Advanced Programming in Quantitative Economics

Introduction, structure, and advanced programming techniques

Charles S. Bos

VU University Amsterdam
Tinbergen Institute

cbos@feweb.vu.nl

23 – 27 August 2010, 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'

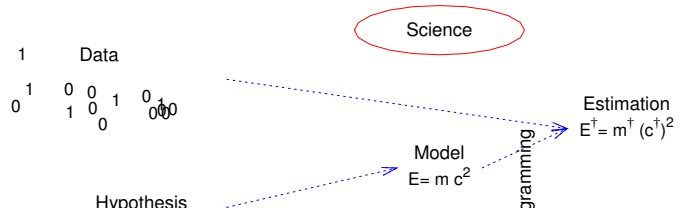
Repeat: What? Why?

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.

Elements to consider

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

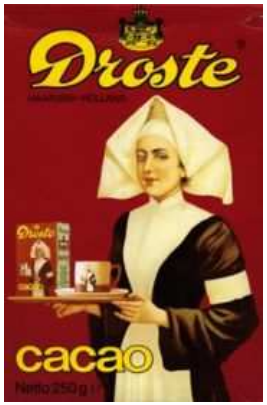
Listing 1: stack/stackols.ox

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

main()
{
  // Magic numbers, and 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*