ITS Information Technology Service

Guide 7

Version 3

An introduction to MATLAB

MATLAB is a high-performance language for technical computing. Its name stands for *matrix laboratory* and its basic data element is the array. MATLAB also is an interactive system that consists of a Development Environment, a mathematical function library, graphics facilities, a programming language and facilities to interact with C and FORTRAN programs.

This guide provides an introduction for beginners to some of the tools of the Development Environment and to the MATLAB programming language.

Previous programming experience will be useful but is not essential. As MATLAB is oriented towards working with arrays and matrices basic knowledge of Linear Algebra is assumed. The guide is based on MATLAB installed on the Windows platform but the user interface on the Linux platform is very similar. Some experience with Windows is therefore assumed.

Durham University

Document code: **Guide 7**
Title: **An introduction to MATLAB**
Version: **3**
Date: **09/10/2009**
Produced by: **University of Durham Information Technology Service**

Conventions:

In this document, the following conventions are used:

- A **bold typewriter font** is used to represent the actual characters you type at the keyboard.
- A ***slanted typewriter font*** is used for items such as example filenames which you should replace with particular instances.
- A typewriter font is used for what you see on the screen.
- A **bold font** is used to indicate named keys on the keyboard, for example, **Esc** and **Enter**, represent the keys marked Esc and Enter, respectively.
- Where two keys are separated by a forward slash (as in **Ctrl/B**, for example), press and hold down the first key (**Ctrl**), tap the second (**B**), and then release the first key.
- A **bold font** is also used where a technical term or command name is used in the text.

Contents

1 Introduction

MATLAB is installed on the ITS Linux and HPC services with a limited number of licences available for research purposes. The ITS Networked PC service (NPCS) has a number of licences available which are for teaching and training purposes only. A Student edition of MATLAB exists, which is essentially the professional package but with certain limitations. For up to date information visit the **Mathworks** web site (www.mathworks.com).

1.1 Starting a MATLAB session

On the Linux service you can start MATLAB from the desktop menu at **Durham > Mathematics > MATLAB** or by typing **matlab** at the prompt in a terminal window.

On the NPCS computers MATLAB is available from the **Start > Programs > Programming Languages > MATLAB** menu.

Further information about availability can be found on the ITS **MATLAB** Application web page.

1.2 Closing MATLAB

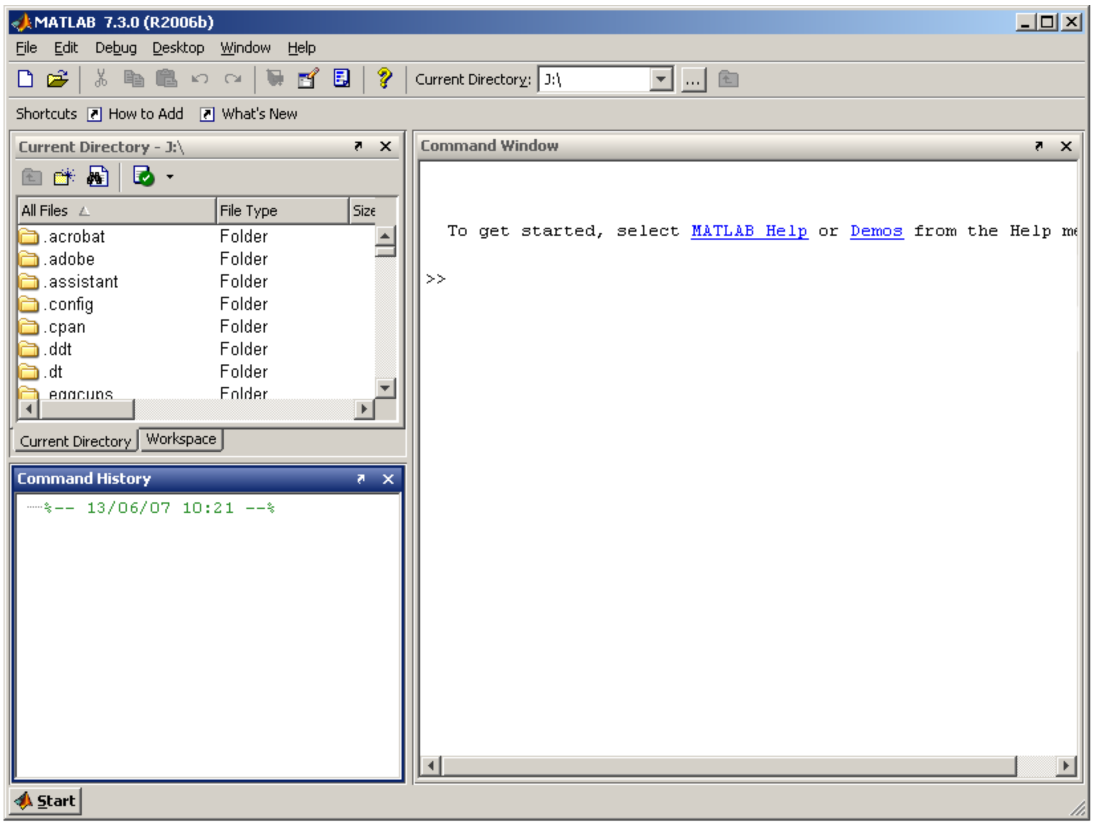
The MATLAB application can be closed in various ways. The program will first ask whether any modified files have to be saved. It is also possible to close individual tools.

The main application can be closed

- By clicking the **Close** button in the top right side of the title bar.
- From the **File** menu select the last item **File > Exit MATLAB**.
- Via the keyboard by pressing **Ctrl/Q**.
- By typing **quit** at the **Command** Window prompt (Section **1.10**).

1.3 The MATLAB Desktop

The Graphical User Interface (GUI) display of MATLAB is called the MATLAB Desktop. It contains various toolbars and windows where the user can interact with the program. Several of these components are outlined in the figure below.



All Files	File Type
.acrobat	Folder
.adobe	Folder
.assistant	Folder
.config	Folder
.cpan	Folder
.ddt	Folder
.dt	Folder
.eggcups	Folder

```
%-- 13/06/07 10:21 --%
```

