A Guide to Using MATLAB for your MSc Dissertation

Charles Rahal

Slides Available: http://github.com/crahal

`Last Updated: 21st June, 2016`

Course Admin - MATLAB

- Student version relatively cheap (£28 +VAT):

  `www.mathworks.co.uk/academia/student_version/`

- Octave is a free alternative to MATLAB:

  `https://www.gnu.org/software/octave/download.html`

- A program written for MATLAB will run in Octave with at most minor changes and, in all likelihood with none.
- Programs written for Octave may not run in MATLAB as some of the functions may require functions from packages which in MATLAB which require additional purchase.
- Creel (2008) is a set of econometrics notes based with applications in Octave.
- For computational routines, it may be *marginally* less efficient.

Course Admin - MATLAB

- MATLAB originated as a MATrix orientated software in the 1970's.
- Now contains over 1000 functions, with various 'toolboxes'.
- We will learn to create some basic functions in this workshop.
- MATLAB, just like econometrics, is still heavily matrix orientated - this can really help to learn various econometric techniques.
- A 'black box' package may be easier: but how do you know what it's doing?
- Typing `ver` into the Command Window tells us what toolboxes are installed.

Outline of this Session

By the end of this session, you should be able to...:

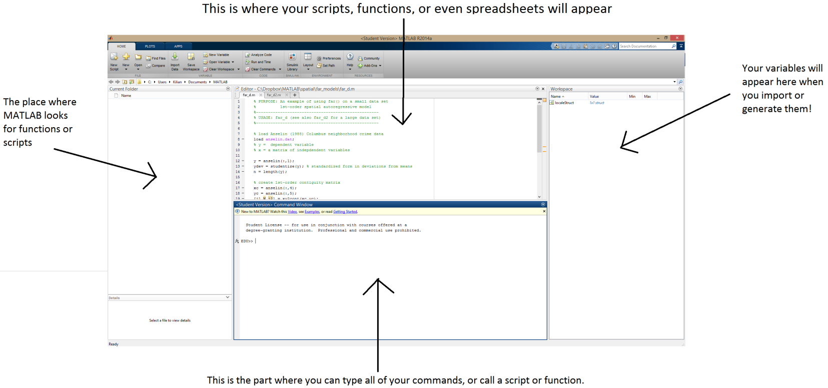
- Section One: Introduction to MATLAB.
- Section Two: Vectors, Matrices and Arrays.
- Section Three: An Example Using OLS.
- Section Four: Decision and Loop Structures.
- Section Five: User Defined Functions.
- Section Six: Applications of the LeSage Toolbox.

Course Admin - MATLAB Resources

- As a guide: Frain (2010), 'An Introduction to MATLAB for Econometrics': `https://www.tcd.ie/Economics/assets/pdf/TEP0110.pdf`
- A free ARCH/GARCH toolbox is available at: `http://http://www.kevinsheppard.com/wiki/MFE_Toolbox`
- Mathworks have issued a new econometrics toolbox at: `http://www.mathworks.com/products/econometrics/`
- LeSage (1999) is a free toolbox which we will utilize heavily: `http://www.spatial-econometrics.com/`
- Many excellent textbooks available for science/engineering.
- In-built help: try '`lookfor inverse`' and '`help inv`'.

MATLAB Windows

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



Preliminary MATLAB Commands

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.

Preliminary MATLAB Commands (Cont.)

Further useful commands to try and keep in mind:

- `mkdir newdir`: Like GitBash, mkdir creates a new 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'.
- `helpbrowser`: Opens the help browser

MATLAB as a Calculator

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

```
EDU»2+2
```

Compare this to:

```
EDU»2+2 ...
    +2
```

Create an object to hold the result of our calculation:

```
EDU»a=2+2;
```

Some Syntax Notes

Some things to note here:

- » is the MATLAB command prompt.
- EDU denotes an educational copy (it may not on UoB terminals).
- $+$, -, *, / and ^ have their usual meanings.
- To fit over multiple lines type ....
- Note that the use of ; suppresses 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 favorite text editor (e.g. Notepad++).

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).
- Compare results to your commands in the command window.

