MATLAB: Quick Start
Econ 837

1 Introduction

MATLAB is a commercial "Matrix Laboratory" package which operates as an interactive programming environment. It is a programming language and a computing environment that uses matrices as one of its basic data types. It is a commercial product developed and distributed by MathWorks. Because it is a high level language for numerical analysis, numerical code can be written very compactly.

- **Starting MATLAB**

  Five windows appear (more can be added: see Desktop in the main menu):

  1. Current Folder (CF)
  2. Command Window (CW)
  3. Workspace (WS)
  4. Command History (CH)
  5. Editor (E)

  Each window can be opened/closed/minimized/maximized/docked/undocked (see upper right corners).

- **Setting the Directory**

  MATLAB with look for files in the CF or the path you specify in your script. Save all your files in the desired directory, and then set up the CF in MATLAB as this above directory, in this way MATLAB can access it directly without having to specify its whole path.

  This can be done intwo ways. By selecting a folder on the CF windows or by adding paths writing for example:

  Add c:/MATLAB/myfiles to the top of the search path:

  addpath('c:/MATLAB/myfiles')

  Add c:/MATLAB/myfiles to the end of the search path:

  addpath ('c:/MATLAB/myfiles -end')

- **Loading Excel data into MATLAB**

  This can be done in at least two ways:

  1. *Manually*
     - *Importing the data.* Copy the file into your CF. Double click filename.xlsx (takes a few seconds), click on "Next", then "Create vectors from each column using "column names" and finally "Finish". The variables will appear in the WS.

       Remark: pay attention to how the excel file is prepared before importing the data.

  - *Saving the WS data.* Enter *save filename* in the CW. The file *filename.mat* appears now in the CF. It can be loaded into the WS at any time by double clicking in the CF, or typing *load filename* in the CW.

2. *Programming*

  - *Importing the data.* Save the original data file in the CF. To import a data file, the easiest way is to get rid of the delimiters and headers and convert it into a ".txt" file. Then type in the CW "load file.txt-ascii" to load the file.

  Remark 1: All the columns have to have the same length, as MATLAB will convert it into a matrix.

  Remark 2: You can import many other kinds of files. Check the help: User's guide/Data import and Export.

To see the list of objects that have been created in your workspace (as well as additional info such as sizes, values, format...), go to the workspace (access it from top panel under "window").

- **Clearing the WS.** Enter *clear* in the CW.

- **Working interactively.** Commands can be entered in the CW. Use ↑ and ↓ to browse through earlier commands. At any time, existing variables are visible in the WS. Double-clicking a variable will open it in a spreadsheet.

- **Script.** A script is just a list of commands to be run in some order (similar to a do file in STATA). Placing these commands in a file that ends in .m allows you to "run" the script by typing its name at the command line. You type the name of the script without the .m at the end.

- **Working with functions (.m files).**

  For most purposes we use functions, saved in .m files. They are sequences of commands to be executed in that order. They are written using the editor (E). Saved functions appear in the CF. Functions can be executed in different ways:

  - from the CF (select the file, right-click>run)
  - as a command in the CW (enter the function's name)
  - from within the editor (click the green button in E)

  A function is capable of taking particular variables (called arguments) and doing something specific to "return" some particular type of result.

  A function needs to start with the line:

  function return-values = functionname(arguments)

  so that MATLAB will recognize it as a function. It is advisable that each function has its own file, and the file needs to have the same name as the function.

For example, if the first line of the function is:

$function\ answer = myfun(arg1, arg2)$

$answer = (arg1 + arg2)./arg1$

then the file must be named $myfun.m$.

One .m file can contain multiple functions. The great advantage of .m files is that they can call other functions (also from other .m files), allowing for transparency by giving coherent tasks to each function. Functions can have input and output arguments (though neither is necessary). It is extremely important to choose transparent names for functions and variables, that is choose as close as possible to the notation used in you textbooks or own writing so that you can easily read your code.

- **Debugging.**

  MATLAB has an extensive debugger that allows you to examine what is going on inside a function when you encounter problems with it. If you type "help debug" at the MATLAB prompt, it will list all of the debugging commands available. For example, breakpoints can be placed inside a function (to the left of any line) to cause execution to halt and allow for variables to be checked. Execution can be resumed from there in many different ways (i.e. buttons to the right of the green button).

  Debugging commands:

dbstop	Set breakpoint.
dbclear	Remove breakpoint.
dbcont	Resume execution.
dbdown	Change local workspace context.
dbstack	List who called whom.
dbstatus	List all breakpoints.
dbstep	Execute one or more lines.
dbtype	List M-file with line numbers.
dbup	Change local workspace context.
dbquit	Quit debug mode

- **Profiling.**

  MATLAB has a profiler (Tools>Open Profiler) to spot bottlenecks regarding computational times (and sometimes it even suggests improvements)

- **Interrupting execution.**

  Place the cursor in the CW and press CTRL+C.

2 Preliminary Comments

- MATLAB is case sensitive. This means that a and A are 2 different objects.

