Advanced Programming in Quantitative Economics

Introduction, structure, and advanced programming techniques

Charles S. Bos

VU University Amsterdam

c.s.bos@vu.nl

20 – 24 August 2012, Aarhus, Denmark

Outline

Day 4 - Morning

9.00L Topics

- Style
- Including packages
- Including magic numbers
- Including graphs

10.30P Estimating a duration model

- Transform $0.5 < \beta_2 < 1$
- Graph the durations
- Advanced:
  - Draw $N = 1000, y_i \sim \mathcal{N}(0, \sigma^2)$ for a σ of choice. Make a QQ plot using `DrawQQ`
  - Make the QQ plot 'by hand' using `DrawXMatrix`, drawing the empirical quantiles of the y's against the theoretical quantiles of the normal density
  - Make a residual plot $E_i = (\Lambda_i y_i)^\alpha$ for your y's of the duration model, and a QQ-plot against the Exp(1) density

Style and neatness, directories

Reread old program: Almost impossible. What can you do?

- Use as clear a style as possible, extremely consistent throughout multiple projects
- Spend time on being neat, including explanations for routines
- Build small, self-explanatory routines
- Use modules/packages for iterative tasks
- Use Hungarian notation
- Use a directory structure! (See next slide)

Order in the main ox file

1. Comments (program, name, date, version)
2. `#include <oxstd.h>` standard packages
3. `#import <maximize>` standard imports
4. `#static decl s_vY;` necessary global variables
5. `#include "include/incinit.ox"` personal code
6. `main()` routine, containing
   - 6.1 `decl`
   - 6.2 magic
   - 6.3 init
   - 6.4 estimate
   - 6.5 output

Order II

```
// Comments (program, name, date, version)

#include <oxstd.h>       // And other basic includes
#import <maximize>       // And other basic imports

#static decl s_vY, s_mX;    // Global variables for Loglikelihood

#include "include/incinit.ox"        // My own initialisation routines
#include "include/incest.ox"         // My own estimation routines

main()
{
  decl ...;

  // Magic numbers
  sData= "data/fx9709.in7";
  asYX= {"FXEU", "C", "FXDK"};

  // Initialise
  InitData(&vY, &mX, asYX, sData);
  InitPars(&vP, vY, mX);

  // Estimate
  ir= Estimate(&mPS, &dLL, vY, mX);

  // Output
  Output(mPS, dLL, ir);
}
```

Directory structure

Example: Project on Foreign Exchange rates (FX) including a jump diffusion process

Directory	Contents
`fxjump`	Project, includes file `logbook.txt`
`fxjump/ox`	Main Ox files, declaration of present setting
`fxjump/ox/data`	Data files, including very small ox-file for graphing data
`fxjump/ox/decl`	Other declarations
`fxjump/ox/include`	Include files with `Ox` routines
`fxjump/ox/test`	Test files
`fxjump/ox/results`	Results of computations
`fxjump/text/v1`	First version of text, main tex file
`fxjump/text/v1/include`	Separate chapters
`fxjump/text/v1/graphs`	Corresponding graphics
`fxjump/text/pres`	Presentation, main tex file
`fxjump/text/pres/graphs`	Corresponding graphics

Reads very much simpler, purpose of each file is clear from its location.

Include

Enlarging the capabilities of ox beyond `oxstd.h` capabilities: Either

```
#include <oxprob.h>
```

(to include the mentioned file literally within the program at that point, and will be compiled in), or

```
#import <maximize>
```

(to import the code when needed; pre-compiled code is used when available)

Ox-provided packages

Package	Purpose
oxprob.h	Extra probability densities
oxfloat.h	Definition of constants
oxdraw.h	Graphics capabilities ($*$)
arma.h	ARMA filters and generators
quadpack.h	Univeriate numerical integration
maximize	Optimization using Gauss-Newton or Simplex methods ($*$)
maxsqp	Maximize non-linear function with sequential quadratic programming
solvenle	Solve a system of non-linear equations
solveqp	Solve a quadratic program with restrictions
database	General class for creating a database
modelbase	General class for building a model
simulation	General class for simulation exercise