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

>>
```

The little pictures that decorate the buttons are called icons and their purpose is to give a visual hint to the button's function.

The main area of the Desktop is the main window which hosts most of the MATLAB Tools. These can be arranged in various ways.

The other objects on the Desktop are

The Title bar

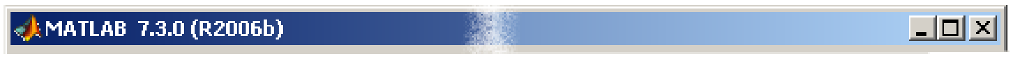
The Menu Bar

The Toolbar

Shortcut Bar

Status Bar

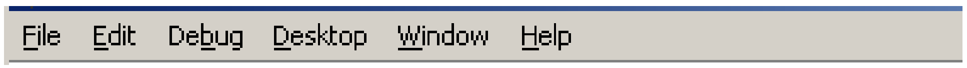
1.4 The Title Bar



In the release of MATLAB for Windows the Title Bar displays the icon, name and version of MATLAB on the left and three system buttons on the right. The three buttons are for minimizing the program, toggling between maximizing and restoring the window and for closing MATLAB respectively. Clicking on the minimize button will hide the window and it can be restored by clicking on the MATLAB button on the Taskbar.

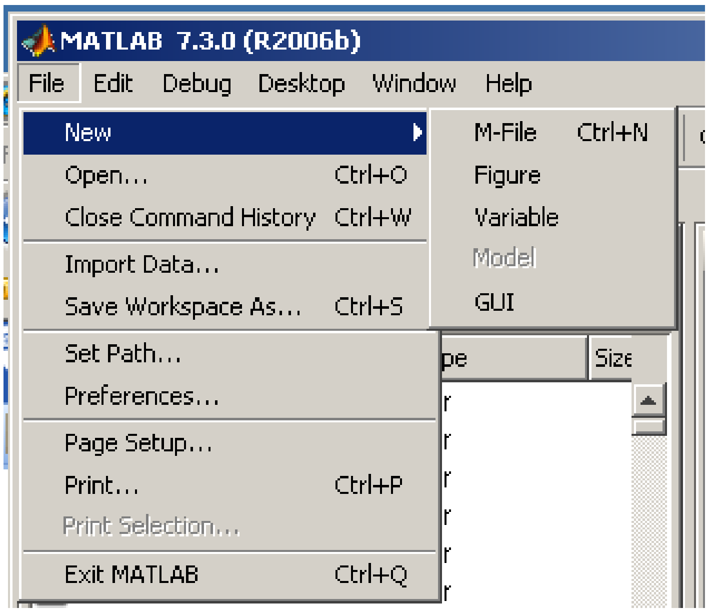
1.5 The Menu Bar

The **menu bar** consists of a number of dynamically changing menu items. The **File**, **Edit**, **Window** and **Help** menus are usually present, whereas additional menus can appear between the **Edit** and **Window** menus depending on the particular MATLAB Tool that is currently active.



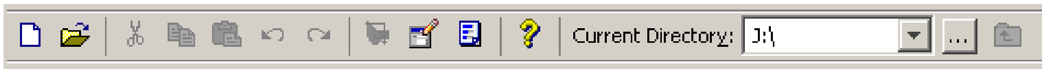
Clicking on an item displays a list of sub-items to toggle a particular setting, or to let the program perform an action or display a further list of menu items.

The expanded **File** menu is shown below as an example.



Where relevant some of the other menus will be discussed in later sections.

1.6 The Toolbar

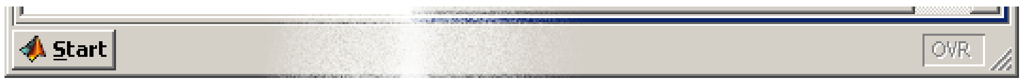


The toolbar is located underneath the Menu Bar and consists of groups of buttons or other Window tools. The buttons are identified with icons and are short cuts to the most used menu items that initiate actions. Holding the mouse briefly over a Button icon displays a Tooltip, a small window with some text giving a brief explanation of the function of the Button.

1.7 The Shortcut Toolbar

This Toolbar contains a number of buttons with short cuts. This is a facility to add custom shortcuts that execute a sequence of commonly used MATLAB commands.

1.8 The Status Bar



The Status Bar occupies the bottom border of the MATLAB Desktop. It contains the Start button on the far left and next to this an area where informative messages are displayed including what action the program is currently performing.

1.8.1 The Start Button

Clicking the MATLAB Start button displays a menu for easy access to tools, demos, documentation and installed Toolboxes.

1.9 Desktop Tools

The MATLAB Desktop contains tools for managing programs, files and applications associated with MATLAB. Some of the tools are

Command Window

Command History

Current Directory

Workspace

Help

Profiler

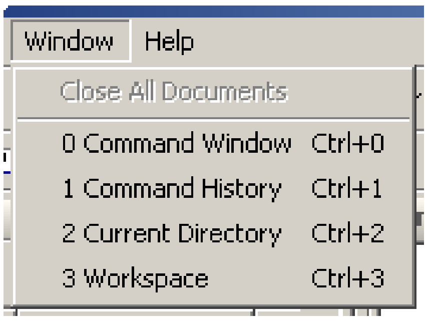
Editor

Start Button

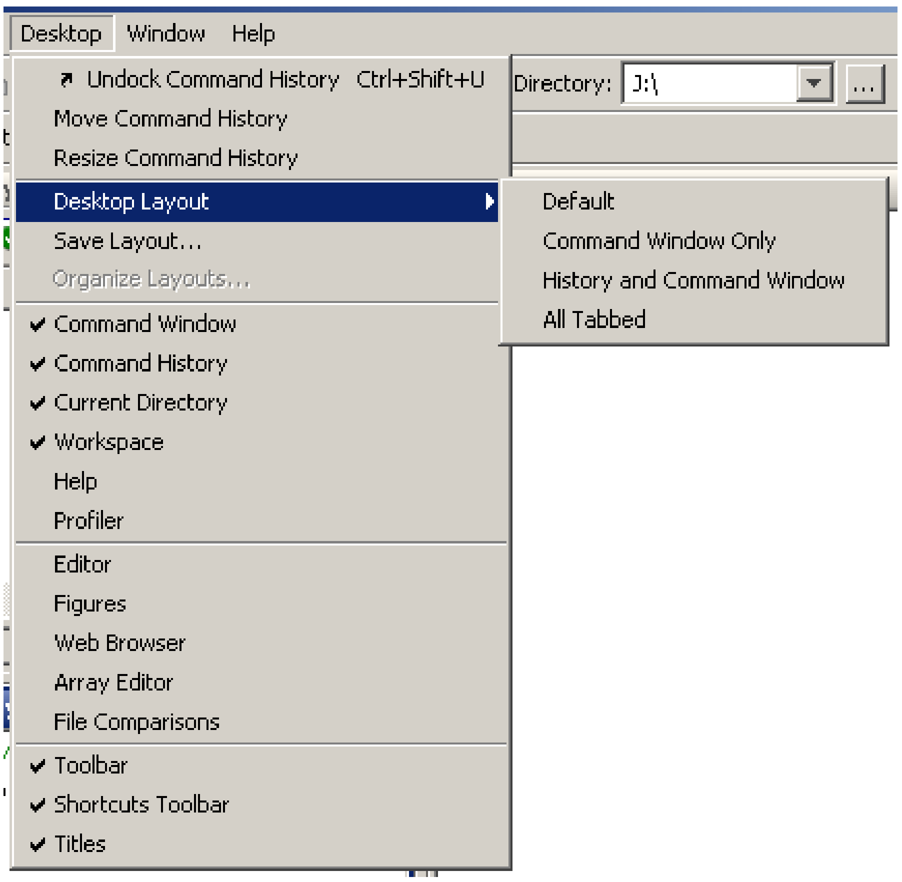
In this basic beginner's guide only the first five tools of this list will be discussed in the next sections.

1.9.1 Activating Tool Windows

To work with one of the Desktop Tool windows it has to be activated first. It will then receive all keyboard input. If a window is active its Title Bar will be highlighted. A particular window can be activated by clicking on it with the mouse, or, with the keyboard by entering a combination of the **Ctrl** key and a digit ranging from 0 to 5, depending on what Desktop Tools are currently visible. In the example below only four Tool windows are visible numbered from 0 to 3.



The user can control which windows are visible and their layout from the **Desktop** menu displayed in the figure below



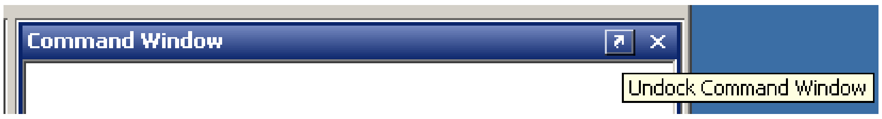
Each Tool appears as a menu item and visible Tools have a “tick” on the left of their name. A Tool can be made visible by clicking on the corresponding menu.

1.9.2 Arranging Tool Windows

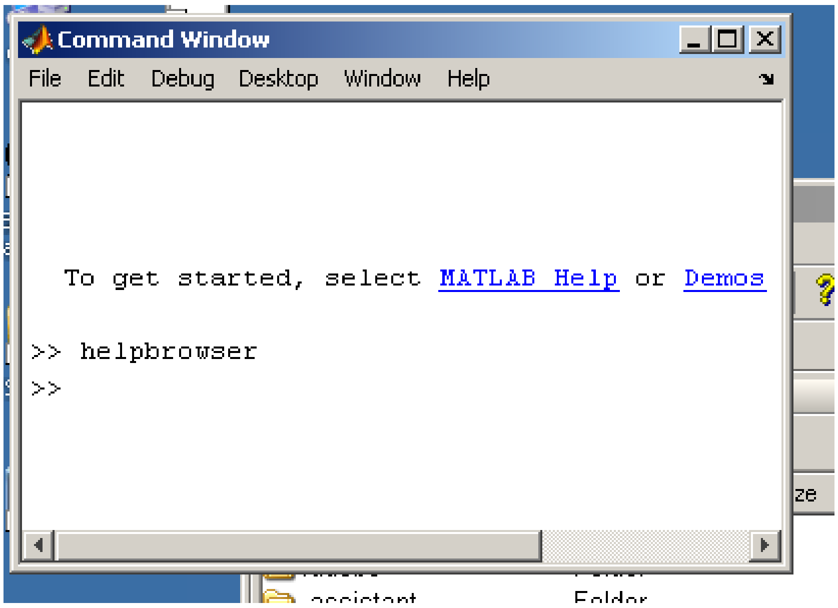
The visible Tools can be laid out in a number of ways.

- Default – Inside the MATLAB Desktop area, attached (“docked”) to the border. Selected from the **Desktop > Desktop Layout** submenu.

- Separate window – Detached (“undocked”) from the Desktop area with the **Dock/Undock** button on the Title Bar:



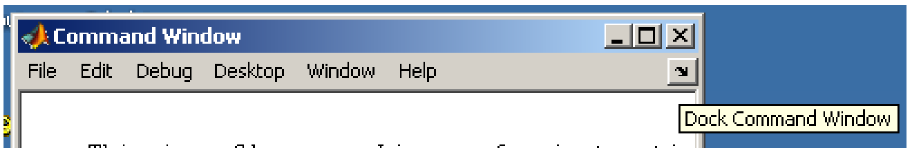
Clicking the button detaches the window from the interior of the main MATLAB window and displays it as a separate window with its own Title Bar and Menu Bar. It can be moved and resized on the Windows Desktop and when minimized appears as a separate MATLAB button on the Windows Taskbar.



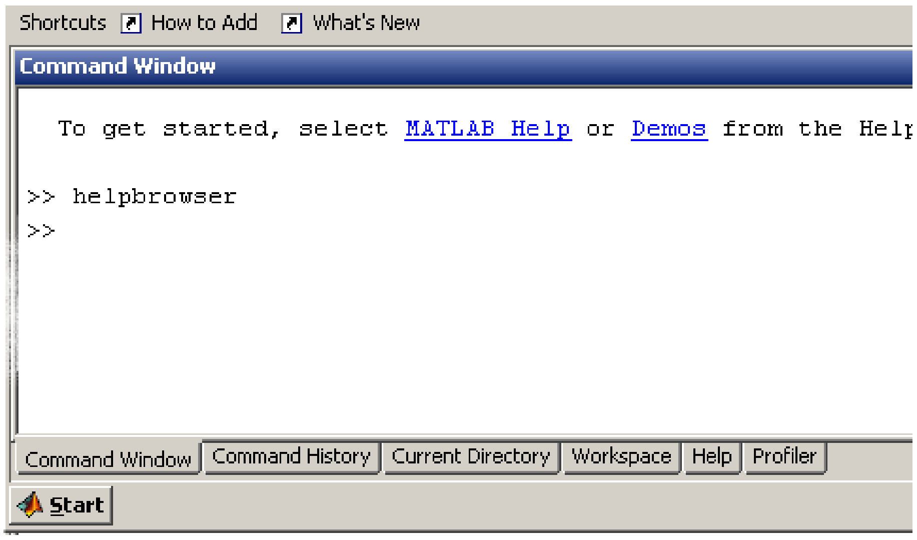
```
  To get started, select MATLAB Help or Demos

