Advanced Programming Tricks

Tyler Ransom

Duke University

June 19, 2012

Plan

- Discuss tips on what to do when optimization fails and you know your code doesn't have a mistake
- Discuss how to recover estimates and standard errors when a user has imposed constraints in her objective function
- Introduce some more advanced, helpful commands

Programming Tricks

- Optimization is more of an art than a science
- You need to know what to do when your optimizer isn't cooperating
- Example:
- Starting values make a huge difference
- The closer your starting values are to the "answer", the better optimizer will perform

General Tips

- Tip 1: Provide as good of starting values as possible
- Use Matlab built-in functions like `regress`, `glmfit`, and `mnrfit` which estimate a wide range of standard econometric models
- Even if your objective function is slightly different (e.g. a slightly modified logit), providing starting values from `mnrfit` output is likely to be better than random noise

General Tips

- Tip 2: "Trick" the optimizer into always giving a "good" guess for certain parameters by imposing a constraint
- Example: In normal MLE regression, a guess of $\sigma \leq 0$ results in an undefined likelihood function and Matlab exiting immediately
- Make a transformation of σ, e.g. $\sigma' = \exp(\sigma)$
- This is always positive, so Matlab will never crash because of a nonpositive σ

Comment

- Could also use `fmincon` with constraint that σ be positive
- I've had limited success with `fmincon` for this type of use
- At the end of optimization, Matlab spits out, e.g. $\hat{\sigma}'$ instead of $\hat{\sigma}$. It also spits out the standard error of $\hat{\sigma}'$ instead of the SE of $\hat{\sigma}$
- After you're done with optimization, you need to make a transformation to both the point estimate and the hessian in order to recover the true parameter estimate and standard error
- We'll go through how to do this a bit later

General Tips

- Tip 3: "Trick" the objective function into returning a really bad value if it tries to guess a parameter value you know is bad
- Example: In the normal MLE example, if Matlab guesses $\sigma \leq 0$, recode the likelihood function to return $+10^4$ (or some other large number)
- This will indirectly tell Matlab that a guess of nonpositive σ is a bad idea

Comment

- This is a pretty indirect way of imposing constraints on the parameters
- Because you're creating a discontinuity in the objective function, it may take substantially longer to converge
- I wouldn't recommend this as a first option, but it is an option

General Tips

- Tip 4: Recode bad values as good values, if the bad values will cause the routine to crash
- Example: In a simple probit, the optimizer might give a guess of β which would yield probabilities of zero. The likelihood formula includes log(P), so it becomes undefined
- Could recode these zero probabilities to e.g. 10^{-200}
- Another example: In the normal MLE scenario, recode σ as .01 if the optimizer tries to guess a nonpositive σ

Comment

- Similar to Tip 3, I wouldn't recommend this in general
- Like Tip 3, it creates discontinuities in the objective function which may cause the optimizer to converge to the wrong spot or not converge at all
- However, I have had more success with Tip 4 than Tip 3, so it is an option

"Tricking" vs Actual Constraints

- The tips we've discussed so far are strictly numerical methods, not theoretical [our optimization should work in theory, after all...]
- They work best for situations in which there are "implicit" constraints on the likelihood function (like $\sigma > 0$ for normal pdf and $P > 0$ for probit/logit)
- If your objective function has formal constraints, it may be better to use fmincon, which is set up to do Lagrangian constrained optimization
- Implementing these "tricks" in my experience causes optimization to take longer (since Matlab has to re-think the problem each iteration)
- Inference is slightly more tricky in this scenario
- Thus, tricking the optimizer should only be done when you can't get a solution any other way

How to do Tip 2 using the Delta Method

Recall the Delta Method: If

$$\sqrt{n}\left(\hat{\theta} - \theta\right) \xrightarrow{d} N\left(0,\ \Sigma\right) \tag{1}$$

then, for any function g with a continuous derivative,

$$\sqrt{n}\left(g(\hat{\theta}) - g(\theta)\right) \xrightarrow{d} N\left(0,\ [\nabla g']\Sigma[\nabla g]\right) \tag{2}$$

where ∇ is the gradient operator.

How to do Tip 2 using the Delta Method

When introducing constraints in multivariate functions, it is important to understand what the constraint function g looks like and what its derivative ∇g looks like. The k-dimensional vector of constraint functions g is

$$g = \begin{bmatrix} g_1(\theta_1, \dots, \theta_k) \\ g_2(\theta_1, \dots, \theta_k) \\ \vdots \\ g_k(\theta_1, \dots, \theta_k) \end{bmatrix}. \tag{3}$$

How to do Tip 2 using the Delta Method

Following rules of differentiation, the derivative of g is

$$\nabla g = \begin{bmatrix} \frac{\partial g_1}{\partial \theta_1} & \frac{\partial g_1}{\partial \theta_2} & \cdots & \frac{\partial g_1}{\partial \theta_k} \\ \frac{\partial g_2}{\partial \theta_1} & \frac{\partial g_2}{\partial \theta_2} & \cdots & \frac{\partial g_2}{\partial \theta_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_k}{\partial \theta_1} & \frac{\partial g_k}{\partial \theta_2} & \cdots & \frac{\partial g_k}{\partial \theta_k} \end{bmatrix}. \tag{4}$$

Most of the time, parameter constraints will only be a function of themselves (e.g. $g(\theta_1) = \exp(\theta_1)/(1 + \exp(\theta_1))$). In these cases, ∇g will be a diagonal matrix. In the case where there are no parameter constraints, $\nabla g \equiv \mathrm{I}$, the identity matrix because $g_i(\theta) = \theta_i$, $i = 1, \ldots, k$.

How to do Tip 2 using the Delta Method

When introducing a constraint (like $\sigma' = \exp(\sigma)$), here are the steps for recovering the correct parameter and standard error:

1. At the top of the function being optimized, insert the constraints (e.g. `theta(end)=exp(theta(end));`)
2. In the file that calls the optimization, restate the constraints just after the line where optimization is called. This will have Matlab display the correct point estimates
3. Now that we have correct point estimates, we can turn to recovering the correct standard errors. Recall that, in Matlab code, the asymptotic variance matrix from (1) is $\Sigma = \texttt{sqrt(diag(inv(hessian)))}$. If we have a constraint function $g(\theta)$, then the delta method formula from (2) says that our new asymptotic variance is $[\nabla g'] \Sigma [\nabla g]$.

How to do Tip 2 using the Delta Method

4. Create a vector of derivatives. In the example given above where `theta(end)=exp(theta(end))`, the derivative would be `theta_prime(end)=exp(theta(end));` (and 1 everywhere else)
5. Once you've applied all of the constraints in this way, create the ∇g matrix by diagonalizing the vector of derivatives, (i.e. `A=diag(theta_prime)`, where `A` is the ∇g matrix).
6. Finally, the correct Hessian can be obtained by directly applying the delta method as outlined above:
`std_err=sqrt(diag(A'*inv(hessian)*A));`

Using loops to create matrices with numbers in the name

- Suppose a user wants to create matrices named X1, X2, ..., Xn
- Matlab has a command named `eval` which accomplishes this
- `eval(s)` causes Matlab to execute a string (s) as Matlab code
- Example:

```
for n = 1:12
   eval(['M' num2str(n) ' = magic(n)'])
end
```

This creates matrices M1 through M12 which are equal to `magic(1)` through `magic(12)`

Advanced flow control: capture errors

- Matlab has a construct similar to Stata's "capture"
- A `try - catch` loop is used to capture error messages that would otherwise cause a program to exit
- The general syntax is

```
try
    [statements]
catch [exception]
    [statements]
end
```