User-provided packages

Package	Author	Purpose
arfima	Doornik, Ooms	Long memory modelling
dcm	Eklof, Weeks	Discrete choice models
dpd	Doornik, Arellano, Bond	Dynamic Panel Data models
financialnr	Ødegaard	Financial numerical recipes
gnudraw.h	Bos	Alternative graphing capabilities
maxsa.h	Bos/Goffe	Simulated Annealing
msvar	Krolzig	Markov switching (outdated)
oxutils.h	Bos	Some convenient utilities ($*$)
oxdbi	Bruche	A database independent abstraction layer for Ox
ssfpack.h	Koopman, Shephard, Doornik	State space models
...	...and many others	
m@ximize	Laurent, Urbain	Use CML optimisation in OxGauss
oxgauss	Doornik	Run Gauss code through Ox

- Packages reside either in `ox-home/packages`, or in a local packages folder.
- After including the package, the package is supposed to work seamlessly with Ox
- Easy and clean way of communicating research

A package: oxutils

What does ‘seamless’ mean?

Standard situation: What is the size of a matrix I’m using?

```
main()
{
  ...
  print (rows(mX)|columns(mX));
}
```

How often would you use this code while debugging?

A package: oxutils II

Alternative: Use a package with some extra functions, not previously available

```
#include <packages/oxutils/oxutils.h>

main()
{
  ...
  print (size(mX));
}
```

Check manual

`<ox-home>/packages/oxutils/doc/oxutils.html`

Other routines I use plenty:

`info`	Measure time an iteration takes, time until end of program
`TrackRoutine`	Routine to profile your program
`printtex`	Replacement for `print`, outputting in LaTeX format
`ReadArg`	Read arguments from command line
`setseed`	Reset the random seed, psuedo-randomly

A package: oxutils III

(From tomorrow's slides on speed)

Use `TrackTime("concat")` to profile a piece of code, get a report using `TrackReport()`

```
#include <packages/oxutils/oxutils.h>

main()
{
  decl iN, iK, mX, j;

  iN= 1000;     // Size of matrix
  iK= 100;

  TrackTime("concat");
  mX= <>;
  for (j= 0; j < iN; ++j)
    mX |= rann(1, iK);

  TrackTime("predefined");
  mX= zeros(iN, iK);
  for (j= 0; j < iN; ++j)
    mX[j][]= rann(1, iK);
  TrackTime(-1);

  TrackReport();
}
```

Output:

```
Ox Professional version 6.00 (Linux_64/MT)
Time spent in routines
concat              2.42    0.99
predefined          0.02    0.01
Total:  2.44
```

Magic numbers and declarations

Magic numbers:

- Those numbers/strings/settings defining what your program will do
- Might be changed regularly (testing different sample sizes, regressors etc.)

Ugly solution: Change program
Better solutions:

1. Specify them 'outside' program?
2. Specify them on command line

Include

Enlarging the capabilities of ox beyond `oxstd.h` capabilities: Either

```
#include <oxprob.h>
```

(to include the mentioned file literally within the program at that point, and will be compiled in), or

```
#import <maximize>
```

(to import the code when needed; pre-compiled code is used when available)
You can also include a declaration file:

```
#include "simox.dec"
```

with special settings for your program.

Declaration file I

Remember previous exercise: Run with $n = 100$, run with $n = 1000$, run with $n = 10000$ etc.

Options:

1. Build program very general, including a loop over different values of n
2. Build a general program for one value of n; indicate the value to use in a *declaration file*
3. Possibly allow settings to be changed on the command line

   ```
   oxl lrdecl n 50 base thisversion
   ```

   e.g. using the `ReadArg` statement from `OxUtils`

Using a declaration file

Wouldn't it be useful to have

Listing 1: stack/eststack.dec

```
/*
**  EstStack.dec
**
**  Purpose:
**    Contain definitions for estimating stack-loss data
**
**  Date:
**    18/9/06
**
**  Author:
**    Charles Bos
*/
static decl g_sData= "data/stackloss.in7";
static decl g_sYVar= "StackLoss";
static decl g_asXVar= {"AirFlow", "WaterTemperature", "AcidConcentration"};
```

Use fewer X-vars: Change array here, leave program untouched

Use different data set: Change `g_sData` and variable names, leave program untouched

⇒ Clean, touch program to change computation, put settings aside in separate file.

