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

Day 1 - Morning

9.30 Introduction
- Target of course
- Science, data, hypothesis, model, **estimation**
- Bit of background

- Concepts of
  - Data, Variables, Functions, Addresses
- Programming by example
  - Gauss elimination
- (Installation/getting started)

11.00 Tutorial: Do it yourself

12.30 Lunch

Target of course

- Learn
- structured
- programming
- and organisation
- (in Ox or other language)

Not: Just learn more syntax...

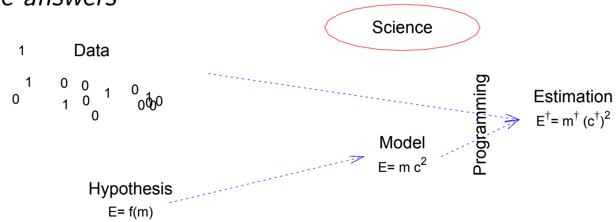
What? Why?

Wrong answer:

For the fun of it

A correct answer

To get to the results we need, in a fashion that is controllable, where we are free to implement the newest and greatest, and where we can be 'reasonably' sure of the answers



Aims and objectives

- Use computer power to enhance productivity
- Productive Econometric Research:
  combination of interactive modules and programming tools
- Data Analysis, Modelling, Reporting
- Accessible Scientific Documentation (no black box)
- Adaptable, Extendable and Maintainable (object oriented)
- Econometrics, statistics and numerical mathematics procedures
- Fast and reliable computation and simulation

Options for programming

	GUI	CLI	Program	Speed	QuanEcon	Comment
EViews	+	-	-	+/-	+	Black box, TS
TSMod	+	-	+/-	+/-	+	Alternative
Stata	+/-	+	-	-	-	Less programming
Matlab	+	+	+	+	+/-	Expensive, other audience
Gauss	+/-	+/-	+	+/-	+	'Ugly' code, unstable
S+/R	+/-	+	+	-	+/-	Graph +, speed -
Ox	+	+/-	+	+	+	Links to C, ectrics
C(++)/Fortran	-	-	+	++	-	Very quick, difficult

Here: Use Ox as environment, apply theory elsewhere

History

There was once...

C-Programmer   Memory leaks   Shell around C   Matrices

...and Ox was born.

More possibilities, also computationally:

Timings for OLS (30 observations, 4 regressors):

2009	Neh 2.67Ghz	64b	670.000†/sec
2008	Xeon 2.8Ghz	OSX	392.000†/sec
2006	Opt 2.4Ghz	64b	340.000†/sec
2006	AMD3500+	64b	320.000†/sec
2006	AMD3500+	4.04	273.000†/sec
2004	AMD3500+	3.40	218.000†/sec
2004	PM-1200		147.000†/sec
2001	PIII-1000		104.000†/sec
2000	PIII-500		60.000/sec
1996	PPro200		30.000/sec
1993	P5-90		6.000/sec
1989	386/387		300/sec
1981	86/87 (est.)		30/sec

Increase:

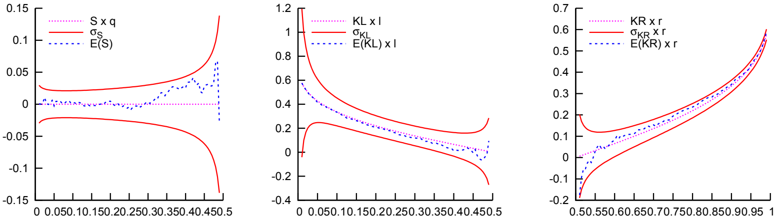
$\approx \times$ 1000 in 15 years

$\approx \times$ 10000 in 25 years.

Speed increase — but keep thinking

$$x \sim \mathrm{NIG}(\alpha, \beta, \delta, \mu) \qquad P(X < x) = \int_0^x f(z)dz = F(x) \qquad x_q = F^{-1}(q)$$

$$\mathcal{S}(q) = \frac{x_{1-q} + x_q - 2x_{\frac{1}{2}}}{x_{1-q} - x_q} \qquad 0 < q < \frac{1}{2}$$



Direct calculation of graph: > 40 min — Pre-calc quantiles: 5 sec

OxMetrics

A P P S	PcGive *+ x12arima* *+ PcNaive*	STAMP *+ SsfPack*	G@RCH	TSP	Ox Packages *DPD, MSVAR* *Arfima, etc.* *Ox programs*
C O R E	**OxMetrics** *interactive graphics* *data manipulation* *results storage* *code editor*				**Ox** *numerical programming* *computational engine* *interface wrapper*

What is programming about?