- If you want to add comments to your program files, use %. Comments are written using % in front of a line:

  Remark: a useful short cut to put/take out % on one or several lines is: $CTRL + R$ (to put %) and $CTRL + T$ (to take it out)

- If you want some information/help about a specific function or command, use help command-name directly in the command window (or go to "help").

- I suggest you write your program in the command editor (from the top panel, go to "file/new"). Then you can either execute it by entering its name in the command window (without the .mat extension, or select part of it and click F9 key to execute it).

- Whenever you assign a value in MATLAB, it will print out on the CW the entire value you have just assigned. If you put a semicolon at the end, it will not print it out which is much faster.

- In this document, I present some useful functions and examples. Many are available in the help section of MATLAB and more specifically the user's guide (which is very well-done!).

3 Basics

- **Create a matrix**

  The simplest way to create a matrix in MATLAB is to use the matrix constructor operator, [ ]. Create a row in the matrix by entering elements. Separate each element with a comma or space. To start a new row, terminate the current row with a semicolon. For example, to construct a 3 row, 5 column (or 3-by-5) matrix of numbers (note that all rows must have the same number of elements!):

  $A = \left[\begin{array}{ccccccccccccccccc} 12 & 62 & 93 & -8 & 22 & ; & 16 & 2 & 87 & 43 & 91 & ; & -4 & 17 & -72 & 95 & 6 \end{array}\right]$

  Defines:

  $$A = \begin{pmatrix} 12 & 62 & 93 & -8 & 22 \\ 16 & 2 & 87 & 43 & 91 \\ -4 & 17 & -72 & 95 & 6 \end{pmatrix}$$

  For example:

  $A = [12; 34; 56];\quad bb = eye(3);\quad cDc = zeros(3,1);\quad i = 4:7;\quad j = 0:0.5:2;$

  Defines the following elements:

  $$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \qquad bb = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad cDc = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

  $$i = \begin{pmatrix} 4 & 5 & 6 & 7 \end{pmatrix} \qquad j = \begin{pmatrix} 0 & 0.5 & 1 & 1.5 & 2 \end{pmatrix}$$

  MATLAB is case-sensitive. Higher-dimensional arrays are also possible.

- **Access to elements of a matrix.**

  To reference a particular element in a matrix, specify its row and column number using the following syntax, where A is the matrix variable. Always specify the row first and column second: $A(row, column)$

  For example:

  $A(3,2)$ is 17. $A(:,2) = \begin{pmatrix} 62 \\ 2 \\ 17 \end{pmatrix}$

  $B(2,1)$ is 3, $B(2,:)$ is $\begin{pmatrix} 3 & 4 \end{pmatrix}$ and $B(2:3,2) = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$.

- **Elementary matrix computations.**

$A + B$	Addition
$A * B$	Multiplication
$A\prime$	Transpose
$inv(A)$	Inverse
$a * A$	Scalar a times matrix A
A/a	Matrix A divided by scalar a:
$A. * B$	Direct multiplication (multiplies each A_{ij} with B_{ij})
$A./B$	Direct division (divides each A_{ij} with B_{ij})
$A.\hat{}k$	Raise to power k each element of matrix A
$diag(A)$	Vector of diagonal elements of A
$sqrt(A)$	Element-by-element square root
sum(A)	Sum by column the elements of A
$abs(A)$	Element-by-element absolute value
$mean(A, 1)$ or $mean(A)$	Mean by column (first dimension) of A
$std(A, 1)$ or $std(A)$	Standard deviation by column (first dimension) of A
$diag(v)$	Diagonal matrix with elements of vector v on the diagonal
$diag(A)$	Creates a column vector that contains the main diagonal of a matrix A
$diag(diag(A))$	Creates a diagonal matrix, with diagonal equals to the main diagonal of a matrix A

  Remark: $inv()$ function for a matrix can be time consuming. Instead you can use left division \ .
  For example:

  Instead of writing: $inv(A) * b$

  Write: $A\backslash b$

- **Some useful functions.**

  MATLAB has a lot of built-in functions. To use a function, just type functionname(arguments), with the appropriate name and arguments. For example:

  $s = sum(C)$

  sums all the columns in matrix A and returns a row vector of the sums.

Here are some of the most used functions in Econometrics:

ones	Creates a matrix or array of all ones
zeros	Creates a matrix or array of all zeros
eye	Create a matrix with ones on the diagonal and zeros elsewhere
diag	Create a diagonal matrix from a vector
rand	Create a matrix or array of uniformly distributed random numbers over (0,1)
randn	Create a matrix or array of normally distributed, $N(0,1)$, random numbers and arrays
disp	Display text or array
length	Returns the length of the longest dimension
size	Returns the length of each dimension
rank	Matrix rank
det	Determinant
trace	Sum of diagonal elements
sum	Sum of array elements
mean	Calculate the mean of
inv	Matrix inverse
chol	Cholesky factorization
lu	LU factorization
$normcdf(z)$	$\Phi(z)$, i.e. the CDF of $N(0,1)$
$cdf('t', z, p)$	CDF of $t(p)$ evaluated at z (similar for F and χ^2)

Remark: There are many other built-in functions. Check out MATLAB help for a comprehensive list of them.

