

Introduction to MATLAB for Econ-606

Jinhui Bai

Georgetown University

First Version: January 20, 2004

Last Update: January 12, 2007

I. Introduction

MATLAB means “MATrix LABoratory.” The name reveals that it is a matrix programming language. A matrix language makes programming of a matrix operation similar to write down the counterparts on the paper, which partly explains its popularity. MATLAB is a high level computer language, which is easier to learn and use than the lower level languages like C or FORTRAN. But in some applications it is significantly slower than those lower level languages.

MATLAB typically refers to both the computer language (MATLAB) and a rich add-on toolbox written based on this language (MATLAB Toolbox). MATLAB supports, among others, Linux, Microsoft Windows, and Mac OS platforms. The most recent version of MATLAB as a language is 7.3, with the full package including tens of toolboxes called “Release R2006b.”

II. Start to use MATLAB

1. Get to know your friend

You can launch MATLAB in the same way as any other program in the Microsoft Windows. After you launch MATLAB, you will see a *Command Window*, where you can write simple commands and read the output from the command.

You may also see other windows besides the Command Window. For example, in *Launch Pad* you can see main products included in your copy of MATLAB. *Workspace* shows the basic properties of all variables generated by your commands, and you can double click each variable to see the values of those variables. *Command History* records all commands you have typed in the *Command Window*. *Current Directory* shows your working directory, which MATLAB will search once you start to execute a program. Therefore, you should put your own program in the *Current Directory*. You can change the current directory by clicking the “browse button” beside that.

To launch a window, just click the *View* menu and choose the one you want. Try to play around with those menus for a while. As a computer user, you will probably know how to do that even without learning.

2. Edit and run a program

In MATLAB, the file containing code is called an M-File. For example, the function to do least square regression is in the M-file “regress.m.” If you want to read the code inside, you can (a) type “type regress.m” in the Command Window; or (b) double click “regress.m” in the folder where it locates; or (c) type “edit regress.m”. For either (b) or (c), you are led to a code editor of MATLAB. There you can edit the code in your desired way.

To run an M-file, just type “*file name*” in the Command Window. For example, after you finish coding “shooting.m” to solve a second-order nonlinear difference equation using shooting algorithm, type “shooting” (without dot m) in the Command Window.

3. Get help on MATLAB

Usually the first thing to learn about a computer language is to know where to find help. There are at least three ways to find a help on MATLAB.

First, you can click menu “Help\MATLAB Help.” There you can find a very good introduction. For example, in “Getting Started” you can find an introduction for beginners; in “MATLAB Functions Listed Alphabetically” you can find the list of all functions.

The second way is to type “help *function name*” or “help *toolbox name*” in the Command Window. For example, if you want to know how to use the function “regress” in Statistics Toolbox to do OLS, just type “help regress”. Then you will see the introduction to the “regress” function. If you type “help stats”, you will find the whole list of functions in the Statistics Toolbox.

The third way to get help is to go to MATLAB user group website (<http://www.mathworks.com/matlabcentral/>). In the online forum there, a lot of friendly people are willing to answer your questions.

4. Set search path

If you want to use a toolbox from a third party (e.g., the Econometrics Toolbox written by James LeSage on the website www.spatial-econometrics.com), you need to set search path to make MATLAB treat it in the same way as its own toolbox. To do this, just click menu “File\Set Path\Add Folders (or “Add with Subfolders” if there are multiple folders there)\Save”. After this, you can call the functions in that toolbox in the same way as other toolboxes.

III. Programming: Simple matrix operation

Now it is time to learn to write your own code. Sometimes you need only to write a one line code to get what you want. You can do it right in the Command Window and click the Enter key. Those simple tasks typically involve simple matrix operation or a function call.

1. Creating a matrix

There are at least three ways to generate a matrix.

First, you can input a matrix manually. For example, type “a = [3, 4, 5; 2, 3, 4];” in the command window. This will generate a 2 by 3 matrix “a” in the workspace. We use “[]” to enclose the matrix elements; use comma (or alternatively space) to separate the elements within the same row; and use “;” to represent the end of a row. One thing to keep in mind is that MATLAB is case-sensitive. Therefore, “A” and “a” are two different matrices. Another thing to know is that a line without “;” will both run the code and show the result. As a practice, try to type “A = [1 2; 3 5]” with and without a semicolon at the end of this command. What have you seen?