> *Managing* DATA, *in the form of* VARIABLES, *usually through a set of predefined* FUNCTIONS *or* ACTIONS

Of central importance: Understand *variables*, *functions* at all times...

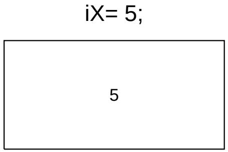
So let's exagerate

Variable

- A *variable* is an item which can have a certain *value*.
- Each variable has *one* value at each point in time.
- The value is of a specific *type*.
- A program works by managing *variables*, changing the *values* until reaching a final *outcome*

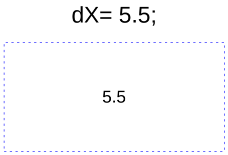
[Example: Paper integer 5]

Integer



- An integer is a number without fractional part, in between -2^{31} and $2^{31} - 1$ (limits are language dependent)
- Distinguish between the *name* and *value* of a variable.
- A variable can usually *change value*, but never *change its name*

Double



- A double is a number with possibly a fractional part.
- Note that 5.0 is a double, while 5 is an integer.
- A computer is not ‘exact’, careful when comparing integers and doubles
- If you add a double to an integer, the result is double (in Ox at least, language dependent)

[Example: `dAdd= 1/3; dD= 0; dD= dD + dAdd;` etc.]

String

- A character is a string of length one.
- A string is a collection of characters.
- The `"` are not part of the string, they are the *string delimiters*.
- One single element of a string, `sY[3]` for instance, is an integer, with the ASCII value of the character.
- Multiple elements of a string are a string as well, `sY[0:4]`, also `sX[0:0]` is a string.

[Example: `sX= "Hello world";`]

'Simple' types

- Integer
- Double
- Character/String

'Derived' type

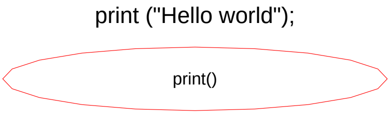
- boolean, integer 0 is `FALSE`, integer 1 is `TRUE`

[Example: `print (TRUE);`]

‘Difficult’ types

- Function
- Address
- Matrix
- Array
- File
- Object

Function



- A *function* performs a certain task, usually on a (number of) variables
- Hopefully the name of the function helps you to understand its task
- You can assign a function to a variable, `fnMyPrintFunction= print;`

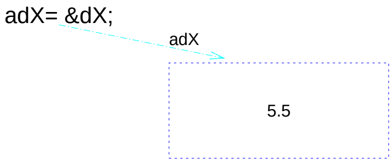
[Example: `fnMyPrintFunction("Hello world");`]

Address, real world

School of Economics and Management
University of Aarhus
Building 1322
DK-8000 Aarhus C

A building at the university The address

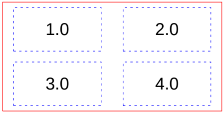
Address



- Now the *address* is the value (of variable `adX`)
- Any variable has an address (`&iX`, `&dX`, `&sX` etc)
- Each object exists only once: Whether I use `dX` or *what's at the address* `adX`, it is the same thing.

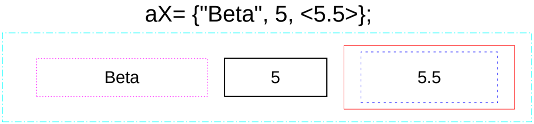
Matrix

```
mX= <1, 2; 3, 4>;
```



- A *matrix* is a collection of *doubles*.
- A matrix has two *dimensions*.
- A matrix of size $k \times 1$ or $1 \times k$ we tend to call a *vector*, `vX`.
- Later on we'll see how matrix operations can simplify/speed up calculations.

Array

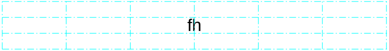


- An *array* is a collection of *other objects*.
- An array itself has one *dimension*.
- An element of an array can be of any type (integer, double, function, address, matrix, array)
- An array of an array of an array has *three* dimensions etc.

