

Advanced Programming in Quantitative Economics

Introduction, structure, and advanced programming techniques

Charles S. Bos

VU University Amsterdam
Tinbergen Institute

`c.s.bos@vu.nl`

15 – 19 August 2011, Aarhus, Denmark

Outline

Include packages

Magic numbers

Declaration files

Alternative: Command line arguments

OxDraw

Speed

Day 4 - Morning

9.00L Topics

- ▶ Finish standard errors, see Day 3b
- ▶ 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

Day 4 - Afternoon

12.00 Lunch

14.15 Empty slot?

16.00L Added capabilities

17.00 The end for today

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	Univariate 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	Simulated Annealing
mc2pack.h	Bos	Markov chain Monte Carlo estimation
msvar	Krolzig	Markov switching
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

```
ox1 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, 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 data, draw XY with X=julian date, and use `DrawAdjust(ADJ_AXISSCALE, AXIS_DATE)`;
- ▶ 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

Speed

- ▶ Use matrices, avoid loops
- ▶ Use the const argument qualifier
- ▶ Use built-in functions
- ▶ Optimise inner loop
- ▶ Avoid using 'hat' matrices/outer products over large dimensions
- ▶ Matrices are stored by row
- ▶ Link in C or Fortran code

Speed: Loops vs matrices

Avoid loops like the plague.

Most of the time there is a matrix alternative, like for constructing dummies:

Listing 7: speed_loop2.ox

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

main()
{
    decl iN, iR, vY, vDY, i, r;

    iN= 10000;      iR= 1000;
    vY= rann(iN, 1); vDY= zeros(vY);

    TrackTime("Loop");
    for (r= 0; r < iR; ++r)
        for (i= 0; i < iN; ++i)
            if (vY[i] > 0)
                vDY[i]= 1;
            else
                vDY[i]= -1;

    TrackTime("Matrix");
    for (r= 0; r < iR; ++r)
        vDY= vY .> 0 .? 1 .: -1;
    TrackTime(-1);    TrackReport();
}
```

Speed: const

Listing 8: speed_const.ox

```
SomeFuncConst(const mX)
{ // Do nothing
}

SomeFuncNotConst(mX)
{ // Do nothing
}

main()
{
    decl iN, iK, iR, iRr, mX, r;

    ...
    mX= rann(iN, iK);
    for (r= 0; r < iR; ++r)
        SomeFuncNotConst(mX);

    for (r= 0; r < iR; ++r)
        SomeFuncConst(mX);
}
```

Speed: Built-in functions

Listing 9: speed_builtin.ox

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

MyOlsC(const vY, const mX, const avBeta)
{
    avBeta[0] = invert(mX' mX * mX' vY);

    return !ismissing(avBeta[0]);
}

main()
{
    decl iN, iK, iR, vY, mX, vBeta, r;

    // Generate regression data
    ...

    for (r= 0; r < iR; ++r)
        MyOlsC(vY, mX, &vBeta);

    for (r= 0; r < iR*iRr; ++r)
        ols2c(vY, mX, &vBeta);
}
```


Speed: Inner/optimize

Target: Compute $SSR = y - X\beta$ for a series of q different vectors of β 's:

Listing 10: speed_outer.ox

```

...
mBeta= vBeta + ranu(iK, iQ);
mE= vY - mX*mBeta;

TrackTime("Outer");
for (r= 0; r < iR; ++r)
    vE2= diagonal(mE'mE);

TrackTime("Inner - matrix");
for (r= 0; r < iR; ++r)
    vE2= sumc(mE .* mE);

TrackTime("Inner");
for (r= 0; r < iR; ++r)
    vE2= sumsqrc(mE);
TrackTime(-1);

```

Speed: Rows/columns

Target: Successive fill either a large row or column of a matrix

Listing 11: speed_rows.ox

```
...
iN= 10;           // Size of matrix
iK= 10000;
iR= 100;         // Number of repetitions

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

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

Speed: Concatenation or predefine

In a simulation with a matrix of outcomes, predefine the matrix to be of the correct size, then fill in the rows.

The other option, concatenating rows to previous results, takes a lot longer.

Listing 12: speed_concat.ox

```
...
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);
```

Speed: Ox vs C vs more optimised C

Replace heavy loops for C-code

Listing 13: speed_c.ox

```
...  
TrackTime("SsfLik Ox");  
for (r= 0; r < iR; ++r)  
    ir= SsfLikOx(&dLnLik, &dVar, vY, mPhi, mOmega, mSigma);  
  
TrackTime("SsfLik C");  
for (r= 0; r < iR; ++r)  
    ir= SsfLik(&dLnLik, &dVar, vY, mPhi, mOmega, mSigma);  
  
TrackTime("SsfLikEx optimised C");  
for (r= 0; r < iR; ++r)  
    ir= SsfLikEx(&dLnLik, &dVar, vY, mPhi, mOmega, mSigma);
```

Speed: Overview

Method	I	II	III
Loop vs matrix	95	5	
Const argument	100	0	
Built-in function	69	31	
Inner loop/hat matrices	60	32	8
Columns vs rows	75	25	
Concatenation vs predefined matrix	98	2	
Ox vs C vs more optimised C	92	5	3

Conclusions:

- ▶ If your program takes more than a few seconds, optimise
- ▶ Track the time spent in parts of the program, optimise what takes longest
- ▶ Declare your arguments as `CONST`
- ▶ C is a lot faster in *loops* than Ox, for matrices it doesn't matter much.