EENG 212-M01
EENG 221-M01 Prof. L. Amara

Tutorial #2
Matlab: Vectors, Matrices and Arrays
Array and Matrix Operations
Basic Plotting

This is a continuation to the introduction to Matlab in tutorial 1. At this time you must all be familiar with Matlab environment and know how to launch Matlab, be able to code arithmetic operations, create an M-file, and execute a Matlab script.
As a reminder:

-Variables have to be defined before being used in a program otherwise their use will result in an error message. A variable name can have up to 31 characters that may consist of the decimal digit 0,..,9, the Latin letters a-z and the underscore (_); however it must start with a letter.
For example we can have name such a CamelCap, Camel_Cap, Day_2, Vx_1.

-Matlab is case sensitive. This is true for variables, functions and Matlab commands.
-A semi-colon (;) at the end of a command saves the result but suppresses the screen output
-All computations are carried out in double precision. The numbers appear according to the format you defined. For example, after entering any of the following command, the entry **2*pi** will results into the number to the right.

>>format short	6.2832
>>format short e	6.2832e+000
>>format long	6.28318530717959
>>format long e	6.283185307179586e+000
>>format hex	401921fb54442d18
>>format rat	710/113
>>format bank	6.28

Important to know : all comments start with **%**. They can be placed on lines or at the end of any Matlab command. If you are using the Matlab editor, all comments show with a green color.

1-Working with Arrays

Matlab is best suited for working with matrices and arrays. In fact Matlab stands for Matrix Laboratory.

1.1 Listing the elements of a matrix

A matrix is a variable that has multiple rows and columns, while a vector variable is a special matrix with only one column or one row. A matrix with one row and one column

is also called a scalar. An m by n matrix is a rectangular array of numbers having m rows and n columns. When m=2 and n=3, we have a 2x3 matrix, say for example A,

When using Matlab we can enter A as: A= [32 -14 89 ,
10 23 4]

Both vectors and matrices are also referred to as being arrays, a name commonly used in programming languages.

```
>> a= [  1  2  3  ]      %  is a row vector with 3 elements  listed between square brackets
```
The vector a can also be written as:
```
>>a=[ 1, 2, 3]     % now matlab returns the following

a=

    1     2     3

>>  b= [ 2 ; 4 ; -5]     % is a column vector with 3 elements
 b =
     2
     4
    -5
```

A scalar does not need brackets
```
>> g=9.12
```

The Colon notation **(:)** is a practical way to produce vectors;
So
```
>>%vector with starting element 1 and ending element 6 and by default  progression1.
>>x=1:6
```

Results in :
```
   Ans=  1   2  3  4  5  6
```

And in the following form:
```
   x= a: b: c
```
Means ‘a” is the starting number, b, the increment and c the final number of the vector

```
 >> x=1:0.2:2
```

Results in
```
 ans= 1  1.2 1.4 1.6  1.9  2
```

Now a 3 by 3 matrix can be entered as:
```
>>x=[1.3  1.4  4.5 ;  0   4   7 ;  1.8  -4.5   3 ]
```

```
x =

   1.3      1.4   4.5
   0        4     7
   1.8     -4     3
```

You can add 4 to each element of the vector “a” defined previously:

```
>> a+ 4      %  vector “a” as defined before.

 ans =

    3    4    5
```

You can add (or subtract) two vectors of the same size

```
>> c=[ 0  2  6 ]   ; another vector of the same size as vector a defined before
>> a + c
ans =

    1    4    9
```

Of course you can not add (or subtract) a row vector to a column vector

```
>> a + b
??? Error using ==> +
Matrix dimensions must agree.
```

A matrix can also be entered row by row as follows

```
>>z= [   1,       9,      5
         20       0,    -5
          3,      21,     4]
```

If the data can not fit on one line, you can go to the next row after adding a comma and three periods:

```
>> t=[ 1, 3, 4,   5, 6, …
       2, 4, 6, 10, 0]
```

You can always define a new matrix by using another matrix. If we want to redefine the matrix p, knowing that p=[1 2 2 2 2 4], we can for example enter the command:

```
>> p=[p,p]
p =

    1    2    2    2    2    4    1    2    2    2    2    4
```

Each element of the matrix can be retrieved by using a reference number identifying the position of the element. Thus A(i,j) refers to the element of the matrix A, located at the intersection of row i and column j. This is basically index specification.

```
>>p(6)    % position six
ans =
    4
```

