

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

1/33

Day 3 - Morning

9.00L Optimization

- ▶ Newton-Raphson and quadratic optimisation
- ▶ Hessian: Importance and problems
- ▶ Loglikelihood and covariance
- ▶ MaxBFGS

10.30P Estimating a duration model

- ▶ Likelihood
- ▶ Optimization
- ▶ Covariance

12.00 Lunch

3/33

Outline

Optimization

Optimization pitfalls

Maximize

Standard deviations

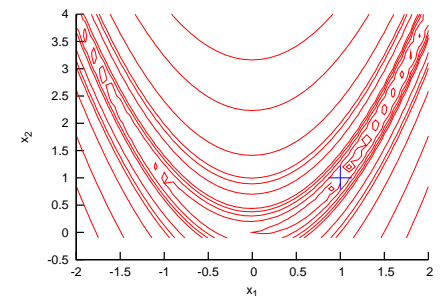
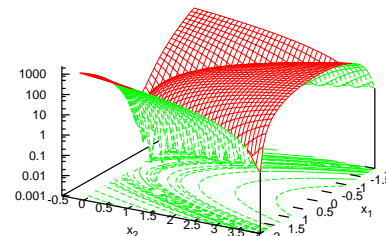
2/33

Maximization: Theory

Rosenbrock function:

$$g(x) = 100 * (x_2 - x_1^2)^2 + (1 - x_1)^2$$

$$f(x) = -g(x)$$



Minimum of $g(x)$ at $(1, 1)$ — Steep function — minimum in 'narrow crooked alley'

4/33

Wrong approach: Purely random

Suppose:

- ▶ Search in area $[-10, 10]^2$
- ▶ Throw darts randomly
- ▶ Continue until you get precision $\epsilon = 0.25$ from minimum, in both coefficients

How many darts would you need?

$$P(|x_i - x_i^*| < \epsilon) = 2 \frac{\epsilon}{20} = .025$$

$$P(|x_i - x_i^*| < \epsilon, i = 1, 2) = P(|x_i - x_i^*| < \epsilon)^2 = 4 \frac{\epsilon^2}{20^2} = 0.000625$$

$$E(n) = 1/p = 1600$$

Way too many...

See `rb_darts.ox`

5/33

Ingredients

$$f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$$

Function

$$f'(x) = \nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_n} \right]^T \equiv g \quad \text{Derivative, gradient}$$

$$f''(x) = \nabla^2 f(x) = \left[\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right]_{i,j=1}^n \equiv H \quad \text{Second derivative, Hessian}$$

If derivatives are continuous, then

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i} \quad H = H^T$$

Hessian symmetric

7/33

Better approach: Use function characteristics

- ▶ Start in some point $x^{(k)}$
- ▶ Choose a direction s
- ▶ Move distance α in that direction, $x^{(k+1)} = x^{(k)} + \alpha s$
- ▶ Increase k , and possibly continue from 1

Direction s : Linked to gradient?

Optimum: Gradient 0, second derivative positive definite?

6/33

Newton-Raphson

- ▶ Approach $f(x)$ locally with quadratic function

$$f(x+h) \approx q(h) = f(x) + h^T f'(x) + \frac{1}{2} h^T f''(x) h$$

- ▶ Minimise $q(h)$ (instead of $f(x+h)$)

$$q'(h) = f'(x) + f''(x)h = 0 \quad \Leftrightarrow \quad f''(x)h = -f'(x) \quad \text{or} \quad Hh = -g$$

by solving last expression, $h = -H^{-1}g$

- ▶ Choose $x = x + h$, and repeat as necessary

Problems:

- ▶ Is H positive definite/invertible, at each step?
- ▶ Is step h , of length $\|h\|$, too big or small?

8/33

Problematic Hessian?

Algorithms based on NR need $H^{(k)} = f''(x^{(k)})$. Problematic:

- ▶ Taking derivatives is not stable (...)
- ▶ Needs many function-evaluations
- ▶ H not guaranteed to be positive definite

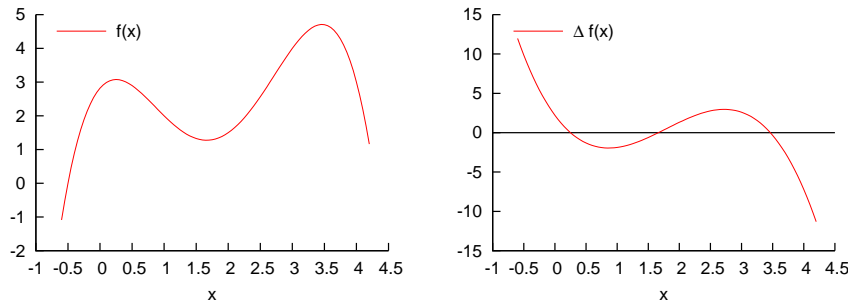
Problem is in step

$$s_k = -H^{(k)^{-1}} g_k$$

Replace $H^{(k)^{-1}}$ by some M_k , positive definite by definition?

