Chapter 1

Introduction

Chapter 1
Introduction

Topics:

- Basic MATLAB concepts
- Book Outline

1.1 Basic MATLAB concepts

MATLAB is coined from **MAT**rix **LAB**oratory. It is a powerful engineering environment and language for problem solving, data analysis, modeling and visualization. It is used by scientists and engineers in research, development and design.

There are many different toolboxes available which extend the basic functions of MATLAB into different application areas.

MATLAB is supported on Unix, Macintosh, and Windows environments; a student version of MATLAB is available for personal computers. For more information on MATLAB, contact the MathWorks.

Basic Mathematical Operators

The following mathematical operators are supported by Matlab:

Addition (a + b)	+
Subtraction (a-b)	-
Multiplication (a . b)	*
Division (a ÷ b)	/ or \
Exponentiation (a^b)	^

Expressions

Single lines (calculations or commands) of Matlab have one of two general forms:

variable = expression

```
y=sin(3.14);
z=(y*4.6)^12.9 – 13.456;
```

or

expression

```
plot(x,y,'-+');
```

Statements

Semicolon **;** terminates the current expression **and** suppresses output (to the screen) of the result.

The comma **,** can be used to separate multiple statements on the same line. The value of any variables will be echoed to the screen.

e.g., x=5.7, y=89.12

will echo those variables and their names back on the screen.

Long statements can be split over lines by 3 periods

e.g., incomeInHand = salary – tax – unionFee …

- superAnnuation;

Notice that, so long as you don't assign a variable a specific operation or result, MATLAB will store it in a temporary variable called "ans".

Vectors

Let's start off by creating something simple, like a vector. Enter each element of the vector (separated by a space) between brackets, and set it equal to a variable. For example, to create the vector a, enter into the MATLAB command window

```
>> a = [1 2 3 4 5 6 9 8 7]
```

a = 1 2 3 4 5 6 9 8 7

Let's say you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2 (this method is frequently used to create a time vector):

```
>> t = 0:2:20
```

t = 0 2 4 6 8 10 12 14 16 18 20

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a'. The equation for that looks like:

```
b = a + 2
```

b = 3 4 5 6 7 8 11 10 9

To add two vectors together, if the two vectors are the same length:

```
>> c = a + b
```

c = 4 6 8 10 12 14 20 18 16

Subtracting vectors of the same length works exactly the same way.

Matrices

Entering matrices into MATLAB is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
>> B = [1 2 3 4;5 6 7 8;9 10 11 12]
```

B =

1 2 3 4

5 6 7 8

9 10 11 12

```
>> B = [ 1  2  3  4
 5  6  7  8
 9 10 11 12]
```

B =

1 2 3 4

5 6 7 8

9 10 11 12

Matrices can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
>>C = B'
```

C =

1 5 9

2 6 10

3 7 11

4 8 12

It should be noted that if C had been complex, the apostrophe would have actually given the complex conjugate transpose. To get the transpose, use .' (the two commands are the same if the matrix is not complex). Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
>> D = B * C
```

D =

30 70 110

70 174 278

110 278 446

```
>> D  = C * B
```

D =

107 122 137 152

122 140 158 176

137 158 179 200

152 176 200 224

You can multiply the corresponding elements of two matrices using the .* operator (the matrices must be the same size to do this).

```
>>E = [1 2;3 4]
>>F = [2 3;4 5]
>>G = E .* F
```

E =	*F =*	*G =*
1 2	*2 3*	*2 6*
3 4	*4 5*	*12 20*

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
>> E^3

   ans =

     37    54

     81   118
```

To cube each element, just use the element-by-element cubing.

```
>> E.^3

   ans =

      1     8

     27    64
```

You can also find the inverse of a matrix or its Eigen-values:

```
>> X = inv(E)

   X =

    2.0000    1.0000

    1.5000    0.5000

>> eig(E)

   ans =

    0.3723

    5.3723
```

