# Econometrics II (ECON 372)
# Lecture 2
# Programming in MATLAB

Teng Wah Leo

## 1 Matlab Programming

Although you can use STATA for programming needs, it is a less intuitive language to learn. Another advantage of MATLAB is its accuracy, but the drawback being it raises the computational time required for each task consequently

### 1.1 Entering Matrices

You can enter matrices into MATLAB in several different ways:

1. Enter an explicit list of elements.
2. Load matrices from external data files.
3. Generate matrices using built-in functions.
4. Create matrices with your own functions in M-files.

In writing or creating your own matrix, you need to follow a few basic conventions:

1. Separate the elements of a row with blanks or commas.
2. Use a semicolon ";" to indicate the end of each row.
3. Surround the entire list of elements with square brackets, [.].

**Example 1** *A*= [16, 3, 2, 13; 5, 10, 11, 8; 9, 6, 7, 12; 4, 15, 14, 1] *and* MATLAB *displays the matrix you entered as*

$$A = \begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{bmatrix}$$

*it should noted that what you have entered is known as a Durer matrix or what is commonly referred to as the magic square. A magic square yields an interesting feature, the sum of its rows or columns or its diagonals always yield the same number. You can also get 4×4 magic square by writing B=magic(4).*

## 1.2 Sum, Transpose and Diagonal

The command sum(.) in MATLAB sum the elements of each column within a matrix. To get the row sums, the easiest way is the use the command on the transpose of a matrix, and transpose is indicated by the symbol, ′. The sum of diagonals can be obtained by the command diag(.), which extract the diagonals and writes it as a column vector. So that diag($A$) gives

$$\begin{bmatrix} 16 \\ 10 \\ 7 \\ 1 \end{bmatrix}$$

To obtain the sum of the counter diagonal, the command fliplr(.) flips a matrix from left to right. Combining the fliplr, with diag, and sum yields the sum required.

## 1.3 Subscripts

You can call any of the elements in a matrix be defining the location in the matrix, so for example if you what the number 7 in $A$, you call it in MATLAB by typing

$$A\,(3,3)$$

where the first '3' is for the row location, and the other is for the column location.

## 1.4 The Colon Operator

1:10 gives a row vector of integers from 1 to 10.

To obtain non-unit spacing, you should specify an increment. 100:-7:50 yields

100 93 86 79 72 65 58 51

and $0 : \pi/4 : \pi$ yields

0 0.7854 1.5708 2.3562 3.1416

To obtain a portion of the matrix, say for example, you wish to extract rows 1 to 3 on the second column, you type

$A\,(1{:}3{,}2)$

The colon by itself refers to all the elements in a row or column of a matrix, so for example you want all of column 3, you type

$A\,(:{,}3)$

## 1.5 Operators

1. +, Addition
2. −, Subtraction
3. ∗, Multiplication
4. /, Division
5. \, Left Division
6. ˆ, Power
7. ′, Transpose
8. pi, 3.14159265...
9. $i$, imaginary unit, $\sqrt{-1}$
10. Inf, Infinity
11. NaN, Not a number

12. .*, Element by element multiplication
13. ./, Element by element division
14. .\, Element by element left division
15. .^, Element by element power
16. >=, greater than or equal to
17. <=, less than or equal to
18. <, less than
19. >, greater than
20. ==, logical equality

## 1.6 Generating Matrices

1. **zeros**, matrix of all zeros
2. **ones**, matrix of all ones
3. **rand**, uniformly distributed random elements
4. **randn**, Normally distributed random elements

You determine the dimension of the matrix but stating the number of rows, and number of columns you want. Of course in Econometrics, and Statistics, we typically use datasets that are drawn from experiments, or actual surveys, so that the data would have to be brought into MATLAB. MATLAB datasets have the **.mat** extension, and the manner in which you load them is unsurprisingly,

**load *dataname*.mat**

You would recall that plenty of our formulas in Econometrics requires the knowledge of the number of observations of the dataset, as well as the number of variables. Recall the standard errors of the OLS coefficients? I assure you there will be many other instances. When creating a program, it is often useful to write one that is "universally" usable, in the sense that regardless of the dataset, as long as you wish to perform a particular procedure,

you want to be able to use it again, and have the program find the write answers to the number of observations and variables. This is easily achieved with the command **size($X$)** for a matrix $X$. This will give a row vector with two values, the first element being the number of rows (or the number of observations for you in our case or type of usage), and the second element is the number of columns (in your case typically this is the number of variables.). You might not need both of them all the time, so that all you need is the number of rows, the command is **size($X$,1)**, and if you need the number of columns, it's **size($X$,2)**. Typically 1 refers to rows and 2 to columns in MATLAB commands.