Our Second .m File

Lets 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)];
forecast modelyesseasonal_f
disp(string)
```

Then you can re-run this to check different values of r.

Our Second .m File (Cont.)

What features can you spot here that we havent seen before so far?

- `% Comments are in green and dont affect the code`
- `echo off`
- `*`
- `string`
- `‘putting text inside apostrophes’`
- `num2str`
- `disp`
- Note the colours if you’re using the MATLAB editor.

Save this in your default working 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, note that .m files will regularly adapt the same structure (as on the last slide):

1. Get/Process/Prepare the data.
2. Estimate some form of calculation e.g. $\hat{\beta} = (X'X)^{-1}X'y$ through some MATLAB functions.
3. Report the outputs in terms of graphs and tables.
4. Re-run the estimations changing parameters or specifications to check 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’.
- Easiest way to load .csv/xls files - the ‘Import Data’ tool.
- Try it with the data.xls. Try the ‘Generate Script’ function.
- If you cant find it - run load_data.m and save it as a .mat:

```
save('datamat','IVOLCA', 'IVOLCH', 'IVOLGB', ...
'IVOLJP', 'IVOLNO', 'IVOLSE', 'IVOLUS', 'IVOLXM')
```

- We could also save a matrix as a csv/xls file e.g. `M=[ivolca,ivolgb]` and then `csvwrite('filename',M)`

Vectors, Matrices and Arrays

The basic variable is an Array. Scalars are 1×1, 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 2 by 3 matrix:

```
x=[1,2,3;4,5,6]
```

- An empty array:

```
x=[]
```

A Quick Word on Number Formats

- First lets define an arbitrary number:

```
x=82.828282828282828282
```

- Some commands which determine how this is 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
```

A Quick Word on 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 six digits displayed, where 4 are floating point decimals.
- the \n indicates the cursor then moves to a new line.

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

Basic Matrix Operations

- Let's define two matrices and play with them:

```
x = [1,2;3,4]
y=[3,7;5,4]
```

- Addition:

```
a=x+y
```

- Subtraction:

```
b=x-y
```

- Multiplication:

```
c=x*y
```

- Remember that matrices must 'conform'!

Matrix Inversion

- Lets find the inverse of matrix `x=[1,2;3,4]`:

```
a=inv(x)
```

- To verify the inverse:

```
b=x*inv(x)
```

- Multiply y by the inverse of x:

```
c = y*inv(x)
```
or
```
d=y/x
```

- Pre-multiply y by the inverse of x:

```
e=inv(x)*y
```
or
```
f=x\y
```

Kronecker Product

- Lets now show an example of how to use Kronecker products.
- Define a matrix as before:

```
x=[1,2;3,4]
```

- Then define the always useful identity matrix:

```
I=eye(2,2)
```

- To use this operator:

```
a=Kron(x,I)
```

Element by Element

- As before, define `x=[1,2;3,4]` and `y=[5,6;7,8]`.
- To do element by element multiplication:

```
a=x.*y
```

- For element by element division:

```
b=y./x
```

- If you try and divide by zero, you will get a warning!
- Define a scalar (e.g. `z`=2) then try:

```
c=x+z
d=x-z
e=x*z
f=x/z
g=x^g
```

- `log`, `sqrt`, and `exp` are all element by element.

Other Matrix Commands

- `det(x)`, `rank(x)` and `trace(x)` work as expected.
- `diag(x)` where X is a matrix puts diagonal of X in a vector.
- `diag(x)` where X is a vector outputs a matrix with a diagonal of X and zeros elsewhere.
- `sum(x)` returns the sum of all elements of a vector, or a row vector of the column sums of a matrix.
- The function `reshape(A,m,n)` returns the m×n matrix B whose elements are taken column wise from A. This is more complicated, so let's see an example:

```
x=[1,2,3;4,5,6]
reshape(x,3,2)
```

Other Matrix Commands (Cont.)

The command `blkdiag(A,B.C)` constructs a block diagonal from matrices. Try:

```
a=[1,2;3,4]
b=[5]
c=[6,7;8,9]
d=blkdiag(a,b.c)
```

In order to produce a diagonal matrix D of generalized eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that A*V = B*V*D:

```
[V,D]=eig(A)
```

Sequences

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

```
first:increment:last
```

- Try:

```
a=[1:3:10]
```

- Or to make simple row or column vectors:

```
b=[1:4]'
```

- Note that ′ creates a transpose of a vector or a matrix.