>> helpbrowser
>>
```

To return it to the docked state click on the **Dock/Undock** button.



- Tabbed form - All windows are stacked inside the Desktop area and a row of tabs at the bottom enables selection of a particular Tool and brings it to the front. This is selected from the **Desktop > Desktop Layout > All Tabbed** submenu.



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

>> helpbrowser
>>
```

1.10 The Command Window

The Command Window is used to enter MATLAB statements and commands. It works like an ordinary command line window and commands are entered after the **command prompt**

>>

Commands can be entered from the keyboard, loaded from binary files using the **load** command or by opening an **M-file**. It is also possible to select, copy and paste a previous command from the **Command History** window into the Command Window.

1.11 The Command History

The Command History window records the commands that have been entered into the Command Window. The Command History displays a list of all these previously executed commands including those of previous sessions, separated by time stamps. A double click on a command will copy it to the Command Window and execute it.

1.12 The Current Directory Browser

The **Current Directory** browser window is self-explanatory (the default directory is your ITS home space).

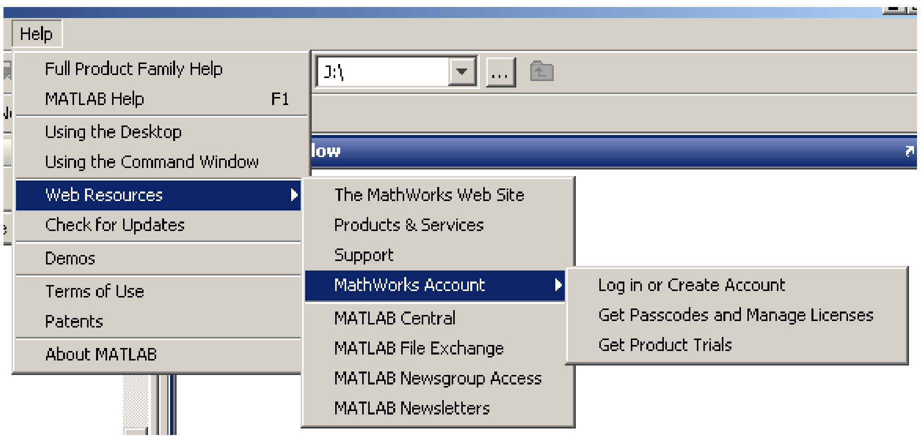
1.13 The Workspace Browser

The **Workspace** sub-window is a list of the variables, vectors and matrices currently in use.

1.14 The Help Browser

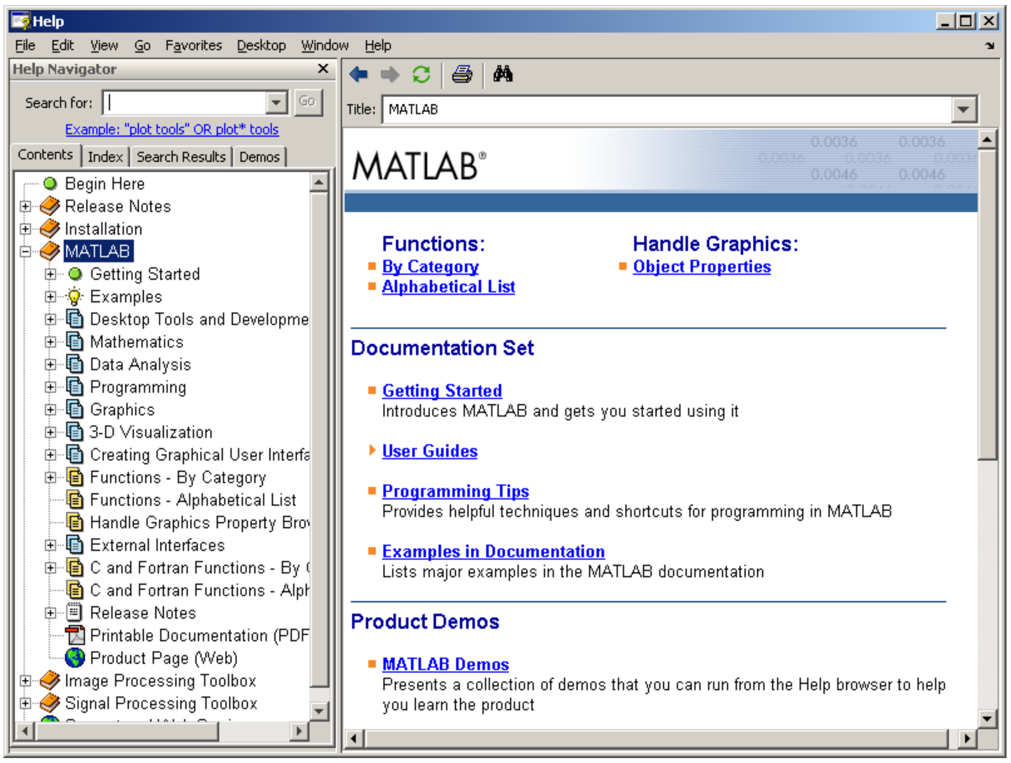
The Help Browser is an on-line help facility in HTML format. It can be opened in several ways.