9/33

Pitfalls I



- ▶ Local optima: Different starting points? Simulation-based optimisation (simulated annealing, anyone?)

11/33

BFGS

Broyden, Fletcher, Goldfarb and Shanno (BFGS) thought of following trick:

1. Start with $k = 0$ and positive definite M_k , e.g. $M = I$
2. Calculate $s_k = -M_k g_k$, with $g_k = f'(x^{(k)})$
3. Find new $x^{(k+1)} = x^{(k)} + h_k$, $h_k = \alpha s_k$
4. Calculate, with $q_k = g_{k+1} - g_k$

$$M_{k+1} = M_k + \left(1 + \frac{q_k' M_k q_k}{h_k' q_k}\right) \frac{h_k h_k'}{h_k' q_k}$$

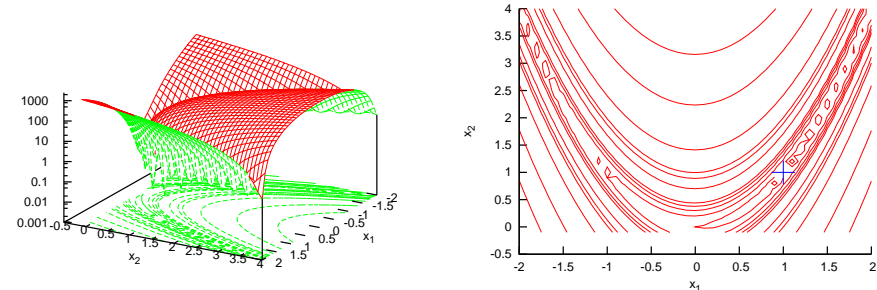
Result:

$$-\frac{1}{h_k' q_k} (h_k q_k' M_k + M_k q_k h_k')$$

- ▶ No Hessian needed
 - ▶ Still good convergence
 - ▶ No problems with negative definite H_k
- ⇒ MaxBFGS in OX, similar routines in Matlab/Gauss/other.

10/33

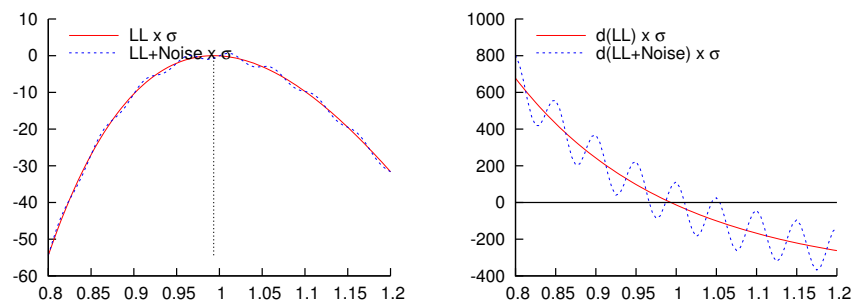
Pitfalls II



- ▶ Flat surface: Different starting points; transformation of parameter ($\theta = \log \sigma$ tends to optimise a lot better...)

12/33

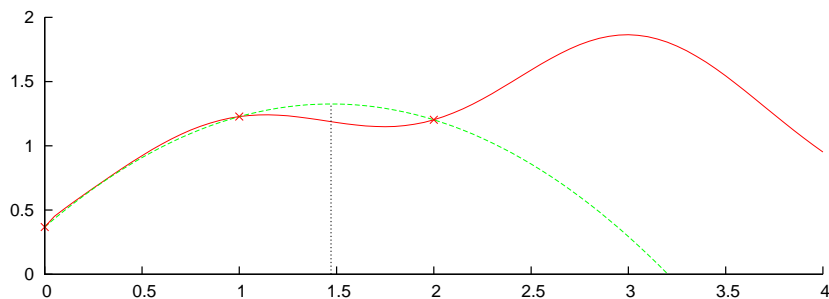
Pitfalls III



- ▶ Noise on derivatives/function: Only smooth functions can be optimised using NR-approach. Use analytical derivatives. Robust programming of target function (optimise AVERAGE LOG likelihood, never likelihood itself).

