A New Optimization Approach to Maximum Likelihood Estimation of Structural Models

Kenneth L. Judd
Hoover Institution, University of Chicago, and NBER

Che-Lin Su
Northwestern University

November 17, 2006

Structural Estimation

- Great interest in estimating models based on economic structure
  - Dynamic programming models
  - Games
  - 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

Basic Problem - DP Example

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

$$L(\theta; X)$$

  where θ is set of parameters

- Augmented likelihood function for data X

$$\mathcal{L}(\theta, \sigma; X)$$

  where θ is set of parameters and σ is decision rule

- Rationality imposes a relationship between θ and σ

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

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

Nested Fixed Point Algorithm

- Given θ, compute σ - in practice, this means writing a program $\sigma = \Sigma(\theta)$
- Solve likelihood

$$\max \mathcal{L}(\theta, \Sigma(\theta); X)$$

  - Problems: Must compute $\Sigma(\theta)$ to high accuracy for each θ examined
  - 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). ....[S]ome recent research ... proposes computationally simple estimators for structural models including auctions, demand in differentiated product markets, dynamic discrete choice and dynamic games. The estimators ... use a two-step approach. ....The two-step estimators can have drawbacks. First, there can be a loss of efficiency. .... Second, stronger assumptions about unobserved state variables may be required. .... However, two-step approaches are computationally light, often require minimal parametric assumptions and are likely to make structural models accessible to a larger set of researchers.

- Is this true?
  - Are structural models so computationally difficult that it is necessary to turn to statistically inferior methods?
  - Are economists experts on computational feasibility?
- In this paper, we argue that the answer to the first question is an emphatic NO!

Simple Consumer Demand Example

- Data and Model
  - Data on demand, q, and price p, but demand is observed with error ε.
  - True demand is $q - \varepsilon$.
  - Assume a parametric form for utility function $u(c; \beta)$ where β is a vector of parameters.
  - Economic theory implies
$$u_c(c; \beta) = u_c(q - \varepsilon; \beta) = p$$
- Standard Approach (from Econ 712, University of Wisconsin, 1979)
  - Assume, for example, a functional form for utility
$$u(c) = c - \beta c^2.$$
  - Solve for demand function
$$c = (1 - p)/(2\beta)$$
  - Hence, i'th data point satisfies
$$q_i = (1 - p_i)/(2\beta) + \varepsilon_i$$
    for some ε_i.
  - To estimate β, choose β to minimize the sum of squared errors
$$\sum_{i=1} \left(q_i - (1 - p_i)/(2\beta)\right)^2.$$

- Limitations
  - Need to solver for demand function, which is hard if not impossible
  - For example, suppose

$$u(c) = c - \beta\left(c^2 + c^4 + c^6\right)$$

    with first-order condition

$$1 - \beta\left(2c + 4c^3 + 6c^5\right) = p$$

  - There is no closed-form solution for demand function.
  - What were you taught to do in this case? *Change the model*!
- Proper Procedure
  - Deal with the first-order condition directly since it has all the information you can have.
  - Recognize that all you do is find the errors that minimize their sum of squares but are consistent with structural equations.

- Examples
  - For our consumption demand model, this is the problem

$$\begin{aligned} \min_{\varepsilon_i,\beta} \quad & \sum_{i=1} \varepsilon_i^2 \\ \text{s.t.} \quad & u_c\left(q-\varepsilon_i;\beta\right) = p_i \end{aligned}$$

  - In the case of the quadratic utility function, this reduces to

$$\begin{aligned} \min_{c_i,\varepsilon_i,\beta} \quad & \sum_{i=1} \varepsilon_i^2 \\ \text{s.t.} \quad & 1-2\beta c_i = p_i \\ & q_i = c_i + \varepsilon_i \end{aligned}$$

  - Degree-six utility function produces problem

$$\begin{aligned} \min_{c_i,\varepsilon_i,\beta} \quad & \sum_{i=1} \varepsilon_i^2 \\ \text{s.t.} \quad & 1-\beta\left(2c_i+4c_i^3+6c_i^5\right) = p_i \\ & q_i = c_i + \varepsilon_i \end{aligned}$$

– Even when you can solve for demand function, you may not want to.

* Consider the case

$$\begin{aligned} u(c) &= c - \beta_1 c^2 - \beta_2 c^3 - \beta_3 c^4 \\ u'(c) &= 1 - 2\beta_1 c - 3\beta_2 c^2 - 4\beta_3 c^3 \end{aligned}$$

* Demand function is

$$\begin{aligned} q &= \frac{1}{12\beta_3} W - \frac{1}{4} \frac{8\beta_1\beta_3 - 3\beta_2^2}{\beta_3 W} - \frac{1}{4}\frac{\beta_2}{\beta_3} \\ W &= \sqrt[3]{\left(108\beta_1\beta_2\beta_3 - 216\beta_3^2 p + 216\beta_3^2 - 27\beta_2^3 + 12\sqrt{3}\beta_3 Z\right)} \\ Z &= \sqrt{Z_1 + Z_2} \\ Z_1 &= 32\beta_1^3\beta_3 - 9\beta_1^2\beta_2^2 - 108\beta_1\beta_2\beta_3 p + 108\beta_1\beta_2\beta_3 \\ Z_2 &= 108\beta_3^2 p^2 - 216\beta_3^2 p + 27 p\beta_2^3 + 108\beta_3^2 - 27\beta_2^3 \end{aligned}$$

* Demand function is far costlier to compute than the first-order conditions.

