

# Dissertation Week Software Classes

## MATLAB

**Vilane G. Sales**

University of Birmingham

June, 2019

# Outline

In this course you will see:

- ▶ Section 1: Introduction to MATLAB
- ▶ Section 2: Working with Vectors, Matrices and Arrays
- ▶ Section 3: Decision and Loop Structures
- ▶ Section 4: Functions
- ▶ Section 5: Examples
- ▶ Section 6: Toolboxes

# Information

- ▶ MATLAB (Matrix Laboratory) numerical computational software
- ▶ The most programming intensive of all the main econometric softwares
- ▶ It has become what is known as 'computational economics' However, it is undoubtedly the most useful for students doing heavily empirical work
- ▶ Student version (29 +VAT) with no add-ins:

[www.mathworks.co.uk/academia/student\\_version/](http://www.mathworks.co.uk/academia/student_version/)

# Information

- ▶ Free alternatives to MATLAB:
  - ▶ Octave at <https://www.gnu.org/software/octave/>
  - ▶ Julia at <https://julialang.org/>
  - ▶ Scilab at <https://www.scilab.org/>
  - ▶ R at <https://www.r-project.org/>
- ▶ MATLAB (not Matlab or MatLab) originated as a MATrix orientated software in the 1970's. Contains over 1000 functions, with several 'toolboxes'
- ▶ MATLAB, just like econometrics, is still heavily matrix orientated - this can really help to learn various econometric techniques

# Information

- ▶ This course is based on Frain (2010), 'An Introduction to MATLAB for Econometrics'

<https://www.tcd.ie/Economics/assets/pdf/TEP0110.pdf>

- ▶ Mathworks have issued a new econometrics toolbox at:

<http://www.mathworks.com/products/econometrics/>

- ▶ LeSage (1999) is a free toolbox which we will utilize in this course:

<http://www.spatial-econometrics.com/>

- ▶ Many excellent textbooks available for science/engineering.  
In-built help: try 'lookfor inverse' and 'help inv'

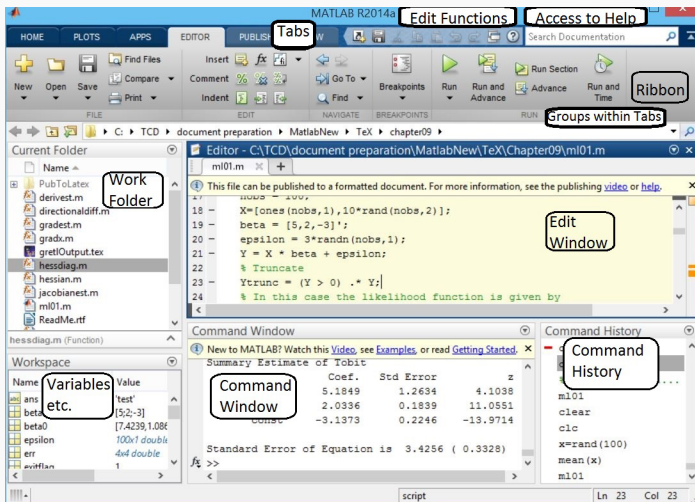
# Information

All the material used in this course is available at

<https://github.com/vgs549/MATLABCourse>

# Information

- Your MATLAB may look different! This is 2014a (student)







# Information

Some things to keep in mind (which will make debugging easier):

- ▶ `addpath`: Adds directory to places MATLAB looks for things
- ▶ `path`: Displays the current path
- ▶ `parh2rc`: Adds a path to the places MATLAB searches
- ▶ `rmpath`: Removes a directory from the search path
- ▶ `cd`: Changes the current directory
- ▶ `clc`: Clears the contents of the Command Window
- ▶ `clf`: Clears the contents of the Figure Window
- ▶ `exit` or `quit`: close MATLAB

# Information

cont...:

