ECON 512: Empirical Methods **Fall 2018**

Lecture 1: Constrained Optimization

Instructor: Paul Grieco *Date: Nov 28/Dec 5*

These lecture notes are for my personal use. If you are reading them, I decided to distribute them as an experiment. There are typos and probably outright errors, if you find them please accept my apologies and report them to me.

1.1 Background

This week we will look at the estimation of structural models via constrained optimization. This is an alternative to nested fixed point estimation (NFXP), which you have used earlier in this course. I have found that sometimes it is able to realize significant speedup over NFXP, sometimes it does not. I use this technique in the paper "Borders, Geography, and Oligopoly: Evidence from the Wind Turbine Industry" (2015, Review of Economics and Statistics).

Some sources are:

- KNITRO user manual, for how to use the optimization package KNITRO.
- Judd and Su (2010, Econometrica), example using Rust model (with code!)
- Dube, Fox, and Su (2012, Econometrica), example using BLP (with code!).
- Ortega and Rheinbold, *Iterative Solutions to Nonlinear Equations of Several Variables*, Chapters 7 and 8, for the math.

Where might this show up in Economics:

- Utility optimization with budget constraints.
- Principal-agent problems and mechanism design more generally.
- Structural estimation with equilibrium constraints (agent optimality conditions).
- Structural estimation employing numerical inversions (BLP).
- Estimation with moment conditions (equalities and inequalities).

1.2 Review: Unconstrained Optimization is Solving FOCs

Recall that we handled non-linear optimization problems of the form,

$$\min_x f(x)$$

for $f : \mathbb{R}^N \to \mathbb{R}$, by essentially using iterative techniques to solve the system of equations,

$$\nabla f(x) = 0,$$

or just the first order conditions of the optimization problem. One way to solve these is Newton's Method:=

$$x^{k+1} = x^k - H(x^k)^{-1}\nabla f(x^k) \tag{1.1}$$

Where H represents the Hessian of f,

$$H(x^k) = \frac{\partial^2 f(x)}{\partial x \partial x'}.$$

To numerically we just iterate (1.1) until some terminal condition is satisfied, i.e.,

$$\nabla f(x) < \varepsilon \approx 10^{-6}.$$

If the problem is convex, then it has a unique minimum and we will find it. If it is not, than we hope it converges, and hope even more it converges to the global minimum. (In practice, you need multi-start).

Since unconstrained optimization is really solving a system of equalities, constrained optimization shouldn't be much harder.

1.3 Lagrangian and Karush-Kuhn-Tucker (KKT) Conditions

So now we want to solve the constrained problem:

$$\begin{aligned} &\min_x f(x) \\ \text{subject to: } &C^E(x) = 0 \\ &C^I(x) \leq 0 \end{aligned}$$

Good news: If you can write your problem in this form, and can supply appropriate derivatives, constrained optimization packages (e.g., KNITRO, fmincon) can be applied to your problem.

What will these programs do? Write the Lagrangian:

$$\mathcal{L}(x, \mu, \lambda) = f(x) + \mu' C^E(x) + \lambda' C^I(x)$$

Where,

- μ is an $\ell \times 1$ vector of Lagrange multipliers for the ℓ equality constraints.
- λ is a $m \times 1$ vector of Lagrange multipliers for the m inequality constraints.

Then the KKT conditions are:

1. Stationarity (the FOC of Lagrangian) wrt x:

$$\nabla_x \mathcal{L} = \nabla f(x) + \mu' J^E(x) + \lambda' J^I(x) = 0$$

   where $J_E(x) = \left[\frac{\partial C^E(x)}{\partial x}\right]$ and $J^I(x) = \left[\frac{\partial C^I(x)}{\partial x}\right]$

2. Primal Feasibility:

$$C^E(x) = 0, C^I(x) \leq 0$$

3. Dual Feasibility:
$$\lambda \geq 0$$

4. Complementary Slackness:
$$\forall i = 1, \ldots, m : \lambda_m C_m^I = 0$$

So again, we just need to solve this system of equations. Notice life is *much* simpler if we don't have to deal with inequality constraints (often in economics, we don't). When we have inequalities, we'll just convert the problem into something similar with only equalities.

1.4 Interior-Point (a.k.a. Barrier) Methods

Add m "slack" variables to deal with the inequalities, as follows:

$$\begin{aligned} &\min_x f(x) - \beta \sum_{m=1}^{m} \log s_i \\ \text{subject to: } &C^E(x) = 0 \\ &C^I(x) + s = 0 \end{aligned}$$

- The added term is called a "barrier function".
- It is basically a reward for keeping the constraints satisfied and away from binding.
- We can approximate the solution to the original problem by solving a sequence of these barrier problems such that $\beta_k \to 0$ as $k \to \infty$.