Prepare data set

Listing 2: stack/loadstack.ox

```
main()
{
  decl mX, asNames;

  mX= loadmat("data/stackloss.mat");       // Read the data into rows
  mX= mX|1;                                 // Add in a constant
  asNames= {"AirFlow", "WaterTemperature", "AcidConcentration",
            "StackLoss", "Constant"};
  savemat("data/stackloss.in7", mX', asNames);  // Save data in columns
  print ("%r", asNames, meanr(mX)~varr(mX));    // Check data
}
```

to reload it in estimation program with

Listing 3: stack/eststack_ols.ox

```
InitData(const avY, const amX, const sYVar, const asXVar, const sData)
{
  decl db;

  db= new Database();
  db.LoadIn7(sData);              // Read database
  avY[0]= db.GetVar(sYVar);       // Extract Y from database
  amX[0]= db.GetVar(asXVar);      // Extract Xs from database

  delete db;
```

Using the declarations

Initialise your settings in a separate routine, reading out the declaration file, e.g.

Listing 4: stack/eststack_ols.ox

```
Initialise(const asYVar, const aasXVar, const asData)
{
  asData[0]= g_sData;
  asYVar[0]= g_sYVar;
  aasXVar[0]= g_asXVar;
}
```

Preferably: Touch globals as little as possible, in few places.

Is this the only way of specifying the settings? No...

From the manual of OxUtils (dd 14/9/07)

ReadArg

```
#include <packages/oxutils/oxutils.h>
ReadArg(const aiX, const sX, const iType);
```

`aiX`

Pointer to output variable, with value on command line if the string indicator is found, or unchanged otherwise. Boolean values default to FALSE, if the argument is not found.

`sX`

in: String, command line argument to look for.

`iType`

in: Integer, indicating type of element to look for;

- <-1= array of strings,
- -1= string,
- 0= boolean (aiX= TRUE if string argument found, FALSE otherwise),
- 1= real value,
- >1= row vector with (at most) iType elements.

Return value

Integer, indicating if number of elements read for the command line argument.

Description

This function checks the command line arguments for the occurrence of the string `sX`, and if it is found the value of the next argument(s) is/are returned in `aiX`, or, if `iType == 0`, `aiX` is returned with the value of `TRUE` straight away.

Using ReadArg/ReadArgUsed

Listing 5: stack/eststack_ols.ox

```
Initialise(const asYVar, const aasXVar, const asData)
{
  ...
  ReadArg(asData, "data", -1);  // Read string with data file
  ReadArg(asYVar, "y", -1);     // Read string with y-variable
  ReadArg(aasXVar, "x", -5);    // Read array of strings with x-variable

  ReadArgUsed();                // Show the arguments
}
```

Call `Ox` from command line using

```
oxl eststack_ols data data/gnp.in7 y GNP x Constant IP
```

(if `Ox` installed within the path) or use `OxRun` to indicate the parameters.

A package: OxDraw (or GnuDraw...)

Ox graphics are displayed within OxMetrics. Needs the professional version for Windows.

Alternatively, use GnuDraw: Displays graphics in GnuPlot on Windows, OSX, Unix. Compatible in usage, easy to switch.

Listing 6: stack/drawstack.ox

```
#include <oxdraw.h>
// #include <packages/gnudraw/gnudraw.h>    // Alternatively

// Draw stackloss regressors on Y, stackloss itself on X axis
DrawXMatrix(0, mX', asXVar, vY', sYVar, 1, 2);
SaveDrawWindow("graphs/stackloss.eps");
ShowDrawWindow();
```

From the manual:

```
DrawXMatrix(const iArea, const mYt, const asY, const vX, const sX, ...);
DrawXMatrix(const iArea, const mYt, const asY, const vX,
    const sX, const iSymbol, const iIndex);
```

OxDraw (or GnuDraw...) II

- Graphing appears in *graphing area*, first argument
- Draws *rows* at a time
- Puts in a label. For multiple Y-values, give an array of labels `{"yHat", "y", "cons"}`
- Can draw XY data, time series data, densities, QQ-plots etc.
- Takes extra arguments specifying line types, colours etc.
- *After* drawing the graph, and before showing it, the last graphing command can be adjusted using `DrawAdjust(...)`
- For daily time series data, use e.g. `DrawTMatrix(iArea, mY, asYVar, vDates, 0, 0);`
- Save the graphics in `eps`, `pdf` or `gwg` format (oxdraw), or also `plb`, `png`, `tex` and others (gnudraw)
- Can show the graph on the screen (professional version of Ox)
- Close the graph if necessary before continuing