Special Matrices

- For example, an identity matrix:

```
a=eye(4)
```

- Or a matrix of ones:

```
b=ones(4,2)
```

- Or a matrix of zeros:

```
c=zeros(3)
```

- To return the size of a matrix (rows, columns):

```
[nr,nc]=size(x)
```

- We can store random numbers in matrices, vectors or scalars using something like `d=rand(5)` for random uniform or `e=randn(5)` for draws from a standard normal distribution.

Matrix Manipulation

- We can isolate individual elements e.g.:

```
x=[1:1:5]
y=x(3)
```

- Or entire elements using the colon operator e.g.:

```
x=[1,2;3,4]
y=x(2,:)
```

- We can also use sub-matrices on the left:

```
x=[1,2,3,4;5,6,7,8,9;10,11,12,13]
x(1:2,[1,4]=[100,99;98,97]
```

- We can also make some block assignments:

```
x(1:2,1:4)=1
```

- We can also stack: `x=[1,2;3,4]` and `y=[5,6;7,8]`:

```
z=[x,y]
```

Regression Example

Following Frain (2010), we can now run a simulated OLS:

$$y_t = \beta_1 + \beta_2 x_{2,t} + \beta_3 x_{3,t} + \varepsilon_t \tag{1}$$

- $x_{2,t}$ is a trend variable with values (1,2,...,30).
- $x_{3,t}$ is uniformly distributed on [3,5].
- ε_t are idd normal random - zero mean, constant variance.
- β_1=5, $\beta_2 = 1$, $\beta_3 = 0.1$.
- ε_t are iid(0,0.4)
- We will attempt to estimate a large number of regression outputs and compare them with EViews to ensure consistency.

Regression Example - Prepare and Process

- First lets define the size of our variables:

```
nsimul=50
```

- And our vector of coefficients:

```
beta=[5,1,0.1]'
```

- Then lets generate our three variables:

```
x1=ones(nsimul,1) % an array of constants
x2=[1:nsimul]' % a trend
x3=rand(nsimul,1)*2+3% Uniform on [3,5]
```

- And finally some other equations:

```
e=randn(nsimul,1)*0.2
X=[x1,x2,x3]
y=X*beta+e
[nobs,nvar]=size(X)
```

Regression Example - Understanding

Let's check everyone is up to speed on what's happening:

1. Why is x_3 is defined like this?
2. Why is the residual (e) defined like this?
3. Note how we stacked variables into a matrix.
4. Note that all of our answers will of course be different!

Regression Example - Estimating the Model

- Introducing structures:

```
ols.betahat=(X'*X)\X'*y
ols.yhat=X*ols.betahat
ols.resid=y-ols.yhat
ols.ssr=ols.resid'*ols.resid
ols.sigmasq = ols.ssr/(nobs-nvar)
ols.covbeta=ols.sigmasq*inv(X'*X)
ols.sdbeta=sqrt(diag(ols.covbeta))
ols.tbeta=ols.betahat./ols.sdbeta
```

- Structures can be thought of as branches, or subfolders.
- Useful if you want to try different specifications, or estimators

Regression Example - Evaluating the Model

- Lets now evaluate our model using some common metrics

```
ym=y-mean(y)
ols.rsqr1=ols.ssr
ols.rsqr2=ym'*ym
ols.rsqr=1.0-ols.rsqr1/ols.rsqr2
ols.rsrq1=ols.rsqr1/(nobs-nvar)
ols.rsqr2=ols.rsqr2/(nobs-1)
ols.rbar=1-(ols.rsqr1/ols.rsqr2)
ols.ediff=ols.resid(2:nobs)-ols.resid(1:nobs-1)
ols.dw=(ols.ediff'*ols.ediff)/ols.ssr
```

- Now we have estimated everything we will need to proceed.

