Matlab Tutorial

AGEC 637 - Summer 2012

I.   Using this tutorial

This tutorial will walk you through some basic steps in Matlab. Do not simply reproduce the lines of code and move on. Be sure that you understand what is going on. At each step you should be able to alter the program to deal with a slightly different problem.

II.   Matlab as a tool, not a crutch

Matlab (or any other symbolic algebra program) can be an invaluable tool, helping to save hours of frustrating pencil time and avoiding careless mistakes. But it can also be a crutch. It is often the case that valuable intuition arises in the process of solving a problem. Sometimes computers will jump over these intermediate steps or simplify a solution in a fashion that robs the work of any intuition. I have reviewed papers for journals in which algebra was clearly done by a computer and the result was the author had little understanding of what was actually being done in the process.   Don't let this happen to you. *Use Matlab to help you understand; don't let it keep you from understanding.*

III.   The big picture: An example of some Matlab code

Just to give you an idea of the type of thing that Matlab can do for you. Here's an example of simple code to solve an consumer utility maximization problem and obtain the Marshallian demand function using Roy's identity (**do not try this yet**)

```
% Initialize variables
syms a b c x y z l m px py pz

% Specify utility function
u = a*log(x)+b*log(y) + c*log(z)

% Lagrangian and FOCs
L = u -l*(m-px*x-py*y-pz*z)
dLdx = diff(L,x)
dLdy = diff(L,y)
dLdz = diff(L,z)
dLdl = diff(L,l)

% Solve the system for x, y and z
[xstar ystar zstar lstar ] = solve(dLdx, dLdy, dLdz, x, y, z, l)
% Indirect utility function and Roy's Identity
V = a*log(xstar)+b*log(ystar) + c*log(zstar)
dVdm = diff(V,m)
dVdpx = diff(V,px)
xM = -dVdpx/dVdm
```

IV.   Getting Started

1. Start Matlab. We will begin by using the command-line interface. A more sophisticated programming approach will be discussed later. You know you're in the command-line interface if you see a >> prompt.

2. Any variable to be used must be introduced with a syms command, e.g.,
   >> syms a b c x

3. If a function or variable is introduced on the left hand side of an equation, it does **not** need to be specified first, e.g.,
   >> f=a+b*x+c*x^2
   This creates a new variable, f, that is a function of a, b, c and x.

4. Note that if you put a semicolon (;) at the end of a line, it does not print the output to the screen. Compare the output of the previous line with
   >> f=a+b*x+c*x^2;

5. In the command window, using the up and down arrows on your keyboard, you can access previous command lines and then edit them if necessary.

6. Note that Matlab is case sensitive. For example, if you simply enter the command
   >> f
   it will reproduce the equation. If you type F, on the other hand, you will get an error message. Commands are similarly case sensitive.

V.   Differentiation

7. To find the derivative of f, simply type
   >> diff(f)
   or define a new function
   >> dfdx=diff(f)

8. If you don't specify the variable with respect to which you want to take a derivative, Matlab will guess. So it's a good idea to be specific. To make sure that you're differentiating with respect to x, use
   >> dfdx=diff(f,x)

9. Try
   >> dfda=diff(f,a)

VI.   Integration

Integration is where programs like Matlab really begin to pay off because integrating can be very difficult and prone to mistakes.

10. An indefinite integral (Note that Matlab drops the constant term) can be done using the command
    >> fi=int(dfdx)
    which assumes you're integrating over x
    >> fi=int(dfdx,x)
    where the integrand is explicit. Compare fi with f. Why are they different?

11. Definite integrals
    >> fi = int(f,a,b) or fi = int(f,x,a,b) or you could give it numbers, e.g., fi = int(f,-1,1)

12. Try
    >> f = 1/x^2
    >> fi = int(f,x,1,inf)
    to integrate from 1 to ∞.
    (Note that by putting f on the LHS above, it now has a new meaning 1/x^2)

13. Also try to integrate
    >> f = 1/x^1.1 and f = 1/x
    from 1 to ∞
    (remember, you can access and edit previous commands with the up & down arrows)

14. In these cases Matlab should have no trouble evaluating the integrals, but be careful.
    All too frequently the program will not find an integral, yet sometimes a solution does
    exist.

VII. Solving systems of equation

15. Suppose we want to find x and y such that $x+y=0$ and $x-y/2=\alpha$
    >> syms x y alpha
    >> f1 = x+y
    >> f2 = $x-y$/2$-$alpha
    >> [x,y]=solve(f1,f2)

16. Now, let's start over with a clean slate by using the
    >> clear all
    command.

17. After completing 15. Redo the previous set of commands, redefining f1 as follows
    >> syms x y alpha
    >> f1 = x^2*y^2 −1
    >> f2 = $x-y$/2$-$alpha
    >> [x,y]=solve(f1,f2)

    You should no longer get a specific scalar value for x and y? What do you think the
    meaning of this result is? Think about what you know about the solution to nonlinear
    equations.

18. A solve statement can also be written putting the RHS of f1 and f2 inside single
    quotation marks as follows:
    >> [x,y]=solve('x^2*y^2-1','$x-y$/2$-$alpha')
    which will solve for the values of x and y that make the functions equal to zero.

VIII. Differential equations

19. Suppose we want to solve the differential equation $\dot{y} + 4y = e^{-t}$ with $y(0)=1$
    >> y=dsolve('Dy+4*y=exp(-t)', 'y(0)=1')
    Note the capital D is used to indicate that it is a derivative with respect to t.

