 + 2*x + 1', [-2 2])
```

The command *plot* produces 2D graphics. Before using plot command, define the interval for the independent variable x and the function of the form $y=f(x)$. Then *plot (x,y)* command is called to obtain the figure of $f(x)$ with respect to x :

```
>> x = 0:0.1:2*pi;  
>> y = sin(x);  
>> plot (x,y)
```

```

>> x = 1:5;
>> y = [0 -2 4 11 3];
>> z = 2:2:10;
>> plot3(x,y,z,'k*')
>> grid

```

Each time you execute a plotting command, MATLAB erases the old plot and draws a new one. If you want to overlay two or more plots, type **hold on**. Figures are displayed in Figure Window which has its own plot editor. You can format your figure using this editor.

A useful plotting function is subplot, which creates a matrix of plots in the current Figure Window. Three arguments are passed to it in the form subplot(r,c,n); where r and c are the dimensions of the matrix and n is the number of the particular plot within this matrix.

MATLAB has several other plotting functions: *fplot(similar to plt)*, *subplot(multiple plots on the same window)*, *plot3(3D plots)*, *ezplot3(3D plots)*, *mesh(3D plots)*, *surf(3D plots)*, *contour*, and, *ezcontour*

You can have a title on a graph, label each axis, change the font and font size, set up the scale for each axis and have a legend for the graph. You can also have multiple graphs per page.

5 User defined functions in matlab

A user defined function is a Matlab program that is created by the user, saved as a function file, and then can be used like a built-in function. A function in general has input arguments (or parameters) and output variables (or parameters) that can be scalars, vectors, or matrices of any size. There can be any number of input and output parameters, including zero. Calculations performed inside a function typically make use of the input parameters, and the results of the calculations are transferred out of the function by the output parameters.

5.1 Writing a Function File

A function file can be written using any text editor (including the Matlab Editor). The file must be in the Matlab Path in order for Matlab to be able to locate the file. The first executable line in a function file must be the function definition line, which must begin with the keyword function. The most general syntax for the function definition line is:

Syntax:

The first line of a function m-file has the form:

function [outArgs] = funName(inArgs)

outArgs are enclosed in []

- outArgs is a comma-separated list of variable names
- [] is optional if there is only one parameter
- functions with no outArgs are legal

inArgs are enclosed in ()

- inArgs is a comma-separated list of variable names
- functions with no inArgs are legal

Example:

The quadraticRoots function is contained in quadraticRoots.m and can be called from the command line, or another m-file.

```
>> c1 = 1;
>> c2 = 5;
>> c3 = 2;
>> r = quadraticRoots(c1,c2,c3);
r =
    -0.4384    -4.5616
```

```
function x = quadraticRoots(a,b,c)
% quadraticRoots Compute the two roots
% of the quadratic equation
%
% Input: a,b,c = (scalar) coefficients of
% the quadratic equation
% a*x^2 + b*x + c = 0
%
% Output: x = vector of the two roots:
%
d = sqrt( b^2 - 4*a*c);
x1 = (-b + d)/(2*a);
x2 = (-b - d)/(2*a);
x = [x1, x2];
end
```

Example: Defining the function

Let us write simple, user-defined functions for computing the root mean square error (RMSE) and mean absolute error (MAE) between two input vectors of the same size.

```
function [rmse, mae]=calcerrors(x, y)
% The function computes the RMSE and MAE between two input
% vectors of the same time
rmse=sqrt(sum((x-y).^2)/length(x));
mae=sum(abs(x-y))/length(x);
```

Example of calling the function

```
>> t=0:0.1:5;
>> u=exp(-t).*cos(2*pi*t);
>> plot(t,u)
>> noise=0.05*randn(size(t));
>> un=u+noise;
>> hold on;
>> plot(t,un,'r');
>> [err1, err2]=calcerrors(u,un)
```

5.2 Anonymous Functions

An anonymous function is a simple, typical a single line, user-defined function that is defined and written within the computer code (not in a separate file) and is then used in the code. Anonymous functions can be defined in any part of MATLAB (in the Command Window, in script files, and inside regular user-defined functions). Anonymous functions have been introduced in MATLAB 7. They have several advantages over inline function (to be discussed next). Right now both anonymous and inline functions can be used, but inline function will gradually be phased out. An anonymous function is created by typing the following statement:

functionName = @(var1,var2,...) expression

where 'functionName' is the name of the anonymous function, 'var1', 'var2', etc. are a comma separated list of arguments of the function, and 'expression' is a single mathematical expression involving those variables. The expression can include any built-in or user-defined functions.

The above command creates the anonymous function, and assigns a handle for the function to the variable name on the left of the = sign. Function handles provide means for using the function, and passing it to other functions. The expression can include predefined variables that are already defined when the anonymous function is defined.

For example, if three variables a, b, and c have been assigned values, then they can be used in the expression of the anonymous function:

