Exercises in Advanced Microeconometrics

By Søren Leth-Petersen, Bertel Schjerning and Jakob Johansen

Week 1 — Autumn 2012

Contents

The purpose of this week's exercise is twofold: First, to introduce the matrix programming language Matlab. Second, to generate a synthetic data set, estimate the parameters of a linear regression model by OLS, and to carry out a simple Monte Carlo experiment.

Section 1 introduces common features and commands in Matlab. Section 2 consists of three exercises. In the first exercise, we generate a synthetic data set. In the second, we estimate the parameters of a linear regression model by OLS. In the third, we carry out a simple Monte Carlo experiment.

1 A short introduction to Matlab

```
[Matlab code is written in verbatim]
```

Matlab's Desktop
Matlab's desktop consists of a number of windows:

1. Command window
   - Execute commands.
   - Carry out tasks such as creating variables.
2. Workspace
   - View created variables and functions.
   - Edit and plot variables from here.
3. Command history
   - Lists all previously executed commands.
4. Current directory
   - Navigate your directories
   - Set current directory

Entering matrices
To create a (1x6) row vector called x write

```
x = [1 2 3 4 5 6 7 8 9]
```

To create a (6x1) column vector called x write, use semicolons (;) to separate the rows

```

x = [1;2;3;4;5;6;7;8;9]
```

Creating a matrix, called x, is easy as a vector, use semicolons (;) to separate the rows of the matrix.

```
x = [1 2 0; 2 5 -1; 4 10 -1]
```

Indexing a matrix, use ()
Consider matrix x from above

$$x = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 5 & -1 \\ 4 & 10 & -1 \end{pmatrix}$$

We have

```
a = x(2,3)
```

$$a = -1$$

```
b = x(:,3)
```

$$b = \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$$

```
c = x(2:3,2)
```

$$c = \begin{pmatrix} 5 \\ 10 \end{pmatrix}$$

```
d = x(2,:)
```

$$b = \begin{pmatrix} 2 & 5 & -1 \end{pmatrix}$$

```
e = x(2:3,1:2)
```

$$e = \begin{pmatrix} 2 & 5 \\ 4 & 10 \end{pmatrix}$$

Matlab also allows you to index a matrix by writing `x(i)` — this is particularly useful when working with *vectors* but not if the input turns out to be a matrix.

Concatenating

We would like to concatenate $a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $b = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$

Horizontally:

```
[a,b] or horzcat(a,b)
```

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

Vertically:

```
[a;b] or vertcat(a,b)
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Matrix Operations
Matrix algebra

$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$'$	tranposition
$\hat{}$	exponentiation
$==$	equal
$\tilde{}=$	not equal
$>$	greater than
$<$	less than
$>=$	greater than or equal
$<=$	less than or equal

When doing matrix algebra one needs to consider **conformability** and matrix operations vs. **element-by-element** operations. Consider a matrix a that is the outcome of an operation of the matrices b and c. First, conformability

Operation	b	c	a
$a = b * c$	3x2	3x1	illegal
$a = b' * c$	3x2	3x1	2x1
$a = b + c$	3x2	3x1	illegal
$a = b + c$	3x2	3x2	3x2

Second, matrix operations vs. element-by-element operations. Sometimes we need to do element-by-element operations, this is done by prefixing the operator with a dot.

Operation	b	c	a
$a = b. * c$	3x1	3x1	3x1
$a = b./c$	3x1	3x1	3x1

For instance, let $b = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$ and $c = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ then

```
1 a=b.*c
```

$$a = \begin{pmatrix} 5 \\ 2 \end{pmatrix} * \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 15 \\ 8 \end{pmatrix}$$

Special matrix types

`y = normrnd(mu,sigma,m,n)`	Creates a mxn matrix of random numbers from the `normal distribution` with mean mu and standard deviation sigma.
`y = eye(m,n)`	Creates a $m \times n$ indentity matrix.
`y = ones(m,n)`	Creates a $m \times n$ matrix of ones.
`y = zeros(m,n)`	Creates a $m \times n$ matrix of zeros.
`y = inv(x)`	The inverse of matrix x
`y = size(x)`	Dimension of matrix x, size(x,1) = no. of rows, size(x,2) = no. of columns
`y = diag(x)`	Creates a column containing the diagonal of x.
`y = mean(x,k)`	Mean of the rows if $k = 2$, mean of the columns if $k = 1$.

Looping
A loop have the following structure

```
while logical_expression
  ...
  statements
  ...
end;
```

where `logical_expression` must be a logical expression (i.e. be either true (1) or false (0)). Another loop is the `for` loop,

```
for counter=start:step:end
  ...
  statements
  ...
end;
```

where `counter` is an integer variable that is initialized as `counter = start` and then indented by `step` at the time until it reaches the value `end`.
Example 1: Generate a $n \times 1$ vector x of standard normal numbers. Then, create a binary vector y that equals 1 if $x > 0$ and 0 if $x \leqq 0$. We look at two different ways to do this.