- ▶ `mkdir nameofdirectory`: `mkdir` creates a new directory
- ▶ `rmdir nameofdirectory`: `rmdir` removes the directory
- ▶ `Ctrl+C`: Breaks out of loops
- ▶ `Diary <filename>/Diary on/Diary off`: Creates a log of all your commands, similar to Stata
- ▶ `who/whos`: Displays a list of variables
- ▶ `disp(<variablename>)`: Displays the contents of a variable
- ▶ `docsearch('phrase')`: Searches the help browser for 'phrase'

# MATLAB as a Calculator

Lets open MATLAB and try some commands of our own. The simplest use of MATLAB is as a calculator:

$$2+2$$

Compare this to:

$$\begin{array}{r} 2+2 \dots \\ +2 \end{array}$$

Create an object to hold the result of our calculation:

$$a=2+2;$$

Now try:

$$\begin{array}{r} 2*2 + 4 \\ 2*2 + 4/2 \end{array}$$

# Syntax Notes

Some things to note here:

- ▶ `»` is the MATLAB command prompt
- ▶ `+`, `-`, `*`, `/` and have their usual meanings
- ▶ to fit multiple lines type `...`
- ▶ using `;` suppresses the process and output

You can recall commands with the up and down arrow keys.

# Our first .m file

To create our first .m file, we have two choices:

1. Open the MATLAB editor: New -> Script
2. Open up your favourite text editor

In either, type the same commands from the previous slide:

```
2+2  
2+2 ...  
+2  
a=2+2
```

Save/run ('run' in MATLAB', or double click your saved .m file in the file directory - your working directory). Compare results to your commands in the command window

## Our second .m file

Let's run a slightly more complex .m file to check out more features:

```
% Example adapted from Frain (2010)
echo off
r=3;
volume=(4/3)*pi*r^3;
string=['volume of a sphere of radius ' ...
num2str(r) ' is ' num2str(volume)];
disp(string)
```

Re-run this to check different values of **r**.

## Our second .m file (Cont...)

What features can you spot here that we haven't seen before so far?

- ▶ `echo off`
- ▶ `string`
- ▶ `'putting text inside apostrophes'`
- ▶ `num2str`
- ▶ `disp`

Save this in your default working directory (current directory) as `volume_of_a_sphere.m`, and then type `volume_of_a_sphere` into your command window.

# A note on m. files

In general, m. files will regularly adapt the same structure:

- ▶ Get/Process/Prepare the data
- ▶ Estimate some form of calculation through MATLAB functions
- ▶ Report the outputs in terms of graphs and tables
- ▶ Re-run the estimations changing parameters or specifications to check other solutions or for robustness



# Data input/output

The default data file in MATLAB is a .mat file.

- ▶ **save filename, var1, var2** will save var1 and var2 in 'filename.mat'
- ▶ **load filename, var1, var2** loads 'filename.mat'
- ▶ **Import Data** tool is the easiest way to load .csv/xls files (Try with data.xls - Output type: Column Vectors - Click in *Generate Script*)
  - ▶ Save your script as new\_load\_data.m
  - ▶ Load new\_load\_data
  - ▶ Save your results as new\_load\_data.mat

```
save('data','IVOLCA','IVOLCH','IVOLGB',... 'IVOLJP',  
      'IVOLNO','IVOLSE','IVOLUS','IVOLXM')
```
- ▶ We could also save a matrix as a .csv/xls file using **csvwrite**

# Vectors, Matrices and Arrays

The basic variable is an Array. Scalars are  $1 \times 1$ , column vectors as  $n \times 1$  and matrices as  $n \times m$ . Examples:

