(Concepts of) Machine Learning

Familiarising with MATLAB

This activity introduces you to Matlab programming language and the simulation environment. After finishing this activity, you should be able to:

- Use basic Matlab commands;
- Define variables in Matlab;
- Use Matlab expressions;
- Work with matrices;
- Create plots;
- Use program control constructs in Matlab.

What is MATLAB?

The name MATLAB stands for MATrix LABoratory. MATLAB is a high-performance language for technical computing. It integrates computation, visualisation, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Maths and computations;
- Algorithm development;
- Modelling, simulation, and prototyping;
- Data analysis, exploration, and visualization;
- Scientific and engineering graphics;
- Application development, including Graphical User Interface building;

Help and Online Documentation

There are several different ways to access information about MATLAB functions. The basics ways include:

- The use of the help command
- The use of the help window
- The MATLAB Help Desk

The help Command

The `help` command is the most basic way to determine the syntax and behaviour of a particular function. Information is displayed directly in the command window. For example:

```
Type: >> help exit

           EXIT   Exit from MATLAB.
           EXIT terminates MATLAB.
```

The command 'help' by itself lists all the directories of ready made functions provided by Matlab, followed with a short description of each function category

```
Type: >> help

           matlab/general
           matlab/ops
           ...
```

The Help Window

The MATLAB help window is available on PCs and Macs by selecting the Help Window option under the Help menu, or by clicking the question mark on the menu bar. It is also available on all computers by typing

```
>> helpwin
```

To use the help window on a particular topic, type

```
>> helpwin topic
```

The help window gives you access to the same information as the help command, but the window interface provides convenient links to other topics.

The lookfor Command

The 'lookfor' command allows you for keyword-based searching. It searches through the first line of help text of each Matlab function, which is known as the H1 line, and returns the H1 lines containing a specified keyword. For example, MATLAB does not have a function named `inverse`. So the response from:

```
>> help inverse

inverse.m not found.
```

But if you use the `lookfor` command:

```
>> lookfor inverse
```

It finds over a dozen matches. Depending on which toolboxes you have installed, you will find entries like:

```
INVHILB  Inverse Hilbert matrix.
ACOSH  Inverse hyperbolic cosine.
ERFINV Inverse of the error function.
INV  Matrix inverse.
…
```

Another simple example, which shows the difference between the Help and Lookfor command, is the following one:

```
>> help sqrt

   SQRT Square root.
   SQRT(X)is the square root of the elements of
   X.Complex results are produced if X is not
   positive.
   See also SQRTM.
   Overloaded methods help sym/sqrt.m
```

When we type:

```
>> lookfor sqrt

   SQRT Square root. SQRTM Matrix square root.
   SQRT  Symbolic  matrix  element-wise  square
   root.
```

Other important commands

- Read information page by page:

  ```
  >> more on
  ```
- Disable pagination of information:

  ```
  >> more off
  ```
- Load Matlab editor

  ```
  >> edit
  ```

  For example, type: `>> edit appcr1`

Expressions

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices.

Variables

- They are defined and allocated dynamically;
- Assignment: `variable =value`;
- Assignment into an existing variable overrides its old value.
- They are case-sensitive.

Examples:

1. `b` and `B` are different variables;
2. Using an undefined variable causes `error`;
3. The assignment `num_students = 25`
   creates a 1-by-1 matrix named `num_students` and stores the value `25` in its single element.

Numerical operations

- addition • '+'
- subtraction • '–'
- multiplication '*'
- division '/'
- power '^'; for example 2^4 in Matlab is produced by the command: >> **`2^4`**

For more about the operators and special characters type:

```
>> help +
```

Functions

MATLAB provides a large number of standard elementary mathematical functions, including `abs`, `sqrt`, `exp`, and `sin`. For a list of the elementary mathematical functions, type:

```
>> help elfun
```

Working with Matrices

You have only to follow a few basic conventions to define a matrix:

- Separate the elements of a row with blanks or commas.
- Use a semicolon ';' to indicate the end of each row.
- Surround the entire list of elements with square brackets, **`[ ]`**.

