MATLAB, QUICK START(Not yet complete but give it a try)

Mail comments to:joby@dsplab.ece.iisc.ernet.in

Here I am trying to put in things I have learned while using and helping others use matlab. Many of the things are which are not easily found in other references. I will try to be as terse as matlab is. Best way to learn matlab is to do a thorough dissection of a matlab .m file from their own toolboxes. But it is some times laborious. Here I have done that and is putting in the results to be read. As far as possible you will be instructed to use what ever demos and helps that comes along with the matlab.

NOTATION for this note

Green --> Section headings
Brown --> Ordinary blah blah blah talking to you
Red --> Things to be noted among the blah blah blah...
Black --> What ever to be typed in at the MATLAB prompt >> by you (There will be the >>prompt also typed in at the beginning)
Gray --> What ever matlab outputs.

How to use these notes?

The notes will be in small small segments with out much coherence. So you can
a) Look up the list of contents and go to the portion which you think might help you(not ready yet so read it in full) or
b) Read it in full.

Beginning(May be silly for you but may be not for some)

If you are on
a) Windows --> Look for matlab icon , double click, and you see the new window popping up with >> prompt.
b) Linux,Unix --> Type matlab at command prompt, you see the MATLAB logo and >> prompt coming in the same window.

That done let us make it clear that you have already learned the most difficult thing in matlab. Rest of it you **know** since you know **basic operations in matrices(consider it as array of numbers)** and **the mathematical equations in which ever field you work in**.

Now the rest of it.

IMPORTANT

Try this at the matlab prompt and do what ever matlab instructs you to do or what ever you please.

```
>>intro
```

At the end of this introduction by matlab, you must be able to

1) Create some matrices like

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

2) Use ijth elements of the matrices by A(i,j), Multiply(A*B), Add(A+B), Subtract(A-B), Transpose(A'), Rise matrix to power(A^2), Raise elements to some power(A.^B), Multiply element by

element(A.*B), plot some figures(plot(A(1,:))), see the variables at the command prompt(whos) and so on.. If you have not done this go back and type in intro at the prompt and learn it.

You must also have noticed that putting ; at the end of matlab command will prevent output from being printed on the screen.

Things to be noted 1(VERY USEFUL THINGS TO BE NOTED)

Check out each of them as you go along reading.

1) In the matlab window If you press UP ARROW previous commands that you executed in will appear at the matlab prompt.

If you type some letters and press UP ARROW then previous commands starting with those letters will appear at the matlab command prompt.

2) If you type whos you will see all the variables present in the workspace and their details like size and dimension. Try

```
>>whos
```

3) All the variables you used in a session are available there for use till you clear them. If you use clear all, then all the variables in the workspace is cleared. If you use clear A B, only A and B are cleared. Try these..

```
>>A=[1 2  3  0.1]; B=[ -1 10^(-5)];
>>whos
  Name      Size         Bytes  Class

  A         1x4             32  double array
  B         1x2             16  double array

Grand total is 6 elements using 48 bytes

>>clear A
>>whos
  Name      Size         Bytes  Class

  B         1x2             16  double array

Grand total is 2 elements using 16 bytes
```

4) ls or dir will list files in the current directory where matlab is operating.

5) cd can be used to change directory just as in any shell or DOS prompt.

6) ! some _command will cause some_command to be passed on to the shell or DOS for execution.

IMPORTANT in WINDOWS : Create a directory now in your name using >>!mkdir and change to it using >>cd . This is done to prevent you from creating files in the matlab directory where you would in all probability be when you start of with matlab in WINDOWS.

7) Putting ; at the end of a command will prevent the output being written to the screen.

8) Putting % before anything will comment anything on that line coming after it. Useful for commenting when writing a .m file.

9) Three dots like **...** and you can continue your command on the next line. Good for long equations.

