Introduction to Matlab

Part 1

Luke Dickens

Imperial College London

January 15th, 2015

Overview

Matlab is an intuitive, easy-to-learn, high performance language for *numerical* computation and data visualisation. It can handle:

- Numerical mathematics and computation
- Algorithm development
- Data acquisition
- Modelling, simulations, and prototyping
- Data analysis, exploration, and visualisation
- Scientific and engineering graphics
- Application development, incl. graphical user interfaces

MATLAB stands for MATrix LABoratory. Its basic variable is arrays, i.e. vectors and matrices. Matlab also has many built-in functions, as well as specialised add-on tool boxes.
Unlike other mathematical packages, such as MAPLE or MATHEMATICA, MATLAB cannot perform symbolic manipulations without the use of additional Toolboxes.

Some similar tools to MATLAB are:

- Octave is an open source high-level interpreted language like MATLAB. Carefully written MATLAB scripts will also run on Octave and vice versa. See:

  `www.gnu.org/software/octave/`

- R is an open-source software programming language and software environment for statistical computing and graphics. Different in syntax to MATLAB, and with an emphasis on statistical methods. See:

  `www.r-project.org/`

- python is a widely used, general-purpose, high-level, open-source programming language. With the numpy, scipy and matplotlib libraries, python can recreate most of the functionality of MATLAB and provide additional flexibility. See:

  `www.python.org`, `www.scipy.org`, `www.numpy.org`, `matplotlib.org`

The 5 main parts to Matlab:

1. Desktop tools and development environment

   Mainly graphical user interfaces, editor, debugger, and workspace

2. Mathematical function library

   Basic maths functions such as sums, cosine, complex numbers
   Advanced maths functions such as matrix inversion, matrix eigenvalues, differential equations

3. The language

   High-level language based on arrays, functions, input/output, and flow statements (for, if, while)

4. Graphics

   Data plotting in 2d and 3d, as well as image analysis and animation tools

5. External interfaces

   Interaction between C and Fortran programs with Matlab, either for linking convenient routines from Matlab in C/Fortran, or for Matlab to call fast C/Fortran programs

Different ways to use Matlab

1. Interactive mode
   just type commands and define variables, empty workspace with command `clear`
2. Simple scripts
   M-file (name.m) with list of commands
   Operate on existing data in work space, or create new data
   Variables remain in workspace (until cleared)
   Re-useable
3. M-file functions
   M-file as with scripts
   May return values
   Re-usable
   Easy to call from other functions (make sure file is in Matlab search path)

Variables

Variables do not need to be declared. Simply assign values to variable names, e.g.

```
>> x = [1 2 3 4 5]
x =
  1 2 3 4 5
```

For quiet assignment, terminate expression with a semi-colon. To display a variable, simply use the variable name on its own.

```
>> x = [1 2 3 4 5];
>> x
x =
  1 2 3 4 5
```

`x` is a row vector.

Sequences and Vector Elements

Create a natural integer sequence (vector) by specifying the first and the last element separated by a colon (`:`), e.g.

```
>> u = [0:8]
u =
  0 1 2 3 4 5 6 7 8
```

A different increment can be specified as a third (middle) argument, e.g.

```
>> v = [0:2:8]
v =
  0 2 4 6 8
```

Vector elements can be referenced with parentheses.
Be careful: Indices start at 1.

```
>> v(2)
ans =
  2
```

A number of elements can be referenced using the colon notation, e.g. `v(1:3)`, `u(2:2:6)`. What will these give?

References can also be standard lists/vectors, e.g. `v([1,5,4])`.

Elements are referenced for both read and write operations. For instance,

```
>> v([1,5,4]) = u([5,3,1])
```

References can also be boolean arrays with the same shape as the original, e.g. try

```
>> v([true,false,true,false,true])
```

or (perhaps more usefully)

```
>> u((u <6) & (u > 2))
```

Vectors

Matlab distinguishes between row and column vectors. For row vectors, spaces or commas (`,`) separate elements. For column vectors, semi-colons (`;`) or new-lines separate elements (rows), e.g.

```
>> y = [6; 7; 8; 9; 10]
```

Or transpose a row vector with the prime symbol (`'`), e.g.

```
>> y = [6 7 8 9 10]'
```

A dot product is a row vector multiplied by (`*`) a column vector, e.g. `x*y`, `y'*x'`, `x*x'` or `y'*y`.
What happens when we multiply a column vector by a row vector, e.g. `y*x`?