Second, we can use existing functions to get a matrix. For example, “a = zeros (3, 4)” generates a 3-by-4 matrix with zero; “b = ones (2, 2)” generates a 2-by-2 matrix with each element equal to one; “c = rand (2, 4)” generates a 2-by-4 matrix with each element withdrawn from a uniform distribution between 0 and 1; “d = randn (2,4)” is similar to the command rand (2, 4), but with standard normal distribution generated instead.

Third, load matrix from an outside data set. In MATLAB, the data file is called “*.mat” file. If you want to load the data file “a.mat”, type “load a;” in the Command Window.

2. Some useful functions

You can manipulate the matrix you have created by calling an existing function. Here are some frequently used functions.

“size (A)” returns the size of a matrix.

“max (A)” returns a row vector containing maximum value of each column. “min (A)” returns the minimum value.

“plot (x, y)” gives a graph of y as a function of x. Disp (x) displays the value of the variable “x.”

As a reminder, you can always type “help *function name*” to find more information about a function.

3. Select a submatrix and merge matrices

Quite often we need to use only part of a matrix, i.e., a submatrix. If we want to select one element of a matrix “a”, just type “a (i, j)” where i and j are row and column index of that element. To get a submatrix with more than one element, we need to specify both the row index and column index of the submatrix. For example, “b = a([1 3], [2, 4])” picks up the row 1 and 3, column 2 and 4 of matrix “a” and put that in the matrix “b”. As a practice, create an arbitrary matrix “a” and get some elements out.

If we want to pick up some rows or columns consecutively, it is easier to use colon operator “:”. For example, “1:3;” means row vector (1 2 3), while “1:.5:2;” means row vector (1 1.5 2). You may have already inferred that the middle number means the size of increment, with default size as increment 1. Now can you tell the meaning of “b = a (1:3, 2:4)” or “b = a (1:3, :)”? Type those commands to find it out.

If you want to delete one row (or column) of a matrix, just type “a (i, :) = []” (or “a (:, j) = []”).

Sometimes we also need to merge available matrices. To concatenate a matrix, just treat each matrix as a number and do the concatenation in the same way as manually inputting a matrix. Try to type the following “A = ones (2,3); B = zeros (2,4); C = zeros (3,3); D = zeros (3,4); E = [A B; C D]” and read the output.

4. Matrix operators

4.1. Arithmetic operator

Here is a list of main arithmetic operators on matrices.

Table 1: Arithmetic Operators

Operator	+	-	*	/	\	'	^
Meaning	Addition	Subtraction	Multiplication	Division	Division	Transpose	Power

The division operators deserve some explanation. You can think of the division operator as a generalized inverse, which means that the matrix to be divided is not necessarily a square matrix. For example, $y = A \setminus b$ is the solution of $A * x = b$, while $x = b / A$ is the solution of $x * A = b$. As a practice, use division operator to solve for the OLS estimators. Which division operator should you use?

If you want the operation above to be entry-wise, you need only precede those operators by “.”. For example, “X .* Y;” is element-by-element multiplication of two matrices.

4.2. Relational operators

Table 2: Relational Operators

Operator	<	<=	>	>=	==	~=
Meaning	Less than	Less than or equal to	Greater than	Greater than or equal to	Equal to	Not equal to

Here the only tricky stuff is “= =.” Be careful about the difference between “=” and “= =.” The “A = B” checks whether two matrices are equal, while “A = B” gives value of B to A. As a practice, generate two matrices as in the previous part and use both “= =” and “=” to operate on two matrices. What do you see?

4.3. Logical operators

Table 3: Relational Operators

Operator	&		~
Meaning	and	or	not

IV. Programming: Write an M-file

Although you can do simple jobs just by typing commands in the Command Window, for complicated jobs or frequently repeated jobs, you’d better write a code and store it in an M-file. There are basically two kinds of M-file: a script and a function. As you might know, a function returns results for some given input variables (e.g., function zeros (m, n)). Script is just a patch of code which gives some results, but it does not accept input arguments.

In terms of grammar, the difference between function and script lies in the first line. The first line of a function starts with “function *output* = *name* (*input*);”. Try to type “edit fzero.m” to see what is on the first line of the “fzero” function. One thing to note is that the name of an M-file and of the function inside should be the same.