- ▶ A 1 by 4 matrix (row vector):  $x = [1 \ 2 \ 3 \ 4]$
- ▶ A 4 by 1 matrix (column vector):  $x = [1;2;3;4]$
- ▶ A 3 by 2 matrix

$$x = [1 \ 2; \ 3 \ 4; \ 5 \ 6]$$

- ▶ An empty array

$$x = []$$

- ▶ A matrix of ones

$$x = \text{ones}(2,3)$$

- ▶ A matrix of zeros

$$x = \text{zeros}(2,3)$$

- ▶ An identity matrix

$$x = \text{eye}(3,3)$$

- ▶ A random matrix

$$x = \text{rand}(2,3)$$

# Number Formats

First, let's define an arbitrary number:

$$x=82.8282828282828282$$

We have some commands which determine how  $x$  can be displayed:

**format loose %** adds in blank lines to space output

**format compact %** suppresses blank lines

**format long %** displays to 14 decimal places

**format short e %** exponential or scientific format

**format long e %** long exponential

**format short g %** short decimal format

**format bank %** currency format of 2 decimals

# Fprintf

The fprintf command prints the results to the command window in a certain way. For example, in the following, % indicates the start of a specification. There will be **6** digits displayed, where **2** are floating point decimals. The \n indicates the cursor then moves to a new line.

```
fprintf ('%6.2f\n',x)
```

# Basic Matrix Operation

Let's define three matrices:

$$\begin{aligned}x &= [1 \ 2; \ 3 \ 4] \\y &= [5 \ 6; \ 7 \ 8] \\z &= [9 \ 10; \ 11 \ 12; \ 13 \ 14]\end{aligned}$$

In Matlab:

- ▶ Addition  $a = x + y$
- ▶ Subtraction  $b = x - y$
- ▶ Multiplication  $c = x * y$
- ▶ Division  $d = x / y$
- ▶ **try**  $e = y * z$
- ▶ **try**  $f = x.*y$
- ▶ **try**  $g = x'$
- ▶ Indexing  $a(1,2)$

## Basic Matrix Operation (Cont...)

Let's define A:

$$A = [3 \ 4 \ -1; \ 1 \ 0 \ 6; \ -2 \ 9 \ -1]$$

In Matlab, compute:

- ▶ `numel(A)`
- ▶ `size(A)`
- ▶ `ndims(A)`
- ▶ `length(A)`
- ▶ `diag(A)`
- ▶ `triu(A)`
- ▶ `tril(A)`
- ▶ `sum(A)`

# Matrix Inversion

Let's find the inverse of matrix A:

$$A_{\text{inv}} = \text{inv}(A)$$

To verify the inverse, we check:

$$A_{\text{check}} = A * A_{\text{inv}}$$

Let's now show an example of how to use Kronecker product.  
Using the same matrices as before **y** and **z**

- ▶ **try** again  $e = y * z$
- ▶ **try**  $e = \text{kron}(y,z)$

Finally, let's see how matrices behave with vectors. As before, consider matrix  $x = [5 \ 6; 7 \ 8]$  and  $t = [2]$

- ▶ **try**  $h = x + t$
- ▶ **try**  $i = x / t$

# Sequences

The colon operator is the easiest way to make a sequence , with the general syntax of

**first:increment:last**

In Matlab, try:

- ▶  $j = [1:2:10]$
- ▶  $k = [10:20]$  and  $l = [10:1:20]$
- ▶  $m = [1:5; 2:2:10]$



# Decision and Loop Structures - Start Programming

- ▶ Boolean Expressions - True (1) or False (0)
  - ▶ Tests Symbols ( ==, ~=, <, > , <=, >=)
  - ▶ Operator Symbols (&, &&, |, ||)
  - ▶ Return Symbols (isempty, isequal, isnumeric, ischar)
- ▶ if - elseif - else Structures

**if** <bool.1 exp>

<commands.1>

**elseif** <bool.2 exp>

<commands.2>

**else**

<commands.3>

**end**

if x > 100

answer = 'X is bigger than 100'

elseif x < 10

answer = 'X is less than 10'

else

answer = 'X is between 10 and 100'

end

# Decision and Loop Structures - Start Programming (Cont...)

- For statements

```
for variable = expression  
    statements  
end
```

To see how it works, let's consider the following example:

```
Balance = 1000;  
for year = 1:30  
    BalanceVec(year) = (1.08)*Balance;  
    Balance = BalanceVec(year);  
end  
plot(1:30, BalanceVec)
```