- **Loops**

  Sometimes, you do need to use some kind of loop to do what you need, rather than just operating on an entire matrix or vector at once.

  - *While*

    The syntax for a while loop is

    ```
    while (some logical expression)
        do something;
        do something else;
    end
    ```

    To keep this from going on forever, you should probably be changing some variable in the logical expression within the body of the loop so that it eventually is not true.

  - *For*

    To execute statements a specified number of times, use the loop "for". Example: create a Hilbert matrix using nested for loops:

    ```
    k = 5;  hilbert = zeros(k, k);          % Preallocate matrix
    for m = 1 : k
        for n = 1 : k
            hilbert(m, n) = 1/(m + n − 1);
        end
    end
    ```

This creates:

$$hilbert = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix}$$

– *If*

The syntax for an if statement is

```
if (logical expression)
    matlab command
elseif (other logical expression)
    another matlab command
else
    a matlab command
end
```

Remark: You don't need an elseif or an else, but you do need an end.

Remark1: Loops can be nested.

Remark 2: Loops in general are quite time consuming, so try to reduce the number of loops you use. That is, try to work with matrix and vector operations, avoiding loops if possible.

For example:

```
n=10000;   K=5;   X=randn(n,K);          % produces a n x k matrix of draws from N(0,1)
%X'X by scalar operations:
tic
XX=zeros(K,K);
for i=1:K
   for j=1:K
       for s=1:n
          XX(i,j)=XX(i,j)+X(s,i)*X(s,j);
      end
   end
end
toc
% X'X by matrix operations:
tic
XX=X'*X;
toc
```

4 Plotting.

- The basic syntax to get a plot in matlab is: $plot(x_1, y_1)$

  Notice that the x values always come before the y values, x_1 and y_1 represent variables that are stored in your data.

- If you type a second plot command later, it will clear your first plot. If you type "*hold on*" it will hold the current plot so you can add plots on top of one another (until you reset it by typing "*hold off*".)

  You can plot multiple values with $plot(x_1, y_1, x_2, y_2)$ and you can specify the color and line type of a plot as something like $plot(x1, y1, 'w*')$ to get white *'s for each data point.

- To split your plot into a bunch of smaller plots, you can use the subplot command to split it up into rows and columns. $subplot(r, c, n)$ will split the plot window into r rows and c columns of plots and set the current plot to plot number n of those rows and columns. For example, $subplot(2, 1, 1)$ splits the plot window into two rows in a single column and prepares to plot in the top plot. Then your plot command will plot in the top plot. Then you could switch to the bottom plot with $subplot(2, 1, 2)$ and use another plot command to plot in the bottom plot.

  You can add titles, labels, and legends to plots:

  title('This is a Title')

  xlabel('My X axis')

  ylabel('My Y axis')

  legend('First Thing Plotted','Second Thing Plotted')

  Remark: legend creates a legend box (movable with the mouse) that automatically uses the right symbols and colors and sticks the descriptions in the legend command after them.

  To *remember:*

$plot(x, y)$	plot the couples (x_i, y_i) (*Note: x and y need to be of the same size*)
$hist(x, n)$	plot the histogram of a vector x using n bins
hold on/off	option to plot another curve on the same/different figure

5 Help and Other tools

- **Places to get help.**
  - Typing "help" at the MATLAB prompt gives you a list of all the possible directories MATLAB can find commands in (which also tells you its "search path", or a list of the directories it is looking in for commands.)
  - Typing "help directoryname" gives you a list of the commands in that directory and a short description of them.
  - Typing "help commandname" gives you help on a specific command.

  - Typing ”lookfor keyword” gives you a list of commands that use that keyword (e.g. ”lookfor integral” lists commands that deal with integrals). It’s pretty slow, choose the word wisely. You can use CTRL+C to stop searching when you think you’ve found what you need.

- **Some useful Tools.**
  - If you accidentally reassign a function name to a variable (i.e., you try saying sum = 3 and then you get errors when you try to use the sum function because it doesn’t know it’s a function anymore), you can restore it to its normal state using ”clear functionname”. You can also use clear to get rid of all variable values with ”clear all”.
  - *who:* will tell you all the variables you have currently defined.
  - *whos:* will tell you the variables, their sizes, and some other info.
  - *eps:* is a function that returns MATLAB’s smallest floating point number. This is useful if you have a vector that might contain zeros that is going to wind up in the denominator of something. If you add *eps* to the vector, you aren’t actually adding anything significant, but you won’t run into divide by zero problems anymore.
  - *format long* and *format short:* switch between the long and short display format of numbers. Either way MATLAB uses the same number of digits for its calculations, but normally (*format short*) it will only display the first four digits after the decimal point.
  - Typing *type functionname* for any function in MATLAB’s search path lets you see how that function is written.