```
a = 3; b = 4; c = 5;
parabola = @ (x) ( a * x + b ) * x + c
```

It is important to note that MATLAB captures the values of the predefined variables when the anonymous function is defined. This means that if subsequently new values are assigned to the predefined variables, the anonymous function is not changed. So in the above example, the parabola is always defined so that the three coefficients, a, b, and c are given by 3, 4, and 5, respectively, even though the values of a, b, and c may be altered subsequently. The anonymous function has to be redefined in order for the new values of the predefined variables to be used in the expression.

For example,

```
>> FA = @ (x) exp(x.^2)./sqrt(x.^2+5)
gives the response
```

```
FA = @(x)exp(x.^2)./sqrt(x.^2+5)
```

Then this anonymous function can be used as follows:

```
>> p = FA(2)
```

which gives the result

```
p = 18.1994
```

and

```
>> Pvec = FA([1 0.5 2])
```

which gives the result

```
Pvec = 1.1097 0.5604 18.1994
```

5.3 Inline Functions

Similar to anonymous function, inline function is a simple single-line user-defined function that is defined without creating a separate function (M-file). Inline functions are gradually being replaced by anonymous functions. An inline function is created with the **inline** command according to the following format:

```
name = inline('mathematical expression typed as a string')
```

A simple example, is:

```
CUBE = inline('X.^3')
```

which calculates the cubic power of an input scalar, vector, or matrix.

5.4 Recursive Functions

Functions can be recursive, that is, they can call themselves. Recursion is a powerful tool, but not all computations that are described recursively are best programmed this way. We will consider the problem of adaptive numerical quadrature as an example of using a recursive function. Numerical quadratures are techniques for numerically computing the integrals of functions. I wrote two such quadratures based on the Gauss-Legendre algorithm, one using a 10-point and the other one a 12-point formula:

gauss10pts(fcn, a, b) gauss12pts(fcn, a, b)

Each of these functions compute numerically the integral of the function fcn from a to b. The function fcn is either a function handle or a string containing the name of the function handle of the integrand function. The more points one uses the more accurate the result is expected to be. Using these two quadratures, I wrote an adaptive quadrature function that computes the integral of a function from a to b to an accuracy specified by a tolerance, tol.

```
function [I,Ct] = adaptiveQuadG2(fcn,a,b,tol,Ct)
% Gander-Gautschi adaptive quadrature with robust
% machine independent termination criterion, &
% avoids arbitrary limits on depth of recursion.
% % fcn = (string) name or file-handle of integrand function
% a = the lower limit.
% b = the upper limit.
% tol = the tolerance (largest possible error).
% Ct = counter to keep track of the total number of times adaptiveQuadG2 calls itself.
% Total number of function evaluations is 2*(10+12)*ct=44*ct.
% Usage: [I,Ct] = adaptiveQuadG2('invSqrt',0,1,eps,0)
% K. Ming Leung, 08/11/03
% I1 = gauss2pts(fcn, a, b); % optimal choice of points
% I2 = gauss4pts(fcn, a, b); % depends on integrand smoothness