[Example: `aX= {};`]

File

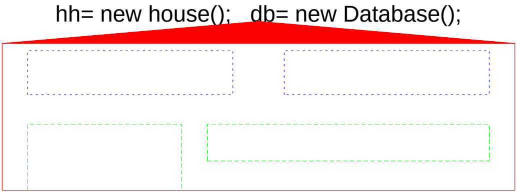
fh= fopen("data/aex-trades-0711.csv", "r");



- A *file* variable ‘points to’ an opened file
- This can be of use to read or write a file e.g. line-by-line
- Useful for successively writing results, or handling enormous data-files

[Example: `fh= fopen("data/mydata.csv", "r");`]

Object



- An *object* variable is an 'object'
- It can have certain characteristics or function members, which can be changed in turn. E.g. `hh.OpenWindow();` or `db.GetVar("Returns");`
- Useful for building higher level programs, with functionality hidden away in member functions.
- Communication of research (`Arfima` example)

Ox and other languages

Concepts are similar

- Ox (and Gauss/Matlab) have automatic typing. Use it, but carefully...
- C/C++/Fortran need to have types and sizes specified at the start. More difficult, but still same concept of variables.
- Precise manner for specifying a matrix differs from language to language. Ox rather similar to C in many respects
- Remember: An element has a value and a name
- A program moves the elements around, hopefully in a smart manner

Keep track of your variables, know what is their scope

All languages

Programming is exact science

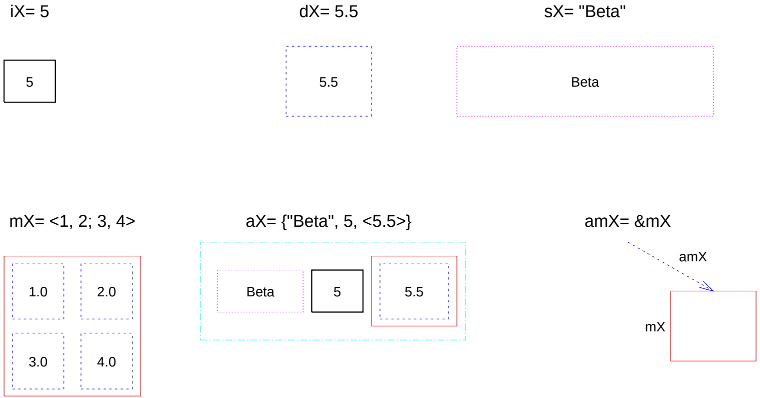
- Keep track of your variables
- Know what is their scope
- Program in small bits
- Program *extremely* structured
- Think about algorithms, data storage, outcomes etc.

Elements to consider

- Comments: `/* (block) */` or `// (until end of line)`
- Declarations: Up front in each routine
- Spacing
- Variables, types and naming in Ox:

scalar integer	`iN= 20;`
scalar double	`dC= 4.5;`
string	`sName="Beta1";`
matrix	`mX= <1, 2.5; 3, 4>;`
array of X	`aX= {1, <1>, "Gamma"};`
address of variable:	`amX= &mX;`
function	`fnFunc = olsr;`
class object	`db= new Database();`

Imagine elements



Every element has its representation in memory — no magic

Try out elements

Listing 1: oxelements.ox

```
#include <oxstd.h>

main()
{
  decl a, mX, sX;

  a= 5;
  println ("Integer: ", a);

  a= 5.5;
  println ("Double: ", a);

  a= sX= "Beta";
  println ("String: ", a);

  a= mX= <1, 2; 3, 4>;
  println ("Matrix: ", a);

  a= &mX;
  println ("Address of matrix: ", a);

  a= &sX;
  println ("Address of string: ", a);

  a= olsr;
  println ("Function: ", a);
}
```

Hungarian notation prefixes

prefix	type	example
`i`	integer	`iX`
`b`	boolean (`f` is also used)	`bX`
`d`	double	`dX`
`m`	matrix	`mX`
`v`	vector	`vX`
`s`	string	`sX`
`fn`	Function	`fnX`
`a`	array or address	`aX`
`as`	array of strings	`asX`
`am`	array of matrices	`amX`
`c`	class object variable	`cX`
`m_`	class member variable	`m_mX`
`g_`	external variable with global scope	`g_mX`
`s_`	static external variable (file scope)	`s_mX`

Use them *everywhere, always*.

Possible exception: Counters `i`, `j`, `k` etc.

