QUICK INTRODUCTION TO MATLAB
PART II

Department of Mathematics
University of Colorado at Colorado Springs

MATLAB is a versatile programming language designed primarily for operating with numerical data. In this second worksheet, you will learn the basics of generation, storage and analysis of such data. You will also get acquainted with an essential tool in MATLAB programming: script m-files and function m-files.

Basic Data Analysis

The following functions can be used to perform data anaylsis

- **max, min** (maximum and minimum of a set of values)
- **find** (find indices of nonzero elements)
- **mean, median, std** (find the average or mean, median and standard deviation)
- **sort** (sort in ascending order)
- **sortrows** (sort rows in ascending order)
- **sum, prod, diff** (sum, product and difference between elements)

MATLAB has a strong preference to work with columns rather than with rows. Most of the functions work with columns when given matrix arguments. For example

```
a= rand(3) % generates a square matrix (3 by 3) of random number
between
           0 and 1
```

```
a =
    0.2028    0.2722    0.7468
    0.1987    0.1988    0.4451
    0.6038    0.0153    0.9318
```

```
m=max(a)
```

```
m =
    0.6038    0.2722    0.9318
```

max returns a vector containing the maximum value of each column. When given a vector, max returns the maximum value

```
max(m)
```

```
ans =    0.9318
```

To find the index correspoding to the maximum value, supply two output arguments:

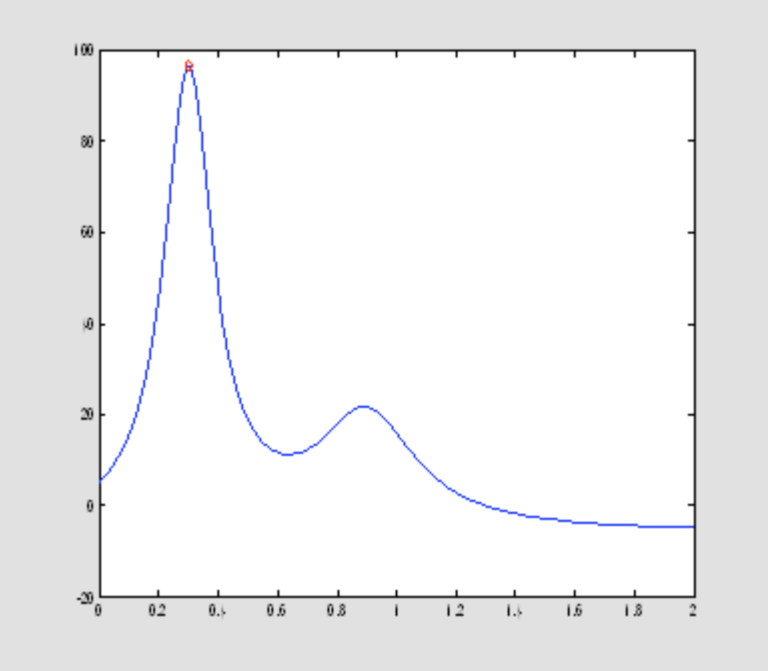
```
[v,ind]=max(m)
```

```
  v =    0.9318
ind =     3
```

The first one is the maximum value and the second is the index of the maximum value. If the maximum element is multiple, ind is the index of the first element. Here is another example:

```
x=0:.01:2;
y=humps(x);
plot(x,y)
[v,ind] = max (y)
hold on
plot(x(ind),y(ind), 'rd')
x(ind), y(ind)
```

```
  v =   96.5000
ind =   31
ans =    0.3000
ans =   96.5000
```

