A Brief Introduction to Matlab for Econometrics Simulations

Greg Fischer
MIT
February 2006

Introduction

First, Don't Panic! As the problem set assured you, the point of the programming exercises is to help you develop intuition and quantitative skills in econometrics, not to drop you into some Sartre-esque morass of endless help files, inscrutable code, and excessive Mountain Dew consumption. We'll try to introduce key programming tips *before* you write your problem sets. Occasionally we'll cruelly leave something out, offering you a teachable moment during which to familiarize yourself with Matlab's or the program of your choice's help files, but odds are we just forgot to explain it. Again, if something seems unduly time consuming, don't be shy about asking for advice

I am in no way a Matlab wizard. For most applied work, you'll want to work with software, such as Stata, that has more user-friendly canned packages. But our goal in 382 is to familiarize you with the nuts & bolts of econometrics and these packages make it hard to do so.[1] In this regard, Matlab is great as its strength is matrix manipulation and it's also a very intuitive programming language.

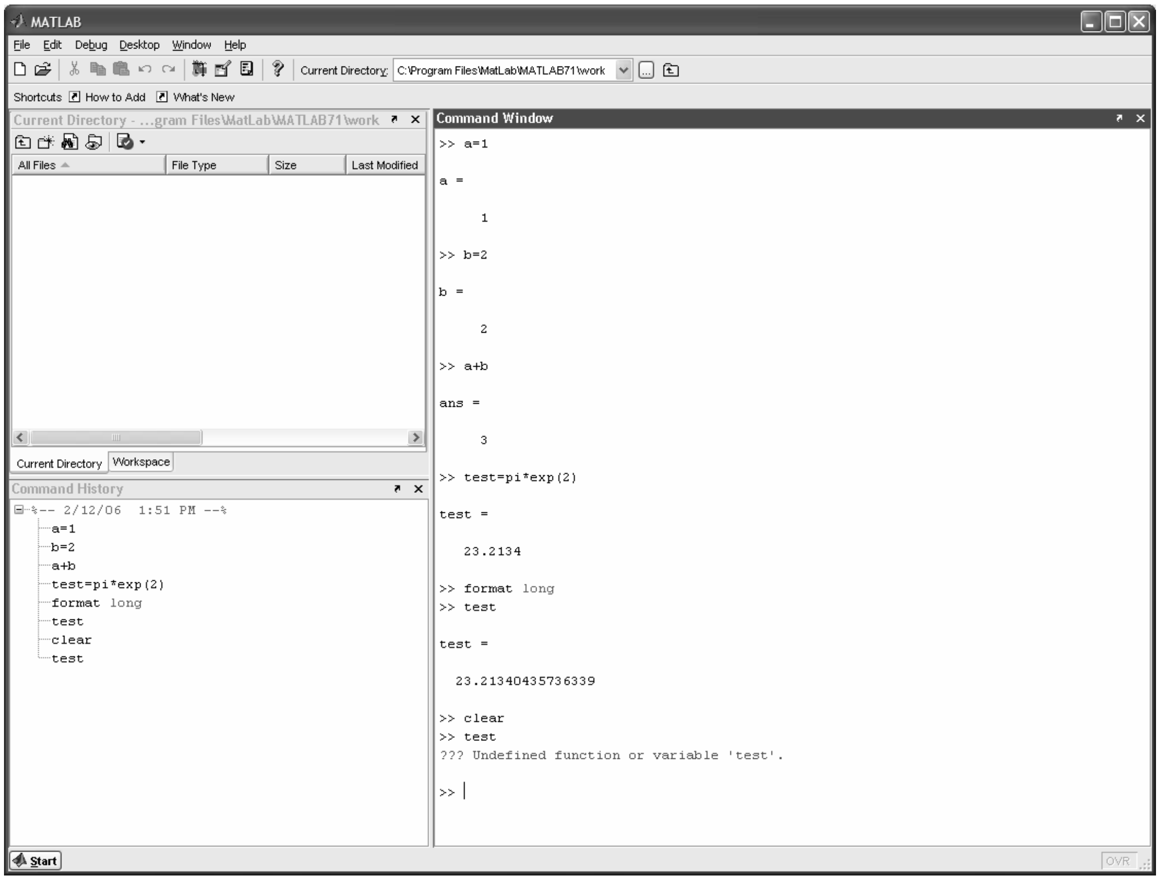
There are some decent on-line tutorials and references available from which I will occasionally draw (without reference) and which offer a more comprehensive introduction that this will. A decent place to start is http://www.math.utah.edu/lab/ms/matlab/matlab.html. For something more thorough, try http://www.math.siu.edu/matlab/tutorials.html. It's the best I've seen (though I'm not sure that says much) and is a useful starting place if you really want to figure out the