Hungarian 2

Ox does not force Hungarian notation: Correct but *very* ugly is

Listing 2: oxnohun.ox

```
#include <oxstd.h>
main()
{
  decl sX, iX;

  iX= "Hello";
  sX= 5;
}
```

Instead, *always* use

Listing 3: oxhun.ox

```
#include <oxstd.h>
main()
{
  decl sX, iX;

  sX= "Hello";
  iX= 5;
}
```

Installation

1. Install the appropriate version (academic/professional), `http://www.doornik.com`, for Ox and possibly OxMetrics
2. Make the Ox documentation the homepage in your browser (`c:\program files\oxmetrics6\ox\doc\index.html`)
3. Install the necessary *tools* for OxEdit, if needed

Optional steps:

- Continue with downloading and installing extra packages `ssfpack`, `arfima`, `gnudraw`, `dpd` etc. into the Ox directory `c:\program files\oxmetrics6\ox\packages\ssfpack` etc, each in its own subdirectory below `ox\packages`.

Installation (advanced)

What if:

- No graphics, no OxMetrics license

Then:

- Install GnuDraw package with Ox, and
- Install GnuPlot (google it for a download) in `c:\program files\gnuplot`

Programming by example

- Enough theory
- Example: How to solve a system of linear equations
- Goal: Simple situation, program to solve it
- Broad concepts, details follow

Setup: Linear system

Solve for $\mathbf{x}$: $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, with

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Solution:

$$x_n = b_n / a_{nn}$$

$$x_i = \left(b_i - \sum_{j>i} a_{ij} x_j \right) / a_{ii}, \qquad i = n-1, .., 1$$

I.e.: Start at the end, solve backwards.

But ... *only works for upper triangular* $\mathbf{A}$...

Elimination

Hence: Create triangular matrix...

$$\begin{pmatrix} 2 & 1 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} 2 & 1 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Subtract multiple a_{jk}/a_{kk} times equation k from rows $j = k+1, ..., n$, such that $a_{jk}^{(k)} \equiv 0$.
Note: The x's don't change, only elements of $\mathbf{A}$ and $\mathbf{b}$.
Extended matrix:

$$(\mathbf{A}, \mathbf{b}) = \begin{pmatrix} a_{11} & \cdots & \cdots & a_{1n} & b_1 \\ a_{21} & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \vdots & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} & b_n \end{pmatrix}$$

Example elimination

$$[\mathbf{A}|\mathbf{b}] = \left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \\ 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \end{array}\right)$$

$$\overset{\text{iteration 1}}{\Leftrightarrow} [\mathbf{A}|\mathbf{b}]^{(1)} = \left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & -12 & 8 & 1 & -27 \\ 0 & 2 & 3 & -14 & -18 \end{array}\right)$$

Let's concentrate on one row at a time: How to eliminate the row starting with 12?

(See `ge0.ox`)

Program by Example 0

- Use commenting
- One main function: `main() {}`
- Declarations on top (...)
- Get the matrices, `mA= <1, 2; 3, 4>;`
- Concatenate, `mAB= mA ~ vB;`
- Debug $\rightarrow$ `println()`

Recognize *Magic Numbers*, initial settings

PbE 1: Eliminate a row

- What row/column are we working with? Start counting at 0...
- Calculate multiplicity
- Subtract a row at a time

PbE 2: Eliminate a row in a function

As we might want to eliminate more rows, it could be programmed as a separate function...

- Function header: Define what goes in/out
- Use commenting
- First use of address `amAB= &mAB;`

PbE 3: Eliminate multiple rows

- Use a loop around the function,
  `for (`**start condition**`; `**check**`; `**increment**`)`

PbE 4: Eliminate multiple columns

PbE 4: Eliminate multiple columns

- Use a loop around the loop. What columns should be eliminated?

PbE 5: Use another function

- Use a function to eliminate a column
- Call the function multiple times from the loop

Resulting program:

- Clean
- Readable chunks
- Debugging was done step by step, function/action at a time
- In future, functions are easily re-utilizable.

Chapter 1: Getting started

Exercise:

1. Copy the file <ox-home>/samples/myfirst.ox to your personal directory.
2. Open the file in OxEdit (e.g. Windows Explorer, walk there, right mouse button, Send To - OxEdit)
3. Run the program (through Modules - Run - Ox)

(If there is no Ox option under the Run menu, load the .tool file from the students directory, using Tools - Add/remove modules - Load from)

Output

```
Ox version 5.10 (Linux_64/MT) (C) J.A. Doornik, 1994-2008
two matrices
       2.0000       0.0000       0.0000
       0.0000       1.0000       0.0000
       0.0000       0.0000       1.0000

       0.0000       0.0000       0.0000
       1.0000       1.0000       1.0000
```