13/33

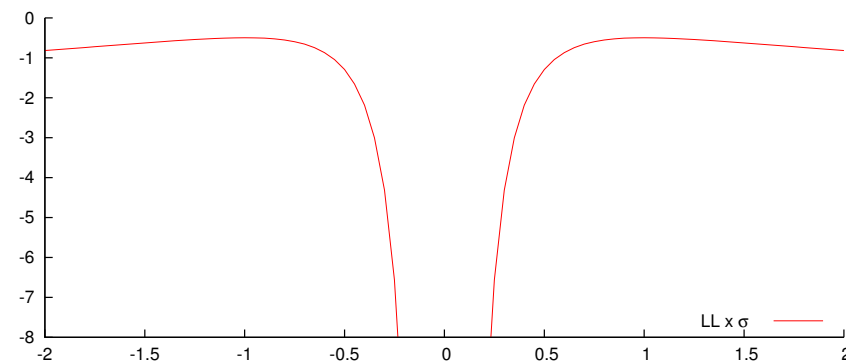
Pitfalls V



- ▶ Bad Taylor-approximation: Sometimes transformation of parameters helps a bit, or it is just bad luck

15/33

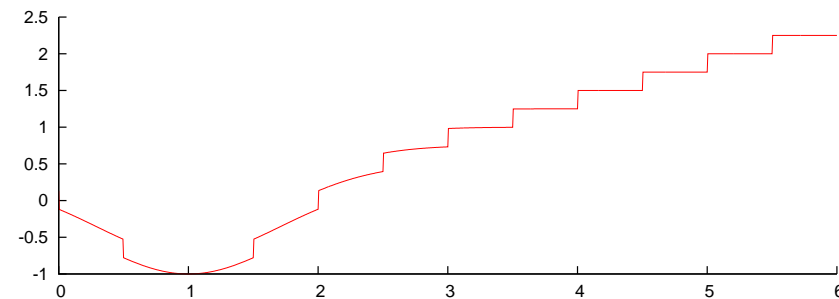
Pitfalls IV



- ▶ Non-uniqueness of optimum/indeterminacy in model specification: Optimise σ , both $\sigma = -1, \sigma = 1$ give same likelihood...

14/33

Pitfalls VI



- ▶ Other numerical trouble: AVERAGE LOG likelihood, check range of optimisor, use grid search for some parameters, estimation by parts, move to simulation-based optimisation, improve starting values...

16/33

A package: Maximize

Doing Econometrics \equiv estimating models, e.g.:

1. Optimise likelihood
2. Minimise sum of squared residuals
3. Mimimise difference in moments
4. Do Bayesian simulation, MCMC

Options 1-3 evolve around

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \pm f(y; \theta)$$

Inputs:

- ▶ f , use *average log* likelihood, or *average* (negative) sum-of-squares.
- ▶ Starting value θ_0
- ▶ Possibly f' , analytical first derivatives of f .

17/33

Maximize II

The maximize package provides features to maximize a function.

So maximize

$$\operatorname{AvgLnLRegr}(\theta) = LL(\theta)/N$$

First write (and test!) this function:

Listing 1: eststack_ml.ox

```
#include <oxstd.h>
#import <maximize>
// Two globals for likelihood function
static decl s_vY, s_mX;

AvgLnLklRegr(const vP, const adLnPdf, const avScore, const amHess)
{
    adLnPdf[0]= ...;

    return !ismissing(adLnPdf[0]); // Check if not missing
}
```

19/33

Maximize I: Regression

$$y_i = X_i\beta + \epsilon_i \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

ML maximises likelihood (other options: Minimise sum-of-squares, optimise utility etc):

$$\begin{aligned} L(y; \theta) &= \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - X_i\beta)^2}{2\sigma^2}\right) \\ &= (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left(-\frac{1}{2\sigma^2}(y - X\beta)'(y - X\beta)\right) \end{aligned}$$

In this case, $\theta = (\beta, \sigma^2)$

18/33

Maximize III

Function to optimize

```
/*
** AvgLnLklRegr(const vTheta, const adLnPdf, const avScore, const amHess)
**
** Purpose:
**   Compute average loglikelihood of the regression model
**
** Inputs:
**   vTheta           k+1 x 1 vector of parameters
**   s_vY             global, n x 1 vector of regressor
**   s_mX             global, n x k matrix of explanatory variables
**
** Outputs:
**   adLnPdf         address, on output the average loglikelihood
**   avScore          (not used for now)
**   amHess           (not used)
**
** Return value:
**   ir              boolean, TRUE if computation succeeded, FALSE otherwise
**
*/
```

20/33

Syntax

Function to maximize should have format

```
fnFunc(const vP, const adLnPdf, const avScore, const amHess)
```