[1] Rumor has it that Stata 9 actually does a very good job with matrix manipulation, but since few of us have access to it let's pretend it doesn't exist for now.

program. Finally, let us and your classmates know if you find a better source. We'd all be thankful. With that, let's get started.

Getting Started

Open up Matlab. You'll see something that looks like Figure 1. The pane to the right is the Command Window and allows you to enter simple commands. Go ahead and try it. Enter something like `a=1` and hit enter. You'll see the results of your handiwork both in the Command Window and in the Command History on the lower left. Now set b equal to 2. Add a plus b. Voila. a + b = 3. It doesn't get all that much harder. You'll notice in the remaining commands shown in Figure 1 that Matlab truncates by default to 4 decimals. You can change this either by entering "format long" in the command line or by changing your preferences under the File-Preferences menu.


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Program Files\MatLab\MATLAB71\work
Shortcuts How to Add What's New
Current Directory - ...gram Files\MatLab\MATLAB71\work
All Files
File Type
Size
Last Modified
Current Directory
Workspace
Command History
Command Window
Start
OVR

```
%-- 2/12/06  1:51 PM --%
a=1
b=2
a+b
test=pi*exp(2)
format long
test
clear
test
```

```
>> a=1

a =

     1

>> b=2

b =

     2

>> a+b

ans =

     3

>> test=pi*exp(2)

test =

   23.2134

>> format long
>> test

test =

  23.21340435736339

>> clear
>> test
??? Undefined function or variable 'test'.

>> 
```

Figure 1

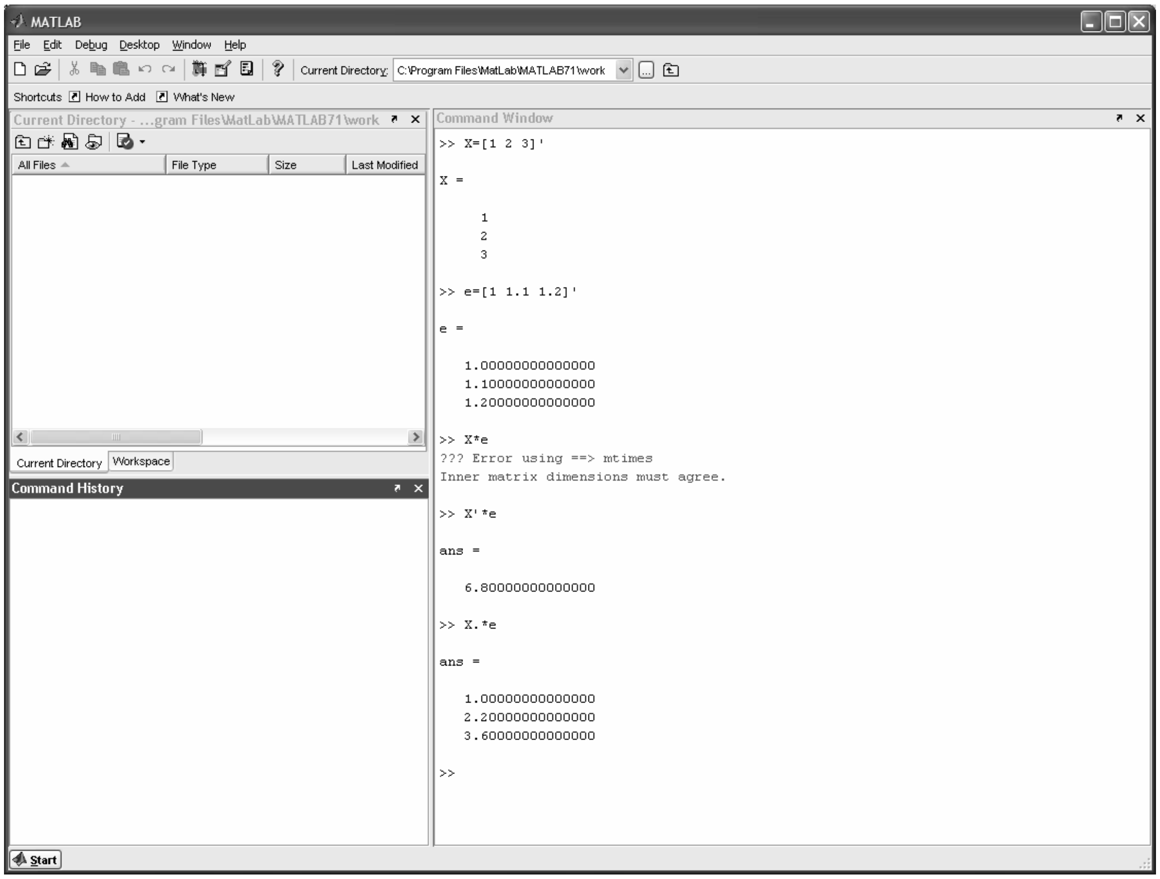
2 See Chris Smith's "How to use programs on blackmarket or shadydealings" posted on the GEA website's Computer Help section for help in setting up a Matlab session; available only to Course 14 students.