I1 = gauss10pts(fcn, a, b);
I2 = gauss12pts(fcn, a, b);
Ct = Ct+1; % increase counter Ct by 1.
if abs(I2-I1) < tol % works better in Matlab
I = I2;
else
m = a + 0.5*(b-a);
[T1,Ct] = adaptiveQuadG2(fcn,a,m,tol,Ct);
[T2,Ct] = adaptiveQuadG2(fcn,m,b,tol,Ct);
I = T1+T2;
end
```

6 Visuallization and plotting in Matlab

Plotting in Matlab

To use the 'plot' function in Matlab, you should first make sure that the matrices/vectors you are trying to use are of equal dimensions. For example, if I wanted to plot vector $X = [3 \ 9 \ 27]$ over time, my vector for time would also need to be a 1x3 vector (i.e. $t = [1 \ 2 \ 3]$).

Syntax

To plot the example vectors above in a new figure:

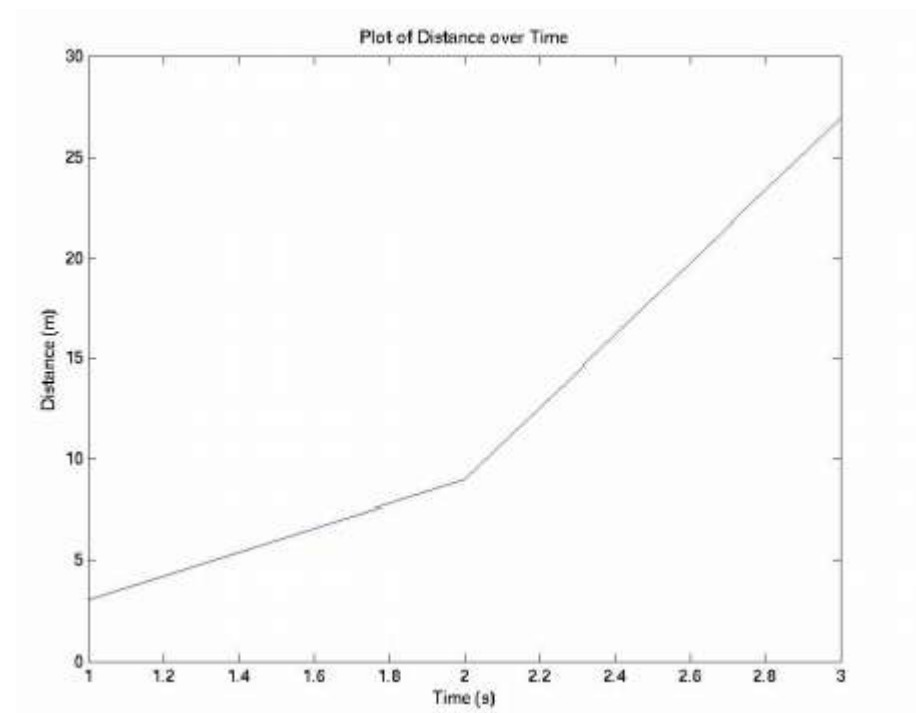
```
clear all % clear all previous variables
```

```
X = [3 9 27]; % my dependent vector of interest  
t = [1 2 3]; % my independent vector figure  
figureplot(t, X) % create new
```

Labeling Axes

To give the above figure a title and axis labels:

```
title('Plot of Distance over Time') % title  
ylabel('Distance (m)') % label for y axis  
xlabel('Time (s)') % label for x axis
```



Legends

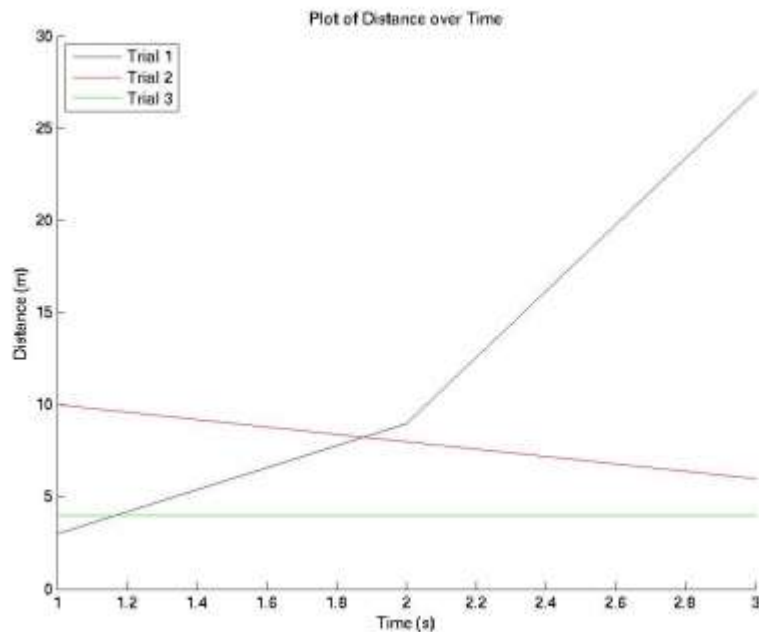
Legends If you have plotted multiple dependent vectors on the same plot and want to distinguish them from each other via a legend, the syntax is very similar to the axis labeling above. It is also possible to set colors for the different vectors and to change the location of the legend on the figure.

