First Introduction to MatLab

MatLab is a computational language, which is just like a foreign language: it can only be learned by practice. It is *much* easier than learning a foreign language since it uses components known from another foreign language that you've been learning: math – matrices and the operations defined on them - using (almost) exactly the same notation as in mathematical texts. MatLab = **Mat**rix **Lab**oratory.

1. **Basic organization**

MatLab is a functional language that uses two main objects:

Matrices (which is a two dimensional "array") that store numerical or logical values.
Higher order arrays can also be used.

Functions that store a set of operations to be performed on inputs to produce outputs

You will refer to matrices in a "general" way in a function, and
Functions can be written flexibly to manage any dimension of input.

(There are more complicated storage options, but you will learn about them as you need them. A notable one is a "structure" that is like a variable with several named "pockets" to store things in.)

2. **Large library of mathematical functions and operations**

Most of the matrix operations you have ever heard of and several besides are already coded into MatLab, as are convenient versions of most statistical concepts (e.g. distributions, densities etc.).

Use them; do not code them yourself unless you are doing it to learn.

There are several numerical approaches to inverting a matrix, some are several orders of magnitude faster and more robust than others, the usual things are already optimized.

The "basic" MatLab does not contain prepackaged versions of time series estimations. The recent "econometrics toolbox" (included in our license) has some functionality, but few pre-coded diagnostics. To estimate basic ARIMA or simple VAR models, R remains a more convenient option, although it has its own shortcomings relative to a corporate product lice Eviews.

3. Syntax

The following will become intuitive very quickly, but is useful to have as a reference when you're just starting out. In what follows below X may be a scalar, vector, matrix or a n-dimensional array of real or complex numbers. It may also be empty.

The most basic first:

> Use a single "equals to" to assign or change values
> (this is an assignment instruction you make to MatLab).
> e.g. `X=5` will create an object with name `X` that has value equal to the scalar `5`
>
> Use two "equals to" signs when performing a logical test
> (this is a question you are asking MatLab)
> e.g. `X==5` is a logical query. The answer will be the logical "True" if X is equal to 5, "False" otherwise.

To suppress output (usually in a long, many iteration code) end a statement with a semi-colon

```
X=5;
```

stores a matrix with name X and scalar value 5 in the memory without producing output. The mere requirement of printing characters on the screen slows down a program tremendously, so use this religiously.

To create a matrix with specific elements

```
X = [1 2 3; 4 5 6]
```

This assignment command creates a 2 x 3 matrix

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

– spaces or commas separate entries in the row, semi-colons separates rows

The indexing convention is the same as in math: $x_{ij} \in X$

> First index refers to row, second to column, third to "page" etc.
> If X is an m x n matrix, it has m rows and n columns.

To change/assign a value to a specific entry in a matrix, use its coordinates:

```
X(2,3)=1
```

If typed after the command above, this replaces the entry in row 2 and column 3 with 1.

If typed on its own (before an object named X exists) it creates a matrix with 2 rows and 3 columns, zeros everywhere except element (2,3) which is 1.

A colon used as an index is used to refer to a whole dimension:

`X(:,2)=0` changes the entries in all rows of the second column to zeros, i.e. refers to column 2 only.

`X(2,:)=0` changes the entries in all columns of the second row to zeros, i.e. refers to row 2 only.

The choice of parenthesis/bracket/brace matters: [is not (or {

Square brackets, curly braces and parentheses are used for distinct purposes and are NOT interchangeable.

Parentheses on the Left-Hand Side of equations are typically used used for coordinates. As in: `X(2,3)=1`

Parentheses on the Right-Hand Side of equations may enclose arguments of a function *or* the coordinates of a matrix.

If a function has more than one output, square brackets are used on the LHS to name them:

```
[m n]=size(X)
```

displays (and stores in variables named "m" and "n") the number of rows and columns in the array X.

Special Matrices have dedicated functions – use the help file to explore

`A=eye(5)` creates a 5 x 5 identity matrix
`B=zeros(1,5)` creates a 1 x 5 vector of zeros
`C=magic(5)` creates a 5 x 5 matrix with some interesting features you can explore on your own

Other useful generating functions:

`ones(n), nan(n), inf(n)`

`rand(n), randn(n)` – pseudo random number generators – there are functions for almost any distribution you can think of. I have not found a distribution that is not available in MatLab (although it may require the "Statistical Toolbox", which is included in the University licence)

Reshaping matrices is often necessary. It takes some care, but there are strong techniques for this.

Matrix notation applies roughly as you would expect:

`B=A'` creates a matrix B that is the transpose of A

`C=A*B` attempts to find the result of the matrix multiplication of A and B. Thus is only defined if the matrices are conformable: if A is [m x n] B must be [n x k]

`D=A^(-1)` or `D=inv(A)` creates a matrix D which is the inverse of A, if it exists.
Note: this is not the most efficient or accurate way to solve linear equations: use `x=A\b` rather than `x=inv(A)*b`.

`E=A^2` attempts to take the "matrix square" and is thus only defined for square matrices and is equal to A*A

`F=A.^2` operates on an **element-by-element** basis. I.e. it squares each element individually, so is defined for any size A. This "dot notation" goes through for all other simple operations (elementwise versions: multiplication ".*", powers ".^", division "./")

Infinity is a "value" in MatLab ("inf") and is distinct from numbers that are "not defined" which is stored as NaN (Not-a-Number).

Be careful: MatLab tends to store "missing values" if you import a dataset as "NaN" and does not remember this for you. Usually, any function of a NaN gives either an error or NaN as answer.

4. Graphing

MatLab has an incredibly useful and powerful graphics capability.

For example: look up "cone plot" in the help file.

And I will illustrate with a figure (and movie) that I created to practice my control of the graphical options (and to teach myself how to think in spherical coordinates).

A first exercise: OLS

To illustrate the notational convenience of MatLab, it is worthwhile writing your own code to estimate a basic regression.

1. *The true data generating process:*

For a simple regression, we will need to generate the data we want to use in the estimation.

Suppose the true data generating process is given by:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \epsilon_i$$

If we define an N x 3 matrix X:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{i,1} & x_{i,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} \end{bmatrix}$$

And vector:

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

We can write the true process as (with y and epsilon Nx1 vectors):

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

We will use the intuitive structure of MatLab's matrix operations to construct the test data.

2. *Generating a sample:*

Let us choose $\mathbf{x_1}, \mathbf{x_2}$ and $\boldsymbol{\epsilon}$ to be independent mean zero normal random variables.

You may choose the sample size N to be whatever you wish – it is worthwhile experimenting to see how larger samples lead to better estimates.

Use commands:

```
N=100

x1=randn(N,1)

x2=randn(N,1)

epsilon =randn(N,1)
```

Next, choose the true parameter vector (it has to be 3 x 1 to be conformable) and create the data matrix X

```
beta = [1;-0.5;3]
```

```
X = [ones(N,1),x1,x2]
```

Check the dimensions of the X matrix by typing:

```
size(X)
```

Last, generate the dependent variable:

```
y=X*beta + epsilon
```

3. *Estimating regression coefficients:*

Recall the definition of the OLS estimator in matrix form?

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

You can type it directly, except that you must explicitly type parentheses to collect the correct operations and the multiplication signs between any two arrays.

In matlab syntax (note: I use the “inferior” way of solving the linear equation just to illustrate the similarity between the syntax and the mathematical notation you are used to):

```
beta_hat = ((X'*X)^(-1))*X'*y
```