Defining & Manipulating Matrices and Vectors

You can define matrices and row vectors in much the same way you do a scalar. Say you wanted to enter a column vector (4,5,6)'. You can type `a=[4; 5; 6]`, `a=[4,5,6]'`, or `a=[4 5 6]'`. Matlab interprets semicolons as the end of a row and interprets commas or spaces as spaces between entries within a row. So you would define the 2 x 2 identity matrix as `X=[1 0; 0 1]`. You can also use a short cut and just say `X=eye(n)`. This will generate an n x n identity matrix. Here are some of the matrix commands you may find useful:

```
Transpose:              X'
Inverse:                inv(X)
Identity:               eye(n)
Element:                X(i,j)
Operators:              +, -, etc. work as you'd expect
Element-wise operator:  Insert a . before the operator
```

The element-wise operator. Do the following: define X to be a 3 x 1 column vector equal to (1, 2, 3)' and e to be a 3 x 1 column vector equal to (1, 1.1, 1.2)'. Now run through the commands shown in Figure 2. If you try to multiply X by e, you'll get an error since the matrices don't conform. `X'e` will give you the inner product, and `X.*e` will give you element-wise multiplication. This can be quite handy.

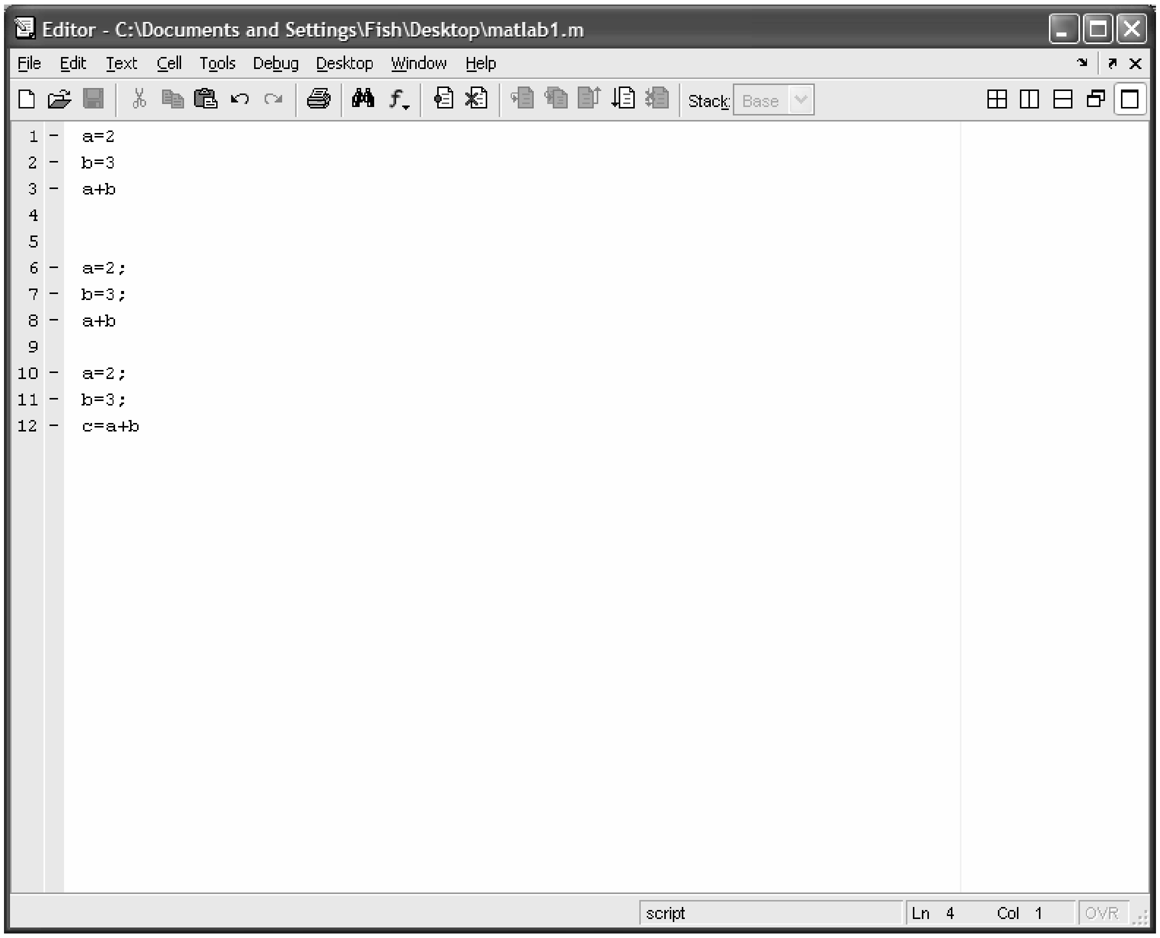

MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Program Files\MatLab\MATLAB71\work
Shortcuts How to Add What's New