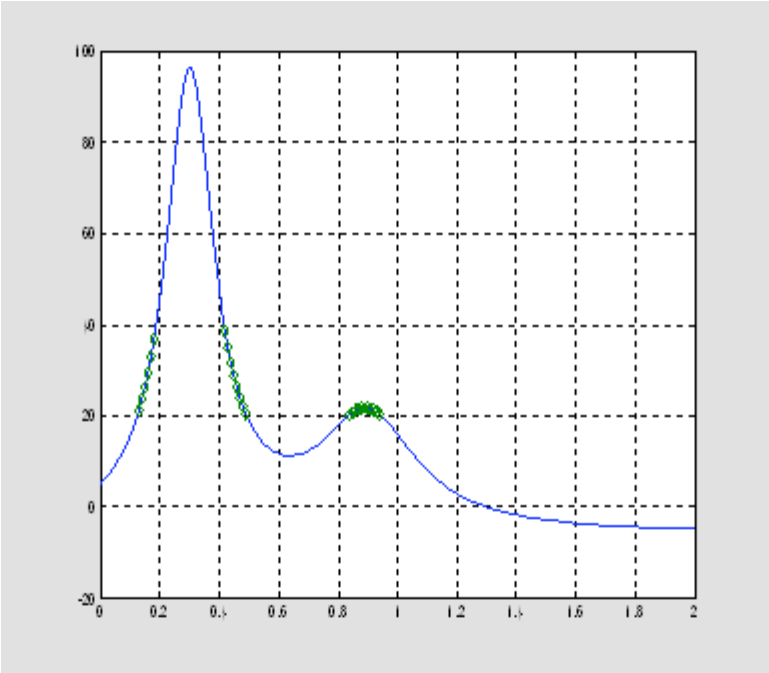


- Logical and relational operators
    - **"=="** equal to
    - **"~="** not equal to
    - **"<", ">", "<=", ">+"** – the usual less than, greater than, less than or equal to and greater than or equal to
    - **"&"** AND
    - **"|"** OR

- **"~"** NOT
- **xor** – Exclusive or
- **any** – True if any element is non-zero
- **all** – True if all elements are non-zero

can be used in conjunction with the function **"find"**. Here is an example

```
clf
ind = find(20<=y & y <=40);
plot(x,y,x(ind),y(ind),'d')
grid
```



The "find" function can be used also with a matrix argument to locate and list the elements that satisfy a logical test.

```
s= spiral(4)
s<8
find(s<8)' % the transpose "'" displays the column vector as a row
s(s<8)'  % list the elements of the matrix which are less than 8
```

```
s =
     7     8     9    10
     6     1     2    11
     5     4     3    12
    16    15    14    13
ans =
     1     0     0     0
     1     1     1     0
     1     1     1     0
     0     0     0     0
ans =
     1     2     3     6     7    10    11
ans =
     7     6     5     1     4     2     3
```

MATLAB has predefined mathematical functions such as

- **sin, cos, tan, sec** : trigonometric functions
- **asin, acos, atan** : inverse trigonometric functions
- **sinh, cosh, tanh, sech** : hyperbolic functions
- **exp, log, log10** : exponential functions

Example:

```
 a1=cos(1+2*i) % MATLAB functions work for complex arguments,
a2=exp(2+3*i)
a3=sech(-2:1:2) %                  and for vector arguments,
a4=asin([0,1; -1, 0.5; -0.5,0])%   even for matrix arguments.
% In all cases, the output of MATLAB functions reproduces the input
structure.
```

```
a1 =   2.0327 - 3.0519i
a2 =  -7.3151 + 1.0427i
a3 =    0.2658    0.6481    1.0000    0.6481    0.2658
a4 =         0    1.5708
       -1.5708    0.5236
      -0.5236         0
```

Other useful functions are

- **real, imag, conj** : real, imaginary part and complex conjugate of a complex number
- **abs, angle** : absolute value and phase angle of a complex number

M- Files

For simple tasts, you learned so far how to input MATLAB commands directly. For more complex tasks though, it is more convenient to store the typed input into a file with extension ".m" called an *m-file*. You can tell MATLAB to get its inputs from these files. These are the *script m-files*. There are also m-files which accept input and produce outputs and are called *function m-files*. Often they are reffered to as simply scripts and functions.

- script m-files
  - can include anything the user writes in the Command window.

  - a script m-file can be called inside other m-files. The variables are shared between the parent and the children m-files.
  - **echo on** – all inputs appear on the screen while they are being executed

- function m-files
  - each function is saved as a separate m-file
  - a function m-file starts with the line

    function [y1,y2,…,yN] = functionName(x1,x2,…,xM)

    where
    - x1,x2,…,xM are M input arguments
    - y1,y2,…,yN are N output values
    - functionName – same as the name of the m-file

As for new variables, when introducing new function m-files, make sure that the name does not conflict with already defined functions.