Likewise for the 3 by 3 matrix z, we can read the element in row 3 and column 2 by entering:

```
>> z(3,2)
ans =

   21
```

Also we can expand the vector a by adding another element in position 13 as follows:

```
>>p(13) = -4
```

And the new vector **a** is now:

```
a =

    1    2    2    2    2    4    1    2    2    2    2    4   -4
```

1.2 Dot Product:

The dot product operator works only on elements of arrays with equal dimensions.
If u and n are 2vectors with **n** elements each, then the following scalar operations can be performed:

```
u.*v =  [ u(1).v(1 )  u(2).v(2)  u(3).v(3) ……    u(n).v(n) ]
u./v =[u(1)/v(1)   u(2)/v(2)    …u(n)/v(n)]
u.^2= [ u(1)^2    u(2)^2   u(3)^2 …u(n)^2]
```

Dot Product and Matrices :

If the matrices A and B have the same size then A.*B will results in a new matrix, say C, such that $C_{m,n}=A_{m,n*}B_{m,n}$ for any set (m,n) where m is the row number and n is the column number.

```
>> A=[ 1  4  0 ;  2  5  7;   9 10   4 ];
>> B=[-2  3  2;  1  -5 4; 10 21 56];
>> C=A.*B

C =

   -2    12     0
    2   -25    28
   90   210   224
```

1.3 Size of a Matrix

The size(dimensions) of a matrix can be obtained by using the command **size.**

```
>>B= [ 3 5 7,
>>      -1  0 9] ;   % 2 by 3 matrix
```

The command
>>size(B) %will result in
ans =
2 3
That means it is a 2 x 3 matrix.

1.4 Tanspose of a Matrix

Transposing a vector means changing a row to a column and the concept works also for a matrix. For example
>> X1= [1 0 54 90] %and its transpose is
>>X1'=
>> 1
0
54
90

And
>>B'=
3 -1
5 0
7 9

1.5 Special Matrices

We know list a certain number of commands associated to special matrices:
ones(m,n) results in an mXn matrix of 1's
so **:**
>>ones(2,2)=
>> 1 1
1 1

zeros(m,n) results in an mXn matrix of 0's
so:
>> **zeros (3,2) =**
>> 0 0
0 0
0 0

1.6 The identity Matrix (noted I in mathematics)

An **n** by **n** matrix of zeros except for having ones along its leading diagonal is called an identity matrix. The command for the identity matrix is **eye(n)**
For example **I=eye(3)** results in
I= 1 0 0
0 1 0
0 0 1

Multiply s=[2; 3; 6] by I or S*I and see result.
If the diagonal has to have different numbers that 1's, we can use the command **diag(d)** where d is vector .
For example if

```
>>d=[  -23  45  21]      %then
>>diag(d)=
          -23     0     0
           0     45     0
           0      0     21
```

Now if A is any matrix (does not have to be squarre), then **diag(A)** extracts its diagonal numbers

1.7 Building Matrices

Larger matrices can be build with smaller ones.
If A=[1 3; 3 6; 0 -1] and B=[6; 7;12]
Then F=[A B] =

```
1    3    6
3    6    7
0   -1   12
```

1.8 Tabulating Functions

Let's illustrate this procedure with an example:

```
>>% tabulate  the functions  x= 2 sin(t) and y= 3 cos (3t) for t=1, 2,3….,10
>>t=1:1:10;
>>x=2*sin(t);
>>y=3*cos(3*t);
>>disp( 'Tabulated Functions');
>>disp('  t        x          y');
>>       [  t'       x'         y']
ans=

   1.0000    1.6829   -2.9700
   2.0000    1.8186    2.8805
   3.0000    0.2822   -2.7334
   4.0000   -1.5136    2.5316
   5.0000   -1.9178   -2.2791
   6.0000   -0.5588    1.9810
   7.0000    1.3140   -1.6432
   8.0000    1.9787    1.2725
   9.0000    0.8242   -0.8764
  10.0000   -1.0880    0.4628
```

Question: Compute A*A and A.*A . Notice the difference in the results. Make sure you know why the results are different!