```
clear all  
X = [3 9 27]; % dependent vectors of interest  
Y = [10 8 6];  
Z = [4 4 4];  
t = [1 2 3]; % independent vector  
figure  
hold on % allow all vectors to be plotted in same % figure  
plot(t, X, 'blue', t, Y, 'red', t, Z, 'green')
```

```

title('Plot of Distance over Time') % title
ylabel('Distance (m)') % label for y axis
xlabel('Time (s)') % label for x axis
legend('Trial 1', 'Trial 2', 'Trial 3')
legend('Location','NorthWest') % move legend to upper left

```



Subplots

It can sometimes be useful to display multiple plots on the same figure for comparison. This can be done using the subplot function that takes arguments for number of rows of plots, number of columns of plots, and plot number currently being plotted:

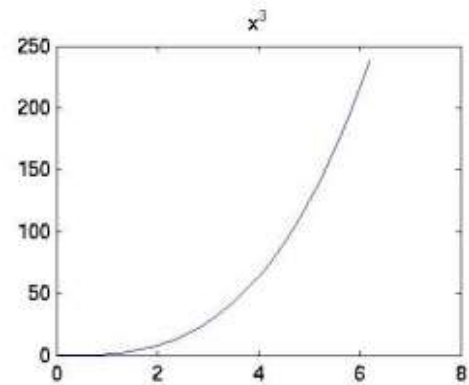
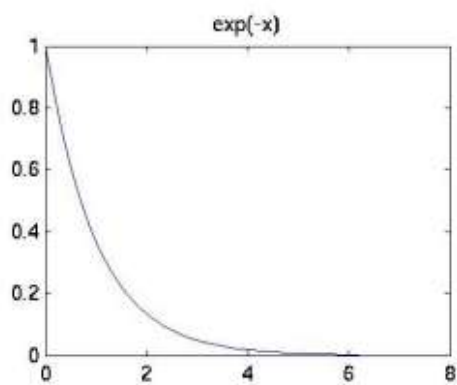
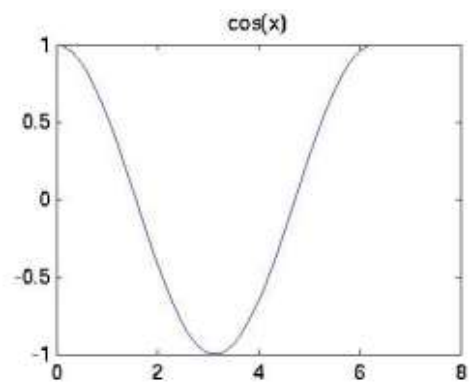
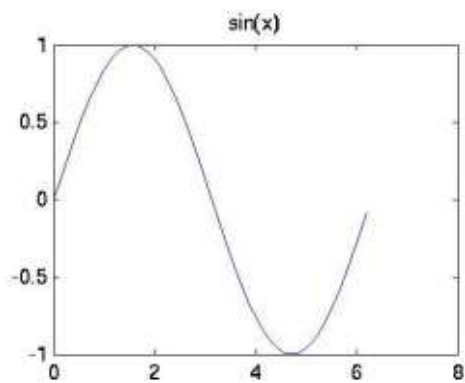
Example:

clear all close all

```

% subplot (nrows,ncols,plot_number)
x=0:.1:2*pi; % x vector from 0 to 2*pi, dx = 0.1
subplot(2,2,1); % plot sine function
plot(x,sin(x));
subplot(2,2,2); % plot cosine function
plot(x,cos(x));
subplot(2,2,3) % plot negative exponential function
plot(x,exp(-x));
subplot(2,2,4); % plot x^3
plot(x, x.^3)

```



Plotting in 3-D

There are also ways to plot in multiple dimensions in Matlab*. One type of 3-D plot that may be useful is a surface plot, which requires you to generate some kind of x-y plane and then apply a 3rd function as the z dimension.

Example:

```
clear all
```

```
close all
```

```
[x,y] = meshgrid([-2:.2:2]); % set up 2-D plane
```

```
Z = x.*exp(-x.^2-y.^2); % plot 3rd dimension on plane
```

```
figure
```

```
surf(x,y,Z,gradient(Z)) % surface plot, with gradient(Z)
```

```
% determining color distribution
```

```
colorbar % display color scale, can adjust
```