Using your text editor, create a file called **mfile1.m** containing the following lines:

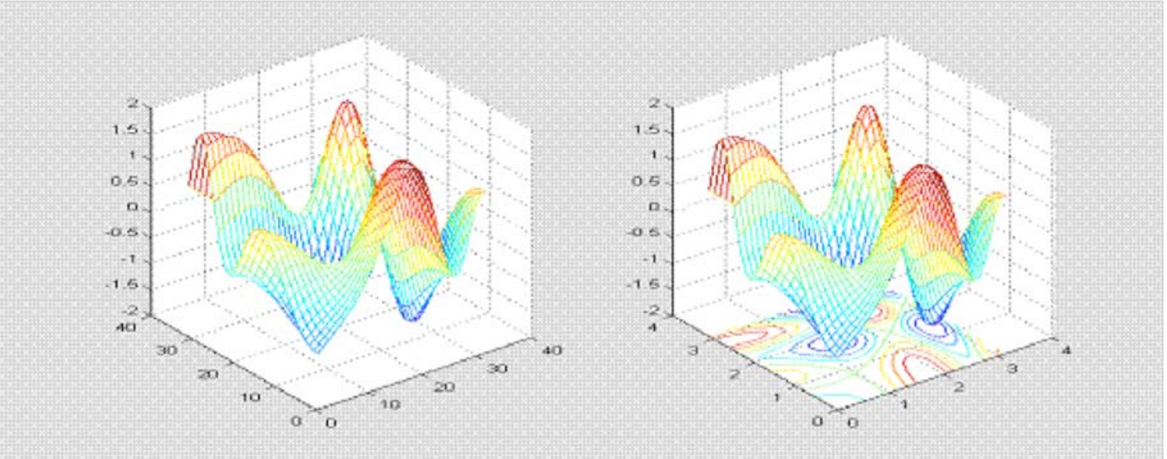
```
x=0:.1:pi; y=0:.1:pi;
[X,Y]=meshgrid(x,y); % produce matrices X,Y such that each row of X is
a % copy of the vector x and each column of Y is a copy of the vector
y.

Z = sin(Y.^2+X)-cos(Y-X.^2);
subplot(121)
mesh(Z) %

subplot(122)
meshc(x,y,Z) %
```

Now type in the MATLAB window.

```
mfile1
```



The following is an example of a user-defined function:

```
function f_avg = trapez(f,x)

% Computes the weighted average of the function f at N points x(1), …,
x(N), using the formula
%       f_avg=(f(x(1))+2 f(x(2))+…+2 f(x(N-1))+f(x(N)))/(2N-2)
% where the input x =[x(1),…,x(N)] is a vector of dimension N.

N=length(x);
fx=feval(f,x);
f_avg=(2*sum(fx)-fx(1)-fx(end))/(2*N-2)
```

The name of the m-file must coincide with the name of the function on the first line, in the present case it must be entitled **trapez.m** . Have you encountered such weighted averages before? They are useful in the numerical integration of functions, e.g. the trapezoidal rule. The comment lines which follow the definition line in a function m-file can be dispayed in the MATLAB window simply by typing **`help trapez`** at the command prompt.

The function 'trapez' can be called in a different script or simply at the command prompt, e.g.

```
x=0:.1:pi;
trapez('sin',x)
```

```
f_avg =
    0.6443
ans =
    0.6443
```

Reading and Writing Operations

There are several ways to communicate data with Matlab. The simplest one is

- input : takes the input data from the user.

```
input('Do you want to continue (y or n)?','s') % 's' indicates Matlab to expects a string
response
```

```
Do you want to continue (y or n)?
```

or

```
r=input('Enter the initial value:') % Matlab expects a numerical response
```

```
Enter the initial value: 4
```

```
r =   4
```