1.9 Data files

Another way of listing the elements of a matrix is by using information stored in a data file. Data files of the ASCII type are created with a word processor and saved with extension **.dat** or by MATLAB editor. To save the ascii data file data_1, we enter the command:

>>save data_1.dat ; here the extension .dat must be added to the name of the file

To load the file, use the load command as follows:

>>load data_1.data; data is now stored in the matrix data_1

Example: Suppose some numeric data is stored in le 'table.dat' in the form of a table, as shown below.

```
100 2256
200 4564
300 3653
400 6798
500 6432
```

The three commands,

```
>> fid = fopen('table.dat','r');
>> a = fscanf(fid,'%3d%4d');
>> fclose(fid);
```

respectively

1. open a file for reading (this is designated by the string 'r'). The variable **fid** is assigned a unique integer which identifies the file used (a file identifier). We use this number in all subsequent references to the file.
2. read pairs of numbers from the file with file identifier **fid**, one with 3 digits and one with 4 digits, close the file with file identifier **fid.**
3. This produces a column vector with elements, 100 2256 200 4564 ...500 6432. This vector can be converted to 5 x 2 matrix by the command A = reshape(2,2,5)';.

1.10 More on the column operator

This is also a very useful operator that helps you read the elements of a matrix or create new matrices.

The column operator can be used to create new matrices

>>c= [1 2 3; 0 4 5; -1 2 9]

```
>>c =

    1.00    2.00    3.00
     0      4.00    5.00
   -1.00    2.00    9.00


>> r=c(:,1)  ; % read only first column of matrix c
>>r =

    1.00
     0
   -1.00
>>r2=c(2,:)          % read only second row of c
r2 =

     0      4.00    5.00
```
We can also create a submatrix
```
>>w=c(:,2,3) ;       % select columns 2 and 3  for the partial matrix w
>>w =

    2.00    3.00
    4.00    5.00
    2.00    9.00

>>w0=w(:)     % check the elements of the new matrix  w0
>>ans =

    2.00
    3.00
    4.00
    5.00
    2.00
    9.00
```
The following are more operations with the : operator
```
>>a=[ 100  -12  23  27; 0  -84  19   3  ; 32  -53  120  -43;  76  -23 10  -54];
>>a(3,:)                  %  selects the third row of matrix a
>>a(:, 1: 3)              %  selects the first through third columns of matrix a
>>a([ 1   2  ], : )       %  selects the  first and second row of matrix a
```

1.11Two special functions to create vectors

There are two frequently used built-in functions that are used to create vectors
linspace (a,b,n) generates a linearly spaced vector of length n from a to b

```
>>y=linspace (1, 2*pi, 10);

>>y =

    1.00    1.59    2.17    2.76    3.35    3.94    4.52    5.11
5.70    6.28
```

The other function is logspace(a, b, n) generate a logarithmically spaced vector of length n. It is the same operation as 10.^(linspace(a,b,n))

```
>> x=logspace(0, 4, 5);
>>x =

    1.00    10.00    100.00    1000.00    10000.00
```

1.12 User input

The element of a matrix can also be entered through keyboard by using the input command

```
>>f=input ('Enter values for f  in brackets:')    % Matlab will wait for you to key in the
                                                  %  value of  f
Enter values for f in brackets:
>>f=[ 1  4 ; 0 -3 ]
>>f =

    1.00     4.00
      0     -3.00
```

1.15 Null or Empty matrix

Matlab allows empty matrices

```
>>A=[  ]  ; matrix has no elements
```

A matrix that contains only zeros is not an empty matrix.

When the rows and columns of a matrix are computed in a loop , a null matrix can be used to initialize the matrix in question.

1.16 Large matrices

They should be initialized to a zero matrix first such that the PC will reserve the necessary memory for its elements. A matrix A with m rows and n columns can be initialized with the command

```
>>A=zeros(m,n);
```

1.17 Some utility matrices

eye (m, n)	returns an m by n matrix with 1's on the main diagonal
zeros(m, n)	returns an m by n matrix of zeros
ones(m, n)	returns an m by n square matrix of ones.
diag(z)	generates a diagonal matrix with vector z on the diagonal
diag(Z)	extracts the diagonal of matrix Z

```
>>eye(3)
ans =

     1.00          0          0
       0        1.00          0
       0          0        1.00
>> c=[3*ones(3)  zeros(3,2)   ; zeros(2,3)  2*eye(2)];
>> c =

     3.00       3.00       3.00          0          0
     3.00       3.00       3.00          0          0
     3.00       3.00       3.00          0          0
       0          0          0         2.00          0
       0          0          0            0       2.00

>>diag(c)'    %transpose of elements of diagonal of c
ans =

     3.00       3.00       3.00       2.00       2.00
```