# Basic Graphs

## Graph Example 1

```
% Load the data for x, y, and yfit
load fitdata x y yfit
% Create a scatter plot of the original x and y data figure
scatter(x, y, 'k')
% Plot yfit
line(x, yfit, 'Color', 'k', 'LineStyle', '-', 'LineWidth', 2)
% Plot upper and lower bounds, calculated as 0.3 from yfit
line(x, yfit + 0.3, 'Color', 'r', 'LineStyle', '-', 'LineWidth', 2)
line(x, yfit - 0.3, 'Color', 'r', 'LineStyle', '-', 'LineWidth', 2)
% Add axis labels
xlabel('X')
ylabel('Noisy')
```

# Basic Graphs

## Graph Example 2

```
% Load Morse data
load MDdata dissimilarities dist1 dist2 dist3
% Plot the first set of data in blue
figure
plot(dissimilarities, dist1, 'bo')
hold on
% Plot the second set of data in red
plot(dissimilarities, dist2, 'r+')
% Plot the third set of data in green
plot(dissimilarities, dist3, 'g^')
% Add title and axis labels
title('Morse Signal Analysis')
xlabel('Dissimilarities')
ylabel('Distances')
```

# Decision and Loop Structures - Start Programming (Cont...)

- While statements

```
while conditions  
    statements  
end
```

To see how it works, let's consider the following example:

```
balance = 50;  
year = 0;  
while balance < 100  
    balance = (1.09)*balance;  
    year = year+1;  
end  
year, balance
```

# User Define Functions

- ▶ MATLAB gives you the possibility to write your own functions
- ▶ They can be used just like built-in functions
- ▶ Until this moment, we've used *script* files
- ▶ **NOTE:** The name of the file must match the name of the first function in the file
- ▶ From Mathworks:

`function[y1,...,yN] = myfun(x1,...,xM)`

declares the function called **myfun** that accepts as inputs `x1,...,xM` and returns outputs as `y1,...,yN`

- ▶ **NOTE:** Outputs are in `[]`, and inputs are in `()`

# User Define Functions

Let's write a function which estimates the density function of a normal distribution. Recall:

$$\frac{1}{\sqrt{2\pi} \sigma} \exp - \frac{(x - \mu)^2}{2\sigma^2}$$

How many inputs do we have in this equation?

# User Define Functions

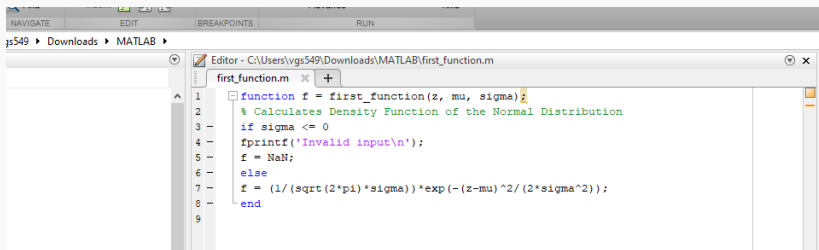
Just as with scripts, click 'New -> Function' or save as **first\_function.m** also on Canvas

```
function f = first_function(z, mu, sigma);  
% Calculates Density Function of the Normal Distribution  
if sigma <= 0  
    fprintf('Invalid input\n');  
    f = NaN;  
else  
    f = (1/(sqrt(2*pi)*sigma))*exp(-(z-mu)^2/(2*sigma^2));  
end
```