10) Complex numbers can be represented by using i or j for (-1)^0.5 . See these..

```
>> A=[1  -1+j*3   0]

A =

   1.0000          -1.0000 + 3.0000i        0
```
Matlab will take care of all the operations when complex numbers are present in your arrays, you don't have to bother about real and imaginary parts separately.
Matlab computes with full 15digit precision all the time but will only display in the above format by default. If you need it to display in other formats you can use
command calld format.Matlab will take care of all the operations when complex numbers are present in your arrays, you don't have to bother about real and imaginary parts separately.
```
>> format long; A

A =

  1.00000000000000                  -1.00000000000000 + 3.00000000000000i                0
```
now back to ususal format...
```
>> format
```
11)To save the values of the variables in the current matlab workspace you can use save, and load can be used to retrieve it. Try these..
```
>> clear all; A=[1 -1+j*3   0]; save A A                % Clearing the workspace and creating variable A
>> whos                                                      % Checking the variable A
  Name      Size         Bytes  Class

  A         1x3            48  double array (complex)
  ans       1x135          270  char array

Grand total is 138 elements using 318 bytes
>> ls                                                          %Checking if the file A.mat has been created.
  A.mat
   jfdj
   fkdj
     :
```
so on the filenames in the present directory.
```
>> clear all; whos                                          %  Clearing the workspace and checking if all is cleared
>> load A; whos                                             %  Loading the previously saved variable A and verifying using whos.
  Name      Size         Bytes  Class

  A         1x3            48  double array (complex)
  ans       1x135          270  char array
```
12) If the output is long then it will fly past on your screen. To prevent this execute more on at the prompt before trying the command. ENTER will go on giving the output in this mode.
```
>> more on
```
To turn off this option say when you want the screen to fly use >> more off
SKIP THE SECTIONS BELOW ON STRUCTURE AND CELL ARRAYS TILL YOU ARE WELL OF IN OTHERS.

Two other data types available are a)structures and b)cellarray. These are available only from version 5.0 on wards. Structures can contain a variety of data types like arrays, strings, other structures ....etc. You can form an array of structures, but then the structures forming the array should have identical data fields meaning one type, but size of each can be anything.

```
 >> mystru.a=A;                          %Here  mystru  is the structure name  and  a  is a field in it.
>>mystru.bore='A sentence';      %Here another field called  bore  is added to the same structure.
>>mystru.a(1,:)
     ans =

         1.0000   -2.0000    0.2000    0.5012
>>mystru.bore(1:4)
    ans =

        A se
>>packymyst=[mystru   mystru];    %Illustrates formation of array of structures called packymyst.
>>packymyst(2).bore(2:4)
    ans =

        se
```

Cell arrays can pack any type of data arrays and its elements can be addressed as elements in an array is addressed, the only difference being that curly brackets '{ }' are used instead of '[ ]'. Hence more versatile than structures.

```
>> G{1,2}=A*B; G{2,2}=mystru; G{1,1}=packymyst;      %Assigning data to a cellarray called G.
Note the variety of data
>> G{1,1}(2).bore(:)'
     ans =

        A sentence
```

==========

How to use matlab help is the next thing you should learn. Try this at your prompt and ENTER and watch the output.

```
>>help

  HELP topics:

    matlab/general       -  General purpose commands.
    matlab/ops           -  Operators and special characters.
    matlab/lang          -  Programming language constructs.
    matlab/elmat         -  Elementary matrices and matrix manipulation.
       :
       ..
```

so on. Here general, ops, elmat etc.. are matlab tool boxes. Now try this for elmat which is one of the toolboxes.

```
>>help elmat

  Elementary matrices and matrix manipulation.
```

Elementary matrices.
  zeros   - Zeros array.
  ones   - Ones array.
  eye   - Identity matrix.
  repmat - Replicate and tile array.
          :

.. so on. These , zeros, ones, eye, etc .. are the functions in the toolbox elmat. Now try for help on ones..

>>help ones

ONES   Ones array.
  ONES(N) is an N-by-N matrix of ones.
  ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones.
  ONES(M,N,P,...) or ONES([M N P ...]) is an M-by-N-by-P-by-...
  array of ones.
  ONES(SIZE(A)) is the same size as A and all ones.

  See also ZEROS.

Let us analyse this output by matlab.
Line 1) ONES Ones array.--> Describes the functions purpose.
Line 2)ONES(N) is an N-by-N matrix of ones.--> Describes one way of using the function. Try

>>ones(2)

NOTICE THAT LOWER CASE IS USED AT THE PROMPT THOUGH THE MATLAB GIVES UPPER CASE IN THE HELP.
Line 2,3,4..) These are other ways of using the function ones(). The ,... after the arguments in th function indicate that any number of further things can be put in there
Line last) See also ZEROS. -->This gives you names of related functions and is VERY USEFUL.