Referencing into column vectors preserves the column structure. For instance, try outputting:

`y(2:4)` or `y([1,4,5])`

Automatic conversion occurs when assigning from row to column vector (or vice versa), e.g. try

```
>> x([2,4]) = y([1,3])
```

Matrices

In fact, row vectors are simply $1 \times n$ matrices, and column vectors are $n \times 1$ matrices.
We can define more general matrices using spaces (or commas) to separate column entries, and semi-colons (or new-lines) to separate rows, e.g.

`A=[1 2; 3 4;]` is the 2×2 matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

`B=[4 5; 6 7; 8 9;]` is the 3×2 matrix $\begin{pmatrix} 4 & 5 \\ 6 & 7 \\ 8 & 9 \end{pmatrix}$

Reference the ith-row jth-column element of matrix `A`, with `A(i,j)`. We can even pull out a submatrix with the colon notation (just a colon gives the full row/column), e.g.

`B(2:3,:)` is the 2×2 matrix $\begin{pmatrix} 6 & 7 \\ 8 & 9 \end{pmatrix}$

A prime (') transposes a matrix.

Block Matrices

We can use vectors to build larger vectors by placing them in square parenthesis and separating them by spaces (commas), called *horizontal* concatenation, so

`>> z=[x y']` is the row vector $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}$

Semi-colons (carraige returns) are for *vertical* concatenation, so

`>> C=[x(1:3) ; y(3:5)']` is the matrix $\begin{pmatrix} 1 & 2 & 3 \\ 8 & 9 & 10 \end{pmatrix}$

Likewise we can block together compatible matrices to build larger matrices.
What will `[A B']` and `[A ; B]` produce?

Scalar Operations

We can multiply every element of a matrix by a scalar, e.g. `3*A`, with the obvious results. So if `r` is a real number, and

>> `D` is the matrix $\begin{pmatrix} d_{11} & \dots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{m1} & \dots & d_{mn} \end{pmatrix}$

then

>> `r*D` is the matrix $\begin{pmatrix} r*d_{11} & \dots & r*d_{1n} \\ \vdots & \ddots & \vdots \\ r*d_{m1} & \dots & r*d_{mn} \end{pmatrix}$

Similarly, if we add or subtract a scalars and matrices together, e.g. `A+3` or `1-B`, then the same scalar operation is applied to each element in the matrix.

Elementwise Operations

Two matrices of the same dimension can be added together with +. The operation is *elementwise*, so the two matrices must have the same dimension. For the two matrices $D, E \in \mathcal{R}^{m \times n}$, where

>> `D` is the matrix $\begin{pmatrix} d_{11} & \dots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{m1} & \dots & d_{mn} \end{pmatrix}$

>> `E` is the matrix $\begin{pmatrix} e_{11} & \dots & e_{1n} \\ \vdots & \ddots & \vdots \\ e_{m1} & \dots & e_{mn} \end{pmatrix}$

then

>> `D+E` is the matrix $\begin{pmatrix} d_{11}+e_{11} & \dots & d_{1n}+e_{1n} \\ \vdots & \ddots & \vdots \\ d_{m1}+e_{m1} & \dots & d_{mn}+e_{mn} \end{pmatrix}$

Other elementwise operators include: subtraction (e.g. `C'-B`), multiplication (`.*`), division (`./`), and power (`.^`).

More on Elementwise Operators

The two matrices must be compatible, so this is fine

```
>> B'+C
ans =
 5  8  11
 13 16 19
```

but this is not

```
>> B+C
???  Error using ==> plus
Matrix dimensions must agree.
```

The elementwise power operator (`.^`), applies a power to each element of a matrix, so for instance `A.^2` is the same as `A.*A`. What will `2.^A` give?

Matrix Multiplication

Standard Matrix multiplication, with `*`, requires matrices to have compatible dimensions, so

`>> B*A` gives the matrix $\begin{pmatrix} 19 & 28 \\ 27 & 40 \\ 35 & 52 \end{pmatrix}$

but

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

Can also take the power of any square matrix with the hat symbol (`^`), e.g. `A^2` is equivalent to `A*A`.

