MATLAB Tutorial

a whirlwind tour

Hinke Osinga

h.m.osinga@bris.ac.uk

Engineering Mathematics

Queen's Building 2.12

The Basics

MATLAB can do everything a calculator does

```
>>(1+4)*3

ans =

      15
```

As is standard:

- `+` and `-` are addition,
- `/` is division and `*` is multiplication,
- `^` is an exponent

Everything is a Matrix!

In MATLAB, a scalar is actually a 1×1 matrix
a vector is an $n \times 1$ or $1 \times n$ matrix

```
>>a = 3

a =

     3
```

a is the number 3, but also the 1×1 matrix $[3]$
Here is how to create a row vector

```
>>v = [1 2 3]

v =

     1   2   3
```

Everything is a Matrix!

A semicolon tells MATLAB to start a new row, so

```
>>w = [4; 5; 6]
w =
     4
     5
     6
```

is a column vector.
With a ′ you turn a column vector into a row vector

```
>>w'
ans =
     4   5   6
```

Multiplication of matrices

You can multiply the vectors `v` and `w`

```
>>v*w
ans =
        32
```

Recall: 1×3 **times** 3×1 **gives** 1×1
Similarly, 3×1 **times** 1×3 **gives** 3×3

```
>>A = w*v
A =
     4     8    12
     5    10    15
     6    12    18
```

Matrix operations

Standard multiplication is really matrix multiplication

Dimensions must match!

Try typing

```
>>v*v
```

However, MATLAB can do 1×1 times a matrix:

```
>>3*A
```

Check what MATLAB does when adding a scalar and a matrix

```
>>A  +  2
```

Elementwise operations

Elementwise operations are done using a . before the operator. For example, each element is squared using

```
>>sqv = v.^2
sqv =
     1   4   9
```

If two vectors (or matrices) have the same dimensions you can perform an elementwise product

```
>>v.*sqv
ans =
     1   8   27
```

Functions operate elementwise

Functions operate on each element in a matrix:

```
>>exp(v)
ans =
      2.7183   7.3891   20.0855

>>log(ans)
ans =
      1   2   3

>>sqrt(v)
ans =
      1.0000   1.4142   1.7321
```

Special constants

The variable `pi` is a permanent variable with value π

```
>>pi
ans =
       3.1416

>>y = tan(pi/6);
```

MATLAB suppresses the output if you end with `;`
MATLAB saves the last output in the variable `ans`.

```
>>ans
ans =
       3.1416
```

Dealing with Matrices

To create a matrix, you could do something like:

```
>>M = [-3 0 1; 2 5 -7; -1 4 8]
```

The semicolons indicate the end of a row.
All rows have to be the same length.

The element in the third row, first column, is `M(3,1)`
You get the elements in rows 2 through 3 and columns 1 through 2, by typing

```
>>M(2:3,1:2)
ans =
     2     5
    -1     4
```

Extract a submatrix

So, to get the entire second column, you type

```
>>M(1:3,2)
```

which is the same as

```
>>M(:,2)
```

which is literally telling MATLAB to use all rows in the second column

You get a whole row of a matrix with

```
>>M(1,:)
```

Linear Algebra

At the heart of MATLAB is a powerful range of linear algebra functions. For example, you can solve the linear system `M*x = w` with unkown vector `x`

```
>>x = M\w
x =
   -1.3717
    1.3874
   -0.1152
```

Is `M*x` indeed equal to `w`?

```
>>M*x,w
```

Eigenvalues

The eigenvalues of `M` can be found using `eig`

```
>>e = eig(M)
e =
```

$$-2.8601$$

$$6.4300 + 5.0434i$$

$$6.4300 - 5.0434i$$

Here i is the imaginary unit, $\sqrt{-1}$.