- By clicking the help button **?** on the **Toolbar.**
- From the **Help** menu, click the **MATLAB Help** menu.



- Pressing the F1 key on the key board.
- Type **`helpbrowser`** in the Command Window.

The Help Browser window looks like this


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Search for:
Go
Example: "plot tools" OR plot* tools
Contents Index Search Results Demos
Begin Here
Release Notes
Installation
MATLAB
Getting Started
Examples
Desktop Tools and Developme
Mathematics
Data Analysis
Programming
Graphics
3-D Visualization
Creating Graphical User Interfa
Functions - By Category
Functions - Alphabetical List
Handle Graphics Property Bro
External Interfaces
C and Fortran Functions - By
C and Fortran Functions - Alph
Release Notes
Printable Documentation (PDF
Product Page (Web)
Image Processing Toolbox
Signal Processing Toolbox
Title: MATLAB
MATLAB®

Functions:
- By Category
- Alphabetical List

Handle Graphics:
- Object Properties

Documentation Set

- Getting Started
  Introduces MATLAB and gets you started using it
- User Guides
- Programming Tips
  Provides helpful techniques and shortcuts for programming in MATLAB
- Examples in Documentation
  Lists major examples in the MATLAB documentation

Product Demos

- MATLAB Demos
  Presents a collection of demos that you can run from the Help browser to help you learn the product

2 Basic manipulations

It is important to remember that, unlike in most other programming languages; MATLAB expressions involve operations on entire matrices and arrays.

2.1 Variables

MATLAB is **case sensitive**, this means for example that x and X are two different variables:

```
>> x = 3 , X = 5
x =
      3
X =
      5
>> x + X
ans =
      8
```

To view the value assigned to a variable, enter the variable name,

```
>> x
```

Note

- The variable or expression is evaluated when the **Return** key is tapped.
- MATLAB will print the result of the expression unless the expression is terminated with a **semi colon** (**;**).

  ```
  >> x = 3 , X = 5 ;
  ```

- Expressions can be combined on a single command line by using the **comma** separator or the **semi colon** (**;**).

Variable names consist of a letter followed by any number of letters, digits or underscores. Only the **first 31 characters** of a variable name are used to identify the variable.

MATLAB does not require type declaration of variables or the dimensions of arrays. If a new variable is introduced MATLAB will decide its type at the moment one or more values are assigned. If a variable already has been assigned a value then changing its value may also change its type.

```
>> x = 6 ;
```

Here MATLAB will create a 1 by 1 array and store the value 6 in its single element. To create a row vector MATLAB has the **[ ]** notation:

```
>> x = [ 3 , 5 , -7  ] ;
```

will now change x into a 1 by 3 array (a **row** vector). This is shown in the way MATLAB prints the vector

```
>> x
x =
```

```
      3        5       -7
```

Note that the row elements are separated with the comma character (**,**) although white space (by entering the **Space** bar on the keyboard) can also achieve this if there is no confusion.

MATLAB distinguishes between **row** and **column** vectors. A row vector is transformed into a column vector by transposition. The notation for the transposition operator is an apostrophe (**'**) or single quote.

```
>> x'

ans =
      3
      5
      -7
```

2.1.1 Matrices

Matrices can also be entered in the **Command** Window with the **[ ]** notation. They are entered on a row by row basis where rows are separated by a semi colon (**;**). The following statement will make A into a 4 by 3 matrix (4 rows and 3 columns),

```
>> A = [ 3 5 7; 2 4 6 ; 3 4 5 ; 11 8 6 ]

A =

    3    5    7

    2    4    6

    3    4    5

   11    8    6
```

Its transpose is

```
>> A'

ans =

    3    2    3   11

    5    4    4    8

    7    6    5    6
```

2.1.2 Matrix Algebra

As MATLAB is a matrix algebra oriented package some of the familiar arithmetic operations behave differently from what one would expect when working with numbers (scalars). The standard characters for multiplication and division, ***** and **/** have a different meaning in MATLAB; they now stand for the corresponding operations of matrix algebra. The same holds for the exponentiation operator, **^**

For example if another vector y is added to the vector x defined before, the two vectors are added element by element:

```
>> y = [ -1 , 6, 7 ] ;
>> x + y

ans =

    2   11    0
```

Similarly for the operation of subtraction

```
>> x - y

ans =

    4   -1    -14
```

The same rules of addition and subtraction apply to matrices.

The multiplication operator ***** for two vectors can have different meanings, it could be the element by element multiplication of the vectors, giving a new product vector of the same length, or, it could be the **scalar product** which is the sum of the elements of the product vector. MATLAB uses the * for the latter

```
>> x * x'

ans =

   83
```

where as the operator for *element by element multiplication* is **.*** which will return a vector with the same length as x

```
>> x .* x

ans =

    9    25    49
```

The built-in function **sum** calculates the sum of columns in a vector or array

```
>> sum ( x .* x )

ans =

   83
```

The same rules apply to matrix-vector and matrix-matrix multiplication. For example set the vector **x** and matrix **A** as

```
>> x = [ 3 , 5 , -7  ] ;
```

and

```
>> A = [ 3 5 7; 2 4 6 ; 3 4 5 ; 11 8 6 ]
```

Then the matrix-vector multiplication

```
>> A * x'
```

```
ans =

   -15

   -16

   -6

  31
```

Note MATLAB will check that the dimensions of vectors and matrices conform to the rules of matrix algebra. If not it will report an error.

It is not required to specify the size of an array in advance but as changing the size takes processor time, the size can be specified if known in advance with the **zeros** function

>> B = zeros (4, 3);

and followed by assignment statements.

2.2 Some special Matrices

MATLAB has a number of built in functions to generate some of the common matrices. Examples are

- Null matrix, with all elements set to zero. To initialize a matrix with n rows and m columns use

  x = **zeros** (n , m) ;

- A matrix with ones for every element

  x = **ones** (n , m);

- The unit or identity matrix **eye**. This is a matrix with zeros everywhere except for ones on its diagonal

  x = **eye** (n , m);

  and similarly,

  x = **eye** (n);

  returns a n by n identity matrix.

2.2.1 Some Examples of Matrix Operations on Square Matrices.

MATLAB supplies a number of built-in functions and M-files that solve problems in **Linear Algebra** defined with matrices. Below are some examples of functions for square matrices. (A square matrix has an equal number of rows and columns).

The inverse of a square matrix A can be calculated with the **inv** function

x = inv (A);

The eigenvalues of a square matrix can be calculated with the function **eig**

e = eig (A)

2.3 Scripts, Functions and M-files

MATLAB provides a large number of intrinsic or built-in functions corresponding to the commonly used functions in mathematics.

The user can also create reusable code in the form of M-files. M-files are text files that contain MATLAB statements. To indicate this, their name should end with the **.m** extension. Two kinds of M-files can be distinguished, depending on the content of the file:

- Scripts.
  These contain lines of code that are executed as if they had been entered directly in the Command Window. They do not accept input arguments nor return output arguments but can use variables that have been defined before the script is invoked.

- Functions.
  These accept input arguments and can return output arguments. Variables created inside the function are local to the function.

The **type** command in the Command Window lists the content of an M-file.