So we can solve this problem, by considering the lagrangian

$$\mathcal{L}(x, s, \mu, \lambda) = f(x) - \beta \sum_{m=1}^{m} \log s_i + \mu' C^E(x) + \lambda'(C^I(x) + s)$$

Which has only equalities and so the simpler set of KKT conditions:

1. Stationarity (the FOC of Lagrangian) wrt x:
$$\nabla_x \mathcal{L} = \nabla f(x) + \mu' J^E(x) + \lambda' J^I(x) = 0$$

2. Slackness (derivative wrt $\log s$):
$$-\beta e + S\lambda = 0$$
where e is and $m \times 1$ vector of ones and S is a diagonal matrix with the vector s along the diagonal (n.b., $\frac{\partial s_i}{\partial \log s_i} = s_i$).

3. Primal Feasibility:
$$C^E(x) = 0$$
$$C^I(x) + s = 0$$

Notice a Trick: if you want something to be positive (like our slackness variable), don't hope it is and take the logarithm, redefine the variable as $\log x$ and exponentiate it.

This is a set of equalities, so we can apply Newton's method just like in Section 1.2. Here a newton step is

$$\begin{bmatrix} \Delta x \\ \Delta \log s \\ \Delta \mu \\ \Delta \lambda \end{bmatrix} = H^{-1} \nabla \mathcal{L}$$

Where H and $\nabla \mathcal{L}$ are the Hessian and Jacobian of the Lagrangian respectively. The Jacobian we've already written down, the Hessian looks like a mess, but notice it is actually pretty sparse:

$$H(x, s, \mu, \lambda) = \begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & J^{E'} & J^{I'} \\ 0 & \Lambda & 0 & S \\ J^E & 0 & 0 & 0 \\ J^I & I & 0 & 0 \end{bmatrix}$$

Where Λ is the diagonal matrix of λ, I is the identity, and everything else is already defined.

- In practice we'll use BFGS to approximate $\nabla^2_{xx}\mathcal{L}$, just like we approximated the hessian of the unconstrained problem.
- It is usually straight forward to program the Jacobian of the constraint and supply it directly.
- Even if you use BFGS, it is often helpful to program up the sparsity pattern in order to save memory, especially if x is of high dimension.

Now we know how to solve the barrier problem for a fixed $\beta > 0$, the true problem is $\beta = 0$.

- We could just solve a bunch of these and let β shrink until the solution "didn't change much".
- But why waste time "completely" solving the barrier problems, most implementations lower β once they are "close enough" to solving the barrier problem's solution conditions, essentially interspersing barrier and newton steps.
- The most advanced solvers don't use interior-point methods to completely solve the problem, but "cross-over" to active set methods once they are confident which inequalities bind ($s = 0$) and which don't ($s > 0$).

1.5 Active Set Method

An older alternative to the Barrier method for constrained optimization is the Active-Set method, it turns out the two are complements.

- Recall the issue with inequality constraints is translating them into a system of *equalities* on which to apply (quasi-)Newton's method.
- If we knew which constraints were binding, and which weren't we could ignore the binding (active) constraints and solve:

$$\begin{aligned} &\min_x f(x) \\ \text{subject to: } &C^E(x) = 0 \\ &C^I_a(x) = 0 \end{aligned}$$

Where $C^I_a(x)$ are active constraints we expect to bind, and $C^I_i(x)$ are inactive constraints we expect are slack (and so ignore).

- Once we did this, we'd want to check whether $\lambda_a > 0$ to make sure the active constraint is binding in the "correct" direction. We'd also want to check that inactive constraints $C^I_i(x) < 0$.
- Now we essentially have a "guess and check" algorithm. Pick the set of active constraints, solve problem. Swap out constraints for which $\lambda_a < 0$, swap in constraints such that $C^I_i(x) \geq 0$.
- Like with barrier methods, it is not useful to fully solve the active set problems, instead, you swap constraints in and out in conjunction with newton's method. (Here lies lots of engineering).

It's clear that active set methods work well when you have a good idea what constraints are binding. As a consequence, many implementations start with barrier methods and then "crossover" once it is easy to guess the active set.

1.6 Implementing Constrained Optimization in MATLAB

There are many constrained optimization packages in many different programming languages. Since we focus on MATLAB I'll describe two:

- **fmincon** is MATLAB's native constrained optimization solver, part of the optimization toolbox. If you have the student version of MATLAB, you have it.
- **KNITRO** is a proprietary optimization package that can be called from MATLAB. It is available on the PSU cluster. But you must load the appropriate module (i.e., type `module load knitro` at the bash prompt before starting MATLAB). The MATLAB command to call KNITRO is `knitromatlab`.[1]

Since it is everywhere I will focus on `fmincon`, this should *not* be taken to mean that is the only command you need to know. That said, they are all pretty similar in that you need to structure your problem as follows:

$$\begin{aligned} &\min_x f(x) \\ \text{subject to: } &C^E(x) = 0 \\ &C^I(x) \leq 0 \\ &A^I \cdot x \leq b^I \\ &A^E \cdot x = b^E \\ &\ell \leq x \leq u \end{aligned}$$

[1]As always, once you know a command, there is a wealth of information on how to use it in MATLAB's help files. Much more than I will cover here.