Once you have gotten your matrix, you might also want to get a feel of the dataset and variables in terms of the number and types of realizations in the variables. You can do so with the command **unique**, which tell you the unique numbers that can be found in the desires columns or rows of a matrix. Suppose $X$ is a matrix, and denote r as the row index and c as the column index, you can find the unique values of a particular row r of the matrix $X$ by typing **unique(X(r,:))** where the colon tells MATLAB that you want to perform the matrix for all columns of $X$. Very logically, for column c only for all rows, the command is **unique(X(:,c))**.

Finally, sometimes you may find yourself wanting to generate a running vector of integers. One possibility is when you want to account for trends in time series data, and as you will find, you can do so by numbering the first year in your sample as 1, and the second as 2 and so on until all the years are exhausted. The command for generating $n$ number of integers beginning at 1 is just $1 : n$. Of course if you wish to generate a set of integers by having it begin at some other point, you can still do so by replacing 1 with that point, and adjusting the terminal point as well.

Sometimes, particularly in nonparametric regression which you may learn about in the future, is to cut or divide your support into equi-distant points. The command that would achieve this aim is **linspace**. So if you wish to divide the support of zero to one into 100 points, you would type linspace(0,1,100).

## 1.7 Finding Sub-matrices or Subsets of Matrices

It is not uncommon that when you have an exceedingly large dataset that you might wish to test you program on a smaller subset of you dataset. Or it may be that you wish to focus on a segment of the dataset where a particular variable of interest takes on a certain value or sets of values. How then can you find this matrices or extract them

in a quick manner without resorting to creating a flow control program. **You should always keep in mind that the shorter your program is, or the more you work directly with matrix manipulations, the less your program needs to hog the cache memory on your computer, and the faster your program you run!**. There are two useful build-in commands in MATLAB that does the trick very fast, they are the **find** and **ismember** commands.

The **find** command basically helps you find the location of the set of elements you are interested in. For instance, you are working on returns to education regressions, and want to focus on individuals with incomes less than $\$x$, and the variable for income is on the $k^{\text{th}}$ column. Name the data matrix as $DATA$, then you can get this submatrix, calling is $DSMALL$ by typing $DSMALL = DATA(find(DATA(:,k) <= x),:)$. What **find** does is it gives you the location, which in this case is the row numbers they are on.

The **ismember** command generates a vector of numbers like **find** which helps in locating the elements of interest. Unlike **find** which gives you the row or column numbers, **ismember** gives you 0 and 1. For instance, you what to focus on only certain income realizations, and you save this set of numbers in a vector of matrix, call it $S$. If the income realization, using the previous example, is inside this set, you will get a 1, else 0. When you put this into a command such as $DSMALL = DATA(ismember(DATA(:,k),S),:)$, you will get that subset you want since all elements that is 0 in column $k$ is dropped.

## 1.8 M-Files

You can write your programs as a m-file.

## 1.9 Deleting Rows and Columns

You can delete entire rows or columns as follows; suppose you want to delete the second row of A,

$$A\,(2,:) = [.]$$

while if you want to delete the third column of A,

$$A\,(:,3) = [.]$$

## 1.10 Flow Control

**if statement** **if** statement evaluates a logical expression and executes a group of statements when the expression is *true*. For example, MATLAB algorithm for generating a magic square of order n involves three different cases: when n is odd, when n is even but not divisible by 4, or when n is divisible by 4.

```
if rem(n,2) ~= 0
    M=odd_magic(n)
elseif rem(n,4) ~=0
    M=single_even_magic( n)
else
    M=double_even_magic(n)
end
```

Other functions that may be useful with the **if** statement include,

- isequal
- isempty
- ismember

**for statement** **for** loop repeats a group of statements a fixed, predetermined number of times. A matching **end** delineates the statements.

```
for n=3:32
    r(n) =rank(magic (n));
end
```

**while statement** **while** loop repeats a group of statements an indefinite number of times under control of a logical condition. To illustrate the use of while, we can convert the above for loop to while loop.

```
n=3;
while n<=32
    r(n) =rank(magic (n));
    n=n+1;
end
```

## 1.11 Creating a Simple Program

Before writing a program, there are several pertinent questions you have to ask yourself, of course assuming you have all the formulas that you need in front of you. It is a non-trivial exercise, writing programs that is. You not only need to know the commands of the software, you have to also understand the statistics (or more generally, what the program needs to be able to do) behind the program you are trying to write. Of course there are no hard and fast rules, but below are some questions that will help you.