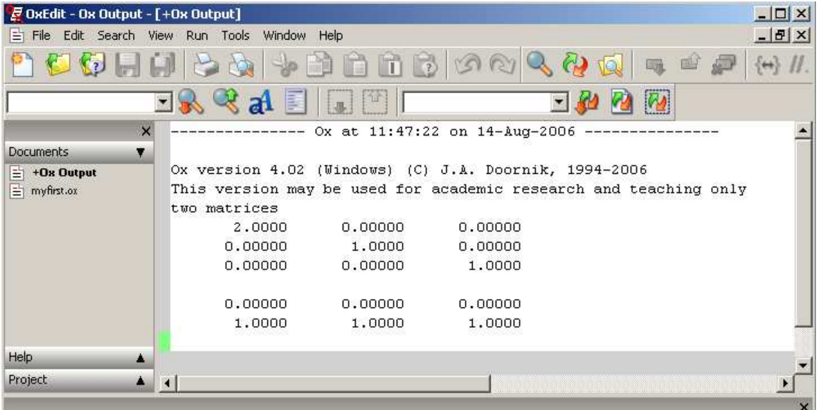
Using OxEdit

One tab has program

Running the program puts output in separate file/sheet

Errors in code can appear in output file

Workspace indicates opened files



Type of errors

1. Compilation errors: Like the above, error in the syntax of Ox

Listing 4: myfirst_err.ox

```
print "two matrices", m1, m2);
    // gives compile-time error
----------------
Ox version 5.10 (Linux_64/MT) (C) J.A. Doornik, 1994-2008
myfirst_err.ox (12): ';' expected but found '<string>'
myfirst_err.ox (12): ';' expected but found ')'
myfirst_err.ox (12): ')' out of place
```

2. Runtime errors: Impossible computations or commands

Listing 5: myfirst_err.ox

```
print ("product of two matrices", m1 * m2);
    // gives run-time error
----------------
Ox version 5.10 (Linux_64/MT) (C) J.A. Doornik, 1994-2008
...
Runtime error: 'matrix[3][3] * matrix[2][3]' bad operand
Runtime error occurred in main(14), call trace:
myfirst_err.ox (14): main
```

One error can lead to multiple messages: Start solving first in list.

Chapter 2: Syntax - Comments

```
/*  This is standard comment,
    which /* may be nested */.
*/
decl x;  // declare the variable x
```

Use them well, use them extensively, use them consistently

```
/*
**  olsc(const mY, const mX, const amB)
**
**  Purpose:
**    Performs OLS, expecting the data in columns.
**
**  Inputs:
**    mY        iT x iN matrix of regressors Y
**    mX        iT x iK matrix of explanatory variables X
**
**  Outputs:
**    amB       address of iK x iN matrix with iN sets of OLS coefficients
**
**  Return value:
**    integer, 1: success, 2: rescaling advised,
**              -1: X'X is singular, -2: combines 2 and -1.
**
**  Example:
**    ir = olsc(mY, mX, &mB);
**
**  Last changed
**    21-04-96 (Marius Ooms): made documentation
**    06-08-09 (Charles Bos): adapted documentation
*/
```

Use explanation, consistently, before *every* function, detailing *name, purpose, inputs, outputs, return value* (and possibly *date, author*, once per file)

Program layout

A minimal complete program is:

Listing 6: oxtut2b.ox

```
#include <oxstd.h>

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

Contains:

1. Include statement, to define all standard functions in Ox; between $<$ and $>$ to indicate `oxstd.h` is an intrinsic part of Ox
2. One function header, called `main`, taking no arguments `()`
3. Function body for `main()`, enclosed in `{}`, with a `println` statement

Note: Syntax terribly similar to C or Java.

Statements

Listing 7: oxtut2c-hun.ox

```
#include <oxstd.h>

main()
{
  decl iN, dSigma, mX, vBeta, vEps;

  iN = 4;
  dSigma = 0.25;
  mX = 1 ~ ranu(iN, 2);
  vBeta = <1; 2; 3>;

  vEps = dSigma * rann(iN, 1);

  print("x", mX, "beta", vBeta, "epsilon", vEps);
}
```

(note: Stick to Hungarian, don't follow the *Introduction to Ox* literally here)

- Declaration: Automatic typing
- Assignment: Integer, double, matrix-function, matrix-constant, function result.
- Print statement