Recall that for matrix multiplication of two matrices A and B, where A_{ij} is the element in the ith row and jth column of A. Then matrix multiplication of the two gives $C = AB$ such that

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Which of the following are valid using the matrices from the slides: `A*A`, `A*B'`, `B'*A'`?

Matrix Division

For a square invertible[1] $n \times n$ matrix `M`, and compatible vector `b`, we have the matrix equivalent of division,

- `>> a = M\b` is the solution of the equation $Ma = b$
- `>> a = b/M` is the solution of the equation $aM = b$

More generally, see the `eig` built in function.

1: A matrix $M \in \mathcal{R}^{n \times n}$ is invertible, if $\forall x \in \mathcal{R}^n$

$$Mx = 0 \Rightarrow x = 0$$

Summary of Arithmetic Operators

Below is a list of the arithmetic operators available in MATLAB.

+	addition
-	subtraction
*	multiplication
^	power
\	left division
/	right division
.*	elementwise multiplication
.^	elementwise power
.\	elementwise left division
./	elementwise right division
'	transpose

Comparison Operators

As with addition and subtraction. *Relational operators* can be applied to a scalar and a scalar; a matrix and a scalar; or to two matrices of the same dimension. The following are available:

`<`	less than
`>`	greater than
`<=`	less than or equal
`>=`	greater than or equal
`==`	equal
`~=`	not equal

Boolean values (or matrices) may be connected by the following *logical operators*.

`&`	and
`|`	or
`~`	not

You will explore these in the lab.

Built-in Functions

There are numerous built-in functions (i.e. commands) in MATLAB. There is only room here to describe a few of them. We separate them roughly into categories.

Special Functions:

`help`: Displays help information for any MATLAB command.
`lookfor`: A keyword search function. Useful for finding unknown commands.
`who`: Lists the current variables in the workspace.
`whos`: Lists current variables with detailed information.
`clear`: Clears current variables.
`exit`: Closes interactive mode.

I strongly recommended you look up any built-in function with `help` before using them or see the online support at:

`http://www.mathworks.co.uk`

Scalar Functions

Certain MATLAB functions are essentially used on scalars, but operate element-wise when applied to a matrix (or vector). They are summarized below (angles are in radians).

`sin`	trigonometric sine
`cos`	trigonometric cosine
`tan`	trigonometric tangent
`asin`	trigonometric inverse sine (arcsine) ...
`exp`	exponential
`log`	natural logarithm
`abs`	absolute value
`sqrt`	square root
`rem`	remainder
`round`	round towards nearest integer
`floor`	round towards negative infinity
`ceil`	round towards positive infinity

Some examples of these functions in use are:

```
>> sin(1)
ans =
 0.8415
>> cos([-1:0.5:1]*pi)      (pi is a built-in constant)
ans =
 -1.0000 0.0000 1.0000 0.0000 -1.0000
>> rem(12,5)
ans =
 2
>> floor(1.4)
ans =
 1
>> ceil(1.4)
ans =
 2
```

Vector Functions

Other MATLAB functions operate essentially on vectors returning a scalar value. Some of these functions are given in the table below.

`max`	largest component
`min`	smallest component
`length`	length of a vector
`sort`	sort in ascending order
`sum`	sum of elements
`prod`	product of elements
`median`	median value
`mean`	mean value
`std`	standard deviation

These can be very effective applied to matrices too, acting across rows or columns, but read the `help` pages first.

Let z be the following row vector.

```
>> z = [0.0099, 0.1389, 0.2028, 0.1987, 0.6038];
```

Then

```
>> max(z)
ans =
 0.6038
>> min(z)
ans =
 0.0099
>> sort(z)
ans =
0.0099 0.1389 0.1987 0.2028 0.6038
>> sum(z)
ans =
 1.1541
>> mean(z)
ans =
 0.2308
```

Vector Functions applied to Matrices

Vector functions applied to a matrix act on each column separately to produce a row vector of results. So for matrix

```
>> M = [0.4447, 0.9218, 0.4057; 0.6154, 0.7382, 0.9355;
0.7919, 0.1763, 0.9169];
```

We can get a vector of the sums of each column with

```
>> sum(M)
ans =
 1.8520 1.8363 2.2581
```

We can apply sum to the above row vector by typing

```
>> sum(ans)
ans =
 5.9464
```

Additional input arguments may affect the nature of the function, e.g. sum the columns of `M` with `sum(M,2)`.

