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 3 - Afternoon

13.00L Optimization II

- Restrictions and transformations
- Delta method: Covariance estimation
- (fixing parameters)

14.30P Implementing covariance estimation

- Duration model restricting α and/or β
- Covariance of parameters

16.00 End

18.00 Course dinner at 'Sct. Oluf'

Optimization and restrictions

Take model

$$y = X\beta + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Parameter vector $\theta = (\beta', \sigma)'$ is clearly restricted, as $\sigma \in [0, \infty)$ or $\sigma^2 \in [0, \infty)$

- Newton-based method (BFGS) doesn't know about ranges
- Alternative optimization (SQP) *may be*(?) slower/worse convergence, but simpler

Hence: First tricks for `MaxSQP`/`MaxSQPF`.

Warning: Don't use `MaxSQP` unless you know what you're doing (the function looks attractive, but isn't always...)

Restrictions: MaxSQP

`MaxSQP` is an alternative to `MaxBFGS`

- Without restrictions, delivers exactly `MaxBFGS`
- Allows for sequential quadratic programming solution, for *linear* and *non-linear* restrictions.

```
#import <maxsqp>
MaxSQP(const func, const avP, const adFunc, const amHessian,
    const fNumDer, const cfunc_gt0, const cfunc_eq0, const vLo, const vHi);
MaxSQPF(const func, const avP, const adFunc, const amHessian,
    const fNumDer, const cfunc_gt0, const cfunc_eq0, const vLo, const vHi);
```

Restrictions:

1. `cfunc_gt0(const avF, const vP);` Fill `avF[0]`.
   Restriction: $f(p) > 0$
2. `cfunc_eq0(const avF, const vP);` Fill `avF[0]`.
   Restriction: $f(p) = 0$.
3. `vLo`: Restriction $p >$`vLo`
4. `vHi`: Restriction $p <$`vHi`

MaxSQP II

Advantages:

- Simple
- Implements restrictions on parameter space (e.g. $\sigma > 0, 0 < \alpha + \delta < 1$)

Disadvantages:

- BFGS is meant for *global* optimisation; `MaxSQP` might work worse
- Often better to incorporate restrictions in parameter transformation: Estimate $\theta = \log \sigma, -\infty < \theta < \infty$

So check out transformations...

Transforming parameters

Variance parameter positive?

Solutions:

1. Use σ^2 as parameter, have `AvgLnLiklRegr` return 0 when negative σ^2 is found
2. Use $\sigma \equiv |\theta_{k+1}|$ as parameter, ie forget the sign altogether (doesn't matter for optimisation, interpret negative σ in outcome as positive value)
3. Transform, optimise $\theta^*_{k+1} = \log \sigma \in (-\infty, \infty)$, no trouble for optimisation

Last option most common, most robust, neatest.

Transform: Common transformations

Constraint	θ^*	θ
$[0, \infty)$	$\log(\theta)$	$\exp(\theta^*)$
$[0, 1]$	$\log\left(\frac{\theta}{1-\theta}\right)$	$\frac{\exp(\theta^*)}{1+\exp(\theta^*)}$

Of course, to get a range of $[L, U]$, use a rescaled $[0, 1]$ transformation.

Note: See also exercise `transpar`

Transform: General solution

Distinguish $\theta = (\beta', \sigma)'$ and $\theta^* = (\beta', \log\sigma)'$. Steps:

- Get starting values θ
- Transform to θ^*
- Optimize θ^*, transforming back within LL routine
- Transform optimal θ^* back to θ

```
AvgLnLiklRegrTr(const vPTr, const adLnPdf, const avScore, const amHess)
{
  ...
  vBeta= vPTr[:iK-1];         // Extract parameters
  dS= exp(vPTr[iK]);          //   in terms of sigma
  ...
}

main()
{
  ...
  vP= zeros(iK, 1)|1;
  vPTr= vP[:iK-1]|log(vP[iK]);
  ir= MaxBFGS(AvgLnLiklRegrTr, &vPTr, &dLnPdf, 0, TRUE);
  vP= vPTr[:iK-1]|exp(vPTr[iK]);
}
```

Transform: Use functions

Notice code before: Transformations are performed

1. Before `MaxBFGS`
2. After `MaxBFGS`
3. Within `AvgLnLiklRegrTr`
4. And probably more often for computing standard errors

Premium source for bugs...

Solution: Define

- `TransPar(const avPTr, const vP)`: $\theta \rightarrow \theta^*$
- `ir= TransBackPar(const avP, const vPTr)` $\theta^* \rightarrow \theta$

And test (in a separate program) whether transformation works right. Necessary when using multiple transformed parameters.

Standard deviations

Remember:

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

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

Therefore, we need (average) loglikelihood in terms of θ, not θ^* for sd's...

Transforming parameters II: SD

Question: How to construct standard deviations?

Answers:

1. Use transformation in estimation, not in calculation of standard deviation. *Advantage*: Simpler. *Disadvantage*: Troublesome when parameter close to border.
2. Use transformation throughout, use Delta-method to compute standard errors. *Advantage*: Fits with theory. *Disadvantage*: Is standard deviation of σ informative, is its likelihood sufficiently peaked/symmetric?
3. After estimation, compute bootstrap standard errors
4. Who needs standard errors? Compute 95% bounds on θ, translate those to 95% bounds on parameter of interest. *Advantage*: Theoretically nicer. *Disadvantage*: Not everybody understands advantage.

See next slides.

