

Department of Applied Mathematics and Statistics (AMS)  
University of Copenhagen

## Introduction to Ox.

This is a short introduction to the Ox language used in the course Econometrics, fall semester 2003 at the University of Copenhagen.

You may or may not be able to download Ox from the course homepage, either way Ox is available at the computers in the various computer-rooms at the institute.

At this point the reader is assumed to be familiar with GiveWin and PcGive/PcFiml, and the following will show how to use Ox to extend the features in these two programs.

Ox is a matrix oriented language and the syntax is quite similar to programs such as C++ and R. From our point of view the main advantage with Ox is the useful interface with GiveWin and the build in packages (called classes) for working with databases, graphics, maximization and regression.

Some of the examples in the following are based on the AR(1) process. We will also revisit the hamburger chain data from Worksheet (exercise) 1 and 2. Finally some of the text and examples stem more or less directly from the Ox help-files.

See the end of this introduction for a reference to an extensive Ox manual.

### GiveWin and OxRun.

From GiveWin you start working with Ox by choosing "File"  $\Rightarrow$  "new"  $\Rightarrow$  "Ox program", you will then be give the choice of including one or more headers, for now just choose "Has Main Function" and click "OK". This opens a page where the program is to be written, for those who remember C++ this should look familiar.

GiveWin includes many features to make programming easier, e.g. unmatched parentheses are shown in red.

To run the program, you can click on the Run button (the running person icon on the toolbar). Before "running" a file it must be saved to a disk, if you haven't done this already GiveWin will ask you to do it now.

The output will appear in a new window, entitled OxRun Sesssion. You can also use OxRun to run a specified program by starting OxRun from the "Modules" menu in GiveWin. As before, the output (both text and graphics) will appear in GiveWin.

Note that you may experience the same problems as we all had when we used PcGive for the first time on the University's system: When using Ox for the first time you may have to leave GiveWin and start OxRun from Windows i.e. "Start->Programs->... ect.". You then enter the path leading to any Ox program file e.g "C:\Programmer\Ox\samples\myfirst.ox". This only has to be done once, in the future GiveWin will be aware of the fact that Ox is installed and it can be started from within this program.

## Program layout.

As mentioned the syntax of Ox is modelled on C and C++ (and Java), so much of the following should be recognized.

True to tradition, our first program is:

```
#include <oxstd.h>

main()
{
    print("Hello world");
}
```

This program only prints one line of text, but it is worth discussing anyway:

- The first line includes a *header file*
- The purpose of **oxstd.h** is to declare all standard library functions.
- Any Ox program starts execution at the **main** function.
- A block of code (e.g. a function) is enclosed in curly braces.
- All statements are terminated with a semicolon (;).

Note that Ox is **case sensitive**, it makes a distinction between lower case and upper case letters.

At this time it is a good idea to point out a difference between Ox and C++:

Since Ox is a matrix based language, there is much less need for loop statements than in C++. Indeed, because Ox is compiled and then interpreted, there is a speed penalty for using loop statements when they are not necessary.

Sometimes we can't avoid using loops so the next sections contains a quick introduction to "classic" programming in Ox.

## If, for, while...

From a programming viewpoint Ox is the same as C++, this can most easily be seen in the following examples:

### IF:

```
decl d = ranu(1,1);
  if (d < 0.5)
    println("less than 0.5");
  else if (d < 0.75)
    println("less than 0.75");
  else
    println("greater than 0.75");
```

`ranu()` generates a uniform distribution.

Ox has the usual relational operators. There is "`<`", "`<=`", "`>`", "`>=`", "`==`" and "`!=`" meaning 'less', 'less than or equal', 'greater', 'greater than or equal', 'is equal' and 'is not equal'. These always produce the integer value 1 (true) or 0 (false). If any of the arguments is a matrix, the result is only true if it is true for each element.

### For, while:

The "for", "while" and "do while" loops have the same syntax as in C++. The for loop consists of three parts, an initialization part, a termination check, and an incrementation part. The while loops only have a termination check, look at the following examples to make sure you understand:

```
for(i = 1; i<=100; ++i)
```

```
while(i<=100)
```

```
while(i>=50 && i<=75)
```

The `&&` is logical-and, whereas `||` is logical-or. The `++i` statement is called (prefix) incrementation, and means 'add one to i'. Similarly, `--j` subtracts one from j. There is a difference between prefix and postfix incrementation (decrementation). For example, the second line in

```
i = 3;
j = ++i;
```

means: add one to i, and assign the result to j, which will get the value 4. But

```
i = 3;
j = i++;
```

means: assign the value of i to j then add 1 to i. So j will get the value 3. In both cases i will end up with the value 4. In the incrementation part of the for loop it does not matter whether you use the prefix or postfix form.

