

Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

Vrije Universiteit Amsterdam
Tinbergen Institute

`c.s.bos@vu.nl`

August 2020 – Version Python

Separate lecture slides

Compilation: July 27, 2020

Overview

Principles of Programming in Econometrics

D0: Syntax, example 2⁸

D1: Structure, scope

D2: Numerics, packages

D3: Optimisation, speed

Day 3: Optimisation

9.30 Optimization (minimize)

- ▶ Idea behind optimization
- ▶ Gauss-Newton/Newton-Raphson
- ▶ Stream/order of function calls
- ▶ Standard deviations
- ▶ Restrictions
- ▶ Speed

13.30 Practical

- ▶ Regression: Maximize likelihood
- ▶ GARCH-M: Intro and likelihood

Optimisation

- ▶ Theory: What is (to be) done
- ▶ Inputs
- ▶ Practice/implementation
- ▶ Standard errors
- ▶ Transformations

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,

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

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

SD2: Average likelihood

For numerical stability, optimise *average negative* loglikelihood \bar{l}_n .

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

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

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

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

Listing 1: opt/lib/incstack.py

```
res= opt.minimize(AvgNlnLRegr, vP0, args=(vY, mX), method="BFGS")

mH= hessian_2sided(AvgNlnLRegr, res.x, vY, mX)
mS2= np.linalg.inv(mH)/iN
vS= np.sqrt(np.diag(mS2))
print ("\nBFGS results in ", res.message,
      "\nPars: ", res.x,
      "\nLL= ", -iN*res.fun, ", f-eval= ", res.nfev)
```

SD2: Hessian...

Hessian:

- ▶ is numerically unstable
- ▶ defines your standard errors
- ▶ hence is utterly important
- ▶ should be calculated with care!

But first: Check the gradient (simpler)

SD2: Gradient...

Gradient:

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

- ▶ Central difference *far* more precise than forward difference
- ▶ Step size h_i should depend on θ_i , different per element
- ▶ Rounding errors can become enormous, when h too small
- ▶ Python seems to provide `scipy.optimize.approx_fprime`, forward difference
- ▶ ... and symbolic differentiation (better, slower, not pursued here)

⇒ `lib/grad.py` contains `gradient_2sided()`

SD2: gradient_2sided

⇒ lib/grad.py contains gradient_2sided() (simplified here)

Listing 2: lib/grad.py

```
def gradient_2sided(fun, vP, *args):  
    iP = np.size(vP)  
    vP = vP.reshape(iP)      # Ensure vP is 1D-array  
  
    vh = 1e-8*(np.fabs(vP)+1e-8)  # Find stepsize  
    mh = np.diag(vh)           # Build a diagonal matrix  
  
    fp = np.zeros(iP)  
    fm = np.zeros(iP)  
    for i in range(iP):        # Find f(x+h), f(x-h)  
        fp[i] = fun(vP+mh[i], *args)  
        fm[i] = fun(vP-mh[i], *args)  
  
    vG = (fp - fm) / (2*vh)    # Get central gradient  
    return vG
```

SD2: Gradient II

Listing 3: opt/estnorm_score.py

```
vSc0= AvgNLnLRegr_Jac(vP0, vY, mX)
vSc1= opt.approx_fprime(vP0, AvgNLnLRegr, 1e-5*np.fabs(vP0), vY, mX)
vSc2= gradient_2sided(AvgNLnLRegr, vP0, vY, mX)
print ("\nScores:\n",
      pd.DataFrame(np.vstack([vSc0, vSc1, vSc2]), index=["Analytical", "grad_1sided", "
```

results in

```
Scores:
          0          1          2          3
Analytical -7.965135 -2.863504 -1.502223 -1.341437
grad_1sided -7.965005 -2.863499 -1.502222 -1.341435
grad_2sided -7.965135 -2.863504 -1.502223 -1.341437
```

Q: What do you prefer?

SD2: Hessian II

Back to Hessian:

- ▶ `lib/grad.py` contains `gradient_2sided()` and `hessian_2sided()` (source: [Python for Econometrics](#), Kevin Sheppard, with minor alterations)
- ▶ **DO NOT** use `scipy.misc.derivative`, as it allows only for a single constant difference h , applied in all directions
- ▶ **DO NOT EVER** use the output from `res = opt.minimize()`, where `res.hess_inv` seems to be some inverse hessian estimate. (Indeed, it is *some* estimate, useful for BFGS optimisation, not for computing standard errors)
- ▶ (Same result can be obtained from [NumDiffTools](#). However, here you have to understand what you are doing...)

Conclusion:

1. For standard errors: Feel free to copy code
2. Possibly better: Use improved covariance matrix, sandwich form. See Econometrics course