Functions

Matlab contains literally hundreds of standard built-in functions

Name	Function
abs(x)	Absolute value
acos(x)	Inverse cosine
acosh(x)	Inverse hyperbolic cosine
angle(x)	Phrase angle
asin(x)	Inverse sine
asinh(x)	Inverse hyperbolic sine
atan(x)	Inverse tangent
atanh(x)	Inverse Hyperbolic tangent
ceil(x)	Round towards +infinity
conj(x)	Complex conjugate
cos(x)	Cosine
cosh(x)	Hyperbolic cosine
exp(x)	Exponentiation e^x
fix(x)	Round towards zero
floor(x)	Round towards –infinity
gcd(x,y)	Greatest common divisor
imag(x)	Complex imaginary part
lcm(x,y)	Least common multiple
log(x)	Natural logarithm
log10(x)	Common (base 10) logarithm
real(x)	Complex real part
rem(x,y)	Remainder after division
round(x)	Round towards nearest int
sign(x)	Sign
sin(x)	Sine

sinh(x)	Hyperbolic sine
sqrt(x)	Square root
tan(x)	Tangent
tanh(x)	Hyperbolic tangent

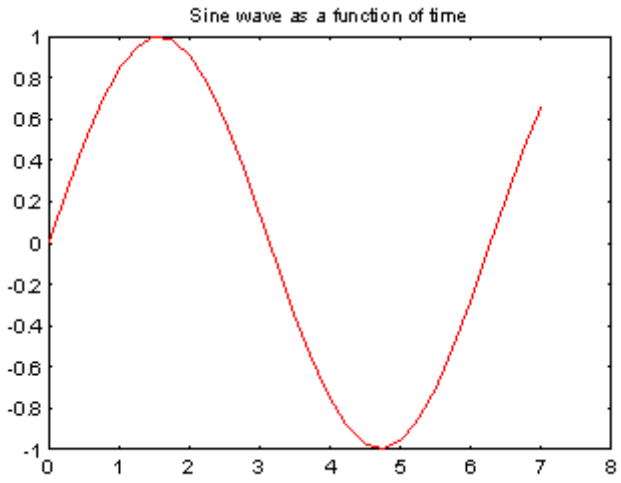
e.g.,

```
distance = sqrt((x1-x2)^2+(y1-y2)^2);

height = velocity*sin(launchAngle)*time - …

         0.5*gravity*time*time;
```

Plotting

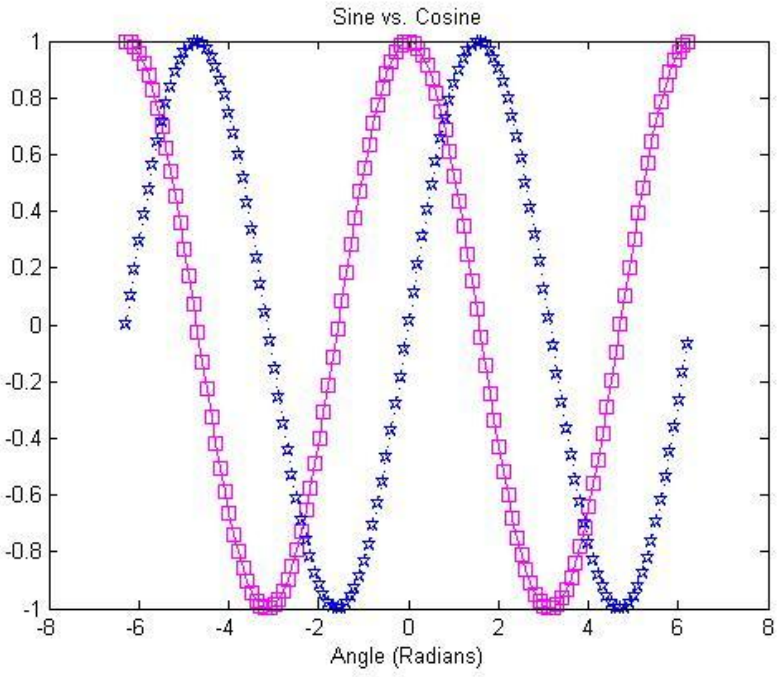
If you wanted to plot a sine wave as a function of time. First make a time vector (the semicolon after each statement tells MATLAB we don't want to see all the values) and then compute the sin value at each time.

```
>> t=0:0.25:7;

>> y = sin(t);

>> plot(t,y)
```