```
n=1000;
x=normrnd(0,1,n,1);
y=zeros(n,1);

i=1;
while i≤n
   if x(i,:)>0
```

```
8       y(i,:)=1;
9     end;
10    i=i+1;
11  end;
```

The same loop can be performed with a `for`-loop

```
1  for i=1:n
2     if x(i,:)>0
3       y(i,:)=1;
4     end;
5  end;
```

Note that `1:n` is short hand notation for the vector `[1 2 ... n]`. Alternative uses of the loop includes writing `for i=1:2:n` which asks Matlab to do the loop in jumps of 2, i.e. `i=[1 3 ... n]`, or writing `for i=n:-1:1` which makes the loop run "backwards".
Hint: In this example, is a loop necesary? No, simply use

```
1  y=x>0,
```

which returns a binary vector (zero/one) where each element is one when the condition is true and zero otherwise. Another cool Matlab feature is that such a binary vector can be used to pick out elements of another vector (of equal size) — so writing `x(x>0)` will return a vector consisting of only the elements in `x` that are positive.
Minimize use of loops, they are time-consuming.

Example 2: Generate an AR(1)-process.

```
1  T=20;
2  rho=1;
3  y=zeros(T,1);
4  epsilon=normrnd(0,1,T,1);
5  y(1,:)=2;
6
7  for t=2:T
8  y(t,:)=rho*y(t-1,:) + epsilon(t,:);
9  end;
10
11 for t=2:T
12 dy(t,:)=y(t,:)-y(t-1,:);
13 end;
14
15 % plot y and dy against t
16 t=1:T;
17 plot(t,y)
18 pause(3)
19 plot(t,dy)
```

Loops are often time-consuming, thus it is important to think about efficiency when using them. When using a `while` loop it is particularly important to make sure that the while-condition will actually be false at some point. Otherwise Matlab will do its very best at frying your computer —

use `<ctrl>+<c>` to quit runaway processes. On the other hand, the `while` loop is useful precisely when you don't know how many steps of an iteration you need, e.g. when working with convergence.

IF-statements

```
if  expression 1
  ...
  statements1
  ...
elseif expression 2
  ...
  statements2
  ...
else
  b...
  statements3
  ...
end;
```

Example 3: Create a $n \times 1$ vector x of uniform random numbers. Then create a $n \times 1$ vector y, where

$$y = \begin{cases} 1 & \text{if } x \leqq 0.25 \\ 2 & \text{if } 0.25 < x \leqq 0.5 \\ 3 & \text{if } 0.5 < x \leqq 0.75 \\ 4 & \text{otherwise.} \end{cases}$$

```
n=100;
x=rand(n,1);
y=zeros(n,1);
for i=1:n
   if x(i)≤0.25
      y(i)=1;
   elseif x(i)>0.25 && x(i)≤0.5
      y(i)=2;
   elseif x(i)>0.5 && x(i)≤0.75
      y(i)=3;
   else
      y(i)=4;
   end;
end;
```

Note that we are using the single-index method of addressing elements of the vectors which only works (sensibly) because `x` and `y` are vectors and not matrices.

An alternative way of obtaining the code above without the use of loops could be the following, where we use elementwise multiplication of the binary vectors to obtain the logical "and" statements (the `&&` operator only works on scalars).

```
z = 4 * ones(n,1); % default category
z = z - 3 * (x≤0.25);
z = z - 2 * ((x>0.25) .* (x≤0.5)); % ".*" gives the "AND"
```

```
4  z = z - 1 * ((x>0.5) .* (x<=0.75));
```

Final comments

- Matlab is case sensitive.
- To stop a running program, press CTRL+C.
- Use arrow up ↑ in the command window to get the most recently executed command.
- Minimize the use of loops, they take time.
- Use comments in your program to make you (and others) remember what a piece of code is doing, or what the content of a variable is. A comment is a piece of code that will not be executed, and do not influence the program in any way. To comment a line put a % in front. To comment out a section, use %{ and %}.
- Sometimes it is convenient to be able to temporary suppress a part of a program. This can conveniently be done by putting a whole section of a program in to a comment. Select the lines you want to comment out. Press CTRL+R. To uncomment press CTRL+T.
- Place the cursor inside the name of a built-in Matlab command that you don't understand or forgot and hit F1 to display the Matlab help for that function.
- Further help can be found in Matlab's on-line help. Here you can also find small video demos.

2 Exercises

2.1 Exercise 1

a. Create a synthetic data set with the following characteristics

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

where $\beta_0 = 1$, $\beta_1 = -0.5$, and $\beta_2 = 2$, $x_{1i} \sim N(0,4)$, $x_{2i} \sim N(5,9)$, $\varepsilon_i \sim N(0,1)$ and where $i = 1, .., 100$.

The code may look something like this:

```
1
2  seed = 1;                                  % set seed to 1
3  randn('seed',seed);                        % set method to seed
4  n=100;                                     % define n=100 - size of data set
5  b=[1;-0.5;2];                              % define parameter values
6
7  x1=normrnd(0,2,n,1);                       % generate x1~N(0,4) -> nx1
8  x2=normrnd(5,3,n,1);                       % generate x2~N(5,9) -> nx1
9  eps=normrnd(0,1,n,1);                      % generate eps~N(0,1)-> nx1
10
11 x=[ones(n,1),x1,x2];                       % collect the expl. variables
12 y=[FILL IN];                               % generates y -> nx1
```

b. Imagine that you had not generated the data set yourself but that you were given a similar data set that was already collected (generated) and ready to analyze. What would you observed in that data set?

2.2 Exercise 2 — OLS

a. Write up the OLS estimator in matrix form.

b. Write up the OLS standard errors in matrix form.

c. Estimate β from the synthetic data set. Furthermore, calculate standard errors and t-values (assuming that the assumptions of the classical linear regression model are satisfied).

The code may look something like this

```
b_hat  = [FILL IN];                          % estimate kx1 vector of OLS parameters
res    = y-x*b_hat;                          % OLS residuals
sigma  = [FILL IN];                          % calculate sigma^2
b_var  = [FILL IN];                          % calculate cover. matrix of b_ols (kxk)
b_stde = sqrt(diag(b_var));                  % extract standard errors from ...
    covar-matrix
t      = b_hat./b_stde;                      % calculate t-values

% PRINT RESULTS
disp('           OLS-estimates                  ');
disp('=========================================');
disp('      b       b_hat    stde_b    t-value  ');
disp([b, b_hat, b_stde, t]);
disp('=========================================');
```

2.3 Exercise 3 — a simple Monte Carlo Experiment

Carry out a Monte Carlo experiment with $S = 200$ replications and $N = 100$ observations to check if the OLS estimator provides an unbiased estimate of β

a. Generate 200 data sets similar to what you did in exercise 1, and estimate β on each of them.

b. Calculate the means of the estimates, the standard errors of the estimates and the means of the estimated standard error - over the 200 replications.

Exercise 3.a and b is most conveniently implemented in a loop. The code could look something like this:

```
s=200;                                       % # replications
n=100;                                       % # observations (as before)
m1=zeros(s,size(b,1));                       % (Sx3) vector to save estiamtes
m2=zeros(s,size(b,1));                       % (Sx3) vector to save estiamtes
for i=1:s                                    % start loop
```

```
6
7       % GENERATE DATA
8       x1=normrnd(0,2,n,1);                    % generate x1~N(0,4) -> nx1
9       x2=normrnd(5,3,n,1);                    % generate x2~N(5,9) -> nx1
10      eps=normrnd(0,1,n,1);                   % generate eps~N(0,1)-> nx1
11      x=[FILL IN];                            % expl. variables    -> nx3
12      y=[FILL IN];                            % dep. variable      -> nx1
13
14      % ESTIMATION
15      b_hat=[FILL IN];                        % OLS estimates -> 3x1
16      res=y-x*b_hat;                          % OLS residuals -> nx1
17      var_b=[FILL IN];                        % Variance-Covariance matrix of beta ...
            -> 3x3
18
19      % STORE ESTIMATES IN m1 AND m2
20      m1(i,:)=b_hat;                          % put OLS estimate i into row i of m1
21      m2(i,:)=diag(var_b);                    % put estimate i of est. var into row ...
            i of m2
22
23  end;                                        % end loop
24
25  % DO MEAN AND STANDARD ERRORS
26  E_b_hat=mean(m1,1);                                 % Mean of OLS estimates
27  E_se_b=[FILL IN];                                   % Mean of estimated standar errors
28  Mcse=sqrt(1/(s-1)*sum((m1-ones(s,1)*mean(m1,1)).^2,1));% Standard deviation of ...
        Monte Carlo Estimates
```

c. Draw a histogram for the 200 estimates of β_1.

```
1   hist(m1(:,2),20);                           % hist(sx1 vector est., number of bins)
```