Example

MATLAB comes with a collection of functions that are actually M-files and one of these is the **factorial** function. The way it is coded in MATLAB can be shown with the command

>> **type factorial**

The command

>> **help factorial**

explains what this function does.

2.3.1 Comments

As soon as a program is stored in a file for later re-use it becomes important to add comments everywhere in order to make the meaning and intent of the program clear. In MATLAB all lines that start with the percentage character (**%**) are treated as comments. These comments can be viewed by the user with the **help** command.

2.3.2 Scripts

A MATLAB script is invoked by typing the name of the script which should be the name of the M-file with the **.m** extension removed. MATLAB will execute the statements as if they were entered in the **Command** Window. They can refer to already existing variables or create new variables which can be used in statements entered after the script has returned.

The following is an example.

```
x = 1 : 1 : 20; % define a vector with a range of values

r = rand ( size ( x ) ); % get a random number between 0 and
1

% for each value of x

colormap ( [ 0.8 0.8 0.8 ] ); % default colour is a shade of
grey

bar ( r ); % plot as a bar graph
```

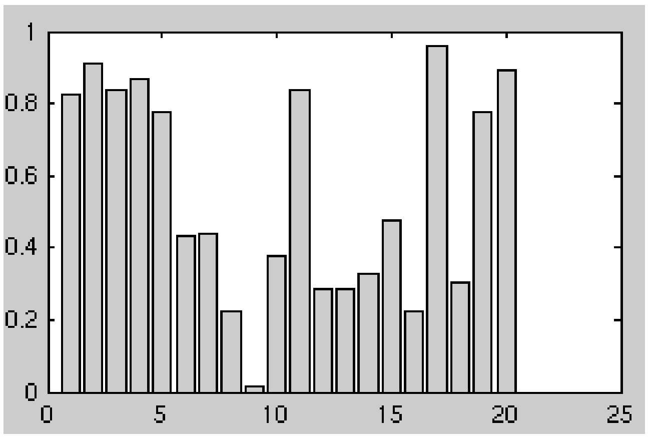
If these MATLAB commands are entered as M-file, **uniform_bar.m**, the content of the file can be displayed in the **Command window** by entering

>> **type uniform_bar**

The script can be executed by typing the statement

>> **uniform_bar**

This will display the random numbers as a bar graph in the **Figures Window**, see below. (Note that the content of the plot will vary every time the script is called because it is produced from random numbers.)



2.3.3 Functions

MATLAB functions combine groups of statements into a single unit of code that can be re-used for different input values. They have the following format, starting with the keyword **function**

function [<output variables> **=**] <function name> (<input variables>)
<MATLAB statements>

The arguments of a MATLAB function are therefore always used for input only. If output from a function is required it must be retrieved through the values returned in the output variables.

Note The name of the M-file (without the .m extension) and that of the function must be the same; otherwise an error message will result.

The following is an example of a function **free_fall_time** that calculates the free fall time in seconds for an object falling from a height h in meters

```
function t = free_fall_time ( h )
g = 9.8;
t = sqrt ( 2.0 .* h ./ g );
```

This code can be entered in a file which must be called **free_fall_time.m** and to call the function enter its name supplied with a value for its argument.

```
>> free_fall_time ( 10000 );

ans =

   45.1754
```

2.3.4 Functions and Global Variables.

By default variables referred to inside a function are local and independent of any variables with the same name that may exist outside the function.

There is a special type of variable that can be created outside a function (for instance in a script or in the **Command** Window) and be accessed from within functions.
The keyword **global** is used for this; this provides a facility for different functions to share data.

For example the following function returns the terminal velocity in a free fall, given the height of falling

```
>> type free_fall_v

function v = free_fall_v ( h )
global G;
v = sqrt ( 2.0 .* G  .* h );
```

This can be used with a value of G set beforehand, as follows

```
>> global G
>> G = 9.8;

>> free_fall_v ( 10000 )

ans =

  442.7189
```

NOTE The global variable must be declared **global** both in any M-files that wish to use it and in the **Command** Window if it will be used in subsequent statements. The assignment of a value to a global variable and its declaration must be in two separate statements.

2.3.5 Functions that return multiple values (output variables)

MATLAB functions can return multiple values in the form of vectors, matrices or arrays. An example is a function that returns both the time and terminal velocity of a free fall.

```
>> type free_fall

function [ t , v ] = free_fall ( h )

g = 9.8;

t = sqrt ( 2.0 .* h ./ g );

v = g .* t;
```

This function is called as follows

```
>> [ t , v ] = free_fall ( 10000 )

t =

   45.1754

v =

  442.7189
```

In the function definition the output arguments have been defined as a vector with two components, t and v. On calling the function a vector compatible with the output variables is used as the left hand side to store the returned values.

This mechanism can be generalized to much more complicated output structures.

Note Built-in function names are not reserved keywords in MATLAB. It is legal to redefine or overwrite any MATLAB variables and built-in functions. The original variable or function can be restored with the clear command.

Example:

```
>> pi

ans =

    3.1416

>> pi = - 3.1416

pi =

   -3.1416

>> clear pi

>> pi

ans =
3.1416
```

2.3.6 Subfunctions

Often it is better to break up a function into several smaller independent logical units. These helper functions are called **subfunction**s in MATLAB. The function that has the M-file name is called the **primary** function and it is the only function that can be invoked from the M-file; any subfunctions are only accessible by the primary function and any other subfunctions in the same file.

- The primary function has to appear first in the M-file, subfunctions can follow in any order.
- Functions within the same M-file cannot access the same variable unless it is declared as **global** or passed as an argument.
- The help facility will only provide the help for the primary function.

Example

```
function [ avg , med ] = mystats ( x ) % the primary function
global n ;

n = length ( x ) ;
% invoke subfunctions:
avg = mean ( x );
med = median ( x );

% here are the subfunctions defined:
function a = mean ( v )
global n ;
a = sum ( v ) / n;

function m = median ( v )
global n ;
w = sort ( v ) ;
if rem ( n, 2 ) == 1
    m = w ( ( n + 1 ) / 2) ;
else
    m = ( w(n/2) + w(n/2+1))/2;
end
```

Functions will be discussed in more details in **Section 4.**

2.1 Exercises

2-1 What happens if the following commands are entered?

x' * x

(Note, **x'** is the transpose of **x**)

A * x

2-2 Enter the complex 2 by 2 matrix

2 – i	1+i
3 i	5

What is the difference between the conjugation operators **‘** and **.’**?

2-3 Code the **uniform_bar** script of **Section 2.3.2** and repeatedly run it. The graph will change in a random way.

2-4 Write a function that has two arguments, the height h and the acceleration of gravity, g, to calculate the fall time and terminal velocity.

3 Graphics

3.1 Introduction

MATLAB can present data as 2 and 3 dimensional plots. The plots can be created from the command line with both linear and logarithmic scales or created and edited interactively with the **Property Editor**.

3.2 2-D plots from data sets

3.2.1 The plot function

MATLAB provides a number of functions for 2 D plots. The three basic versions are

plot (y)

plot (x , y [, x , y])

plot (x , y , 'colour_style_marker' [, x , y , 'colour_style_marker'])

The first form of the MATLAB plot command is plot (y) where y is a vector of values. This plots the values of y against the indices, that is the pairs (n , y (n)) with n =1 , . . . , length (y).

Example

```
>> y = rand ( 1, 10 ); % create a vector of 10 uniform random numbers

>> plot ( y )
```

The general 2-D plot command is based on pairs of (x, y) values as given by the vectors x and y (which must therefore be of equal length).

Example

```
>> x = 0 : 0.1 : 5 ;

>> y0 = exp ( - x );

>> plot ( x , y0 )
```

will plot the exponential decay function in the range from 0 to 5 in steps of 0.1.

3.2.2 Multiple graphs in a single plot

It is possible to create multiple plots in a single graph by successively adding pairs of x and y vectors

>> **y1 = x .* exp (- x);**

>> **y2 = x .^ 2 .* exp (- x);**

>> **plot (x , y0 , x , y1 , x , y2)**

This will produce a graph with a plot of the vectors y0, y1 and y2 with different colours, selected according to the current setting in MATLAB.