Basic form is a line plot connecting values in 2D space.

```
% g1
x=-2*pi:0.1:2*pi;y=cos(x);z=sin(x);
plot(x,y,'ms-',x,z,'bp:');
xlabel('Angle (Radians)');
title('Sine vs. Cosine');
```



Axes & Labels

Matlab incorporates a number of commands for labelling and modifying existing figures:

title(): Give the figure a title

xlabel(): Label the horizontal axis

ylabel(): Label the vertical axis

grid(): Draw grid-lines on graph

box(): Enclose (or remove enclosing) figure in a box

text(): Place text at the specified location on the figure

gtext(): Interactively (with mouse) place text

legend(): Create a legend box

axis(): Control axis scaling, range, orientation, etc.

Printing Figures

It is often useful to obtain a hard-copy of a figure or save it as a file for incorporation in other documents, using **print,** which sends the currently active figure to the printer or saves it to a file as *postscript* . For example:

```
print              % Send current figure to the default printer
print –deps plotex.eps          % Save the current figure to
                                  % a file called plotex.eps
                                  % encapsulated postscript
```

Multiple Figures & sub-plots

By default there is a single figure window created the first time any type of plotting is done, over-written with each new plot (**hold on**). New figure windows may be created with the **figure(n)** where **n** is an integer value. If figure n window already exists it becomes the *active* window.

Individual figure windows can be sub-divided into multiple plots with **subplot(m,n,p)** which splits the current figure into an **m**-by-**n** matrix of plots and makes the **p**'th plot the active one. Numbering is row-major order.

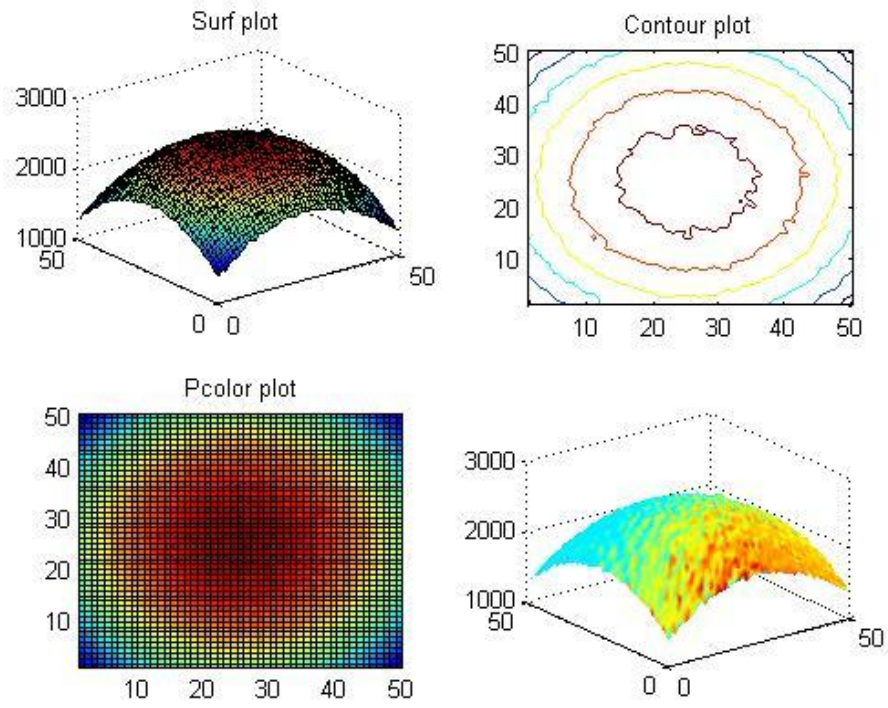
Other 2D plots

While plot is the most commonly used 2D plot function there are a number of others available such as: log scale, area, pie chart, error, bar, histogram, polar, stem.