This is just saying that linear constraints and simple bounds will be handled separately from the nonlinear constraints. It's trivial to guess how the packages simplify your life if you provide them this additional structure.

The MATLAB call to solve this problem is something like

```
[x, fval, exitflag] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Where:

1. `fun` is a function handle to your objective function. In most cases, you will write a function say,

   ```
   function f = mylike(params, data) { ... }
   ```

   And then call it using a anonymous function along the lines of "`@(x) mylike(x, data).`" This lets fmincon repeatedly call your function with whatever first argument it wants.
2. `x0` is the starting point.
3. `A, b, Aeq, beq` are the linear constraint information.
4. `lb, ub` are vectors of lower and upper bounds.
5. `nonlcon` is a function handle that returns both inequality and equality constraints. This function must return two output arguments e.g.,

   ```
   function [c, ceq] = myNonlcon(params, data) { ... }
   ```
6. `options` is a structure of option flags, there are many to work through, see the help file.

The upshot of all this is that once you write up your problem, the solver can take it from there, just like an unconstrained optimization problem. Of course, there are some details:

- Buried in the options are things like stopping criteria, step size tolerance, iteration counts, that you will need to tune to your problem almost certainly. In my experience, not experimenting with these will keep your problem from performing well.
- Notice that so far I didn't provide any derivative information, if I left it like this, then MATLAB would calculate numerical derivatives, which may be very slow. Derivative information is provided by (1) returning gradient (and maybe hessian?) as additional output arguments to `fun` and `nonlcon` and (2) setting the appropriate flag in `options`. E.g.,