3.2.3 Line style of a graph

With the third version of the plot function it is possible to control the style (colour, type of line and marker) of individual graphs as well

plot (x , y , 'colour_style_marker' [, x , y , 'colour_style_marker'])

Here 'colour_style_marker' is a concatenation of up to three character strings; for colour, type of line and marker. The available values are listed in the tables below.

Available line colours are:

cyan	magenta	yellow	red	green	blue	white	black
‘c’	‘m’	‘y’	‘r’	‘g’	‘b’	‘w’	‘k’

Available line styles with their character resentation are:

solid	dashed	dotted	dash-dot
‘-‘	‘--‘	‘:’	‘-.’

Available marker types are

‘+’	‘o’	‘*’	‘x’

Available filled marker types are

square	diamond	up triangle	down triangle	right triangle	left triangle	penta gram	hexa gram
‘s’	‘d’	‘^’	‘v’	‘>’	‘<’	‘p’	‘h’

The order and number of character strings is not important, if a string is absent MATLAB will use default values.

For example, plotting the above three vectors with different colours and markers:

```
>> plot ( x , y0 , 'k-x' , x , y1 , x , y2 , ':m>' )
```

3.2.4 Graphs for complex valued data

If the first form of the plot function, plot (y), is used with a complex valued argument for the vector y then it is equivalent to using the second form as plot (real (y) , imag (y)).

Example

```
>> phi = pi ./ 180 .* [ 0 : 45 : 360 ] % steps of 45 degrees converted to radians

phi =

  Columns 1 through 8

       0    0.7854    1.5708    2.3562    3.1416    3.9270    4.7124    5.4978

  Column 9

    6.2832

>> z = exp (i .* phi )

  Columns 1 through 4

   1.0000             0.7071 + 0.7071i   0.0000 + 1.0000i  -0.7071 + 0.7071i

  Columns 5 through 8

  -1.0000 + 0.0000i  -0.7071 - 0.7071i  -0.0000 - 1.0000i   0.7071 - 0.7071i

  Column 9

   1.0000 - 0.0000i

>> plot ( z )
```

3.2.5 Clearing the content of the Figures Window

To clear the content of the Figures window select the **Edit > Clear Figure** menu item from the Figures window menu bar or from the MATLAB main menu, depending on whether the window is undocked or docked respectively.

However it may be quicker to enter the command

```
>> clf
```

in the Command Window.

3.2.6 Adding Graphs to an existing Plot

The plot command allows the create of several graphs in a single call. It is also possible to add graphs to an already existing plot with the **hold** command.

```
>> hold on
```

will add the graphs created by the next plot commands to the plot until the command

>> hold off

is entered, which switches this feature off.

3.2.7 Controlling the Plot Axes

By default MATLAB uses the given range of x values for the x-axis and selects the range of the y axis such that all graphs will fit.

To change the limits of the x-axis in the Command Window use the function

axis ([xmin xmax])

and

axis ([xmin xmax ymin ymax])

to change both axis limits to the values of the arguments.

An alternative set of functions that apply to individual axes are

xlim ([xmin xmax])

and

ylim ([ymin ymax])

The command

>> **axis auto**

re-enables MATLAB's automatic limit selection.

Axes Aspect Ratio and Visibility

The axes aspect ratio of the graph is controlled with the axis command. Use

>> **axis square**

to make the x and y- axes of the same length on the display. The command

>> **axis equal**

makes the individual tick mark increments the same length.

The axes can be hidden with the command

>> **axis off**

and made visible again with

>> **axis on**

Labels

The axes can be provided with **labels** to indicate their meaning with the xlabel and ylabel commands which add labels to the respective axes. For example

>> **xlabel (‘x’)**

>> **ylabel (‘y’)**

3.2.8 The Graph Title

The graph can be decorated with a title with the function **title** which takes a string as argument, for example

>> **title (‘Plots of various exponentially decaying functions’)**

3.2.9 Descriptive Text in the Graph

Text with descriptive information can be placed anywhere in the graph with the text command.

text (x , y , ‘text string’)

where x and y are the coordinates where the test string has to start. However this can be done more conveniently in **Plot Editing Mode** for a final version of the graph.

Legend
This identifies the plots with some text enclosed in a box

legend (' text' [, 'text']);

3.2.10 Special Symbols in the Graph Annotation using TeX notation

MATLAB has a limited capability to display special symbols, including Greek characters, in any graph text using a TeX notation. For example the above plot of complex values can be decorated with the Greek letter phi

```
>> plot ( exp ( i .* phi ) )

>> text ( 0 , 0 , 'Plot of e^{ i \phi}' )

>> xlabel ( 'cos(\phi)' )

>> ylabel ( 'sin(\phi)' )
```

Example

The M-file plot_complex.m script wraps up the example of a plot of complex numbers on the unit circle

>> **type plot_complex.m**

```
% script to plot complex numbers on the unit circle

phi = pi ./ 180.0 .* [ 0 : 45 : 360 ];

plot ( exp (i .* phi ) )
```

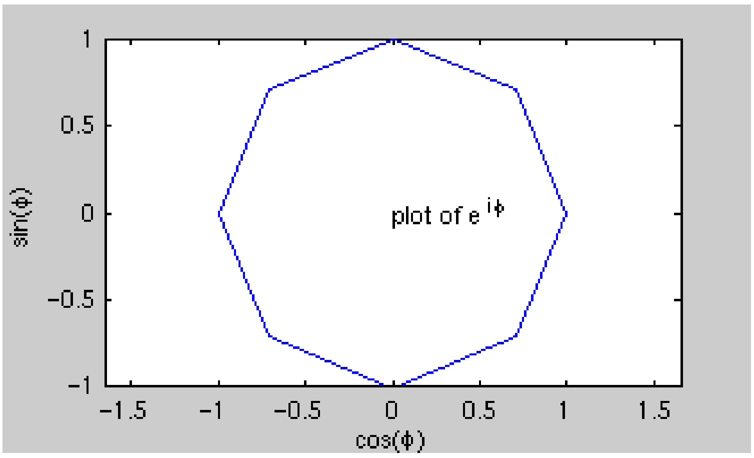
```
text ( 0 , 0 , 'plot of e^{ i \phi}' )

ylabel ( 'sin ( \phi )' )

xlabel ( 'cos(\phi)' )

ylabel ( 'sin(\phi)' )
```

annotates with the Greek letter ‘phi’ as shown in the plot below



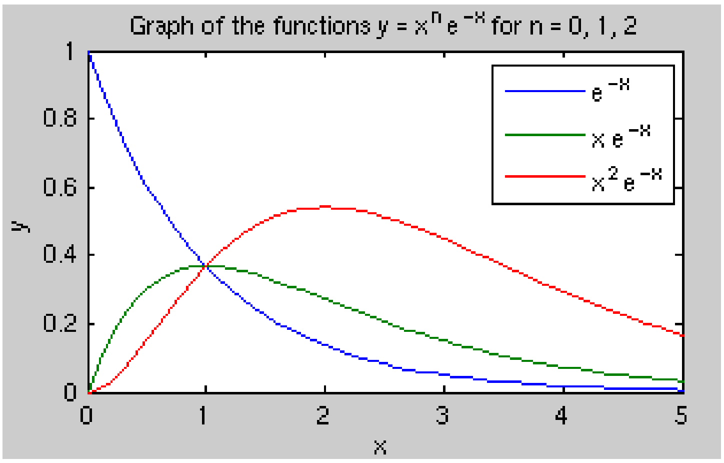
Example

The exponential decay functions in the file exp_decay.m

>> **type exp_decay_plot.m**

```
%exponential decay plots
x = 0 : 0.1 : 5;
y0 = exp ( - x );
y1 = x .* exp ( - x );
y2 = x .^ 2 .* exp ( - x );
plot ( x , y0 , x , y1 , x , y2 )
xlabel ( 'x' )
ylabel ( 'y' )
title ( 'Graph of the functions x^{n} e^{-x} for
n=0,1,2' )
legend ( 'e^{-x}' , 'x e^{-x}' , 'x^{2} e^{-x}' )
```

produce the graph:



3.2.11 Grid Lines

Grid lines can be displayed in the graph with the statement

>> **grid on**

and switched off with

>> **grid off**

3.3 Saving a Figure/Graph to a File

To save a figure, select the **Save As…** command from the **File** menu of the **Figures** window or from MATLAB main menu, depending on whether the window is undocked or docked respectively. Alternatively, clicking the Save Figure button on the Toolbar will achieve the same. This will open the **Save As** dialog box which has a number of options to select the way the graph can be saved to a file. It offers various formats to save the figure in.

