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 - Afternoon

13.30 Structuring your thoughts
- What is programming?
- Preparation of a program

14.30 Tutorial: Do it yourself
- Exercise to hand in
- Work through 'Introduction to Ox Ch 1-5'

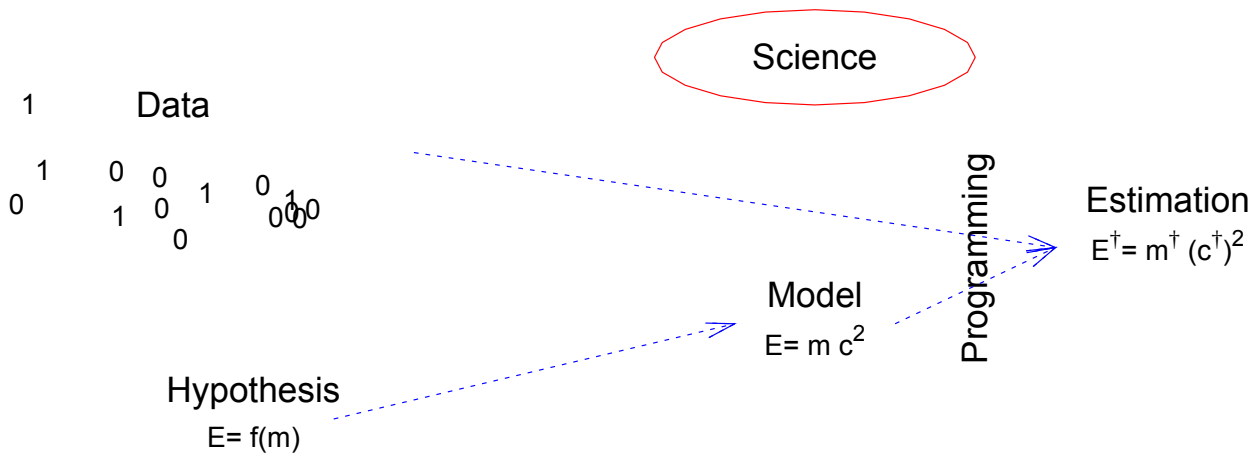
Repeat: What? Why?

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



Programming in Theory

Plan ahead

- Research question: What do I want to know?
- Data: What inputs do I have?
- Output: What kind of output do I expect/need?
- Modelling:
  - What is the structure of the problem?
  - Can I write it down in equations?
- Estimation: What procedure for estimation is needed (OLS, ML, simulated ML, GMM, nonlinear optimisation, Bayesian simulation, etc)?

Closer to practice

Blocks:

- Is the project separable into blocks, independent, or possibly dependent?
- What separate routines could I write?
- Are there any routines available, in my own old code, or from other sources?
- Can I check intermediate answers?
- How does the program flow from routine to routine?

... names:

- How can I give functions and variables names that I am sure to recognise later (i.e., also after 3 months)?
  Use (always) **Hungarian notation**

Even closer to practice

Define, on paper, for each routine/step/function:

- What inputs it has (shape, size, type, meaning), exactly
- What the outputs are (shape, size, type, meaning), also exactly...
- What the purpose is...

Also for your main program:

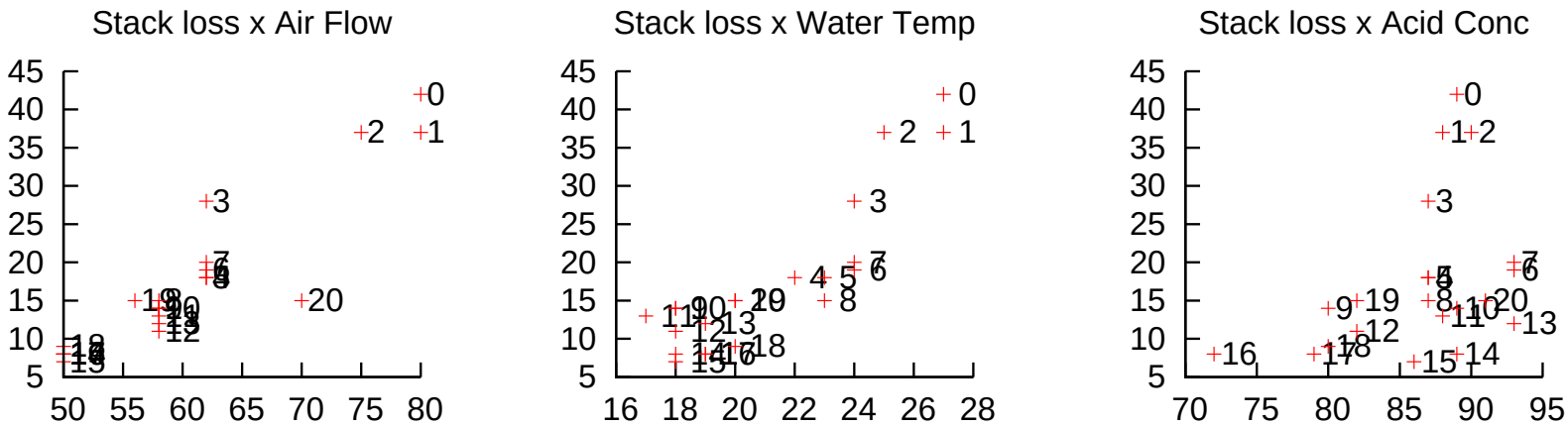
- Inputs can be *magic numbers*, (name of) *data file*, but also specification of model
- Outputs could be screen output, file with cleansed data, estimation results etc. etc.

Intermezzo: Stack-loss data

21 Observations on 4 variables (source: Brownlee (1965)).
It concerns the oxidation of ammonia to nitric acid, as a function of air flow, water temperature, and acid concentration.
See also: Justel & Peña (1996)
Regression model - outliers? - OLS - standard deviation - R^2
Data:

```
80 80 75 62 62 62 ...
27 27 25 24 22 23 ...
89 88 90 87 87 87 ...
42 37 37 28 18 18 ...
```

Intermezzo II: What to do?



Data: Stack loss $= f$(Air flow, Water temperature, Acid concentration)

As a starter: Look at the data.
Check which series is which, ranges, means, outliers, transformations etc.

Conclusion: Regression, indeed?

Intermezzo III: What to do?

Model	$y_i = X_i\beta + u_i, u_i \sim \mathcal{N}(0, \sigma^2)$
Estimation	$\hat{\beta} = (X'X)^{-1}X'y$
Data	$y_i =$ Line **0** of data in `data/stackloss.mat`, running from 7-42
	$X_i =$ Lines **1-3** of data; and constant!

Ugly programming:

Listing 1: stack/stackols_ugly

```
#include <oxstd.h>
main()
{
decl y= <42;37;37;28;18;18;19;20;15;14;14;13;11;12; 8; 7; 8; 8; 9;15;15>;
decl x= <1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;80,80,75,62,62,62,62,
62,58,58,58,58,58,58,50,50,50,50,50,56,70;27,27,25,24,22,23,24,24,23,18,
18,17,18,19,18,18,19,19,20,20,20;89,88,90,87,87,87,93,93,87,80,89,88,82,
93,89,86,72,79,80,82,91>';
print((x'x)^(-1)*x'y);
}
```

Errors...