  ```
  options = optimoptions('fmincon','SpecifyObjectiveGradient',true)
  ```
- `fmincon` is not exactly Newton's method, it is actually a "Newton based method" even better, it actually implements several different algorithms. The default may not be the best for your problem. Have I mentioned the help file?
- Of course, this stuff isn't magic. If you are playing around, there is nothing to stop you from writing your own Newton-based solver, or finding one online. That said, it's nice to use professional code (until it refuses to converge).
- Help pages: `https://www.mathworks.com/help/optim/ug/fmincon.html` for fmincon and `https://www.artelys.com/tools/knitro_doc/3_referenceManual/knitromatlabReference.html` for KNITRO

1.7 Applications

To wrap up, lets consider how constrained optimization can be used in some structural estimation problem.

1.7.1 Mixed Logit - Berry, Levinsohn, Pakes (1995)

This should be fresh in your mind. Recall that BLP solved this model with a nested-fixed point approach. That is, an outer-loop maximized θ while an inner loop solved for unobserved product qualities. We could write this as an unconstrained optimization problem that looks like:

$$\min_{\theta} \xi(\theta)' Z W Z' \xi(\theta)$$

Where $\xi(\theta)$ is a function that solves the share inverse, i.e., $s = \sigma(\xi, \theta)$, in practice BLP solve this function using a contraction mapping. Maybe we could have solved it using Newton's method. At any rate, the problem is that while it's easy to calculate:[2]

$$\sigma_{jt}(\xi, \theta) = \int \frac{x_{jt}\beta_i - \alpha_i p_{jt} + \xi_{jt}}{\sum_{k \in J_t} x_{kt}\beta_i - \alpha_i p_{kt} + \xi_{kt}} dF(i; \theta)$$

The inverse $\xi(\sigma) = \sigma^{-1}(s_j, \theta)$ must be solved numerically. Under nested fixed point this problem (of $J \cdot T$ equations and $J \cdot T$ unknowns) must be solved for each calculation of the objective function.

The **exact same estimator** can be written as a constrained optimization problem:

$$\begin{aligned} &\min_{\theta, \xi} \ \xi' Z W Z' \xi \\ &\text{s.t.: } s - \sigma(\xi, \theta) = 0 \end{aligned}$$

This approach to BLP is explored by Dube, Fox, Su (2012). A few notes:

- Notice that we have greatly increased the number of parameters, there is one ξ dimension for every product in the dataset. We also have as many constraints as products.
- That said, the constraints are fairly sparse (since substitution is only within-market) and also very diagonal dominant.
- Moreover, it is very easy to calculate derivatives of the objective function, which is now just quadratic in ξ.
- For a recent paper, I've implemented both a NFXP and Constrained optimization paper. In that context we eventually went with the NFXP, your milage may vary.

1.7.2 Dynamics - Rust (1987), Judd and Su (2010)

Rust's bus engine replacement problem is a seminal paper on structural estimation in dynamic models. Without getting too bogged down, a dynamic optimization problem, given regularity conditions, can define it's value function as the fixed point of a contraction mapping.

[2]Were θ parameterizes the distribution of (α_i, β_i).

$$EV(x,d) = T_\theta(EV)(x,d)$$

Rust discretizes the dynamic problem (so that the value function is a vector) and exploits the properties of the type-I extreme value distribution in order to write the contraction mapping operator as:

$$T_\theta(EV)(x,d) = \sum_{x'} \log \left[\sum_{d'} \exp\left\{ u(x',d';\theta) + \beta EV(x',d') \right\} \right] p(x'|x,d,\theta)$$

In Rust's implementation, for a given θ, one iterates this operator to solve for the vector of valuations for each state-decision $EV(x,d)$. Then uses this to derive choice probabilities in each state and compute the likelihood. The constrained optimization approach maximizes the likelihood and solves the value function jointly,

$$\begin{aligned}
&\max_{\theta, EV} \sum_{i=1}^{N} \sum_{t=2}^{T} \log P(d_t^i | x_t^i, \theta) + p(x_t^i|, x_{t-1}^i, d_{t-1}^i, \theta) \\
&\text{s.t.: } EV - \sum_{x'} \log \left[\sum_{d'} \exp\left\{ u(x',d';\theta) + \beta EV(x',d') \right\} \right] p(x'|x,d,\theta) = 0.
\end{aligned}$$

Where,

$$P(d|x,\theta) = \frac{u(x,d;\theta) + \beta EV(x,d)}{\sum_e u(x,e;\theta) + \beta EV(x,e)}$$

Which just enforces that agent's make optimal choices.

1.7.3 Equilibrium - Coşar, Grieco, Tintelnot (2015)

In this paper, we have data on the location of wind turbine projects and which of 11 companies built the project. We do not observe transaction prices.

To deal with this, we assume that firms post take it or leave it prices and the project manager chooses the manufacturer that maximizes his valuation. We assume a logit-style demand system and derive the following equilibrium condition on the probability manufacturer j wins project i.

$$\rho_{ij} = \frac{\exp\left(d_j - c_{ij} - \frac{1}{1-\rho_{ij}} \right)}{1 + \sum_{k=1}^{|\mathcal{J}|} \exp\left(d_k - c_{ik} - \frac{1}{1-\rho_{ik}} \right)} \qquad \text{for } j \in \mathcal{J}.$$

Where ρ_{ij} is the probability that manufacturer j wins product i. Then, we can use equilibrium conditions to find the likelihood of the data,

$$L(\rho) = \prod_{\ell \in \{D,G\}} \prod_{i=1}^{N_\ell} \prod_{j=0}^{|\mathcal{J}_\ell|} (\rho_{ij}^\ell)^{y_{ij}^\ell},$$

Once we parameterize demand and cost, we can use constrained optimization to estimate this problem as follows:

$$
\begin{aligned}
\max_{\xi,\beta,\,\rho} \quad & L(\rho) \\
\text{subject to:} \quad & \rho_{ij}^{\ell} = \frac{\exp\left(\xi_j - \beta_d \cdot \log(\text{distance}_{ij}) - \beta_b \cdot \text{border}_{ij} - \beta_s \cdot \text{state}_{ij} - \frac{1}{1-\rho_{ij}^{\ell}}\right)}{1 + \sum_{k=1}^{|\mathcal{J}_\ell|} \exp\left(\xi_k - \beta_d \cdot \log(\text{distance}_{ik}) - \ \beta_b \cdot \text{border}_{ik} - \beta_s \cdot \text{state}_{ij} - \frac{1}{1-\rho_{ik}^{\ell}}\right)} \\
& \sum_{k=1}^{|\mathcal{J}_\ell|} \rho_{ik}^{\ell} + \rho_{i0}^{\ell} = 1 \qquad \text{for } \ell \in \{D, G\},\ i \in \{1, ..., N_\ell\},\ j \in \mathcal{J}.
\end{aligned}
$$

See the paper for details.

1.8 A look at the code for the Wind Turbine example

Show students the code from one particular specification of the Wind Turbine Paper:

1. Examine the master program.
2. Go over MPEC main:
   - Set up sparsity pattern.
   - Set up options for call.
   - A little multi-start.
   - Numerical derivative and standard errors.
3. Go over likelihood function, not it's simplicity and lack of dependence on θ.
4. Go over constraints, and their gradients.
5. Quick look at dummy objective, what is it used for.
6. Look at the diary file.