# User Define Functions

Generally, a function looks something like this



The image shows a screenshot of the MATLAB Editor interface. The top toolbar includes buttons for NAVIGATE, EDIT, BREAKPOINTS, and RUN. The breadcrumb path at the top reads 'js549 > Downloads > MATLAB >'. The editor window title is 'Editor - C:\Users\vgs549\Downloads\MATLAB\first\_function.m'. The file name in the tab is 'first\_function.m'. The code in the editor is as follows:

```
1 function f = first_function(z, mu, sigma);
2 % Calculates Density Function of the Normal Distribution
3 if sigma <= 0
4     fprintf('Invalid input\n');
5     f = NaN;
6 else
7     f = (1/(sqrt(2*pi)*sigma))*exp(-(z-mu)^2/(2*sigma^2));
8 end
9
```

# User Define Functions

- ▶ We can get help on this function by typing

```
help first_function
```

- ▶ We can evaluate our function at a point (e.g.zero)

```
first_function(0,0,1)
```

- ▶ And finally, we can plot our function on the interval [-5,5]

```
fplot(first_function(1,0,1), [-3,3])
```

# Examples - Grunfeld

Following Green(2000), we will use Grunfeld investment data set to run a POLS regression.

$$y_{ti} = \beta_i x_{ti} + \epsilon_{ti} \quad (1)$$

Where,

- ▶  $i$  represents the individual agent or country or item for which we are estimating some equation and  $t$  represents the  $t^{th}$  measurement on the  $i^{th}$  unit
- ▶ variance of  $\epsilon_t$  is constant
- ▶  $X_i$  is exogenous for all  $1 \leq i \leq M$ .

## Example Grunfeld (Cont...)

The variables included in the Grunfeld analysis are

- ▶ FIRM : There are 10 firms
- ▶ YEAR : Data are from 1935 to 1954 (20 years)
- ▶ I : Gross Investment
- ▶ F : Value of Firm
- ▶ C : Stock of Plant and Equipment

To reduce the amount of detail we shall restrict analysis to 5 firms

- ▶ Firm no 1 : GM - General Motors
- ▶ Firm no 4 : GE - General Electric
- ▶ Firm no 3 : CH - Chrysler
- ▶ Firm no 8 : WE - Westinghouse
- ▶ Firm no 2 : US - US Steel

## Example Grunfeld (Cont...)

### Load and Generate Data

```
data = xlsread('Grunfeld.xlsx','A2:E201');  
YGM = data(1:20, 3); % I  
XGM = [ones(20,1),data(1:20,[4 5])]; % constant F C  
YGE = data(61:80, 3); % I  
XGE = [ones(20,1),data(61:80,[4 5])]; % constant F C  
YCH = data(41:60, 3); % I  
XCH = [ones(20,1),data(41:60,[4 5])]; % constant F C  
YWE = data(141:160, 3); % I  
XWE = [ones(20,1),data(141:160,[4 5])]; % constant F C  
YUS = data(21:40, 3); % I  
XUS = [ones(20,1),data(21:40,[4 5])]; % constant F C
```

## Example Grunfeld (Cont...)

We now estimate the coefficients imposing various restrictions. Each estimation involves the following steps:

1. Set up the required stacked  $y$  and  $X$  matrices.
2. Estimate the required coefficients.
3. Estimate standard errors, t-statistics etc.
4. Report.

The restrictions imposed by Pooled OLS are that corresponding coefficients are the same across equations. We also assume that the variance of the disturbances is constant across equations.

## Example Grunfeld (Cont...)

We may write the system as follows

$$y = x\beta + \epsilon$$

And  $\beta$  can be estimated as

$$\hat{\beta} = (X'X)^{-1}X'y$$

This is implemented in MATLAB as follows

### POLS Regression

```
Y = [YGM', YGE', YCH', YWE', YUS']'  
X = [XGM', XGE', XCH', XWE', XUS']'  
pols.beta = (X'*X)\X'*Y;  
pols.uhat = Y - X*pols.beta ;  
pols.sigsq = (pols.uhat'*pols.uhat)/(size(X,1)-size(X,2));  
pols.sdbeta = sqrt(diag(inv(X'*X))*pols.sigsq);  
pols.tbeta = pols.beta ./ pols.sdbeta;  
pols.se = sqrt(pols.sigsq);
```