- The conversion between numerical inpout and string input can be done using the functions **str2num** and **num2str:**

```
b1=7328 % this is a number
b2='7328'  % this is a string
b3 = str2num(b2)
b4=num2str(b3)
length(b1) % b1 is a scalar, hence of length one
length(b2) % b2 is am array of four charachers
(b1==b3) % compare the two variables
(b2==b4) % compare the two arrays, on components (must be of same
size!)
```

```
b1 =      7328
b2 = 7328
b3 =      7328
b4 =  7328
ans = 1
ans = 4
ans = 1
ans = 1     1     1     1
```

- **`disp`** is used to display variable on the screen as below:

```
disp([' The value of pi with 6 decimals is ' num2str(pi,6)] )
```

```
 The value of pi with 6 decimals is 3.14159
```

- **`fprintf and sprintf`** are used for more sophisticated formatting such as

```
fprintf('%6.4f\n', pi) % the character "%" requests a field of width 6
                      with 4 decimals. "\n" indicates new line
```

```
3.1416
```

The standard formats (same as in C language) for data output are

"%10.2f" – fixed point format
"%10.2e" –floating point format
"%10.2g" – the shortest of fixed or floating point format
"%10.0f" – fixed point format for a rounded integer part of a number
"%d" - dipspalys an integer
"%s" - displays a string

More examples:

```
fprintf(' %g \n %e', exp(1), -exp(1))
```

```
 2.71828
 -2.718282e+000
```

Other special characters:

- "\n" indicates a new line.
- "\t" tab character
- "\r" carriage return character
- "\\" backslash character
- "%%" percent character

Data File Input and Output

To save all the variables from the worskspace onto disk you should use the command **save**.

```
save datafile % saves the variables to a disk file called datafile.mat
```

To load certain variables, specify them after the filename. For example

```
load datafile x % loads the variable called x from datafile.mat
```

Alternative syntax is save('datafile') and load('datafile','x').

To load a list or table of numbers in ASCII format, stored say in a file data1.dat, one simply type the command load data1.dat. This would load the data into a variable called data1. If the ASCII file contains a table, the variable would be a matrix of the same size as the table.

- fopen, fclose are commands for file opening and closing

```
fid=fopen('filename','w') % open the file "filename" for writing
                             operations
```

fid is an integer, called a file identifier (uniquely identifies the file in a given workspace)

The options are:
'r' – read
'w' –write (create if necessary)
'a' - append (create if necessary)
'r+' - read and write (do not create)
'w+' - truncate or create for read and write

'a+' – read and append (create if necessary)

```
fclose(fid)  % close the file with the file identifier 'fid'
fprintf(fid,'textname') % write the text "textname" into the file with
                     the file identifier "fid"
fwrite(fid, variablename) % write the variable "variablename" into the
                      file with file identifier "fid"
```

Other I/O commands: **`fscanf, fread, fgets, fget.`**
Follow the example below.

```
A = [ 0, 23, 45, 100];
fid=fopen('temp', 'w');
fprintf(fid, '%3.0f degrees Celsius = %3.0f degrees Fahrenheit\n', [A;
9*A/5+32]);
fclose(fid);
```

This creates a file "temp.dat" containing

```
  0 degrees Celsius =  32 degrees Fahrenheit
 23 degrees Celsius =  73 degrees Fahrenheit
 45 degrees Celsius = 113 degrees Fahrenheit
100 degrees Celsius = 212 degrees Fahrenheit
```

To read this file, use the following

```
fid=fopen('temp','r');
X=fscanf(fid,'%f degrees Celsius = %f degrees Fahrenheit')
fclose(fid);
```

```
X =
     0
    32
    23
    73
    45
   113
   100
   212
```

Finally, we can convert the vector output X to the original matrix format

```
X=reshape(X,2,4)'
```

```
X =
     0    32
    23    73
    45   113
```

```
   100   212
```

When working with different directories in MATLAB, the following commands can be useful:

- **cd** - Change to another directory
- **pwd** - Dispray current working directory
- **dir** - Display contents of current working directory
- **what** - Display MATLAB-relevant files in current directory
- **which** – Display directory containing specific function
- **type** – Display file in MATLAB window
- **path** - Display or change the search path
- **addpath** – Add directory to the search path
- **rmpath** – Remove directory from the search path