Introduction to Matlab for Econ 511b

Jinhui Bai
January 20, 2004

I. Introduction

Matlab means "Matrix Laboratory". From the name you can see that it is a matrix programming language. Matlab includes both the computer language and a rich toolbox written by it. The current version of Matlab as a language is 6.5, with the full package including toolbox called "Matlab Release 13".

II. Matlab Basics

1. Start to Use Matlab

1.1. Get to Know your Friend

You can launch Matlab as any other program in the Windows Operation System. When you enter Matlab, you will see "command window", where you can write simple command and read the result. You may also see other windows besides that.

For example, in "Launch Pad" you can see main products included in your copy of Matlab. "Workspace" includes the basic properties of all the variables generated by your process, and you can double click to see the contents of those variables. "Command history" reminds you about the commands you have typed in "Command Window". "Current Directory" gives your working directory, where Matlab will search once you start to execute a program. Therefore, you should put your program in "current directory". You can change current directory by clicking the "browse button" beside that.

To launch a window, just click "View" menu. Try to play around with those menus for a while. As a window user, you will know the meaning of those menu items without learning them.

1.2. Get Help about Matlab

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

First, you can click menu "Help\Matlab Help". There you can find a very good introduction (indeed the best help among software I ever used). 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*" in command window. For example, if you want to know how to use the function to do OLS in Statistics Toolbox, just type "help regress". Then you will see the introduction to "regress" function. If you type "help stats", you will find the whole list of functions in Statistics Toolbox.

The third way to get help is to go to Matlab website. In Matlab Version 6.5, you can click menu "Web\MATLAB Newsgroup Access". In the BBS system there, a lot of friendly people are ready to answer your questions.

1.3. Set Path

If you get 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.

1.4. Edit and Run a program

In the Matlab, the file containing code is called M-File. For example, the function to do bootstrap is called "bootstrp.m". If you want to read its code, you can (a) type "type bootstrp" in the command window; or (b) double click "bootstrp.m" in the folder if you can find it; or (c) type "edit bootstrp". For either (b) or (c), you are led to Matlab editor\debugger. 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 writing a code "shooting.m" to solve a second-order nonlinear difference equation using shooting algorithm, type "shooting" in command window.

2. Creating Matrix

Now it is time to start writing your own code. Loosely speaking, the basic element to work with is matrix, although it can be more than two dimensions. There are four ways to get 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 called "a". We use "[ ]" to enclose the matrix elements; use comma (or equivalently 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. As a practice, try to type "A = [1 2; 3 5]" without a semicolon at the end of this command. What have you seen?

Second, we can use built-in functions to get matrix. For example, "a = zeros (3, 4)" generates a 3-by-4 zero matrix; "b = ones (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 (2,5)" is similar to the above case, but with standard normal distribution.

Third, load matrix from an outside data set. In Matlab, the data file is called "*.mat" file. If you want to use "a.mat", type "load a;".

The last way is to merge available matrices. To concatenate a matrix, just treat each matrix as a number and do the concatenation as manually input a matrix. Try to type the following "A=ones(2,3); B=zeros(2,4); C=rand(3,3); D=randn(3,4); E=[A B;C D]" and read the output.

3. Submatrix

Quite often we need to refer to submatrix. If we want to refer to one element of a matrix "a", just type "a (i, j)" where i and j are row and column number of that element. To get a submatrix, we need to specify both the row number and column number 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".

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).

Here you can see that the middle number means the size of increment, with default case as increment 1. Now you can understand the meaning of "b = a (1:3, 2:4)" or "b = a (1:3, :)".

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

4. Matrix Operation

4.1. Arithmetic Operator

Here is a list of main arithmetic operators on matrix.

Table 1: Arithmetic Operators

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

Here the division operators need 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 \ b is the solution of A * x = b, while x = b / A is the solution of x * A = b. As a practice, you can try to use division operator to solve for the OLS estimators.

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 from "=". The "A == B;" checks whether two matrices are equal, while "A = B" gives value of B to A.

4.3. Logical Operators

Table 3: Relational Operators

Operator	&	|	~
Meaning	and	or	not

III. Programming in Matlab

1. Programming Basics

As said before, the code in Matlab will be stored in an M-File. There are basically two kinds of M-file: script and functions. As you must know, function returns result for some given input variables (e.g. function zeros (m, n)). Script is just a patch of code which gives a result, 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 function starts with "function *output = name (input)*;". Try to do "type rank" to see what is inside of Rank function. One thing to note is that the name of the M-file and of the function should

be the same.

2. Some Useful 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. To see how to get the maximum value of each row and corresponding index, type “help max”.

Plot(x, y) gives a plot of y as a function of x. To know more about plotting, see the help file about this.

3. Flow Control

One of most frequently used programming techniques is flow control. You will use the following three commands quite often.

First, “For” loop. The structure is “for...end;”. It executes a group of statements a fixed number of times. For example, to simulate a path for a first order difference equation $k_{t+1} = 2k_t^{0.5}$ from 0 to T=20 for given $k_0 = a$, we could write the following code:

```
T = 20;
K= a * ones (T+1, 1);
for i =0 to (T-1)
    K(i+1) = 2 * K(i)^(0.5);
end;
```

Second, “if” statement. The structure is “if...elseif...else...end;”. In the example above, we may want to find an initial condition such that K (T) is approximately 20. Assume that we already know that starting from $k_0 = a$, K(T) < 20; if we start from $k_0 = b$, K(T) > 20. And we want to use “bisection method” to find the solution. We can write the following code:

```
c = (a + b)/2;
K(0)= c
"simulate a path from the method in “For loop”"
if K(T)< 20
    a = c;
elseif K(T) == 20
    disp(‘Doesn’t it has measure zero?’);
elseif K(T) > 20
    b=c;
end;
```

Third, “While” loop. The structure is “while...end;”. It executes a group of statements an indefinite number of times, based on some logical condition. Now we can finish our steps by the following code:

```
while (abs (K(T)-20)> 0.01)
    “write the code above”;
end;
```

4. Some Simple Tips

4.1. Make comments in your code as much as possible. Without comments, almost surely you cannot understand your own code later. To write comments, just start the line with "%".

4.2. Define constant as a variable instead of writing it as a number. For example, define subjective discount factor as BETA = 0.95 instead of writing down the number in the equations. You need to change the value of some constants frequently in your code. It will make your life easier.

4.3. Avoid loops whenever possible. In terms of matrix operation, Matlab can be as efficient as (or sometimes better than) C or Fortran. But loops make Matlab much slower.

4.4. Initialize a matrix. It will make speed faster. For example, to do shooting algorithm with time horizon between 0 and T. Initialize your solution path as "k = ones (T + 2, 1);".

IV. Shooting Algorithm: one candidate structure

Now you have got to know enough vocabulary to write your code (at least for this problem). Similar to the case of writing an essay, you need to have a structure to organize your code before you start. Here is one possible way to organize the code.

First, fix a time horizon T, and start to simulate a path given any initial guess of K1. You can do it in a "For" loop. You can put the solution path in either a row vector or column vector of your solution path matrix. Check whether the value of T + 1 is close to zero. If yes, then find the maximum value within the path. If not, start from a new guess (maybe from bisection method from lectures) until you find one satisfactory solution.

Second, check whether the maximum value exceeds 90. If yes, stop. If not, do the case for T+1. If you want to solve for it faster, maybe you can do bisection again for T as a variable. Here again it is a loop, you can embed the first step in this loop.

Good luck and hope it helpful.