## Example Grunfeld (Cont...)

### POLS Regression (Cont...)

```
label = ['Constant '; 'F '; 'C '];  
disp('OLS Results using stacked matrices')  
disp('coef sd t-stat')  
  
for ii=1:size(X,2)  
    fprintf('%s%10.4f%10.4f%10.4f\n',label(ii,:),...  
        pols.beta(ii),...  
        pols.sdbeta(ii), ...  
        pols.tbeta(ii))  
end  
  
fprintf('Estimated Standard Error %10.4f',pols.se)
```



## Example - Macroeconomic Simulation

This example is based on the macroeconomic system in Example 10.3 of Shone (2002). There are 10 equations in this economic model. The equations of the system are as follows

$$c_t = 110 + 0.75yd_t$$

$$yd_t = y_t - tax_t$$

$$tax_t = -80 + 0.2y_t$$

$$i_t = -4r_t$$

$$g_t = 330$$

$$e_t = c_t + i_t + g_t$$

$$y_t = e_{t-1}$$

$$md_t = 20 + 0.25y_t - 10r_t$$

$$ms_t = 470$$

$$md_t = ms_t$$

## Example - Macroeconomic Simulation (Cont...)

### Initialise and Describe Variables

```
N = 15 ; % Number of periods for simulation
c = NaN * zeros(N,1); %real consumption
tax = NaN * zeros(N,1); %real tax
yd = NaN * zeros(N,1); %real disposable income
i = NaN * zeros(N,1); % real investment
g = NaN * zeros(N,1); % real government expenditure
e = NaN * zeros(N,1); % real expenditure
y = NaN * zeros(N,1); % real income
md = NaN * zeros(N,1); % real money demand
ms = NaN * zeros(N,1); %real money supply
r = NaN * zeros(N,1); % interest rate
t=(1:N)'; % time variable
g = 330 * ones(N,1); % policy variable I
ms = 470 * ones(N,1); % policy variable II
y(1) = 2000;
```

## Example - Macroeconomic Simulation (Cont...)

The next step is to simulate the model over the required period. Here this is achieved by a simple reordering of the equations and inverting the money demand equation to give an interest rate equation. Note that the loop stops one short of the full period and then does the calculations for the final period.

### Simulation

```
for ii = 1:(N-1)
tax(ii) = -80 + 0.2 * y(ii);
yd(ii) = y(ii) - tax(ii);
c(ii) = 110 + 0.75 * yd(ii);
md(ii) = ms(ii);
r(ii) = (20 + 0.25* y(ii) -md(ii))/10; % inverting money demand
i(ii) = 320 -4 * r(ii);
e(ii) = c(ii) + i(ii) + g(ii);
y(ii+1) = e(ii);
end
```

## Example - Macroeconomic Simulation (Cont...)

### Simulation (Cont...)

```
tax(N) = -80 + 0.2 * y(N);  
yd(N) = y(N) - tax(N);  
c(N) = 110 + 0.75 * yd(N);  
md(N) = ms(N);  
r(N) = (20 + 0.25* y(N) -md(N))/10;  
i(N) = 320 -4 * r(N);  
e(N) = c(N) + i(N) + g(N);
```

### Output

```
base = [t,y,yd,c,g-tax,i,r];  
fprintf(' t y yd c g-tax i r\n')  
fprintf('%7.0f%7.0f%7.0f%7.0f%7.0f%7.0f%7.0f \n',base')
```