- ▶ Choose your own logical function name
- ▶ vP is a $p \times 1$ COLUMN vector with parameters
- ▶ adLnPdf is a pointer to a variable, which on return should contain the function value, or a missing if function could not be evaluated
- ▶ avScore can be either 0 or a pointer. In the latter case the function should fill it in with the $p \times 1$ score vector
- ▶ amHess can be either 0 or a pointer, but isn't used in MaxBFGS
- ▶ **The function should return either TRUE (if the calculation succeeded) or FALSE, if not.**

21/33

Syntax II

Call MaxBFGS according to

```
#import <maximize>
ir= MaxBFGS(fnFunc, avP, adLnPdf, amInvHess, bNumDer);
```

- ▶ fnFunc is the name of the function
- ▶ avP is a pointer to the initial vector of parameters, on return it will contain the optimal parameters
- ▶ adLnPdf is a pointer which on return will contain the optimal value
- ▶ amInvHess can be 0 or pointer to $k \times k$ matrix with initial inverse Hessian estimate; on output it gives a (bad) estimate of this matrix
- ▶ bNumDer is a boolean, indicating if numerical derivatives have to be used

The return value ir indicates the type of convergence; MaxConvergenceMsg(ir) returns an intelligible message-string.

23/33

Writing the likelihood

Let's do it...

To remember:

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

$$\log L(y; \theta) = -\frac{1}{2} \left(N \log 2\pi + N \log \sigma^2 + \frac{e'e}{\sigma^2} \right)$$

In this case, $\theta = (\beta, \sigma)$ or $\theta = (\beta, \sigma^2)$.

- ▶ Start off your adLnPdf [0] at a missing (just so you won't forget giving it a value)
- ▶ Extract your parameters from the vector, use sensible names
- ▶ Check if your parameters are valid; if not ($\sigma^2 < 0$?) return FALSE
- ▶ At the end, test for a !ismissing(adLnPdf [0])
- ▶ Careful: This routine is used often; don't do unnecessary work
- ▶ And test...

22/33

Optimisation

Approach for general *criterion function* $Q(\beta)$: Write

$$Q^*(\beta) = Q(\beta_{(0)}) + g'_{(0)}(\beta - \beta_{(0)}) + \frac{1}{2}(\beta - \beta_{(0)})'H_{(0)}(\beta - \beta_{(0)})$$

$$g(\beta) = \frac{\partial}{\partial \beta} Q(\beta)$$

$$H(\beta) = \frac{\partial^2}{\partial \beta \partial \beta'} Q(\beta)$$

Optimise approximate $Q^*(\beta)$:

$$g_{(0)} + H_{(0)}(\beta - \beta_{(0)}) = 0$$

First order conditions

$$\Leftrightarrow \beta_{(1)} = \beta_{(0)} - H_{(0)}^{-1}g_{(0)}$$

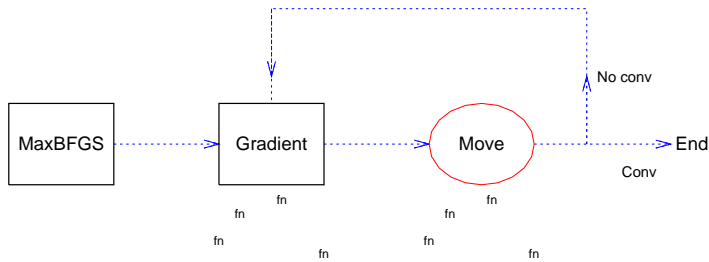
and iterate into oblivion.

24/33

MaxBFGS: Program flow

MaxBFGS follows quasi-Newton method according to Broyden, Fletcher, Goldfarb and Shanno (BFGS).

Can use either numerical or analytical first derivatives. Analytical derivatives are more robust, exact, quicker.



Maximize: Precision

Strong convergence is said to occur if (roughly):

- $|q_j^i \theta_j^i| \leq \epsilon_1, \forall j$, with q_j^i the j th element of the score at θ^i , at iteration i : Scores are relatively small.
- $\frac{|\theta_j^i - \theta_j^{i-1}|}{|\theta_j^i|} \leq 10\epsilon_1$: Change in parameter is relatively small

Note: This also depends on the scale of your parameters...

Preferably $\theta \approx 1$, not $\theta \approx 1e - 15$!

Adapt the precision with `MaxControlEps(const dEps1, const dEps2);`,

default is `dEps1= 1e-4, dEps2= 5e-3`.

Maximize: Average

Why use average loglikelihood?