## Statements, Simulations, Matrices etc.

If you remember the introduction to GiveWin and PcGive, we simulated an AR(1) process, lets see how this is done in Ox:

```
#include <oxstd.h>
#include <oxdraw.h>

main()
{
    decl eps,y,n=500,rho=0.4,i;
    y=zeros(n,1);
    eps=rann(n,1);

    for(i = 1; i<=(n-1); ++i)
    {
        y[i][0]=rho*y[i-1][0]+eps[i][0];
    }

    Draw(0,y');
    ShowDrawWindow();
}
```

Comments:

- **decl** is used to declare the variables of the program.
- **zeros(n,k)** creates a n by k matrix with all elements equal to zero.

- **rann(n,k)** creates a n by k matrix with observations from a  $\mathcal{N}(0, 1)$  distribution.
- A matrix is indexed starting from 0, (i.e. top left element in a matrix A is called by typing A[0][0]).
- **y'** means y transpose.

The results from running the above program can be seen in Figure 1. Try replacing  $\rho=0.4$  with  $\rho=0.7$ ,  $\rho=0.9$ ,  $\rho=1$  and observe when there is a radical change in the output, you should be well aware of the reason for this change by now.

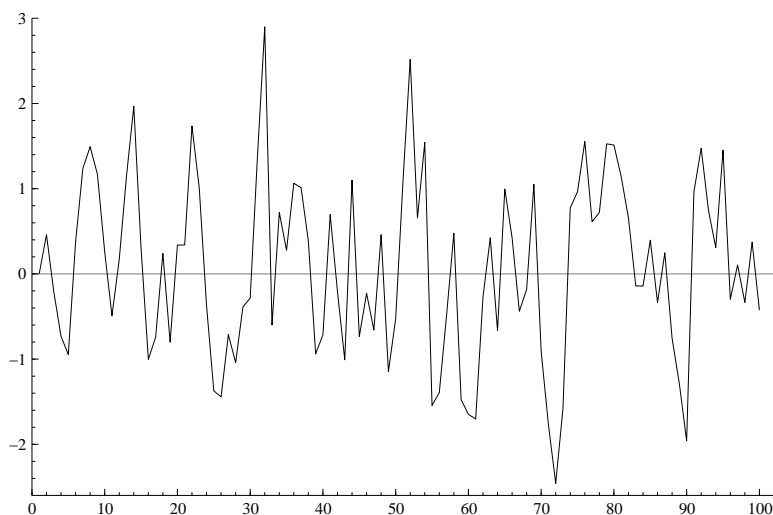


Figure 1: AR(1) process simulated in Ox.

## Linear Regression.

In this section we will begin to understand the true potential of Ox. We will do so by replicating the results from Worksheet 2, where we analyzed data from a hamburger chain.

First we should note that many of the procedures we will need during the course are already present within Ox in the shape of classes. We will not need to make our own programs to perform maximization, OLS, or other forms of simple regression. The following program illustrates:

```
#include <oxstd.h>
```

```

#import <pcfiml>

main()
{
    decl model= new PcFiml();

    model.Load("c:/.../burger.in7");
    model.Deterministic(FALSE);

                                // formulate the model
    model.Select(Y_VAR, { "Receipts", 0, 0 } );
    model.Select(X_VAR, { "Price", 0, 0 } );
    model.Select(X_VAR, { "Advertising", 0, 0 } );
    model.Select(X_VAR, { "Constant", 0, 0 } );

    model.SetSelSample(1, 1, 52, 1);
    model.Estimate();           // estimate the system (VAR)

    delete model;
}

```

You should type this program for yourself and see if you recognize the results from Worksheet 2.

Comments:

- **new** creates a new object of the PcFiml class, the **new** command is needed to make sure that "model" is recognized as a regression model.
- **model.Load** inputs the data from the file "burger.in7".
- **Deterministic()** creates a constant, a trend and seasonal dummies. **Deterministic(false)** creates "normal" seasonals. **Deterministic(true)** makes centred seasonals (with quarterly observations, i.e. 0.75, -0.25, -0.25, -0.25, ...), in which case the names are **CSeason**, **CSeason\_1**, ..., **CSeason\_x**. No seasonals are created by **Deterministic(x)** where  $x < 0$ . If other deterministic processes are needed (e.g.  $x_t = t^2 + t$ ) they must be created explicitly.
- **select** formulates the model, **Y\_var** for dependent and lagged dependent variables. **X\_var** for other regressors, the second argument contains: variable name, start lag, end lag.