3D Graphics – Surface

Matlab has a number of functions for visualising 3D surfaces:

```
% pittedDome.m
% Generate a simple pitted dome
width=50;
pitted=zeros(width,width);
for length=1:width
  for breadth=1:width
    pitted(length,breadth)=width*width-((length-width/2)^2+...
      (breadth-width/2)^2) + width*randn/3;
    end;
  end;
  subplot(2,2,1); surf(pitted); title('Surf plot');
  subplot(2,2,2); contour(pitted); title('Contour plot');
  subplot(2,2,3); pcolor(pitted); title('Pcolor plot');
  subplot(2,2,4); surfl(pitted); shading interp;
```



Data-structures

Cell Array is an array, whose elements can be any type or size

```
>> cellEx{1,1}=[1 2; 3 4];
>> cellEx{1,2}='1st row, 2nd column';
>> cellEx{2,1}=6.4-5.1i;
>> cellEx{2,2}=eye(3);
>> cellEx
cellEx =
    [2x2 double]    '1st row, 2nd column'
    [6.4000- 5.1000i]            [3x3 double]
>> celldisp(cellEx);
```

```
cellEx{1,1} =
    1    2
    3    4
cellEx{2,1} =      6.4000 - 5.1000i
cellEx{1,2} =      1st row, 2nd column
cellEx{2,2} =
    1    0    0
    0    1    0
    0    0    1
```

Structures

A collection of disparate types of information, each with a name

```
>> unit.name='Tank';
>> unit.attack=20;
>> unit.armour=12;
>> unit.health=30;
>> unit(2).name='Alpha';
>> unit(2).attack=14;
>> unit(2).armour=0;
>> unit(2).health=20;
>> unit
unit =
1x2 struct array with fields:
```

```
    name
    attack
    armour
    health
>> unit(1)
ans =
     name: 'Tank'
    attack: 20
    armour: 12
    health: 30
>> fieldnames(unit)
ans =
    'name'
    'attack'
    'armour'
    'health'
```

Object Oriented

Recent versions of Matlab support object oriented programming: classes, objects, methods, inheritance, and aggregation.

Unfortunately the hybrid design of procedural syntax with tacked-on objects leads to a number of difficulties such as method invocation (precedence rules rather than explicit user indication of object that invokes) and requirement to explicitly list other classes in constructor's superiorto(), inferiorto() calls means forward compatibility with new classes is an on-going problem.

GUI Design

Matlab provides support for GUI (Graphical User Interface) : menus, text boxes, buttons, sliders, figures, typefaces, pointer & mouse events, and dialog boxes.

Toolboxes

Functionality is expanded modularly with *toolboxes* which are a suite of functions related to a specific problem area such as :

- signal processing
- image processing
- neural networks
- wavelets
- fuzzy logic
- communications
- optimisation
- Control system

Getting help in MATLAB

MATLAB has a fairly good online help; type

help commandname

Large series of examples use **demo** command

Controlling the Environment

A list of "meta" commands useful for controlling the Matlab environment and execution

<cntrl-c> Terminate currently executing command. Useful if stuck in an infinite loop.

clc Clear the screen and home the cursor.

home Return the cursor to the top left corner

more Force output to be paginated (a page at a time)

diary <fname> Save all Matlab output in the file ***fname*** until a **diary off** command is executed. Useful for recording testing results and saving program output.

1.2 Book Outline

This book is organized into 4 chapters as follows: **Chapter 1** shows basic MATLAB concepts.

Chapter 2 shows the Matlab language basics such as variables & data types, I/O, arrays, operators, selection statements, loops, and functions.

Chapter 3 covers polynomials representation, interpolation & splines, integration and Differentiation.

Chapter 4 shows basic SIMULINK concepts.