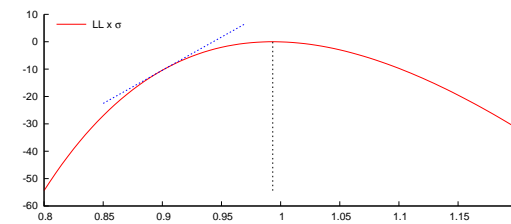
- Likelihood function $L(y; \theta)$ tends to have tiny values \rightarrow possible problem with precision
- Loglikelihood function $\log L(y; \theta)$ depends on number of observations: Large sample may lead to $|\text{large LL}|$, not stable
- Average loglikelihood tends to be moderate in numbers, well-scaled...

Better from a numerical precision point-of-view.

Warning:

Take care with score and standard errors (see later)

Maximize: Scores



Optimising \equiv 'going up'
 \equiv finding gradient.

Numerical gradient, for small h :

$$f'(\theta) = \frac{\partial f(\theta)}{\partial \theta} \approx \frac{f(\theta + h) - f(\theta)}{h} \approx \frac{f(\theta + h) - f(\theta - h)}{2h}$$

Function evaluations: $2 \times \text{dim}(\theta)$

Preferred: Analytical score $f'(\theta)$

Maximize: Scores II

```
AvgLnLklRegr(const vTheta, const adLnPdf, const avScore, const amHess)
{
  ...
  if (isArray(avScore)) // Check if score is requested
  {
    avScore[0]= ???; // Compute the score, in whatever way
  }
}
```

- ▶ Only compute the score when requested
- ▶ Test if avScore is an array (as this looks similar to an address)
- ▶ Or test if avScore is non-zero: if (avScore) ... , should do the same thing
- ▶ Work out vector of scores, of same size as θ .
- ▶ DEBUG! Check your score against Num1Derivative()

Maximize: Scores III

Let's do it...
 To remember:

$$f(y; \theta) = -\frac{1}{2} \left(\log 2\pi + 2 \log \sigma + \frac{e'e}{N\sigma^2} \right)$$

$$e = y - X\beta$$

$$\frac{\partial f(y; \theta)}{\partial \beta} = \dots$$

$$\frac{\partial f(y; \theta)}{\partial \sigma} = \dots$$

- ▶ In this case, it matters whether $\theta = (\beta, \sigma)$ or $\theta = (\beta, \sigma^2)$!
- ▶ Find score of AVERAGE loglikelihood

Maximize: Scores IIb

- ▶ ...
- ▶ DEBUG! Check your score against Num1Derivative()

```
#import <maximize>
...
AvgLnLklRegr(vTheta, &dLnPdf, &vS1, 0); // Compute analytical score
Num1Derivative(AvgLnLklRegr, vTheta, &vS2); // Compute numerical score

print ("Theta and scores: ",
      vTheta~vS1~vS2~(vS1-vS2)); // Compare scores
```

Don't ever forget debugging this
 (goes wrong 100% of the times...)

Standard deviations

Given a model with

$\mathcal{L}(Y; \theta)$	Likelihood function
$l(Y; \theta) = \log \mathcal{L}(Y; \theta)$	Log likelihood function
$\hat{\theta} = \operatorname{argmax}_{\theta} l(Y; \theta)$	ML estimator

what is the vector of standard deviations, $\sigma(\hat{\theta})$?
 Assuming correct model specification,

$$\Sigma(\hat{\theta}) = -H(\hat{\theta})^{-1}$$

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

SD2: Average likelihood

For numerical stability, optimise *average* loglikelihood.

For regression model, e.g. the stackloss model,

$$l(Y; \theta) = -\frac{(y - X\beta)'(y - X\beta)}{2\sigma^2} - N \log 2\pi\sigma^2 + c$$

$$\bar{l}(Y; \theta) = -\frac{(y - X\beta)'(y - X\beta)}{2N\sigma^2} - \log 2\pi\sigma^2 + c'$$

$$H_{\bar{l}} \equiv \frac{\delta^2 \bar{l}(Y; \theta)}{\delta\theta\delta\theta'} = \frac{1}{N} \frac{\delta^2 l(Y; \theta)}{\delta\theta\delta\theta'} \quad \hat{\Sigma}(\hat{\theta}) = \frac{1}{N} (-H_{\bar{l}})^{-1}$$

```

mS2= 0;
if (Num2Derivative(AvgLnLiklRegr, avP[0], &mH))
    mS2= invertgen(-mH, 30)/iN,

avS[0]= sqrt(diagonal(mS2)');

print ("MaxBFGS returns ", MaxConvergenceMsg(ir),
      " with LL= ", adLL[0]*iN,
      " at parameters ",
      "%c", {"Par", "Std"}, avP[0]~avS[0]);

```