

# *Computer Programming in Econometrics*

*Introduction, structure, and advanced programming  
techniques*

*Tutorial 5/6 October 2009, Tinbergen Institute*

Charles Bos

`cbos@feweb.vu.nl`

VU University Amsterdam

Tinbergen Institute

## Day 2 - Afternoon

---

### 13.30 Practical (at VU, 3A05)

- Likelihood implementation
- Optimization
- St. dev
- Restrictions

## Regression model

---

For today, start simple (this will be a good starter for the final exercise, saves time).

Setting:

$$y_i = X_i\beta + \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$$\mathcal{L}(y; \beta, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - X\beta)'(y - X\beta)}{2\sigma^2}\right)$$

Choose your own 'magic numbers' for  $\beta, \sigma, n$ .  
Derive the loglikelihood for this model.

## Step one: Generate data

---

Create a program file, with your magic numbers, with two routines:

1. `GenrX(const amX, const iN, const iK)`, creating a matrix  $X$  of the correct size, with a constant and numbers of choice
2. `GenrData(const avY, const mX, const vBeta, const dSigma)`, simulating the observations

Check that it works: Print or plot the data, use `olsc()` to estimate, see if  $\beta$  can be estimated.

Only afterwards...

## Regression likelihood in steps

---

Perform, in steps, for instance

1. Get the outline of your loglikelihood function. Call it from main, with a valid vector of parameters, and set the likelihood value equal to the average of your  $y$ 's. Think of `static decl` for global data variables  $y$  and  $X$ .
2. Extract the vector of parameters, into  $\beta$  and  $\sigma$ . Print them separately from the loglikelihood function.
3. Check the value of  $\sigma$ . If negative, maybe return a zero?
4. Construct a vector of residuals  $e = y - X\beta$ . Does this work?
5. Construct full (average!) loglikelihood function. Does the value seem 'logical'?
6. Run `MaxBFGS( )`. What return value `ir` do you get, what does it mean? What is `vP`? Does it look similar to  $\hat{\beta}$  from OLS? What is  $\hat{\sigma}$ ?

## Standard errors

For the standard errors, you had to find

$$\Sigma(\hat{\theta}) = -H(\hat{\theta})^{-1}, \quad H(\hat{\theta}) = \left. \frac{\delta^2 l(Y; \theta)}{\delta \theta \delta \theta'} \right|_{\theta = \hat{\theta}}$$

Some standard code could look like

```
ir= MaxBFGS(AvgLnLiklRegr, avP, adLL, 0, TRUE);  
if (Num2Derivative(AvgLnLiklRegr, avP[0], &mH))  
    mS2= invertgen(-mH, 30)/iN,  
avS[0]= sqrt(diagonal(mS2)');
```

*stack/eststack\_ml.ox*

7. Get the standard errors with it. How do they change if you only use the first 10 observations?
8. Beautify the output: Get a nice print with the maximum likelihood you find, the type of convergence, the parameters, standard errors and  $t$ -values (defined as  $t = \hat{\beta} / \hat{s}_{\beta}$ )

## Restriction: $\sigma > 0$ ?

What happens with your program when  $\sigma < 0$ , e.g. as a starting value?  
Robustify, preparing functions

```
TransPars(const avPTr, const vP)  
TransBackPars(const avP, const vPTr)
```

Now optimize the transformed parameter vectors, transforming back and forth where needed:

1. Find normal starting values
2. Transform
3. Optimize  $vPTr$  using `MaxBFGS`
4. In loglikelihood function, receive  $vPTr$ , transform back
5. After `MaxBFGS`, transform back optimal  $vPTr$
6. Print  $vP$

What about standard errors?

## Extra: Analytical score

---

As an extra: Work out the analytical score for the model

9. Find it on paper
10. In the program, add the necessary code (only compute it if asked for it, if `avScore` is an address!)
11. Check it, contrasting your analytical score to `Num1Derivative`
12. Evaluate the number of function evaluations needed when using numerical scores, and when using analytical scores (hint: Define an extra global variable, `s_iEval`)

## 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.

```
#include <oxdraw.h> stack/drawstack.ox
// #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` or `gwg` format (`oxdraw`), or `eps`, `plb`, `png`, `tex` and others (`gnudraw`)
- Can show the graph on the screen (professional version of Ox)
- Close the graph if necessary before continuing

## On graphs...

This means: Some plots could be made by

```
// Area=0, y axis: row of y values, label: "y", start index=1,  
// increment=1, draw marks(=1) instead of lines (0), with color 2  
DrawMatrix(0, vY', "y", 1, 1, 1, 2);  
  
// A density plot of y, together with normal approximation  
DrawDensity(1, vY', "y", 1, 0, 1);  
  
// A cross-plot against one of the x's  
DrawXMatrix(2, vY', "y", mX[][1]', "x1", 1, 2);  
ShowDrawWindow();
```