Additional Input and Output Arguments

The role of additonal arguments depends on the function, for instance given `X`, `Y`$\in \mathcal{R}^{m\times n}$, `max(X,Y)` gives an elementwise maximum array also in $\mathcal{R}^{m\times n}$, e.g.

```
>> max([1:5],[5:-1:1])
ans =
 5 4 3 4 5
```

Additional output arguments can also sometimes be specified.
For instance, `[val,pos] = max(X)` returns the index of the maximum value in `pos`. So,

```
>> [val,pos] = max(z)
val =
 0.6038
pos =
 5
```

Check the `help` page for each function you use.

Array Constructor Functions

We have used the colon notation in square brackets to construct a vector, e.g.

```
>> x1 = [-pi:pi/50:pi]
```

for a vector of 101 evenly spaced points in the range $[-\pi, \pi]$. We can achieve the same results with the `linspace(a,b,n)` command, where `a` and `b` give the two endpoints and `n` gives the number of elements, e.g.

```
>> x1 = linspace(-pi,pi,101)
```

Similarly, to generate n logarithmically spaced elements from 10^a to 10^b, type `logspace(a,b,n)`.

These are very useful for plotting as we will see.

Matrix Constructor Functions

More generally, many functions exist for building matrices, some of which are given in the table below.

`eye`	identity matrix
`zeros`	matrix of zeros
`ones`	matrix of ones
`diag`	extract diagonal of a matrix or create diagonal matrices
`triu`	upper triangular part of a matrix
`tril`	lower triangular part of a matrix
`rand`	randomly generated matrix
`repmat`	Build larger matrices from repeated matrix blocks

In particular, `repmat` (and `bsxfun`) can be very helpful in avoiding computationally expensive `for` loops (use `bsxfun` where possible).

Get a feel for these in the lab. Remember to use the `help` function.

Built in Logical Functions

There are a number of built in logical functions you may find particularly helpful. These can apply to the whole matrix, or elementwise.

`all`	true (= 1) if all elements of vector are true
`any`	true (= 1) if any element of a vector is true
`exist`	true (= 1) if the argument (variable or function) exists.
`empty`	true (= 1) for an empty matrix.
`isinf`	true for all infinite elements of a matrix
`isfinite`	true for all finite elements of a matrix
`isnan`	true for all elements of a matrix that are not a number, `NaN`.
`find`	returns vector of indices of all non-zero elements of a matrix.

Other Operations

Some other useful commands that apply to arrays are

`size`	size of an array
`det`	determinant of a square matrix
`inv`	inverse of a matrix
`eig`	eigenvalues and eigenvectors
`norm`	norm of matrix (1-norm, 2-norm, ∞-norm)
`arrayfun`	map a function to each element of an array
`bsxfun`	Implicitly replicate rows/columns for simple operations (quicker than repmat)

Again, you will get a chance to explore some of these matrix functions yourselves in the lab.

Some more commands that you may find useful

`rank`	rank of a matrix
`rref`	reduced row echelon form
`poly`	characteristic polynomial
`cond`	condition number in the 2-norm
`lu`	LU factorization
`qr`	QR factorization
`chol`	Cholesky decomposition
`svd`	singular value decomposition

The `plot` Function

`plot` is a special MATLAB function to visualise data.
To plot the cosine function, begin by choosing the points along the x-axis, to evaluate `cos(x)`.

```
>> x=-pi:0.01:pi;
```

Smaller increments give a *smoother* curve.
Then define the corresponding y values,

```
>> y = cos(x);
```

Finally, we can plot this function with

```
>> plot(x,y)
```

The plot appears in a separate window.
For more details, you should read the `help` page on `plot`.

Notes on `plot`

`plot` operates on pairs of points (x, y), and connects these with straight lines. For analytic functions, you should use enough points to give the appearance of a smooth curve.