3.4 Activating or Opening a new Figures Window

The plot functions automatically open a new figures window if one is not already open. If one or more are open then the output of the plot functions will be displayed in the currently figure. (The Figure window that is currently active). To start with a new figure window enter the command

>> **figure**

before using the **plot** command.

3.5 3-D plots for data sets

MATLAB provides two methods for plotting data in three dimensions

- Mesh Plots
  This defines a surface of z coordinates of points above a grid in the x – y plane

- Surface plots.
  These plots are like mesh plots but the surface of the plot is no longer transparent. The display can be modified in various ways by controlling the illumination and viewing angle.

3.6 Plots of Functions of One and Two Variables

Plotting of functions requires a **function handle** and this topic will be discussed in section **4.4 Plotting Functions**

3.7 Interactive editing of plots in the Plot Editing Mode

This mode can be selected from the **View** menu by selecting **Property Editor**, or, by clicking the **Edit Plot** button on the Toolbar of the **Figures** window. The various items that are visible in the **Figures** window are called Graphics Objects in MATLAB. These can be the plots, the labels and axes, the legend, the title etc. The Editor offers a number of facilities to both view the graph in different ways and to modify the Graphics Objects that it contains. A small selection is described below.

3.7.1 Zoom In and Zoom Out.

It is possible to zoom in or out of the graph. From the Toolbar buttons on the **Figure** Window select the appropriate button.

3.7.2 Panning a Graph

Panning controls what part of the plot is visible within the **Figure** Window. Select this mode with the Toolbar button.

3.7.3 Selecting and Editing a Graphics Object

The Graphics Objects can be selected with a single click. A double click selects the Graphics Object and also opens a Tool that is suitable to modify the Object's properties.

Right mouse click on a Graphics Object pops up a context menu with submenus from where the properties of the object can be changed one at a time. If several properties have to be changed it is more convenient to open the **Properties Editor**. Either by clicking on the menu or from the **Figures** Window menu via **View > Property Editor** The Property Editor appears as an additional window attached to the bottom of the **Figures** Window.

3.8 Exercises

Simplify the commands in the second example of **Section 3.2.10** by using the expressions in the plot function instead of using the vectors y0, y1 and y2.

4 More about Functions

4.1 Function handles

A variety of problems makes it necessary to use a function that has to perform an action on another function, usually defined by the user. Examples are root finding, numerical integration and optimization.
In those cases a function is used as a global object and in MATLAB a reference or handle to the function has to be created.

The handle is obtained by preceding the function name with the 'at' symbol (**@**). For example a handle to the sine function is defined as

```
>> handle_sin = @ sin ;
```

Once a function handle is available the function can be evaluated with the **feval** function as in

output_variables **= feval (** *function_handle* **,** *function_input_arguments* **)**

For example

```
>> data = [ 0.0 , pi / 3 , pi/4 , pi/2 ];
```

```
>> feval ( handle_sin , data )
```

The function **feval** is an example of a MATLAB "function" function which will be discussed later.

This mechanism also works for functions with more than one input and output variable. For example in the case of the **free_fall** function,

```
>> h_free_fall = @ free_fall ;
```

```
>> [ t , v ] = feval ( h_free_fall , 10000.0 ,  9.8 )
```

4.2 Anonymous Functions

The function handle can be used to define unnamed functions. MATLAB enables the construction of simple functions as a single statement without giving a name but instead a function handle. The format is

<handle> = @ (<argument list>) <anonymous function definition>

Example:

```
>> sqr = @ ( x ) x .^ 2; % returns the square of the input value
>> a = sqr ( 5 )
```

```
a =
    25
```

4.3 Function Functions

The "function" functions in MATLAB enable the use of functions as arguments of other functions via function handles.

For example the following function will have the same effect as calling the sine function for a range of data values

```
function x = plot_fhandle (  fhandle , data )
plot ( data , feval ( fhandle , data ) )
```

and used as

```
>> data = 0 : 0.01 : pi;
```

then

```
>> plot ( data , sin ( data ) );
```

and

```
>> plot_fhandle (  @ sin , data );
```

will produce both a graph of the sine function on the interval from 0 to pi.

In this case plot_fhandle is a MATLAB "function" function.

An important example is quadrature. This is the numerical integration of a function over a prescribed interval. MATLAB provides a number of function functions for this purpose. The simplest is **quad**, which uses adaptive Simpson integration,

value **= quad (** *function_handle* , *lower_bound* , *upper_bound* **)**

Since an anonymous function is actually defined through its function handle, it can also be used as an argument, for example to integrate the function **sqr**, defined in **Section 4.2**, between 0 and 1:

>> **quad (sqr , 0 , 1)**

```
ans =
    0.3333
```

The same can be achieved with using the definition of the function **sqr** inside **quad**

>>quad (@(x) x .^ 2 , 0 , 1)

4.4 Plotting Functions

Functions can be plotted for one and two input arguments.

Note There is an inconsistency in notation. The plot functions of Section **3.2** have the independent variable preceding the depending variable, for function plotting this order is reversed.

4.4.1 2 D

In 2D this is the function **fplot** which is in it simplest format

fplot (*function_handle* , *limits* **)**

where *limits* can be used for the x range only

[*x_min x_max*]

or both X and Y range

[*x_min x_max y_min y_max*]

As an example the function (handle) **sqr** on the interval from 0 to 1,

>> **fplot (sqr , [0.0 1.0])**

4.4.2 3 D

Functions of two variables can also be plotted, with the **ezplot** and **ezmesh** and **ezsurf** functions.

ezplot(fun,[min,max])

ezplot(fun,[xmin,xmax,ymin,ymax]

ezsurf(fun)

ezsurf(fun,domain)

where the domain can be either a row vector [xmin, xmax, ymin, ymax] or row vector [min, max] (where min < x < max, min < y < max).

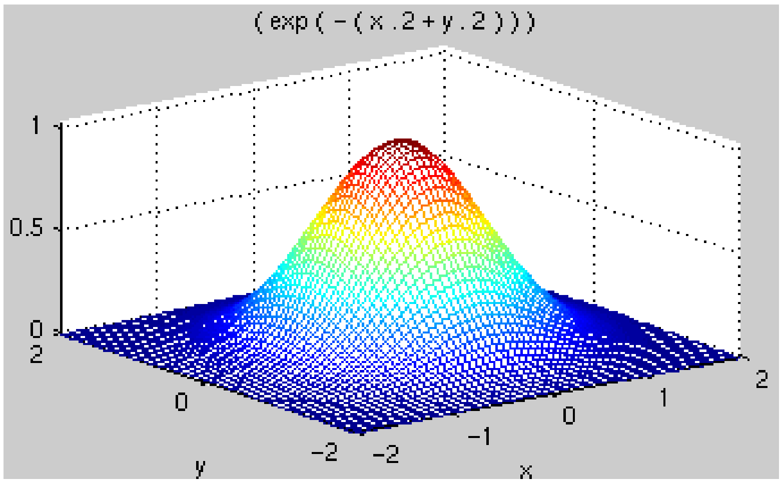
An example is a two dimensional Gaussian function defined as an anonymous function