Example:

Simply type: >> **`A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]`**

MATLAB displays the matrix you just entered

```
A = 16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Subscripts
The element in row `i` and column `j` of matrix `A` is denoted by **`A(i,j)`**.

For example,

```
A(4,2)
```

gives the element in the fourth row and second column. In our case it is the number **15**.
Type in Matlab command prompt `A(4,2)` to take this number.

If you try to use the value of an element outside of the dimensions of the matrix, it will produce an error.

Example:

```
>> t = A(4,5)
```

then you will take the following message in Matlab command window:

```
'Index exceeds matrix dimensions.'
```

This happened because the matrix *A* has only 4 columns!

Entering 1D-arrays (vectors)

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

A =

  1     2     3     4     5     6     7     8     9
```

Entering 2D-arrays (matrices)

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

A =

     1     2     3
     4     5     6
     7     8     9
```

Array operations

```
>> A*A (multiplication)

ans =

    30    36    42
    66    81    96
   102   126   150
```

```
>> A+A (addition)

ans =

     2     4     6
     8    10    12
    14    16    18
```

The above operation follows the general rule for matrix multiplication. You can find more details in Wikipedia:
https://en.wikipedia.org/wiki/Matrix_multiplication

Generating Matrices
MATLAB provides four functions that generate basic matrices.

- **Initialising a matrix to zero**

```
>> Q= zeros(3,2)

Q =

     0     0
     0     0
     0     0
```

- **Create array of all ones**

```
>> Q= ones(3,2)

Q =
     1     1
     1     1
     1     1
```

- **Create an identity matrix**

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

Q =
     1     0     0
     0     1     0
     0     0     1
```

- **Initialising a matrix with uniformly distributed pseudorandom numbers in the open interval (0,1)**

```
>> Q=rand(3,2)

Q =
    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621
```

- **Initialisation with pseudorandom values drawn from the standard normal distribution**

```
>> Q=randn(3,2)

Q =
   -0.4326    0.2877
   -1.6656   -1.1465
    0.1253    1.1909
```

The sequence of numbers produced by **`randn`** is determined by the internal state of a random seed.

- **Calculate all numbers over an interval**

`>> x=-1:0.1:1` <this is for the interval (-1,+1)>

```
x =
 Columns 1 through 7

   -1.0000   -0.9000   -0.8000   -0.7000   -0.6000   -0.5000
-0.4000

  Columns 8 through 14

   -0.3000   -0.2000   -0.1000         0    0.1000    0.2000
0.3000

  Columns 15 through 21
```

```
    0.4000    0.5000    0.6000    0.7000    0.8000    0.9000
1.0000
```

- **Calculate values of a function y**

```
                    >> y=1./(1+exp(-x))
```

Division here is an Entry-wise operation and **exp** defines %the exponential function in Matlab. The following %Wikipedia article is about the Exponential function, e^{-x}, http://en.wikipedia.org/wiki/Exponential_function

```
y =

  Columns 1 through 7

    0.2689    0.2891    0.3100    0.3318    0.3543    0.3775
0.4013

  Columns 8 through 14

    0.4256    0.4502    0.4750    0.5000    0.5250    0.5498
0.5744

  Columns 15 through 21

    0.5987    0.6225    0.6457    0.6682    0.6900    0.7109
0.7311
```

Store a matrix in an M-file

```
                    >> edit
```

% A new file titled Untitled opens in the MATLAB Editor (or default editor).

- Type: **`A= [1 2 3; 4 5 6; 7 8 9]`**
- Save the file: Use the **`SAVE`** command; use "**`data`**" as filename, e.g. you can type **`save('data','A')`** in the command window
- Type in Matlab command window

`>> open data.mat`

- To get the matrix displayed type

`>> open A`

```
A =
     1     2     3
     4     5     6
     7     8     9
```

Graphics

This section describes some of the most important graphics functions and provides examples of some typical applications.

Creating a Plot

The plot function has different forms, depending on the input arguments. If **`y`** is a vector, **`plot(y)`** produces a piecewise linear graph of the elements of **`y`** versus the index of the elements of **`y`**. If you specify two vectors as arguments, **`plot(x,y)`** produces a graph of **`y`** versus **`x`**, which is useful for creating the graph of a *function* - a *function* is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output. The following article in Wikipedia provides further information about functions in mathematics: http://en.wikipedia.org/wiki/Function_(mathematics)

For more details type:

```
>> help plot
```

Plot a function:

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

Example: Create the plot of the function $y = \frac{1}{1+e^{-x}}$, where x takes values in the interval $[-10,+10]$.

Flow Control

MATLAB has five flow control constructs:

- `if` statements
- `switch` statements
- `for` loops
- `while` loops
- `break` statements

If statement

The **`if`** statement evaluates a logical expression and executes a group of statements when the expression is true. The optional **`elseif`** and **`else`** keywords provide for the execution of alternate groups of statements. An **`end`** keyword, which matches the `if`, terminates the last group of statements. The groups of statements are delineated by the four keywords – no braces or brackets are involved.

For more details type:

```
>> help if
```

For

The **`for`** loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

```
>> x=[]; for i=3:-1:1, x=[x,i^2], end
```

How does this work analytically?

```
>> x=[]

x =
     []

>> for i=3:-1:1, x=[x,i^2], end

x =
     9

x =
     9     4

x =
     9     4     1
```