# LeSage Toolbox

- ▶ A 'toolbox' is a related set of MATLAB functions aimed at solving a particular class of problems
- ▶ Toolboxes of functions are useful in signal processing, optimisation, statistics, finance and many other areas available from MathWorks
- ▶ Most popular for econometrics is written by James LeSage:  
[www.spatial-econometrics.com](http://www.spatial-econometrics.com)
- ▶ It is split (handily) into libraries: Regression, Utility Functions, Diagnostics, VAR/VECMs, MCMCMs, Limited Dependents, SEMs, Distributions, Optimizations, etc

# LeSage Toolbox - Installing

- ▶ To install the toolbox, a specific library, or a function, you need to
  1. Download the toolbox
  2. if it's zipped, extract it: create a folder in your documents folder like **econ**
  3. Go to the 'Documents folder ' in the MATLAB window, right click on **econ**, and

Add to path -> Select Folders and Subfolders

## LeSage Toolbox - REGRESS

The first thing which we are going to do is use the REGRESS library. Let's begin with the previous example of Grunfeld dataset.

### Load and Generate Data

```
data = xlsread('Grunfeld.xlsx','A2:E201');  
YGM = data(1:20, 3); % I  
XGM = [ones(20,1),data(1:20,[4 5])]; % constant F C  
YGE = data(61:80, 3); % I  
XGE = [ones(20,1),data(61:80,[4 5])]; % constant F C  
YCH = data(41:60, 3); % I  
XCH = [ones(20,1),data(41:60,[4 5])]; % constant F C  
YWE = data(141:160, 3); % I  
XWE = [ones(20,1),data(141:160,[4 5])]; % constant F C  
YUS = data(21:40, 3); % I  
XUS = [ones(20,1),data(21:40,[4 5])]; % constant F C
```

# LeSage Toolbox - REGRESS

## POLS Regression

```
Y = [YGM', YGE', YCH', YWE', YUS']'  
X = [XGM', XGE', XCH', XWE', XUS']'  
pooled = ols(Y, X);  
vnames= ['I ' ; 'Constant ' ; 'F ' ; 'C '];  
prt(pooled,vnames)
```



## LeSage Toolbox - GRAPHS

Now, let's try a couple of examples from the Graphs library. In your MATLAB command window, write:

### Graphs

```
cstr = cal(2000,1,12) % creates a time-series calendar  
tsplot(X,cstr) % tsplot creates a time-series graph
```

As a second and final example of plotting from the Graphs library, use:

### Graphs (Cont...)

```
pltdens(X)
```

What pltdens does? Check for yourself **help pltdens**

# Matlab Econometrics Package

This toolbox is of interest to those working in the financial econometrics and computational finance. It should be noted that the econometrics toolbox requires that the statistics, optimization and financial toolboxes be also installed.

<http://www.mathworks.co.uk/help/econ/product-description.html>

# Oxford MFE Toolbox

This toolbox covers many time-series and other econometric methods used in financial econometrics and quantitative finance. It also contains some relevant functions similar to those in the MATLAB statistics and financial toolboxes that can be substituted if the MATLAB versions are not available.

[http://www.kevinsheppard.com/MFE\\_Toolbox](http://www.kevinsheppard.com/MFE_Toolbox)

## By yourself

Try with 'NationalIncomeReal.csv'

- ▶ Load csv data
- ▶ Create variable Year
- ▶ Create variable NI
- ▶ Create figure

Draw a graph of the residuals against the trend in the ols-simulation exercise below, put title and labels.

### By yourself

```
nsimul=50;
beta=[5,1,.1]';
x1=ones(nsimul,1); % constant
x2=[1:nsimul]'; % trend
x3=rand(nsimul,1)*2 +3; % Uniform(3,5)
x=[x1,x2,x3];
e=randn(nsimul,1)*.2; % N(0,.04)
y= x * beta +e ; %5*x1 + x2 + .1*x3 + e; [nobs,nvar] = size(x);
betahat=inv(x'*x)*x'*y;
yhat = x * betahat;
beta(1)*x1-beta(2)*x2-beta(3)*x;
resid = y - yhat;
```

# The end...for now!

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

- Martin Fowler