Regression Example - Evaluation Outputs

- Lets now get MATLAB to present the results to us.
- First, lets focus on the evaluation outputs using fprintf:

```
fprintf('R-squared = %9.4f \n', ols.rsqr);
fprintf('Rbar-squared = %9.4f \n ', ols.rbar);
fprintf('sigma^2=%9.4f \n ',ols.sigmasq);
fprintf('S.E.=%9.4f \n ',sqrt(ols.sigmasq));
fprintf('Durbin-Watson = %9.4f \n ' ,ols.dw);
fprintf('Nobs, Nvars =%6d, %6d \n ' ,nobs,nvar);
```

- To get a rudimentary estimation output, type something like:

```
coeffs=[ols.betahat ols.sdbeta ols.tbeta]
```

Regression Example - Coefficient Output

- Now lets produce the formatted output::

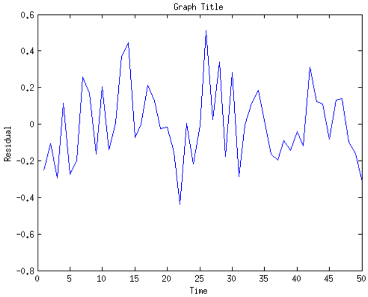
```
fprintf('%12s %12s %12s %12s %12s \n', ...
'Variable','Coefficient','Standard Error','t-statistic')

fprintf('%12s %12.6f %12.6f %12.6f \n','Const', ...
ols.betahat(1), ols.sdbeta(1), ols.tbeta(1))

fprintf('%12s %12.6f %12.6f %12.6f \n', 'Trend',...
ols.betahat(2), ols.sdbeta(2), ols.tbeta(2))

fprintf('%12s %12.6f %12.6f %12.6f \n', 'Var2',...
ols.betahat(3), ols.sdbeta(3), ols.tbeta(3))
```

- A fairly standard regression output as per other softwares.
- Note the results change every time we run the .m file (why?).

Introducing Plots

- This might be a nice time to introduce plots:

```
plot(x2,ols.resid)
title('Graph Title')
xlabel('Time')
ylabel('Residual')
```



Regression Example - Output

- Your command window should now look something like this:

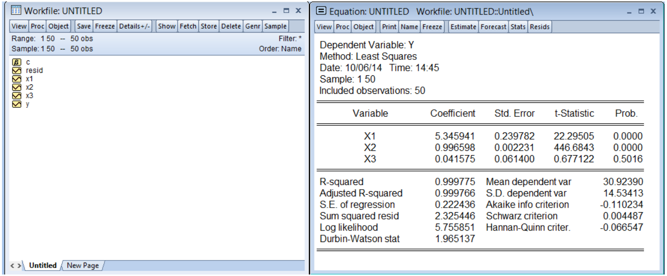
```
R-squared =    0.9998
Rbar-squared =    0.9890
 sigma^2=   0.0495
 S.E.=   0.2224
 Durbin-Watson =    1.9651
 Nobs, Nvars =    50,      3
     Variable  Coefficient Standard Error  t-statistic
        Cons     5.345941     0.239782    22.295047
       Trend     0.996598     0.002231   446.684305
        Var2     0.041575     0.061400     0.677122
EDU>>
```

- (Your results will be different as we're using simulated data!)

Regression Example - Checking Results

- Copy your variables x1, x2 and x3 along with your dependent - y into EViews (undated, 50 obs).

- Then run: `equation eqcheck.ls y1 x1 x2 x3`



Decision and Loop Structures

- **if statements** take the general form of:

```
if condition
  statement
end
```

- Conditions can include the following operators:

$$==, \sim=, <, >, >=, <=, \&, \&\&, |, ||, xor, all, any$$

- We can then extend this to `else` and `elseif`:

```
if conditions1
   statements1
elseif conditions2
   statements2
else
   statements3
end
```

Decision and Loop Structures (Cont.)

- The next 'flow of control' structure we will consider is a **for** loop:

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

- To see how it works, consider the following example:

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

Decision and Loop Structures (Cont.)

- The general format of a **while** statement is similar:

```
while conditions
   statements
end
```

- Lets see an example below:

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

User Defined Functions

- MATLAB gvies you the ability to write your own functions.
- They can then be used just like native functions.
- Up until now, we've used what are called 'script' files.
- Note: The namze of the file should match the name of the first function in the file.
- From MathWorks:

  function[y1,...,yN] = myfun(x1,...,xM) declares a function named myfun that accepts inputs x1,...,xM and returns outputs y1,...,yN.

- Note that the outputs are in [ ], and inputs are in ( ).

General Function Structure

Generally, a function looks something like this.