Current Directory - ...gram Files\MatLab\MATLAB71\work
All Files File Type Size Last Modified
Current Directory Workspace
Command History
Command Window
Start

```
>> X=[1 2 3]'

X =

     1
     2
     3

>> e=[1 1.1 1.2]'

e =

   1.00000000000000
   1.10000000000000
   1.20000000000000

>> X*e
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> X'*e

ans =

   6.80000000000000

>> X.*e

ans =

   1.00000000000000
   2.20000000000000
   3.60000000000000

>>
```

Figure 2

Programs

While you can do quick calculations in the Command Window, for programming you'll want to put your work in an m-file. To open a new file, select File-New-Mfile. Type something trivial in this file, as you see in Figure 3.


Editor - C:\Documents and Settings\Fish\Desktop\matlab1.m
File Edit Text Cell Tools Debug Desktop Window Help
Stack: Base
script
Ln 4 Col 1
OVR

```
 1 -  a=2
 2 -  b=3
 3 -  a+b
 4
 5
 6 -  a=2;
 7 -  b=3;
 8 -  a+b
 9
10 -  a=2;
11 -  b=3;
12 -  c=a+b
```

Figure 3

Save your file somewhere, and then run it by either hitting F5 of selecting Debug-Run. Toggle back to your Command Window and you'll see the results. You may have a lot of entries the results for which you don't want to see in the command window. To suppress the output of these lines them with a semicolon as in lines 6, 7, 10, and 11. This also works in the Command Window.