Eigenvectors

The eigenvectors of `M` are found using two output arguments for `eig`

```
>>[V, D] = eig(M);
```

the columns of the matrix `V` are the eigenvectors
The eigenvalues are on the diagonal of `D`

Eigenvalues and eigenvectors

The first eigenvalue in `D` is associated with the first eigenvector in `V`, so

```
>>ev1 = V(:,1)

>>M*ev1
ans =
        -2.8094
         0.3648
        -0.3931
```

This should be equal to `-2.8601*ev1`

```
>>D(1,1)*ev1
```

Equally spaced values

The colon notation is useful for constructing vectors of equally spaced values

```
>>v = 1:6
v =
     1   2   3   4   5   6
```

Non-unit increments are specified as follows

```
>>w=2:3:10, y=1:-0.25:0
w =
     2   5   8

y =
    1.0000   0.7500   0.5000   0.2500   0
```

Generating matrices

You can build certain types of matrices automatically. Try

```
>>Id = eye(3,3)

>>Y = zeros(3,5)

>>Z = ones(2)
```

Here the first argument is the number of rows
The second argument is the number of columns
With only one argument, the matrix becomes square

Getting help

- Typing “help” at the MATLAB prompt gives you a list of folders where MATLAB can find commands in
- “help foldername” gives you a list of commands in that folder
- “help commandname” gives help on a specific command

Plotting

Plotting the sine function from 0 to 10:

```
>>x = [0:0.1:10];

>>plot(x, sin(x))
```

If you type a second plot later, it will clear your first plot:

```
>>plot(x, sin(x), 'r*')
```

Plotting two graphs in the same figure

You can add plots on top of one another, for example

```
>>plot(x, sin(x))

>>hold on

>>plot(x, sin(x), 'r*')

>>hold off
```

shows both the data points and the curve.

Other plotting options

Other colours and styles can be used as well:

```
>>plot(x, sin(x), 'm',x, sin(x), 'xk')

>>title('The sine function')

>>xlabel('the x-axis')

>>ylabel('the y-axis')
```

For more information type

```
>>help plot
```

Printing a plot

Use “Print Preview” under the “File” menu of the figure window before printing.
You can print directly to the printer, or save to a file first

Saving, and Loading

To interupt a session, the following may be useful

```
>>save filename.mat
```

will save all your variables and values in MATLAB format.

```
>>load filename.mat
```

loads this file back into MATLAB, provided the directory above the command window is properly set.

Useful tools

- "`clear M`" **causes MATLAB to forget about** `M`
  "`clear all`" **clears the entire workspace**
- `who`
  will tell you all the variables currently defined
- `whos`
  prints the variables, their sizes, and other info
- `whos -file filename.mat`
  gives info on filename.mat before loading it
- `format long` **and** `format short`
  switch between long and short display format.

Scripts

A script is a list of commands to be run in some order.

Placing commands in a file that ends in `.m` allows you to "run" the script by typing its name at the command line.
You type the name without `.m`

Scripts — Example

For example, get the file

`mandelbrot.m`

from the

Data Analysis Website

and save it in your preferred folder.
(click on “Website” if you cannot find this file)

Set the “Current Directory” above the command window to this preferred folder, press return and type

```
>>mandelbrot
```

for loop — Example

Inspect the file

`mandelbrot.m`

in your favourite editor.

This is a script file and it also shows how to do a `for` loop.

Functions

You can define your own functions in MATLAB.
A function must start with the line

function *return-values* = *functionname*(arguments)

so that MATLAB will recognize it as a function.

Each function needs to have its own file,
and the file must have the same name as the function.

Functions — Example

For example, get the file

`sierpinski.m`

from the

Data Analysis Website

and save it in your preferred folder.
(click on “Website” if you cannot find this file)

This is a function file and it recursively generates the famous Sierpinski fractal up to any level.

Sierpinski

Set the “Current Directory” above the command window to this preferred folder, press return and type

```
>>level = 5;

>>Pa = [0; 0]; Pb = [1; 0]; Pc = ...
[0.5; sqrt(3)/2];

>>sierpinski(Pa, Pb, Pc, level)

>>title(['Sierpinski fractal:  level '...
num2str(level)], 'FontSize', 16)

>>axis('equal')
```