It is good practice to label the axis on a graph. This can be done with the `xlabel` and `ylabel` commands.

```
>> xlabel('x')
>> ylabel('y=cos(x)')
```

Give the plot a title with the `title` command.

```
>> title('Graph of cosine from -pi to pi')
```

To redraw the graph in green, use

```
>> plot(x,y,'g')
```

The third argument, for the colour, appears within single quotes. We can get a dashed line instead of a solid one with

```
>> plot(x,y,'--')
```

or say a blue dotted line by typing

```
>> plot(x,y,'b:')
```

Colours and Styles

A list of the available colours and styles appears below:

`y`	yellow	`.`	point
`m`	magenta	`o`	circle
`c`	cyan	`x`	x-mark
`r`	red	`+`	plus
`g`	green	`-`	solid
`b`	blue	`*`	star
`w`	white	`:`	dotted
`k`	black	`-.`	dashdot
		`--`	dashed

Multiple Plots

Multiple curves can appear on the same graph. To show this, we define another vector

```
>> z = sin(x);
```

then plot both on the same axis with

```
>> plot(x,y,'r--',x,z,'b:')
```

This gives a plot with the red dashed line for $y = cos(x)$ and a blue dotted line for $z = sin(x)$.
The command `legend` provides a legend (or key) to help distinguish multiple plots, e.g.

```
>> legend('cos(x)','sin(x)')
```

`hold` can also be used for multiple plots on the same axis, e.g.

```
>> plot(x,y,'r--')
>> hold on
>> plot(x,z,'b:')
>> hold off
```

Other Visualisation Commands

Other commands for data visualization that exist in MATLAB include

`subplot`	create an array of (tiled) plots in the same window
`loglog`	plot using log-log scales
`semilogx`	plot using log scale on the x-axis
`semilogy`	plot using log scale on the y-axis
`errorbar`	plot with error bars
`bar`	plot a bar chart
`hist`	plot a histogram
`surf`	3-D shaded surface graph
`surfl`	3-D shaded surface graph with lighting
`mesh`	3-D mesh surface (for surface graphs)

Again you are encouraged to explore these yourselves.

Using `subplot`

The `subplot` command allows you to plot two curves alongside one another. An example is

```
>> x = linspace(0,5,51);
>> subplot(2,1,1)
>> plot(x,sin(x),'-')
>> axis([0 5 -2 2])
>> title('A sine wave')
>> subplot(2,1,2)
>> plot(x,sin(x)+0.1*randn(1,51),'o')
>> title('Noisy points on the sine wave.')
```

The command `randn` is used to generate a random vector with elements sampled from $N(0,1)$ – the normal distribution.

A Surface Plot

A Quick Example of a 3-D Plot is given below:

```
>> [x,y] = meshgrid(-3:.1:3,-3:.1:3);
>> z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
- 1/3*exp(-(x+1).^2 - y.^2);
>> surf(z)
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Peaks')
```

The command `meshgrid` specifies the 2 dimensional grid of points on which to evaluate the function. The ellipsis (`...`) allows for line continuation.

Saving Plots

Plots can be saved by clicking on the appropriate icon on the display, but if you would like to perform this at the command line then you can.
To do this you need to get a handle to the figure you will create, then plot the graph, before finally passing the figure handle to the `print` function. The print command also needs to know the file type and file name. For example, the following commands will plot the sine function to file `myplot.eps`.

```
>> x = linspace(0,5,100);
>> handle = figure;
>> plot(x,sin(x))
>> print(handle,'-depsc','myplot.eps');
```

Check out `help print` for more detail.

Notes on formats

The `fig` file type is MATLAB specific, and means that you can reload into MATLAB later and continue editting.

For LaTeX documents, it is often better to produce (scalable) `eps` files, than `jpeg` or `png` files; these can be converted to `pdf` files with command line tools.

Use `-depsc` for colour eps plots.

`savefig` and `saveas` are also provided

Summary

- Overview of Matlab
- Vector and matrix construction
- Working with vectors and matrices
- Commonly used built in functions. Including:
  - Special functions
  - Operations on scalars, vector and matrices
- A focus on plotting

References

This course has been partly developed from material found in the following sources.

- Getting Started with MATLAB7: A quick introduction for scientists and engineers, Rudra Pratap. Oxford University Press, 2006.
- A Beginner's Guide to MATLAB, Christos Xenophontos. Department of Mathematical Sciences. Loyola College. 2005 version.
- Introduction to Scientific Programming in Matlab, Guy-Bart Stan (lecture slides). Imperial College. 2010.