Example:

```
>>%Conversion for Celsius and Fahrenheit temperatures
>>C=[ 10 : 100]' ;  %Create a column vector
>>F=(9/5)*C + 32; % Convert to Fahrenheit
>>Temperature_Table= [ C  F ]  % make a two column matrix.
```

2-Matrix and Array operations

Likewise when doing operations with matrices (arrays) , always be concerned by their sizes. As you know from basic linear algebra:
A+C and A-C is valid if both matrices A and C are of the same size.
A*B if A's number of columns equal B's number of rows.
A/B is the same as $A.B^{-1}$ for same size matrices. Here **A^{-1} is the inverse matrix of A** and the matrix A has to be square. It is also referred to as left division.
By contrast, the right division A\B means really "B divided by A".
B^2 is valid only if matrix B is square. Here the caret symbol ^ means exponentiation.
det(A) gives the determinant of the square matrix A.
inv(A) gives the inverse of the square matrix A. A must not be singular (determinant is not zero).
rank(A) gives the rank or number of linearly independent rows and columns of matrix A.

2.1 Solving equation Ax=b

Concerning divison of matrices, Matlab allows also the right division (\) besides what we call normal or left division(/). When solving the equation Ax=b in matrix form for x, we

use the command **x=A\b** . It is the same as **x=inv(A)*B** but faster. Inv(A) is the inverse of matrix A. This operation is often used to solve equations resulting from linear systems.

```
>>A=[ 2 , -1; 3 , 8] ;
>>A =

        2.00        -1.00
        3.00         8.00
>>  b=[-1  4]';
>>b =

       -1.00
        4.00
>>x=A\b;

>>x =

       -0.21
        0.58
```

2.2 Array Operations with the dot operator

Element by element multiplication, division, and exponentiation between two matrices or vectors of the same size are done by preceding the operator with a period.

.* element by element multiplication
./ element by element left division
.^ element by element exponentiation
.\ element by element right division

```
>>x=[ 1 , 3 , 4; -1 , 0, 9; 4 , 6 , 8]



x =

        1.00         3.00         4.00
       -1.00            0         9.00
        4.00         6.00         8.00


>>y=[-1 , 3 , -4; 4 , 6 , 9 ; 9, 12, 3]

y =

       -1.00         3.00        -4.00
        4.00         6.00         9.00
        9.00        12.00         3.00
>>c=x.*y     ; matrices of same size 3 by 3
```

```
c =

    -1.00     9.00   -16.00
    -4.00        0    81.00
    36.00    72.00    24.00
>>z=x.^2
z =

     1.00     9.00    16.00
     1.00        0    81.00
    16.00    36.00    64.00
```

```
>>% create a vector t with 10 elements
>>t=1:10;
>>% compute t=sint(t) , z= sin(t^3)/t^2
>> z1=t.*sin(t);
>>z2=sin(t.^3)./(t.^2)
```

Understand the difference between z^2 and z.^2. The operation z^2 is basically z*z but z.^2 consists to compute all elements $(z_{ij})^2$

```
>>z^2
ans =

   266.00   585.00  1769.00
  1297.00  2925.00  5200.00
  1076.00  2448.00  7268.00
```

But

```
>>z.^2
ans =

     1.00    81.00   256.00
     1.00        0  6561.00
   256.00  1296.00  4096.00
```

2.4 Eigenvalues and eigenvectors

A common problem in engineering applications consists to the following:
Given the matrix A, an n × n matrix , then the n numbers λ that satisfy

$$Ax = \lambda x$$

are the eigenvalues of A. Matlab find these eignevalues with the command

```
eig(A)
```

Question: find the eigenvalues for the above z matrix.

2.5 Polynomial product or convolution of polynomials

Consider the polynomials

$a(s)= 2s^4 +10s^3 + 2s +20$

$b(s)= s^2 + 3s +3$

To obtain the product of both polynomials , we take the convolution of the coefficients as follows:

```
>> a=[ 2  10  0  2  20];  % notice the coefficient 0 for the missing term s^2
>>b=[1  3  3];
>>%the product of both polynomials is
>>d=conv(a,b)
>> d =

    2    16    36    32    26    66    60
```

The result is the Matlab representation of

$d(s)= 2 s^6 + 16 s^5 + 36 s^4 + 32 s^3 +26 s^2 + 66 s +60$

The **conv** command can be applied to only 2 polynomials at a time; so if you have three polynomials to multiply, you have to proceed as follows:

Given a third polynomial

```
>>c= [  12  3  45  0  1];
```

Then the product p(s) of the polynomials a(s) , b(s), c(s) is

```
>> p(s)= conv(d, [12 3 45 0 1])
```