1. What is the sequence of the formulas, before arriving at the final result you desire.
2. What are the variables you need to define?
3. Although it is always easy to write loops, you have to keep in mind that each time your program runs a loop, it is hogging the computers memory, thereby slowing the process. Here is where knowledge of mathematics, and statistics comes in very handy. Can you represent the equations in matrix form? You will observe later below when we create the OLS program the reason why it was a great idea to understand the matrix derivation of the procedure.
4. Once the program is complete, you have to ask yourself whether the procedure can be written in a more efficient manner.

You will of course develop you own quirks along the way. Like mathematics, and statistics, practice makes perfect. Remember, cutting and pasting a program someone else has written, even if it is freely available, is not writing a program, and can be detrimental to your understanding of the statistics behind the program, not to mention from the point of view of integrity!

We will now create a program (function) for ordinary least squares that you could use whenever you want to perform the procedure. It is simple, and a great starting point.

### 1.11.1 Ordinary Least Squares

```
function results=ols(y,x)
  if nargin =2
    error('Wrong # of arguments to ols');
  else
```

```
    [nobs nvar] = size(x);
    nobs2 = size(y,1);
     if nobs  = nobs2
       error('x and y must have same # obs in ols');
     end
  end
  r = qr(x,0); % QR Decomposition
  xpxinv = (r' * r)\eye(nvar);
  beta = xpxinv * (x' * y);
  yhat = x * beta;
  er = y - yhat;
  rss = er' * er;
  sigma = rss/(nobs - nvar);
  varbeta = (sigma) * (diag(xpxinv));
  stdbeta = sqrt(varbeta);
  tstat = beta./stdbeta; % The period before the division sign means element-by-element
division.
  ymn = y - mean(y);
  tss = ymn' * ymn;
  Rsq = 1 - (rss/tss); % R^2
  results.beta = beta;
  results.yhat = yhat;
  results.tstat = tstat;
  results.er = er;
  results.rss = rss;
  results.sigma = sigma;
  results.varbeta = varbeta;
  results.stdbeta = stdbeta;
  results.ymn = ymn;
  results.tss = tss;
  results.Rsq = Rsq;
```

### 1.11.2 Statistical Tests

Although all statistical programming packages have a "regression" function, because it is the most basic statistical tool, it is always useful to start from programming that. With an ability to program and statistical know-how and understanding, you can always stay abreast of developments in theory. You would notice that there are elements missing, particularly the p-values for for $t$ and $F$ statistics. This programming ability will be particularly handy when we learn Monte Carlo techniques later in the course. Therefore, as an exercise, you are now required to add additional results including the confidence interval, p-values for both the $t$ and $F$ statistics. To help you with this, you would need to know the following distributions.

Before creating the programs for these tests, we need to take account of all the formulas that we require to calculate the values. You might asking yourself right now whether you know the distribution which you have been using. It is one thing to have heard of the t-distribution, but another to actually "know" the distribution. The density function of the t-distribution for a random variable $x$ with is $\nu$ degrees of freedom,

$$f(x;\nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where $\Gamma(.)$ is just the

$$\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$$

The $t-$distribution function is,

$$\begin{aligned} F(x;\nu) &= \int_{-\infty}^{x} \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}} dt \\ &= 1-\frac{1}{2}\mathcal{I}_{x(t)}(\frac{\nu}{2},\frac{1}{2}) \end{aligned}$$

where $x(t)=\frac{\nu}{t^2+\nu}$

$$\mathcal{I}_x(a,b) = \frac{\beta(x;a,b)}{\beta(a,b)} \text{ , Regularized Beta Function}$$

$$\text{, and where } \beta(x;a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt \text{ , Incomplete Beta Function}$$

$$\text{and } \beta(a,b) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} \text{ , Beta Function}$$

Fortunately for you and I, the $\beta(.)$, $\Gamma(.)$ and $\mathcal{I}_x(.)$ are all standard function in MATLAB (the function names are `beta`, `gamma`, and `betainc` respectively), which reduces our programming burden. Nonetheless, it is still a significant exercise.

Now for the F-distribution, first the density function for a random variable $x$ with that distribution, with $\nu_1$, and $\nu_2$ degrees of freedom,

$$f(x;\nu_1,\nu_2) = \frac{\sqrt{\frac{(\nu_1 x)^{\nu_1}\nu_2^{\nu_2}}{(\nu_1 x+\nu_2)^{\nu_1+\nu_2}}}}{x\beta\left(\frac{\nu_1}{2},\frac{\nu_2}{2}\right)}$$

and the cdf is,

$$F(x;\nu_1,\nu_2) = \mathcal{I}_{\frac{\nu_1 x}{\nu_1 x+\nu_2}}\left(\frac{\nu_1}{2},\frac{\nu_2}{2}\right)$$

You exercise is to write programs so that your OLS program can use these functions to calculate the p-values.