20. Second- and higher-order differential equations can be solved by converting them into a series of first-order differential equations. A second-order differential equation can also be solved replacing 'D' above with 'D2.'

IX. Eigen values

21. Create the matrix A with the command
    >> A=[a,b;c,d]
    note that comma divides the columns and the semicolon divides the rows so that if we instead wrote A=[a;b;c;d] or A=[a,b,c,d] we could get a column or row vector respectively.

22. The Eigen values of the square matrix A are found simply with the command
    >> B=eig(A)
    either symbolic or numeric expressions can be in A.

23. Since the Eigen values of a 2×2 matrix are the values $\boldsymbol{\lambda}=(\lambda_1, \lambda_2)$ that set the determinant of $\mathbf{A}-\lambda\mathbf{I}$ to **0**, it follows that the values if $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, then the Eigen values are $\dfrac{a+d \pm \sqrt{a^2 - 2ad + d^2 + 4bc}}{2}$. Your answer should be

    B =
    [1/2*a+1/2*d+1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]
    [1/2*a+1/2*d-1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]

X. Displaying your results

24. Sometimes its useful to see your results in the "pretty" format
    >> pretty(f)

25. To simplify
    >> simplify(f)

26. The command
    >> simple(f)
    would present a variety of ways, allowing you to decide what you want.

27. For vectors or arrays, you can refer to a single element of the array using the standard row, column order, e.g. "B(1)" will refer to the first Eigen value and "A(1,1)" will display "a"

XI.   Saving your work and comments

28. All work that is carried out in command-line mode can be saved as a workspace file (with a .mat extension) so that you can step right back in where you left off. Try saving your work up to now in such a file.

29. Any time you may want to refer back to your work, it is helpful to insert comments, even when you're working in command-line mode. This can be done by placing a "%" sign before the text that you want Matlab to ignore, e.g.
    >> syms xx yy   % These are two new variables

XII. Scripts (a.k.a. M-files)

30. Up to this point we have been implementing our code using the command-line interface with Matlab. You can also, and usually preferably, use script files that contain a sequence of commands. These used to be called M-files. A script has the advantage that you can print it, and easily reproduce your results.

31. To create a script, start by using the sequence of commands: From the **F**ile menu, choose **N**ew, **script** (or ctrl-N). This will create an empty file in which you can type commands. Type a sequence of commands, such as

    ```
    syms a b c x
    solve('a+b*x+c*x^2=0',x)
    ```

    Save this file under a name such as MyFile.m. For some reason Matlab is very particular about names for these files. Start the file's name with a letter and make sure there are no spaces in the name.

    From the M-file editor, you can press the F5 key and this will run the file. Unless you have stored the file in the default Matlab directory, the program should prompt you to change the directory. Any of the options provided will work fine.

32. Alternatively, from the command window you can run an m file by simply typing the name of the file, without the .m extension. If you are using an m-file in another directory, you will need to change the path. Matlab should prompt you for this need to use the File and Set Path commands.

33. As in the command-line, you can suppress output by putting a semicolon at the end of a line of code. This is particularly important if your code is long.

34. As in command-line mode, you can insert comments into your program using the % symbol before the text that you want to be ignored by the program. This is a good practice in all programming, whether in Matlab or any other language (*and is required on your problem sets*).

35. If you need to break a command into 2 lines, put 3 periods "…" at the end of the line that is to be continued.

XIII. Graphing your results

36. One of the real strengths of Matlab is in its graphing capability. If you want to get
    fancy, you'll need to consult other sources, but for very basic graphing, you follow
    the following approach. a) Define a numerical range for your independent variable.
    b) Set numerical value for all parameters. c) Evaluate the function, yielding a vector
    of values for your dependent variables, then plot the data.

    ```
    >> x = 0:0.2:12;          % defines x for value from 0 to 12 by 0.2
    >> a = 3; b =2; c = -.2;  % parameter values
    >> y = a + b*x + c*x.*x   % values for y. Note x.*x because we want scalar multiplication
    >> plot(x,y)              % plots x on the horizontal axis, y on the vertical axis
    ```
    Which yields the following graph
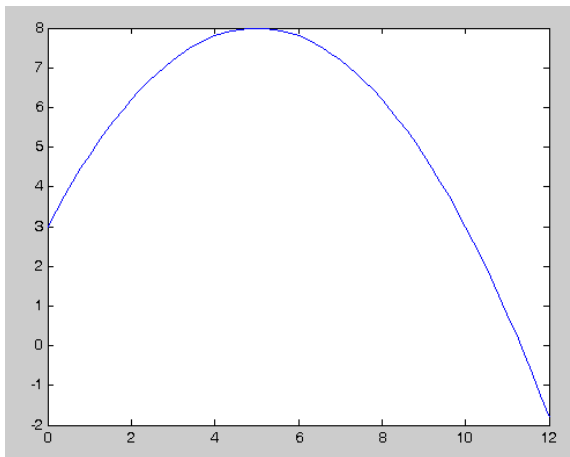
    Note: As far as I know, switching from symbolic expressions to numerical ones
    which can be graphed is not straightforward. If you discover a way to do so, please
    let me know.

XIV. Inline functions

37. Sometimes you may want to actually plug real numbers into a function. In this case
    you use what is called an inline function. Here's an example:
    ```
    >> f = inline('a*x + b*x^2')
    >> a = 3
    >> b = 1
    >> x = 2
    >> f(a,b,x)
    ans =    10
    ```
    Unfortunately, as far as I know, you cannot manipulate inline functions (e.g., using
    diff or int as you do with symbolic functions. Again, if you figure out how to do this,
    please let me know.