Next to evaluate a polynomial like a(s) at a specific s=2, we use the command **polyval**

```
polyval( a,2)
```

In order to divide polynomials, we use the command **deconv.** For example to divide the polynomials a(s) by b(s), we do

```
[q,r] = deconv (a,b)  % q is the quotient and r the remainder
```

2.6 Matrix exponential

The command expm(A) gives the matrix exponential of an n×n matrix A. That is

$e^A = I + A + A^2/2! + A^3/3! + \ldots$

```
>>A=[ 1   0  10;  2  7  -8 ; -2  26  0];
>>expm(A)
```

3-Basic Plotting

There main plotting commands are : figure, plot, zoom . The main image display commands: image, imagesc, colormap .

We will focus at this time on 2D plots and ,later on, revisit this topic for more advanced futures that include figure creation, plotting and image display.

3.1 Plotting commands: the most important commands for basic 2-dimensional plotting are illustrated in the following examples:

```
>>x=0:0.1:10;        % x ranges from 0 to 10 in steps of 0.1
>>y=sin(x);          % y contains the sin of each value of x
>>figure             %  creates an empty figure window
>>plot(y)            %  plots all of the y values
>>plot(x,y)          % plots y against x (replacing the old plot on the same
                     %figure, now showing the units of x on the x-axis)
>> close             % closes the figure window
>> disp              % diplay command
```

A simple example:

```
>>x=linspace(0, 2*pi ,100);
>>plot(x, sin(x))                  % or   plot (x, sin(x), '*')
>>xlabel('x'),ylabel('sin(x)', 'fontsize', 12)
>>title('my plot)          % or   legend(('my plot')
```

3.2 Zooming: The command "zoom [on/off]" command is useful for zooming of a plot with the mousse. Type "zoom" or "zoom on" , drag cursor of the mousse on a plot and click the left button of the mousse. Likewise, you can return to the original state by double clicking the left or right button of mousse. The off state turns off this feature.

3.3 Using Subplot to Layout Multiple Graphs:

A few plots can be placed side by side (not overlay) by using the SUBPLOT command. The format is

subplot (m, n, p)

It divides the graphic window into m by n sub-windows and puts the plot generated by the next plotting command into the p_{th} sub-window where the sub-windows are counted row-wise.

Example:

```
>>subplot(2,2,3), plot (x,y)    % divides the graphics window into 4 sub-windows and plot y
```

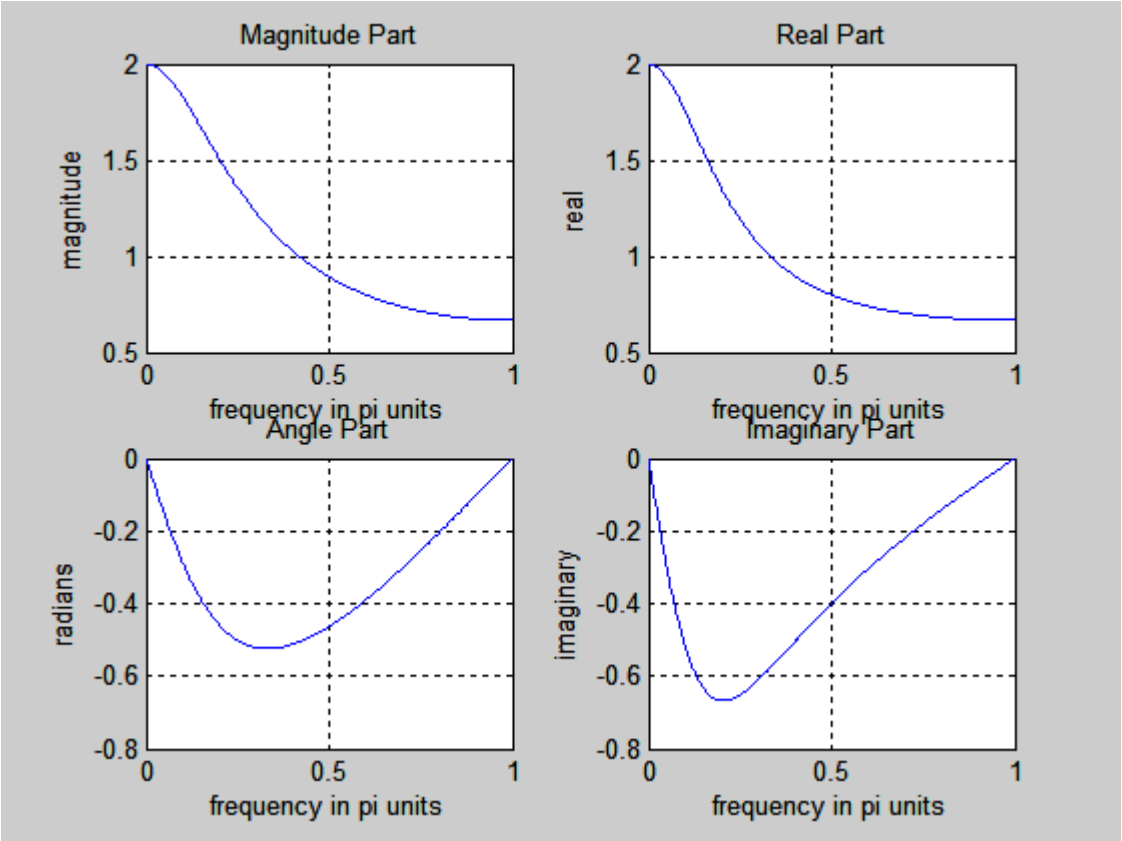
>> % versus x in the 3rd sub-window.

