January 5, 2017
Yunan Liu

MATLAB TUTORIAL

1 What is MATLAB?

Matlab is a software package that integrates computation, data visualization, and high-level programming. Matlab is specifically designed for the manipulation and visualization of matrices and analysis of large amounts of data.

2 How to use MATLAB?

Once you have started up Matlab, you will see the Command Window. Matlab commands are executed by typing them at the >> prompt and then striking the ENTER key. This is followed by Matlab replies. In the sequel the lines with >> are command lines and the rest are Matlab replies.

To quit Matlab, type "quit" from the command line. e.g.,

```
>> quit
```

Typing "help general" will give information on general purpose commands useful for navigating Matlab. UNIX shell commands may also be executed by putting a ! before the command. For example, to edit the file "words.txt" using emacs while running Matlab type "!emacs words.txt".

3 MATLAB Variables and the Workspace

Matlab was originally written as a matrix manipulation program, and therefore tends to try to deal with everything as a matrix. Although it is possible to input equations, assign variables, and use a lot of mathematical functions, in the end it is necessary to understand matrices to understand Matlab. To enter a small matrix, type

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

and Matlab responds with

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

This matrix can also be entered in its usual form with carriage returns replacing the semicolons.

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

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

When entering a matrix, elements are separated by a space, rows are separated by a semicolon, and the matrix is enclosed in square brackets.

The above assignment "A =" creates a variable A with the above matrix as its value.

Matlab stores variables in an initially empty workspace. Once a variable has been assigned it is stored in the workspace and may be referred to again until redefined. To get a list of variables in the workspace type "who".

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

A =
    1 2 3
    4 5 6
    7 8 9

>> b = [1 2 3]

b =
    1 2 3

>> who

Your variables are:

A       b
```

or "whos" for a more detailed description

```
>> whos
Name     Size     Bytes     Class

  A       3x3        72      double array
  b       1x3        24      double array

Grand total is 12 elements using 96 bytes
```

4 Displaying and Suppressing Outputs (Comments)

After typing a Matlab command, the output of that command is displayed.

```
>> c = 5

c =
    5
```

To execute a command and not display the output follow the command with a semicolon, e.g., typing "c;" at the command line does not display the value of c whereas typing c without the semicolon displays the value of c as the following examples illustrate.

```
>> c;
>> c
c =
    5
```

Use the symbol "%" to make comments. Anything after a % is not seen as a command, e.g.,

```
>> 5 % The rest is a comment
c =
    5
```

5 Matrices

In Matlab, a matrix is a rectangular is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. Where other programming languages work with numbers one at a time, Matlab allows you to work with entire matrices quickly and easily.

5.1 Matrix Operations and Solving Matrix Equations

Scalar multiplication of matrices is supported. For example, if A is a matrix, then "2*A" returns the matrix formed by multiplying each element of A by 2.

The usual matrix operations are supported, viz., +, - and *, the last being matrix multiplication.

Exponents are also supported by ^, e.g., 2^3 = 8, 4^0.5 = 2. This works for square matrices too. For example, if A is a square matrix, A^3 returns the cube of A, i.e., A*A*A, and A^0.5 returns the square root of A, i.e., the matrix whose square is A. If A is also nonsingular, this works for negative integer powers of A too, e.g., A^(-2) returns the square of the inverse of A. Alternately, the built-in function inv(A) also returns the inverse of A.

Division is done a little differently. If A is a nonsingular matrix and B is a matrix with the same number of rows as A, then A\b returns (inverse of A)*B. Thus, for example, the solution of the linear equations $Ax = b$ is x = A\b. Similarly, if B is a matrix with the same number of columns as A, then B/A returns B*(inverse of A). Thus, the solution to the linear equations $yA = b$ is y = b/A.

A matrix may be transposed with ', i.e., A' returns the transpose of the matrix A.

If A is a vector, max(A) returns the largest element of A.

If A is a matrix, M = max(A) returns the row vector M containing the maximum element in each column of A. Also, [M, I] = max(A) returns both M and the indeces I of the maximum elements. If there are

several such indices, the index is the smallest one. For example,

```
>> A = [1 3 0
    0 4 0]