>> **gauss2d = @ (x, y) (exp (- (x .^ 2 + y .^ 2))) ;**

This function can be displayed as a mesh with the **ezmesh** function,

>> **ezmesh (gauss2d, [-2 , 2 , -2 , 2])**

as shown in the graph below,



whereas the **ezsurf** command will display the surface as well

>> **ezsurf (gauss2d , [-2 , 2 , -2 , 2])**

Another example is the **sinc** function in 2 dimensions, written as an M-file **sinc2d.m**

```
function z = sinc2d ( x , y )
r = sqrt ( x .^ 2 + y .^ 2 + eps );
z = sin ( r ) ./ r;
```

and plotted with

>> **ezmesh (@ sinc2d , [-10,10,-10,10])**

4.4.3 An Advanced Example

A more advanced example of function functions is given here based on the normal distribution function, familiar for example in statistics. A normal random variable X with zero mean and standard deviation σ has a probability distribution

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma}\right)$$

where x can take any real value. This means that the probability of the value of X to be between a and b is the definite integral

$$P(a \le x \le b) = \int_a^b p(x)\,dx$$

Here $p(x)$ is actually a function of two variables but σ plays a special role as it determines the shape of the distribution. An example of code for an M-file **normal.m** is

```
function r = normal ( x , sigma )
r = 1 ./ sqrt ( 2 .* pi .* sigma) * exp ( - x .^ 2 ./ (
2 .* sigma ) );
```

To plot this function for a range of x values with σ as parameter with the **fplot** function, the value of σ has to be specified

>> **sigma = 1.0;**

>> **fplot (@ (x) normal (x , sigma) , [-3.0 3.0])**

To evaluate the probability between -1 and 1 for this value of sigma the **quad** function can be used

>> **quad (@ (x) normal (x , 1.0) , -1.0 , 1.0)**

```
ans =

    0.6827
```

Sometimes the cumulative probability

$$C(x) = \int_{-\infty}^{x} p(\xi)\,d\xi$$

is required for the normal distribution with standard deviation σ . A function for this purpose could be defined as

```
function f = normal_cum (x,sigma)
% this function returns the cumulative probability
between -Inf and
% x for the normal distribution with standard deviation
sigma
f = quad(@(x) normal(x,sigma ), -10.0,x );
% Note: In this example the lower value of the

integration range

% has been set to -10.0
```

and stored in an M-file **normal_cum.m** A plot of both the normal and cumulative normal distributions could be made with this M script:

```
hold on
fplot(@(x) normal(x,1.0), [-3.0,3.0]);
fplot(@(x) normal_cum(x,1.0), [-3.0,3.0]);
hold off
```

5 Programming in MATLAB

MATLAB provides extensive programming features and the following provides just an introduction.

5.1 Program Flow Control

Every programming language has a number of methods to control when and what statements are executed. Although the keywords are different for different languages these constructs usually provide the same functionality.

Some of MATLAB basic control constructs are

- Conditional Statements
- Repetitive Statements
- Selection Statements

5.2 Conditional Constructs

The **if end** construct evaluates a logical expression and when the result is true executes a group of expressions.

```
if  logical_expression

        statements_block

end
```

As MATLAB is based on matrices and arrays a careful distinction has to be made between comparing two arrays as a whole and comparing on an element by element basis. If a and b are scalars the expression a == b will return either 0 or 1 (false or true). However, if a and b are matrices then this test will return a matrix of zeros and ones where a 1 indicates that the two corresponding elements are equal and a 0 that they are not. To test whether a and b are equal on an element by element basis the function **isequal** has to be used which will return a scalar result.

The **if else end** construct

```
if  logical_expression

        statements_block_1

else
        statements_block_2

end
```

and the **if elseif end** construct

```
if  logical_expression

        statements_block_1
```

elseif
        statements_block_2

... more **elseif** statements

else
        statements_block

end

Example

```
d = b ^ 2 - 4 * a * c;
if d > 0.0
     disp ( 'two distinct real roots' );
elseif d == 0.0
     disp ( 'double real root' );
else
     disp ( 'pair of complex conjugate roots' );
end
```

5.3 The Selection Construct

Often it is required to test a variable or expression that can attain a known set of values. This is best done with the **switch** construct.

```
switch expression
   case value1
      block1
   case value2
      block2
...
   otherwise
      default_block
end
```

Note MATLAB's switch does not "fall through" unlike in e.g. the C language. The statement block of the first matching case is executed and the switch statement terminates. MATLAB has no break statements to define the end of a case block.

5.4 The Repetitive or Iterative Constructs

5.4.1 The for loop

The **for** loop repeats a block a fixed predetermined number of times.

```
for  index =  start_value  : increment :  stop_value
       statement_block
end
```

start value, *increment* and *stop value* take on integer values. The increment can take on positive and negative values; the default value is 1, e.g.

```
for index = start_value : stop_value
      statement_block
end
```

For positive increment execution terminates when *index* is greater than the stop value, whereas for negative values when the index is less than stop value.

The next example is a function that calculates the factorial of a non-negative integer.

Example

```
function f = fact_prod ( n )
% function returns the value of n factorial n! = 1.2.3
... (n-1).n
% for integers n >= 0 and 0! = 1 by calculating the
product

f = 1;
for i = 1 : n
    f = f * i;
end
```

5.4.2 The while loop

The **while** loop tests a block an indefinite number of times under control of a logical condition.

```
while  logical_expression

       statement_block

end
```

In the following example the first integer n is found for which n factorial (n !) is a 100-digit number. (Note that this is a very inefficient way to solve this problem.)

Example

```
n = 1;
while prod ( 1 : n ) < 10e100
    n = n + 1;
end
```

The MATLAB function **prod** returns the product of the integers 1 to n.

>> **help prod**

provides more details.

5.4.3 The continue statement

The **continue** statement passes control to the next iteration of the **for** or **while** loop, skipping any remaining statements in the body of the loop.

5.4.4 The break statement

The **break** statement will cause an early exit from the **for** or **while** loop.

5.5 Example: The Collatz Problem

The Collatz problem considers the construction of a sequence of integers taking any arbitrary positive integer as starting point. This is the algorithm to construct the sequence: If n is an integer in the sequence then the next integer is computed as follows. If n is even divide it by 2, if n is odd multiply it by 3 and add 1. The Collatz conjecture states that the sequence will reach the value 1 in a finite number of steps. The length of the sequence varies and depends on the starting value.

For example, for n = 1 the sequence is [1], for n = 2 it is [2, 1] and for n = 3 it is [3, 10, 5, 16, 8, 4, 2, 1].

This is an example of a function to calculate the Collatz sequence, given a starting value n.

```
function sequence  = collatz ( n )
% The Collatz problem. Attempts to find for any
% positive start value n the sequence of integers.

sequence = n;
i = n;
while i > 1
    if rem ( i , 2 ) == 0 % rem returns the remainder
on division of i by 2
        i = i / 2;
    else
        i = 3 * i + 1;
    end
    sequence = [ sequence , i ];
end
```

The length of the sequence can be plotted against the starting value with the following function:

```
function collatz_plot ( n )
% This plots the length of the Collatz sequence for all
positive integers
% between 1 and n.

clf
for m = 1 : n
    seq  = collatz ( m ) % the Collatz sequence for
integer m
    plot ( m , length ( seq ) )
end
```

5.6 Exercises

5-1 Write a function fact(n) = n! to calculate the factorial using its recursive property, fact(n) = n * fact(n-1).

MATLAB provides a function **factorial** for this purpose and the code can be displayed with the command

```
>> type factorial
```

5-2 Rewrite the example in Section 5.4.2 such that the next value for n uses the previous result (for example use a **for** loop and not the MATLAB function **prod**)

5-3 Code the Collatz problem and test the function for n = 1, 2, 3. Try some bigger values of n. Plot the sequence length for a range of values of n.

5-4 User a quicker way of finding the length of the Collatz sequences by storing previous results.