A Nonlinear Programming Approach to Structural Estimation

Kenneth L. Judd
Hoover Institution and NBER

Che-Lin Su
Kellogg, Northwestern and NBER

INFORMS ANNUAL MEETING
Nov 5–8, 2006, Pittsburgh, PA

Structural Estimation

- Great interest in estimating models based on economic structure
  - Dynamic programming models
  - Games – static, dynamic
  - Auctions
  - Dynamic stochastic general equilibrium
- Major computational challenge because estimation involves also solving model
- We show that many computational difficulties can be avoided by using optimization tools

Structural Estimation

- Specify model with structural parameters, θ
- Find θ such that equilibrium implications of θ match data
- Objectives: Maximum likelihood, matching moments, ...
- Difficulties
  1. Computing an equilibrium implied by θ may be costly
  2. Find all equilibria consistent with θ is usually intractable!
- Current Practice in econometrics
  1. Use less efficient "two-step" methods
  2. Use "speculative" methods - Nested Pseudo Likelihood (NPL)

Basic Problem – DP Example

- Individual (agent) solves a dynamic programming problem
- Econometrician observes state (partially) and decisions
- Likelihood function for data X

$$L(\theta, D; X)$$

where θ is set of parameters and D is decision rule

- Rationality imposes a relationship between θ and D

$$0 = G(\theta, D)$$

- We want to find maximum likelihood θ but impose rationality condition

Hazold Zurcher Model – Data

Bus #: 5297

events	year	month	odometer at replacement
1st engine replacement	1979	June	242400
2nd engine replacement	1984	August	384900

year	month	odometer reading
1974	Dec	112031
1975	Jan	115223
1975	Feb	118322
1975	Mar	120630
1975	Apr	123918
1975	May	127329
1975	Jun	130100
1975	Jul	133184
1975	Aug	136480
1975	Sep	139429

Hazold Zurcher Model

- Time series data: $(x_1, x_2, \ldots, d_T)$ and $(d_1, d_2, \ldots, d_T)$
  - Observed state is x_t: mileage since last overhaul
  - d_t: decision at t, $d_t = 0$ (no repair) or 1 (repair)
- θ - parameters on repair costs, transition probabilities
- $V(x, \epsilon; \theta) = \max_{d \in \{0,1\}} [u(x, d, \theta) + \epsilon(d) + \beta EV(x, d; \theta)]$
- $V(x; \theta)$ - value to repairman of a bus with x, *before* he knows current shock to costs
- Bellman equation
  - $V(x; \theta) = \mathcal{F}(x, V(\theta))$, where $V(\theta) = [V(x; \theta)]_x$
  - $V(x; \theta)$ implies a decision rule, $D(x)$, which implies a transition process, Π_θ, for states and decisions

NFXP: Rust (1987)

- Define $\mathcal{L}(\theta, V(\theta); X)$

$$\theta \longrightarrow V(\theta) \longrightarrow \Pi_\theta \longrightarrow \mathcal{L}(\theta, V(\theta); X) = \text{ likelihood}$$

- Write a program to compute $V(\theta)$
- Write a program to solve $\max_\theta \mathcal{L}(\theta, V(\theta); X)$
- Nesting:
    - Inner loop to computes $V(\theta)$
    - Outer loop to solve $\mathcal{L}(\theta, V(\theta); X)$

Difficulties with NFXP

- Outer loop needs $\dfrac{\partial \mathcal{L}}{\partial \theta}$
  - Analytic derivatives are hard to do by hand
  - Numerical derivatives need high accuracy on $\mathcal{L}(\theta)$ and $V(\theta$ solutions
- Outer loop would like $\dfrac{\partial^2 \mathcal{L}}{\partial^2 \theta}$: No Way!!
- Slow since one must solve $V(\theta)$ for each θ examined in outer loop

Is Structural Estimation Difficult ?

- Current View: Erdem et a. (2004):

  > Estimating structural models can be computationally difficult. For example, dynamic discrete choice models are commonly estimated using the nested fixed point algorithm (see Rust 1994). This requires solving a dynamic programming problem (DP) thousands of times during estimation and numerically maximizing a nonlinear likelihood function. ...

- Our view: Gauss-Jacobi or Gauss-Seidel methods are often used in economics even though they are at best linearly convergent. *Apply the rabid dog principle!*.