A =
    1 3 0
    0 4 0

>> [M, I] = max(A)

M =
    1 4 0

I =
    1 2 1
```

The function min(A) works analogously after replacing max by min everywhere.

Other functions of a square matrix A are built-in too (see a later section), e.g., the matrix exponential expm(A) of A, the eigenvalues eig(A) of A, the determinant det(A) of A, etc.

5.2 Special Matrices (Identity, Zeros, Ones)

The function eye(n) returns the identity matrix of order n.

The function zeros(m, n) returns the m×n matrix of zeros.
The function zeros(n) returns the n×n matrix of zeros.

The function ones(m, n) returns the m×n matrix of ones.
The function ones(n) returns the n×n matrix of ones.

5.3 Selecting Elements and Submatrices of a Matrix

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

>> A(2,3)

ans =
    6
```

Submatrices may be extracted using vectors, e.g.,

```
>> u = 1:2;

>> v = [1 3];

>> A(u, v)

ans =
```

```
    1 3
    4 6
```

gives the submatrix defined by the first two rows and the first and third columns. An entire row or column may be selected by using a colon : instead of an index or a vector, e.g.,

```
>> A(:, 2)

ans =
    2
    5
    8
```

gives the second column of A.

5.4 More Matrix Functions

```
>> help matlab/matfun

Matrix functions - numerical linear algebra.

Matrix analysis.
norm       - Matrix or vector norm.
normest    - Estimate the matrix 2-norm.
rank       - Matrix rank.
def        - Determinant.
trace      - Sum of diagonal elements.
null       - Null space.
orth       - Orthogonalization.
rref       - Reduced row echelon form.
subspace   - Angle between two subspaces.
⋮
```

Typing "help rank" will give information about the function rank, e.g.,

```
>> help rank

RANK    Matrix rank.
   RANK(A) provides an estimate of the number of linearly independent
   rows or columns of a matrix A.
   RANK(A,tol) is the number of singular values of A that are larger
   than tol.
   RANK(A) uses the default tol = max(size(A))*norm(A)*eps.

Overloaded methods
   help sym/rank.m
```

In general, to find out about the predefined function "func", type "help func".

6 Graphs and Plotting

Matlab supports graphing 2-D plots. Suppose y is a matrix. Then plot(y) graphs the columns of y versus their indices.

Suppose also that x is a vector with the same number of elements that y has rows. Then plot(x, y) graphs the columns of y versus x.

```
>> x = -5:5
x =
    -5 -4 -3 -2 -1 0 1 2 3 4 5
>> plot(x, x.^2)
```

plots x on the x-axis and x squared on the y-axis.

The title, axis, axis labels, etc. of graphs can be altered with built-in functions.

Matlab also supports 3-D plots.

There are two areas matlab/graph2d and matlab/graph3d containing built-in functions for graphs.

7 M-Files

We can write Matlab "programs" using m-files. An m-file is a script of Matlab commands in a text file with the extension ".m". You can create m-files using a text editor, then use them as you would any other Matlab function or command.

There are two kinds of m-files:

1. Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace.
2. Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

7.1 Scripts

Scripts store blocks of Matlab commands that can be applied to variables that already exist in the workspace. For example, consider the file "twoA.m" with the single line

```
B = 2*A
```

When in Matlab, create any matrix A

```
>> A = [1 2; 3 4];
```

then run the m-file twoA.m by typing

```
>> twoA
```

Matlab responds with

```
>> B
B =
    2 4
    6 8
```

In general, executing the m-file "filename.m" is done by typing "filename" in Matlab. Note that filename.m must be in the current directory.

7.2 Functions

User-written functions are also supported by Matlab. They are similar to m-files with the following differences:

- the first line of a function file defines the function
- they accept input and output
- they don't affect the workspace

Consider the function "twoA" that produces the same result as the m-file twoA.m

```
function B = twoA(A)
B = 2*A;
```

The first line specifies that this file defines a function "twoA" that takes one input A and returns one output B. The second line defines B to be 2*A. The function may then be used in Matlab. Create any matrix A in Matlab

```
>> A = [1 2; 3 4];
```

and suppose the matrix B also exists

```
>> B = [1 1];
```

The function is called as if it were a built-in function

```
>> A = twoA(A);
>> A
A =
    2 4
    6 8
```

The function multiplied A by 2 as expected. Even though B is used inside the function file, B is not affected in the workspace.

```
>> B
B =
    1 1
```

Obviously, functions provide much greater flexibility and modularity than scripts. Note that the m-file and function names should coincide if the function is to be called from the command line (or from another function or script).