THUS STARTING FROM help AND THEN SUBSEQUENTLY CHOOSING THE RIGHT TOOLBOX (help toolbox_of_my_choice) FROM ITS OUTPUT, AND THEN THE RIGHT FUNCTION FROM THAT TOOLBOX (help function_of_mychoice) AND READING THE HELP FOR THAT FUNCTION YOU CAN PROCEED TO BE A MASTER OF MATLAB .. Do I see a doubt on your face. .. but yes... I am serious :).

==========

.m file (WRITING SCRIPT, OR SAVING YOUR WORK)

So far you have been working at the prompt. Suppose you want to write a sequence of matlab commands and reuse it some time later. Then do the following.

1) Open a file with .m extension, say for example my_firstmfile.m
  a)In Windows: Use notepad, edit,...or some other editor you please.(You can do this from matlab window via File, Open, ...etc.
  b)In Unix: Use editor of your choice and do the same.

2) Write in the sequence of matlab commands you want to be executed and save the file. You can use programming structures like, for .. end, if.. then..elseif.. end, while .. end; etc in your script. Use >>help for, >> help while etc ... to find the exact usage. This will be peanuts if you are familiar with any of the programming languages, but smartness is in NOT USING IT. As you progress through this you will learn ways to do it.

3) At the matlab prompt change to the directory in which you saved the .m file.

```
>>cd directory_in_which_saved
```

4) Execute the matlab file by typing in the **filename without .m extension.** in our case it is..

```
>>my_firstmfile
```

All results will be there as if you had executed each line you wrote in the file as if it had been executed at the matlab command prompt >>

THIS COMPLETES THE BASIC THINGS NEEDED FOR YOU TO GO ABOUT DOING PROGRAMMING IN MATLAB, HOWEVER IT IS GOING TO BE INEFFICIENT IF YOU RESTRICT TO THIS. TO BE MORE EFFICIENT GO THROUGH THINGS BELLOW.

==========

Things to be noted 2 (ACCESSING AND SHAPING MATRIX ELEMENTS AS YOU PLEASE)

Let us start by defining an array and a vector which will serve as our representatives for all the requirements in this note from now on. Enter these at the prompt or in a .m file and be ready. For the present don't use **;** at the end of the commands so that you can see the results.

```
>>A=[1 -2  0.2 10^(-0.3);  0 100  -exp(5)  1/4]
>>B=[ -1 0 2 -0.1]'
```

---

1) To use any full column of A, say for example the second column,

```
>>C=A(:, 2)

C =

   -2
  100
```

Similarly for using a particular row say first row >>C=A(1,:);. Try the following too...

```
>>C=A(1,:)*B
>>C=A(1,:)'*B'
>>A(:,2:end)
```

... etc

In the last example notice the use of end. If you don't know the size of the matrix you can use this end to denote the last element, be it in column or row. (I think this use of end is there only since matlab

version 5)

2) To make a vector out of a matrix by stacking the columns.Try this...

```
>>C=A(:)
  ans =

       1.0000
            0
      -2.0000
     100.0000
       0.2000
 -148.4132
       0.5012
       0.2500
```

---

3) To form a new matrix by pieces from other matrices. Try the following examples and see how C is formed from pieces of A and B..

```
>>C=[A(1:2,1:3)   B(2:3)' ]
C =

    1.0000   -2.0000    0.2000    0
    0        100.0000  -148.4132  2.0000
```

---

4) If you want to chose alternate elements from A and form a matrix try this. Check which element from where came to where

```
>>C=[B(2:3); A(2,1:2:end)  ]
C =

       0       2.0000
       0  -148.4132
```

This need not be alternate elements it can be every 3rd or nth element, by just changing 2 between the **:**s to 3 or any n as you want. Only make sure you have the correct multiple number of elements in the addressed direction, row or column, else matlab might crib (It has reason to, don't it?).