Example: Consider the function $x(e^{jw}) = (e^{jw})/(e^{jw}-0.5)$
Evaluate at 501 equispaced points between [0,pi] and plot Magnitude, Angle, Real and Imaginary Parts on same window.

Solution:

```
>>% Layout of multiple graphs
>>w=[0:1:500]*pi/500;  % [0,pi] axis divided into 501 points
>>x=exp(j*w)./(exp(j*w)-0.5)
>>magX=abs(x); angX=angle(x);
>>realX=real(x); imagX=imag(x);
>>subplot(2,2,1); plot (w/pi,magX);grid   %frequency in pi units
>>xlabel('frequency in pi units'); ylabel('magnitude');
>>title('Magnitude Part')
>>subplot(2,2,3); plot (w/pi,angX);grid   %frequency in pi units
>>xlabel('frequency in pi units'); ylabel('radians');
>>title('Angle Part')
>>subplot(2,2,2); plot (w/pi,realX);grid   %frequency in pi units
>>xlabel('frequency in pi units'); ylabel('real');
>>title('Real Part')
>>subplot(2,2,4); plot (w/pi,imagX);grid   %frequency in pi units
>>xlabel('frequency in pi units'); ylabel('imaginary');
>>title('Imaginary Part')
```

This results in the following graphs.



Practical note: The graphics can be copied and pasted on a MS Word page. To do so, in the graphics window, select EDIT and then "copy figure" and then paste to a word document.

3.4 Formatted output

Use of the **fprintf** function gives more control than the **disp** function when we want to display the output. It is the same format as the one used by C language to print text and matrix values. The format is

fprintf(fid, format, matrices)

For format, the specifiers are %e, %f and %g, respectively exponential form, fixed point or decimal form, and either %e or %f (depending on which one is shorter).

The format string usually ends with \n , which means start new line, to show the end of the location of the printed line, the rest is printed on the next line; **fid** is an integer file opener that works with **open** command (can be of value is 1 for the screen or 2 in case of error).

Example:

>>fprint(**'** The interest is **%f** dollars. **\n' ,** interest)

>>% if the interest is 30, then the print out will list:

The interest is 30 dollars.

>>fprint(**'**The price of a shirt in a luxury store **\n** is %f dollars **\n'**, 200)

the print out will reveal:

The price of shirt in a luxury store

is 200 dollars

let’s recapitulate with an example on temperature conversion

```
>>F= -40:5:100;
>>C=(F-32)*(5/9); % convert to temperature Celsius
>>T=[F  ;  C]'
>>fid=fopen('temperature.table', 'w')  %open existing file or open new file temperature.table
>>fprintf(1, ' Temperature    Table\n');  % fid value is 1 for screen.
>>fprintf(1,'-----------------------\n');
>>fprintf(1, 'Fahrenheit     Celsius \n');
>>fprintf( 1, '%4i          %8.2f    \n', T)    % i: integer format and  f: decimal format
>>fclose(fid)
```

3.5 Capturing output

The session in the workspace area can be captured and saved to a report (MS Word doc for example). To do so, use the **diary** command.

To start the recording session, assuming the name of your file is **report1,**

```
diary report1      % begin session
diary off          % end session
```

That’s all folks for this tutorial!