An Example

Rather than go through commands and code structure in the abstract, let's walk through an example of some code that does something similar to what you'll be asked to do on the first and second problem sets. The file is heavily annotated and should cover directly just about everything you'll need to do to complete the exercise. And no, I'm not posting the m-file itself, but short of that, please don’t hesitate to contact Raymond or me if you have questions, comments, or suggestions. Have fun.

```
%**********************************************************************************************************************
% 14.382, Introduction to Econometrics
% Matlab Introduction
% Written by Greg Fischer
% February 11, 2006
%**********************************************************************************************************************
%   It's good coding practice to descript what you're doing up front.
%   Note the % marks a comment and the asteriks (*) help readability
%   You may also want to change the preferences under editor/debugger-language to autowrap at 120, landscape width
%**********************************************************************************************************************

clear all                                         % clears all variable from current workspace
clear global                                      % type `help clear' to see why you want to do this

%**********************************************************************************************************************
% Problem Summary
% This is NOT your problem set.  Just an edited version of some code I% had lying around
% Though it doesn't show up in the pdf, when you write in the editior you'll note the nifty color coding, green for
% comments, black for text, purple for strings,etc.  You can change these to suit your preferences too.
%**********************************************************************************************************************
% y(i) = x(i)*Beta + epsilon(i)             [1]
% x(i) uniformly distributed on (0,1)
% epsilon(i) = x(i) * v(i) where x & v are independent; v distributed N(0,1)
% true Beta is 1
% Repeat 1,000 times:
% Generate 200 observations from the model, equation [1]
% Test beta=1 using usual variance, s^2 * (X'X)^-1, for betahat
% Test beta=1 using Eicker-White Variance for betahat
% Comment on fraction of times we can reject for each method
%**********************************************************************************************************************

%**********************************************************************************************************************
% Inputs & Counters                               % It's also a good habit to define inputs up front not as you go
%**********************************************************************************************************************
iter        = 1000;                               % Iterations
Beta        = 1;                                  % True beta
n           = 200;                                % Number of replications
UsualCount  = 0;                                  % Defining variables that I'll use later
```

```
WhiteCount  = 0;
result    = zeros(iter,1);                      % sets aside an iter x 1 matrix to store results
%*****************************************************************************************************************
% The Simulation
%*****************************************************************************************************************

for m = 1:iter                                  % this is the formal for all for loops
                                                % Hit Ctrl-c to break out of a loop gone bad
    % Step 1: Generate 200 observations from [1]
    X = rand (n,1);                             % Step 1A: Generate 200(n) x 1 vector of x's distributed U(0,1)
    v = normrnd (0,1,[n 1]);                    % Step 1B: Generate 200(n) x 1 vector of v's distributed N(0,1)
    e = X.*v;                                   % Step 1C: Caculate e (epsilon) where e(i) = x(i) * v(i)
                                                % This is the element-wise multiplication mentioned in text
    y = X*Beta + e;                             % Step 1D: Calculate y per equation [1]
                                                % We've not generated our random sample according to the model above

    % Step 2: Calculate BetaHat & Other statistics
    BetaHat = inv(X'*X) * X' * y;
    e = y - X * BetaHat;
    s2 = e' * e / (n - 1);
    SHat = (X.*e.*e)'*X/n;                      % Just a sneaky, low resource way to get SHat
    result(m,1) = BetaHat;                      % Stores our estimate of BHat as element m of results vector

    % Step 3: Calculate Standard Errors
    var_usual   = s2 * inv(X' * X);             % Don't worry about the particulars for now.  Just get the sense of
    se_usual    = sqrt (var_usual);             % How the programming language is working & decent file structure
    var_white   = n * inv(X' * X) * SHat * inv(X' * X);
    se_white    = sqrt (var_white);

    % Step 4: Test Hypotheses and Count Rejections
    % Step 4A: The Usual Test
    t_usual     = (BetaHat - Beta) / se_usual;
    if abs (t_usual) > 1.96
        UsualCount = UsualCount + 1;
    end

    % Step 4A: The White Test
    t_white     = (BetaHat - Beta) / se_white;
    if abs (t_white) > 1.96
        WhiteCount = WhiteCount + 1;
    end
end

UsualReject = UsualCount / iter;
WhiteReject = WhiteCount / iter;

%*****************************************************************************************************************
% Display Results
%*****************************************************************************************************************
% By now you will have noticed that most of Matlab's displays are quite ugly.  This is just a slight clean up
disp ('  ');
disp(sprintf('# replications =%8.2f', iter));
disp(sprintf('Prob Rejection, Usual = %6.4f', UsualReject));
disp(sprintf('Prob Rejection, White = %6.4f', WhiteReject));

%*****************************************************************************************************************
% Histogram
%*****************************************************************************************************************
BMin = 0.5;                                       % I'm forcing the bins in my histogram, but you can just as easily cal
BMax = 1.5;
Bins = 25;
BRange = [BMin:(BMax-BMin)/Bins:BMax];            % An easy way to define a vector with steps of 1/25 between elem
figure                                            % Creates a Matlab graphics object.  Not necessary if only 1 plot
                                                  % But still a good habit
cdfplot(result)
figure
hist(result,BRange)                               % Makes your histogram
                                                  % Look at ylim, xlabel, ylabel, subplot and title for some beauty tips


```