7.3 Control Expressions and Looping

Matlab supports if, for, and while statements. The following examples show how to use these expressions:

```
if x == 0,
   x = x + 1;
elseif x == 1,
   x = x - 1;
else x = 0;
end

for i = 1:10,
   j = 10*i;
end

i = 0;
while i < 10,
   i = i + 1;
end
```

7.4 Example

The following example combines the concepts of m-files, control expressions, and loops. The function "findMaxMinAvg" takes a vector `A` as input, then returns a vector containing the maximum, minimum, and average value of the components of `A`.

First, the following code is saved as an m-file called "findMaxMinAvg.m".

```
function [maximum, minimum, average] = findMaxMinAvg(A)

% Step 1.  Initialize temporary (local) variables
maximum = A(1);
minimum = A(1);

% Step 2.  Loop through the values of A and update max and min as necessary
for i = 1:length(A)
```

```
    if maximum <= A(i)
       maximum = A(i);
    end
    if minimum >= A(i)
       minimum = A(i);
    end
end

% Step 3.  Calculate the average of the components of A
average = sum(A)/length(A);
```

Now suppose we would like to use the function "findMaxMinAvg" to compute the maximum, minimum, and average of the vector

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

Equivalently, you can create this input vector by entering

```
>> myVector = [1:10];
```

Next, issue the command

```
>> [Max, Min, Avg] = findMaxMinAvg(myVector)
```

Matlab will return:

```
Max =
    10

Min =
    1

Avg =
    5.5000
```

7.5 Built-in functions

Matlab has many built-in functions that can be applied to both scalars and matrices. When applied to a matrix, the function is applied element-wise, and the output is a matrix of the same dimensions of the input. To get a complete list of built-in functions type:

```
help elfun
```

For more advanced functions type:

```
help specfun
```

```
help elmat
```

A few examples of built-in functions are given below:

`sin`	Sine
`cos`	Cosine
`tan`	Tangent
`asin`	Inverse sine
`acos`	Inverse cosine
`atan`	Inverse tangent
`exp`	Exponential
`log`	Natural logarithm
`sqrt`	Square root
`abs`	Absolute value
`floor`	Round towards minus infinity
`ceil`	Round towards plus infinity
`round`	Round towards nearest integer
`gamma`	Gamma function
`beta`	Beta function

8 Random Numbers

It is very easy to generate random numbers in Matlab as follows. The "rand" command produces uniform random numbers between 0 and 1.

```
>> rand
ans =
    0.9501
```

rand(5,1) produces a vector of 5 independent uniform random numbers

```
>> rand(5,1)
ans =
    0.1987
    0.6038
    0.2722
    0.1988
    0.0153
```

randn produces a Normal(0,1) random variable (mean 0 and variance 1).

```
>> randn
ans =
   -0.4626
```

For example, to generate a Normal random variable with mean μ and variance σ^2, recall that if X is a Normal(0,1) random variable, then $Y = \sigma X + \mu$ has a Normal(μ, σ^2) distribution, so to generate a Normal(3,4) random variable, type

```
>> 2*randn + 3
ans =
    2.1349
```

The following example shows how to generate an exponential random variable with parameter μ (or equivalently, with mean $1/\mu$). Using the inversion method, if X is uniformly distributed in (0, 1), then $Y = F^{-1}(X)$ has an exponential distribution with parameter μ, where

$$F(x) = 1 - e^{-\mu x} \qquad \text{and} \qquad F^{-1}(x) = -\frac{1}{\mu}\log(1-x)$$

Therefore,

```
>> -log(1-rand)/2
```

returns an exponential random variable with parameter 2 (mean $1/2$), and

```
>> -log(1-rand(10,1))/2
```

returns a column vector of length 10 whose elements are independent exponential random variables with parameter 2.