Editor - C:\Dropbox\Teaching\introductiontoeconometricsoftware2014\MATLAB\first_function.m

filterto2011.m first_function.m

```
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
```

Our First User Defined Function

- Now we'll write a function which estimates the density function of a normal distribution. Recall:

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

- Note that the three inputs to our function are going to be x, μ, and σ.
- After evaluating it for specific values, we are going to plot over an interval.

A Function for the Density of Normal Distribution

- 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');
if sigma= NaN;
else
   f = (1/(sqrt(2*pi)*sigma))*exp(-(z-mu)^2/(2*sigma^2));
end
```

Density of Normal Distribution (Cont.)

- We can then get help on this function from typing:

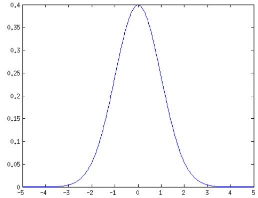
```
help first_function
```

- 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('normaldensity(x,0,1)', [-5,5])
```



LeSage Toolbox

- A 'toolbox' is a related set of MATLAB functions aimed at solving a particular class of problems.
- Toolboxes of functions useful in signal processing, optimization, statistics, finance/a host of other areas available from MathWorks.
- Most popular for econometrics is written by James LeSage:

  `www.spatial-econometrics.com`

- It is split (handily) into libraries: Regression, Utility Functions, Diagnostics, VAR/VECMs, MCMCMs, Limited Dependents, SEMs, Distributions, Optimizations, etc.
- Of particular interest is the *Spatial Econometrics* Library.

Installing Toolboxes or Functions

- To install the toolbox, a specific library, or a function:

1. Download the toolbox.
2. If it's zipped, extract it: at home, somewhere like:

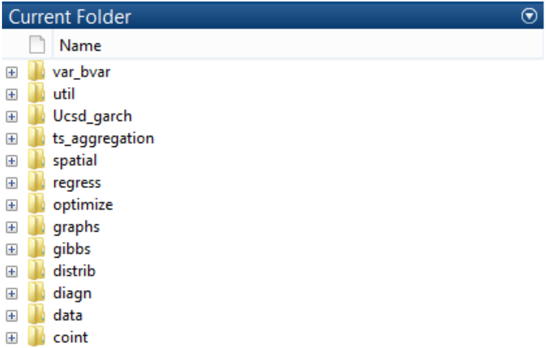
   `C:\Program Files\MATLAB\R2014a\toolbox`

3. Add a new folder, call it 'econ'.
4. Or on the University computers, you will have to extract to somewhere like 'econ' in your documents.
5. Just as before, you will have to navigate to it in the MATLAB window 'current folder' window, right click, and:

   `Add to path → Selected Folders and Subfolders`

LeSage Toolbox

If everything has gone right, you should have a list of the librarys within the Toolbox added to the Current Folder:



REGRESS

- The first thing which we are going to do is use the REGRESS library .
- Then, we'll compare our results with 'first_regression' which we wrote earlier.
- After running this file, we will use the y and X objects.
- After running this file, be sure to delete or rename the 'ols' structure, or MATLAB will get confused, giving you error:

  ```
  Subscript indices must either be real positive integers or logicals.
  ```

- This is because our regression *function* in the LeSage Toolbox is called 'ols' also.

Checking OLS and first_regression

- To use the LeSage function: `results=ols(y,X)`.

Variables - ols1

1x1 struct with 15 fields

Field	Value	Min	Max
betahat	[4.9853;1.0019;0.0925]	0.0925	4.9853
yhat	*50x1 double*	6.3516	55.4448
resid	*50x1 double*	-0.3776	0.3712
ssr	1.2405	1.2405	1.2405
sigmasq	0.0264	0.0264	0.0264
covbeta	[0.0308,-6.7639e-05,-...	-0.0072	0.0308
sdbeta	[0.1756;0.0016;0.0423]	0.0016	0.1756
tbeta	[28.3872;629.2864;2.1...	2.1848	629.28...
rsqr1	1.2405	1.2405	1.2405
rsqr2	213.3338	213.33...	213.33...
rsqr	0.9999	0.9999	0.9999
rsrq1	0.0264	0.0264	0.0264
rbar	0.9942	0.9942	0.9942
ediff	*49x1 double*	-0.5721	0.4847
dw	2.2986	2.2986	2.2986