The output from this program contains all the data we need for further analysis and testing, among other things we find standard deviations of estimators (useful for t-test) and the value of the likelihood function.

**Writing to a file.**

We now expand on the AR(1) program from earlier so that it also creates a .in7 file containing the data, reloads the data and estimates the AR(1) model.

That is, we first simulate some data given by:

$$y_t = \rho y_{t-1} + \mu + \varepsilon_t$$

$$\varepsilon_t \sim iid\mathcal{N}(0, \sigma^2)$$

We then save this data in a .in7 file, and last we estimate the model just like we would have done in PcGive/PcFiml:

```
#include <oxstd.h>
#include <oxdraw.h>
#import <pcfiml>

main()
{
    decl eps,y,n=100,rho=0.4,i,model= new PcFiml();
    y=zeros(n,1);
    eps=3*rann(n,1);

    for(i = 1; i<=(n-1); ++i)
    {
        y[i][0]=7+rho*y[i-1][0]+eps[i][0]; //we have added a constant.
    }

    Draw(0,y');
    ShowDrawWindow();

    savemat("c:/.../AR1.in7",y);

    //A file has now been saved under the name AR1.in7.

    //Now we can estimate the model,
    //note that we now have one lag on the dependent variable.

    model.Load("c:/.../AR1.in7");
    model.Deterministic(FALSE);

    model.Select(Y_VAR, { "Var1", 0, 1} );
```

```

model.Select(X_VAR, { "Constant", 0, 0} );

model.SetSelSample(1, 1, n, 1);
model.Estimate();          // estimate the system (VAR)

delete model;
}

```

After having run this program you should:

1. Go to the appropriate directory and see that the file has been saved as it should.
2. See that the estimation does what it should, i.e. that  $\hat{\rho}$  and  $\hat{\mu}$  are close to the values we selected.
3. Perform the estimation in PcGive and note that it is the same.

### The $p$ -dimensional VAR(k) model

It will be useful later on if we learn how to simulate a VAR(k) model, so let's see how this is (or can be) done, we will work with the following process:

$$Y_t = \sum_{i=1}^k \Pi_i Y_{t-i} + \varepsilon_t$$

Where  $Y_t$  is a  $p$  vector,  $\Pi_i$  is a  $p$  by  $p$  matrix and  $\varepsilon_t$  is drawn from a  $p$ -dimensional normal distribution.

In this program the residuals are all identically distributed (i.e. for  $1, \dots, p$ ), if we want we can multiply a matrix on each row in the eps matrix and thereby introduce a variances/covariance matrix.

```

#include <oxstd.h>

main()
{
    decl p=3,k=2,eps,y,n=500,pi1,pi2,i,output;

    pi1=<0.3,0.0,0.0;0,0.3,0.0;0.0,0.0,0.1>;

    pi2=<0.6,0.0,0.0;0.0,0.2,0.0;0.0,0.0,0.2>;
}

```

```
y=zeros(p,n);  
  
eps=rann(p,n);  
  
for(i = k; i<=(n-1); ++i)  
{  
    y[] [i]=pi1*y[] [i-1]+pi2*y[] [i-2]+eps[] [i];  
}  
  
output=y';  
  
savemat("c:/.../VARK.in7",output);  
  
}
```

The above program simulates a 3-dimensional VAR(2) process, i.e.  $p = 3$  and  $k = 2$ , the results are saved in a GiveWin data file named *VARK.in7*.

You should run this program, then open the file in GiveWin and estimate the model in PcFiml. It is important that you learn how to read the output from a multi-variable estimation, so go to the results window and see if you can find the following:

$$\hat{\Pi}_1, \hat{\Pi}_2, \det(\hat{\Omega})$$

These estimates are scattered throughout the PcFiml output, but with a little experience they are easily found.

### **Conclusion.**

We have now been through the most fundamental parts of Ox, so far we are not able to do anything that couldn't be done in PcGive/Fiml, but later on we will work with non-linear models, and models containing zero/one variables and so on. This is where Ox really becomes useful. The algebra editor in GiveWin is not well suited for longer programs (it can be done though), and working with Ox makes life much easier.

This note should help you get started on working with Ox, it is clear though, that you need a much more detailed list of functions and classes once we get to the more hardcore programming. A 113 page long book by the creator of Ox (Jurgen Doornik) can be found within the help files (This Ox manual

should be installed with the Ox program, if not then it can be downloaded from <http://www.nuff.ox.ac.uk/users/doornik> , start by entering GiveWin, open an Ox file go to "help" in the toolbar, select "Module Help Index.". A web page should now open, in the menu on the left select "Introduction to Ox".