Independent of what kind of M-files you want to write, probably you need to use the following three commands quite often.

1. “for” loop

A “for” loop executes a group of statements a fixed number of times. The structure is

“for i =1:T; *code inside* ; end;”

In the statement, “i” is the index for the rounds and T is the total rounds for running the code inside the loop. For example, to generate and visualize a path for a first order difference equation $k_{t+1} = \alpha\beta Ak_t^\alpha$ from $t = 1$ to $t = 20$, where $\alpha = 0.36$, $\beta = 0.96$, $A = 1.0$, and $k_1 = 0.1$, we can write the following code:

```
T = 20;  
alpha = 0.36;
```

```

beta = 0.96;
A = 1.0;
k_start = 0.1;
k = k_start * ones (T+1, 1);
for i = 1:T;
    k(i+1) = alpha * beta * A * k(i)^alpha;
end;
plot (1:T+1, k);

```

Does the graph look familiar to you? Think about Solow model and a special case of Ramsey model which is consistent with Solow growth model. What if you change the initial value to “k_start = 1.0”? Why does the shape look different? Think about the economic scenario.

2. “while” loop

A “while” loop executes a group of statements an indefinite number of times, based on some logical condition. The structure is

```

“while (logical condition is true); code inside; end;”

```

For example, after generating and visualizing the path in the previous section, we may want to find a “T” such that the difference between k_T and the steady state capital $k^* = (\alpha\beta A)^{\frac{1}{1-\alpha}}$ is smaller than a small number (tolerance level, say 0.0001). To achieve this goal, we can modify the previous code to the following one:

```

alpha = 0.36;
beta = 0.96;
A = 1.0;
kstar = (alpha*beta*A)^(1/(1-alpha));
k_start = 0.1;
k = k_start;
toler = 1e-4;
T = 1;
while (abs (k(T) - kstar) > toler)
    k(T+1) = alpha * beta * A * k(T)^alpha;
    T = T + 1;
end;
disp(T);

```

3. “if” statement

A “if” statement executes a group of statements based on some conditions. The structure is

```

If (the first condition is true); (Execute the first group of commands);
elseif (the second condition is true); . (Execute the second group of commands);
elseif ...;
else; (Execute final group of commands);
end;

```

In the previous example, we may want to know whether the initial value (k_1) is smaller or larger than the steady state value. We can add the following code to the end of the code in the previous section:

```
if  $k\_start > kstar$   
    disp('the starting capital is larger than the steady state');  
elseif  $k\_start < kstar$   
    disp('the starting capital is smaller than the steady state');  
else  
    disp('the starting capital is equal to the steady state');  
end;
```

V. Final comments

1. Get started in writing a more sophisticated code

Now you have got to know enough to start to write a long and more sophisticated code. Like the case of writing a long essay, a good code typically needs a clear structure and style. Your personal style depends on your personal preference and your way of thinking. Independent of what style you like, it is good to stick to your preferred style so that you can understand your own code easily in the future.

Again, like the case of writing an essay, a good starting point is to read and learn tricks from others. The commercial code from the Mathworks (the company behind MATLAB) folks should be among the most professionally crafted programs available. Start to read the code of the functions when you need to use them and gradually get a sense of professional programming. But don't overdo it at the beginning since some codes are too professional to be absorbed by a layman.

2. Some simple tips

(a) Make comments in your code as much as possible. Without comments, almost surely you cannot understand your own (sophisticated) code later. To write comments, just start your words with "%".

(b) Define a parameter as a variable instead of writing it as a number. For example, define subjective discount rate as "beta = 0.96" instead of writing down the number in the equations. You need to change the value of some parameters frequently in your code. It will make your life easier.

(c) Initialize a matrix. It will make speed faster and remind you the dimension of the object you are working with. For example, to do shooting algorithm with time horizon between 0 and T. Initialize your solution path as "k = ones (T + 2, 1)."

(d) Think about the tradeoff between the efficiency of executing a code and the time spent in writing an efficient code. If you spend more time in optimizing a code than that saved from running a more efficient code, maybe you should stop and stick to your less efficient and even ugly code.

Good luck and enjoy your journey!