Transforming: Temporary

Use global indicator `s_bTrans` indicating whether the parameters are transformed or not:

```
s_bTrans= TRUE;
TransPar(&vPTr, avP[0]);
println ("Transforming initial parameters ", avP[0]', " to ", vPTr');
ir= MaxBFGS(AvgLnLiklRegr, &vPTr, adLL, 0, TRUE);

// Get back standard parameters
TransBackPar(avP, vPTr);
s_bTrans= FALSE;
println ("Transforming back estimated parameters ", vPTr', " to ", avP[0]');
```

```
TransBackPar(const avP, const vPTr)
{
  ...
  avP[0]= vPTr;
  if (s_bTrans)
    avP[0][iQ-1]= exp(vPTr[iQ-1]);
  return TRUE;     // Correct transform? Needed for NumJacobian
}
```

`AvgLnLiklRegr(vPTr, ...)` takes parameters, and transforms them to normal parameters at the start.

Transforming: Delta

$$n^{1/2}(\hat{\theta} - \theta_0) \overset{a}{\sim} \mathcal{N}\left(0, V^{\infty}(\hat{\theta})\right)$$

$$\hat{\gamma} = g(\hat{\theta})$$

$$\hat{\gamma} \approx g(\theta_0) + g'(\theta_0)(\hat{\theta} - \theta_0)$$

$$n^{1/2}(\hat{\gamma} - \gamma_0) \overset{a}{=} g_0' n^{1/2}(\hat{\theta} - \theta_0) \overset{a}{\sim} \mathcal{N}(0, (g_0')^2 V^{\infty}(\hat{\theta})) \quad \text{scalar}$$

$$n^{1/2}(\hat{\gamma} - \gamma_0) \overset{a}{\sim} \mathcal{N}(0, G_0 V^{\infty}(\hat{\theta}) G_0') \quad \text{vector}$$

In practice: Use

$$\text{var}(\hat{\gamma}) = \hat{G}\,\text{var}(\hat{\theta})\hat{G}'$$

$$\hat{G} = \frac{\delta g(\theta)}{\delta \theta'} = \begin{pmatrix} \frac{dg(\theta)}{d\theta_1} & \frac{dg(\theta)}{d\theta_2} & \cdots & \frac{dg(\theta)}{d\theta_k} \end{pmatrix} = \text{Jacobian}$$

Transforming: Delta in Ox

Listing 1: stack/eststack_ml_tr2.ox

```
// Estimate transformed parameters
TransPar(&vPTr, avP[0]);
ir= MaxBFGS(AvgLnLiklRegr, &vPTr, adLL, 0, TRUE);

if (Num2Derivative(AvgLnLiklRegr, vPTr, &mH))
  mS2Th= invertgen(-mH, 30)/iN,

// Get matrix of first derivatives of transformation
NumJacobian(TransBackPar, vPTr, &mG);
mS2= mG * mS2Th * mG';

// Transform back parameters
TransBackPar(avP, vPTr);
```

Use `NumJacobian(TransBackPar, vPTr, &mG)` to compute Jacobian.

Note: `TransBackPar` needs to return `TRUE` if transformation succeeded.

Transforming: Bootstrap

- Estimate model, resulting in $\hat{\gamma} = g(\hat{\theta})$
- From the model, generate B bootstrap samples $y_j^s(\hat{\gamma})$
- For each sample, estimate $\hat{\gamma}_j^s = g(\hat{\theta}_j^s), \hat{\theta}_j^s = g^{-1}(\hat{\gamma}_j^s)$
- Report $\text{var}(\hat{\theta}) = \text{var}(\hat{\theta}_1^s, \ldots, \hat{\theta}_B^s)$

I.e, report variance/standard deviation among those B estimates of the parameters, assuming your parameter estimates are used in the DGP.

Simple, somewhat computer-intensive?

Transforming: Bootstrap in Ox

Listing 2: stack/eststack_ml_tr2_boot.ox

```
BootstrapS(const avS, const mX, const vP, const iB)
{
  ...
  for (i= 0; i < iB; ++i)
    {
      // Simulate data from DGP
      GenerateData(&vY, mX, vP);

      s_vY= vY;       // Prepare globals for AvgLnLiklRegr

      TransPar(&vPTr, vP);
      ir= MaxBFGS(AvgLnLiklRegr, &vPTr, &dLL, 0, TRUE);
      TransBackPar(&vPB, vPTr);

      mG[][i]= vPB;  // Record estimated parameters
    }
  mS2= variance(mG');
  avS[0]= sqrt(diagonal(mS2)');
}
```

For the tutorial: Try it out for the normal model?

Fixing parameters

What if some of the parameters are fixed?

1. Rewrite likelihood function for each special case
2. Keep track of fixed parameters in a flexible way

Obviously, option (2) is preferred. Ideas:

- Put full vector of parameters, including restricted, in global `s_vPar`
- Keep track of free parameters by their indices, `s_vIdxFreePar`
- In likelihood function, place free pars within full vector, `s_vPar[s_vIdxFreePar]= vP;`, and continue with `s_vPar` as if nothing happens.
- Use functions `FixPar(const dP, const iIdx)` and `FreePar(iIdx)` to fix or free parameters

Advantage: General, flexible, clean...

Fixing: Ox code

See live coding...

Other options

Study `stack/eststack.ox`, using

- choice of estimation methods
- log-file
- choice of output filename
- saving estimation results for later initialisation

What is meaning of

```
oxl eststack x Constant AirFlow met 1 sa load base output/stb
```