---

5) Reshaping the matrices can be done using function reshape. It takes any matrix, stacks the columns one below the other and then create a matrix of the specified dimension by making columns out of elements of the previously created vector in same order as they appear in the vector. Try this..

```
>>C=reshape(A,4,2)                          %Creates 4X2 matrix out of columns of A
ans =

     1.0000       0.2000
     0         -148.4132
    -2.0000       0.5012
```

```
 100.0000         0.2500
```

6)Suppose you want to access elements in a vector randomly from here and there and form a new vector. First you have to generate a vector which specify the elements you want to access in the order you desire it to be. If you want to form vector of B in the order [2 1 4 1], Then the usage is as bellow,

```
>>C=B([3 1 4 3])

C =

    2.0000
   -1.0000
   -0.1000
    2.0000
```

The vector of ordering which was explicitly specified above might be output of some other function.

==========

Efficient use of conditional checks on elements of matrices. (READ THIS IF EVER YOU USE <,>,==,~=,not,..etc)

1) Any of the operators above can be applied to a matrix as a whole. Watch the following....

```
>>C=A>0                                          %To get check which  elements of A are greater than
0.

C =

     1     0     1     1
     0     1     0     1
```

You have got ones where ever the condition is satisfied. NEVER USE A LOOP IN MATLAB WHEN YOU CAN AVOID IT. So try these.

```
>>C=(A>=0).*A                                    %To make all elements less than zero  to be 0

C =

    1.0000         0         0.2000    0.5012
         0    100.0000         0         0.2500
```

---

2) To choose elements in an array satisfying the required conditions. See the following example.

```
>>C=A(A~=0)                                      %To choose elements in A not equal to zero, and form a
vector of that.
C =

    1.0000
   -2.0000
  100.0000
    0.2000
 -148.4132
```

```
    0.5012
    0.2500
```

3) At the beginning of this section it was told that NEVER USE A LOOP IN MATLAB WHEN YOU CAN AVOID IT. But if your computer has less memory, use of such large array operations might give OUT OF MEMORY problems. Then you either have the option of increasing the memory, swap or USE THE LOOPS AND SUCH PROGRAMMING CONSTRUCTS TO REDUCE MEMORY USE BY MATLAB. (Ref to section called COMPILING)

==========

Familiarize with the kind of functions.

1) Matlab provides kakkathollairam (translated in mallu kids language to, crow's 900), implying large number, of functions. Each function takes some values of inputs and, outputs, none, one or more things. Consider this (and try it out)

```
>>C=sin(A)

C =

    0.8415   -0.9093    0.1987    0.4805
    0         -0.5064    0.6877    0.2474
```

Here the function sin(), returned the sine of all elements in matrix A. Thus when all functions which operate on a scalar, in day to day maths, is called in matlab, and a matrix is supplied to it, will operate on each of the matrix element. See help sin.

Now watch this,

```
>>[C  X]=sort(A')

C =

   -2.0000 -148.4132
    0.2000         0
    0.5012         0.2500
    1.0000  100.0000


X =

     2     3
     3     1
     4     4
     1     2
```

(If the output on the screen is flying of fast, you can put a break to it by executing **>>more on** before you execute the command which has too much to speak)

In the above example the function sort() sorted each column of A' in the ascending order and gave the sorted value in C and their respective indices in X. Every mathematical function which normally operates on a vector if invoked in matlab, and a matrix is supplied to it, will operate on each column of

the input matrix and output is placed in columns of the returned matrix. Try help sort.

3) Below is a list of functions to give you a FLAVOR of functions available in matlab(including toolboxes). Try help of the ones you find interesting and not recognizable at first sight, notice the related function given at the end of each help.

Matrix : reshape(), min(), max(), median(), mean(), var(), sum(), prod(), cumsum(), imag (),ℑ ang(), ...etc

Generators: cos(), atan(), chirp(), rectpuls(), sawtooth(), bessel(), sinc(), exp(), erfc(), gamma(), legendre(), ...etc.