Our Approach: Constrained Optimization

- Use MPEC modeling ideas
- Eliminate "nested" structure
- Eliminate fixed point
- Formulate problem as a constrained nonlinear optimization problem

$$\begin{array}{ll} \max\limits_{(\theta, V)} & \mathcal{L}(\theta, V; X) \\ s.t. & V - \mathcal{F}(V, \theta) = 0 \end{array}$$

Nonlinear Optimization Analogues

- Consider problem

$$\begin{array}{ll} \max\limits_{\theta, y} & f(\theta, y) \\ s.t. & y = g(y, \theta) \end{array}$$

- NFXP is essentially a nonlinear substitution of variables method
  - Define $Y(\theta)$ by $Y(\theta) = g(Y(\theta), \theta)$
  - Substitute out the y variables in objective to get

$$\max_{\theta} f(\theta, Y(\theta))$$

Nonlinear Optimization Analogues

- Nested Pseudo-Likelihood (NPL) – Aguirregabiria and Mira, *Econometrica* (2002):
  - Essentially a Gauss-Seidel method

$$
\begin{aligned}
\theta^{i+1} &= \arg\max_{\theta} f\left(\theta, y^{i}\right) \\
y^{i+1} &= g\left(y^{i}, \theta^{i+1}\right)
\end{aligned}
$$

  - Convergence related to the eigenvalues of g_y; no reason to believe that all the eigenvalues are stable
- These methods are regarded as inefficient in the nonlinear programming literature

J-S Advantages

- J-S evaluates Bellman errors

$$V - \mathcal{F}(V, \theta) \qquad \text{per}\theta,$$

whereas NFXP solves Bellman equations

$$V_\theta - \mathcal{F}(V_\theta, \theta) = 0$$

for each θ

- User only needs to write down Bellman equation for optimizer - NO NEED TO WRITE SOLVER

Therefore, J-S is faster and easier to use

Comparison with Rust Implementation

- Ease of use
  - Rust: Gauss
    - a high-level symbolic language
    - built-in linear algebra routines
  - J-S: AMPL
    - all solvers have access to linear algebra routines
    - flexible approach to matrices, tensors, and indexed sets
- Vectorization
  - Rust: Efficient use of GAUSS requires the user to "vectorize" a program
  - J-S: All vectorization is done automatically in AMPL

Comparison with Rust Implementation

- Optimization Method
  - Rust: BHHH/BFGS
  - J-S: Use solvers far superior to these methods
- Derivatives
  - Rust: compute the value of and its derivatives numerically in a subroutine
  - J-S: Use true analytic derivatives; done automatically and efficiently by AMPL using automatic differentiation.

Comparison with Rust Implementation

- Dynamic programming method
  - Rust: Contraction mapping fixed point (poly)algorithm.
    - combine contraction with Newton-Kantorovich iterations
    - contraction iterations are linearly convergent
    - quadratic convergence is achieved only at final stage.
  - J-S: Newton-style methods
    - globally faster than contraction mapping
    - particularly important if β is close to 1

J-S AMPL Implementation

- Express problem in straightforward language
- Access almost any solver:
  IPOPT, KNITRO, SNOPT, Filter, MINOS, PENNON
- Gradients and Hessians are computed analytically and automatically and efficiently

Time

- 1000 data points
- 120 states
- estimate quadratic cost, replacement cost and transition probabilities

five parameter case

Solver	CPU time
KNITRO	0.5 sec
SNOPT	1.5 sec
IPOPT	2 sec

Sensitivity to Number of States

- 1000 data points
- estimate quad cost, replacement cost and transition probabilities

KNITRO

num. of states	CPU time (in sec.)	Maj. Iter.
120	0.23	18
240	0.42	21
360	0.55	19
480	0.98	21

Strategy for Games

$$\max_{\theta, V^i} \quad \mathcal{L}(V^1, V^2, \ldots, \theta; X)$$

$$s.t. \quad V^1, V^2, \ldots, \text{ satisfy equilibrium equations given } \theta$$

- NFXP
  - Guess θ and compute all Nash equilibrium
  - Multiple equilibria produces intractable problem
- J-S
  - Multiple equilibria reduces to problem of global optimization, which maximum likelihood already has!