Intermezzo III: Nicer programming

```
**  StackOls
**
**  Purpose:
**    Estimate a regression model on the stackloss data set
**
**  Inputs:
**    The program expects the file data/stackloss.mat to contain
**    the data, with size information
**
**  Author:
**    Charles Bos
**
**  Date:
**    16/2/2005
*/
#include <oxstd.h>

main()
{
  decl mStackloss, vY, mX, ir, vBeta;

  mStackloss= loadmat("data/stackloss.mat");    // Load the data
  vY= mStackloss[3][];                          // Read out row 3
  mX= mStackloss[0:2][];                        // Read out row 0-2
  mX= 1|mX;                                     // Append a constant
  ir= olsr(vY, mX, &vBeta);                     // Run OLS on rows

  print ("Ols estimates of Beta: ", vBeta);
}
```

Elements to consider

- Explanation: Be generous (enough)

Listing 2: stack/stackols.ox

```
/*
**  NAME
**
**  Purpose:
**    Short description of main idea
**
**  Inputs:
**    Clearly indicate what should have been prepared
**
**  Author:
**    Who am I
**
**  Date:
**    When did I write this version...
*/
#include <oxstd.h>

main()
{
  ...
}
```

Elements to consider II

- Explanation: Be generous (enough)
- Initialise from main

Listing 3: stack/stackols.ox

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

main()
{
  // Initialisation
}
```

Elements to consider III

- Explanation: Be generous (enough)
- Initialise from main
- Then do the estimation

Listing 4: stack/stackols.ox

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

main()
{
  // Initialisation

  // Estimation
}
```

Elements to consider IV

- Explanation: Be generous (enough)
- Initialise from main
- Then do the estimation
- ... and give results

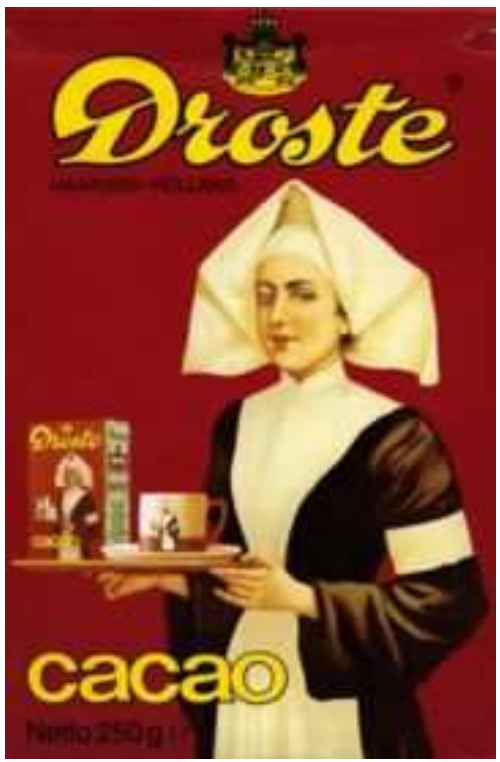
Listing 5: stack/stackols.ox

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

main()
{
  // Initialisation

  // Estimation

  // Results
}
```

NB: These steps are usually split into separate functions

The 'Droste effect'

- The program performs a certain function
- The main function is split in three (here)
- Each subtask is again a certain function that has to be performed



Apply the Droste effect:

- Think in terms of functions
- Analyse each function to split it
- Write in smallest building blocks

Preparation of program

What do you do for preparation of a program?

1. Turn off computer
2. On paper, analyse your inputs
3. Transformations/cleaning needed? Do it in a separate program...
4. With input clear, think about output: What do you want the program to do?
5. Getting there: What steps do you recognise?
6. Algorithms
7. Available software/routines
8. Debugging options/checks

Work it all out, before starting to type...

KISS

KISS

Keep it simple, stupid

Implications:

- Simple functions, doing one thing only
- Short functions (one-two screenfuls)
- With commenting on top
- Clear variable names (but not too long either)
- Consistency everywhere
- Catch bugs before they catch you

Reference:
`http://kerneltrap.org/files/Jeremy/CodingStyle.txt`

KISS: Example

Remember Gauss elimination:

- Eliminate a matrix ≡
- $(K-1)\times$ [eliminate a column ≡
- $(K-k)\times$ [eliminate a single row ≡
- subtracting f times row k]]

Separate actions, separately programmed, *each debugged separately*