Transforms: fft(), fft2(), fftshift(), dct(), dct2(), psd(), spectrum(), specgram(), xcorr(), conv () .....etc

Plotting: figure(), subplot(), holdon, plot(), stem(), mesh(), plot3(), title(), xlabel(), legend (), view(), contour(), print, ...etc

Keyboard Interactions: input(), keyboard, return(), dbstop(), dbquit () .....etc

File operations: save, load, fopen(), fread(), fwrite(), fscan(), fgets(), wavread(), dlmread() ....etc

Random numbers: rand(), rand(), sprand (), ...etc

Polynomial Ops: poly(), roots(), residue(), conv(), zplane(), xcorr() , filter () ....etc

Matrix Factorizations: svd(), qr(), lu(), null(), rank (), ...etc

Set operations: union(), intersect(), setdiff(), setxor(), ismember () .....etc

These are but samples of what is in there, and you are encouraged to explore more and suggest to me something which should be essentially added to the above list. As you can see you only need imagination to use matlab function you never heard of before. You feel like doing something check out if a function for it already exist in matlab. Good guesses as to their matlab function names often work. Suppose you guessed the name as guessedname.m. Then try >>help guessedname . If that fails then try >>lookfor guessdname.

==========

Writing your own functions, and associated things

See help function for knowing the syntax. Also scan through the related function given at the bottom of the help (Better make this a habit till it becomes boring :)). Here are the steps. Let the function you want to write be named premier().

a)Open a new file named premier.m and enter the following lines and save.

```
function [X, C, Y]=premier(Z,B,varargin)
  %  function [X, C, Y]=premier(A,B,varargin)
  %     X,C,Y       -> Are the variables passed back.
  %     Z and B  -> Are the variable which take values necessarily passed to the function
  %                          without which the function when called will give syntax error.
  %     varargin  -> Will have the array of values optional variables you pass in to the
  %                           function.
   Y=size(varargin);
   X=sin(Z(1:end-1));
   C=varargin{1,2}(1,2);                          %(1,2) element of the second variable in varargin.
```

b)Notice the following things about this small programme.

In the first line

keyword&keywords; function -> Saying this ia a function

passed back variables within [..]= -> Saying these are to be returned back

passed into variables in premier(...) -> This is the function name and variables passed to it assume the name given within the brackets within the funtion.

&functionbsp; keyword among variables varargin -> Not necessary, but if present allows you to pass more unspecified number of values to the function .

It is your duty to write the program to use this efficiently.

Next few lines of code you see starting with % are help comments. Try >>help premier. You will see these lines without % sign being printed out by matlab. This is how the matlab help works.

The lines following theses are matlab commands operating on the values passed in and assigning values to the variables to be passed back. Have a close look at the commands and try to figure out what it will do. See below to see if you have figured out correctly.

c) Now execute the function by passing in some values;

```
>>[G H SIZE]=premier(B, A, A,[1 2; 2 4], B')
G =

   -0.8415
         0
    0.9093
   -0.0998


H =

     0


SIZE =
```

1 3

Looking at the result you can see that

1)External variable A and B were assigned to internal variables B and A respectively in the order it was input. Remaining variables passed in where assigned to varargin, whose size as you can see from the result SIZE is 1X3 variables. G was assigned the value X while being passed back from the function. H is the (1,2) element of the 2nd variable in varargin as was assigned to C within the function.

==========

Forming matrices as you want and thus reducing conditionals and loops. (READ THIS TO AVOID for LOOPS IF YOU HAVE REPEATED OPERATIONS WITH VECTORS)

We will approach this section as a case study of a problem.

Consider the following problem: You have a matrix, and you want to create a new matrix with all the rows in the given matrix which is same as a given vector eliminated. In conventional programming you would run a loop over the rows of the matrix and create the new matrix by adding element by element
from the original matrix which satisfies the condition..... Here goes the matlab code...

```
%function [y]=delete_rows(x,a)
%
%  x -> The matrix whos row is to be removed
%  a -> The row vector which is to be removed.
%  y -> The cleaned output matrix.
%Retruns x y with all rows of x which are same as a are removed.
%
function [y]=delete_rows(x,a)
y=x(setdiff((1:size(x,1)),find(sum((x==(a'*ones(1,size(x,1)))')')==size(x,2))),(1:size(x,2)));
```

Just a single line and no loops... Check it out and figure it out as an exercise...

It is not always true that removing loops and making matrices speeds up things. As matrix sizes increase CPU will have to start using the swap memeory space on the hard disk instead of the RAM. These disk accesses are very slow and will become the bottle neck for speed. So there is a trade off between the speed gain from rediucing 'for loops' and slownes from the increased 'matrix size', especially if you are operaing with less RAM.

Keep clearing the variables which are not required for the same reason as above.

==========