Variables - results

1x1 struct with 14 fields

Field	Value	Min	Max
meth	'ols'		
y	*50x1 double*	6.3877	55.2708
nobs	50	50	50
nvar	3	3	3
beta	[4.9853;1.0019;0.0925]	0.0925	4.9853
yhat	*50x1 double*	6.3516	55.4448
resid	*50x1 double*	-0.3776	0.3712
sige	0.0264	0.0264	0.0264
bstd	[0.1756;0.0016;0.0423]	0.0016	0.1756
bint	[4.6325,5.3380;0.9987,...	0.0075	5.3380
tstat	[28.3872;629.2864;2.1...	2.1848	629.28...
rsqr	0.9999	0.9999	0.9999
rbar	0.9999	0.9999	0.9999
dw	2.2986	2.2986	2.2986

- Remember we are generating random variables each time.
- Name our structure anything (e.g. `olsresults=ols(y,X)`).

Application: Spatial Econometrics

- We may turn to the LeSage toolbox for spatial econometrics.
- Spatial econometrics is necessary because of spatial dependence or spatial heterogeneity.
- First of all, we must quantify some locational aspects of our data (i.e. address).
- From this, we obtain their latitude and longtitude.
- We can calculate some kind of weighting matrix.
- 'Spatial' routines need other functions ('Utilities').
- See the following guide to the Spatial Library:

  http://www.spatial-econometrics.com/html/wbook.pdf

Application: Spatial Econometrics (Cont.)

- The most simple example to consider is the FAR model:

$$y = \rho W y + \varepsilon \tag{3}$$

$$\varepsilon \sim N(0, \sigma^2 I_n) \tag{4}$$

- The least squares estimator would yield:

$$\hat{\rho} = (y'W'Wy)^{-1} y'W'y \tag{5}$$

- However, lets show that this estimator is unbiased:

$$E(\hat{\rho}) = (y'W'Wy)^{-1} y'W'(\rho W y + \varepsilon) \tag{6}$$

$$E(\hat{\rho}) = \rho + (y'W'Wy)^{-1} y'W'\varepsilon \tag{7}$$

- Anselin (1988) also establishes inconsistency.

Application: Spatial Econometrics (Cont.)

- Find the **far** and **far_d** function/script in the Spatial Library.
- This uses the dataset of Anselin (1988): Columbus neighborhood crime data.
- The functions use maximum likelihood estimators and sparse matrices.
- Sparse matrices are necessary otherwise inverting the large matrices would be extremely difficult.
- Many W matrices in the literature have many zero elements (e.g. 'nearest neighbor' or contiguity).
- If a problem has < 500 obs, it computes the theoretical information matrix, if > 500, computes a numerical Hessian.

Application: Spatial Econometrics (Cont.)

- Call the example FAR script `far_d`. Hopefully your results are similar to:

Figure: Output Results for far_d

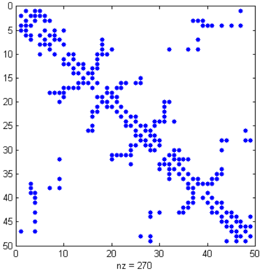
```
First-order spatial autoregressive model Estimates
Dependent Variable =       crime
R-squared          =    0.5394
sigma^2            =    0.4512
log-likelihood     =        -60.25782
Nobs, Nvars        =     49,  1
# of iterations    =     20
total time in secs =    0.0740
time for optimiz   =    0.0030
time for lndet     =    0.0630
time for t-stat    =    0.0010
min and max rho    =  -1.0000,  1.0000
Pace and Barry, 1998 spline lndet approximation used
***************************************************************
Variable      Coefficient  Asymptot t-stat    z-probability
rho             0.778983        5.326578         0.000000
```

Application: Spatial Econometrics (Cont.)

- In addition to this, you can also plot the spatial weighting matrix using the command:

```
spy(D)
```

Figure: Spatial Weighting Matrix for far_d



Optional Homework Assignment

- Taken from Pouliot and Lampis (2014):

> Write a function called mydiag that returns a diagonal matrix if the input is a vector and returns the diagonal entries of the matrix if the input is a matrix. Use the `max`, `min` and `size`.