- The (*bad*) habit of restricting models to cases with closed-form solutions is completely unnecessary.

Basic Problem - DP Example

- Individual solves a dynamic programming problem
- Econometrician observes state (with error) and decisions
- Augmented likelihood function for data X

$$\mathcal{L}(\theta, \sigma; X)$$

  where θ is set of parameters and σ is decision rule
- Rationality imposes a relationship between θ and σ

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

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

Judd-Su Approach - Use MPEC ideas

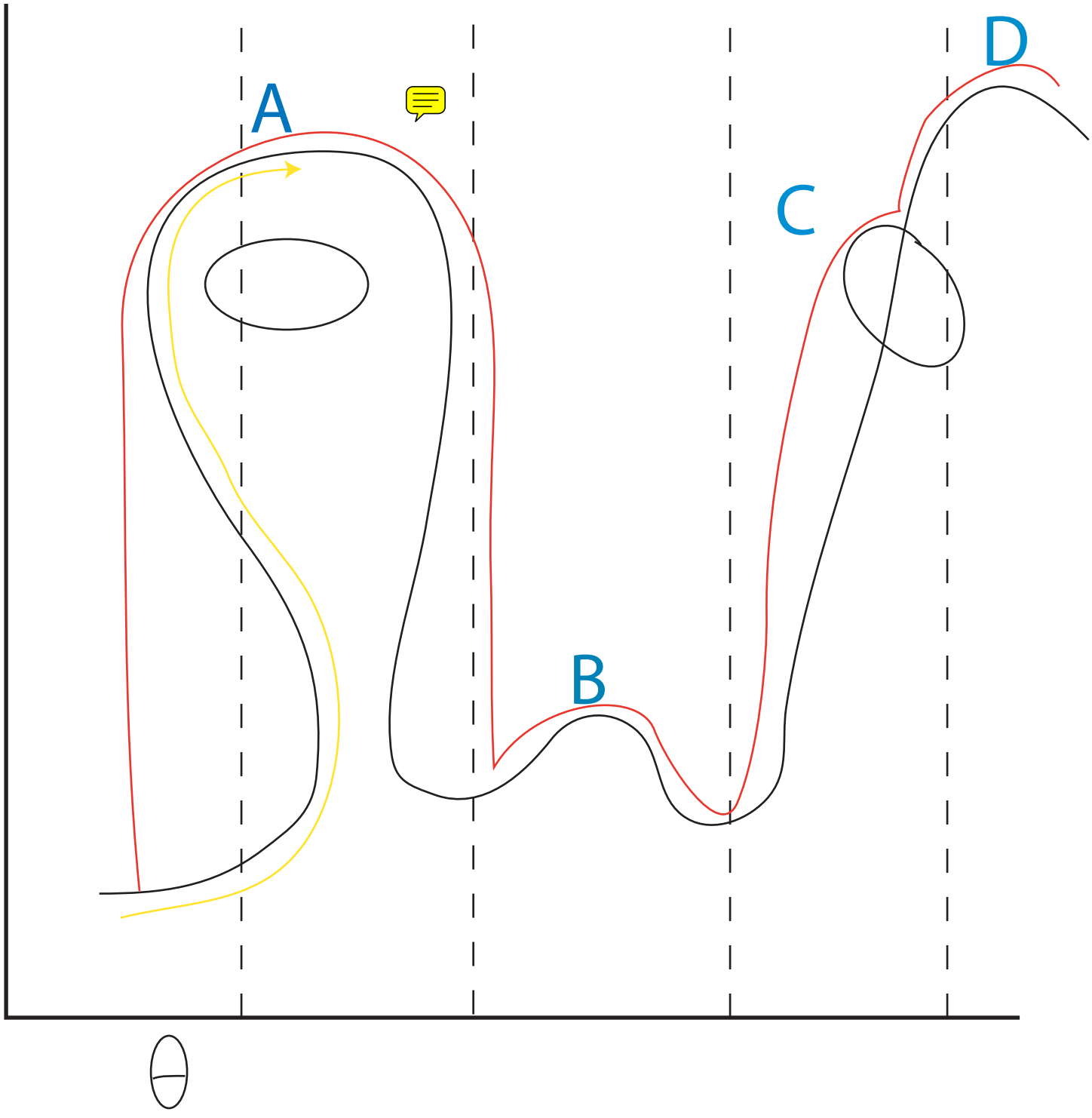
- Suppose that an economic model has parameters θ.
- Suppose that equilibrium and optimality imply that the observable economic variables, x, follow a stochastic process parameterized by a finite vector σ.
- The value of σ will depend on θ through a set of equilibrium conditions

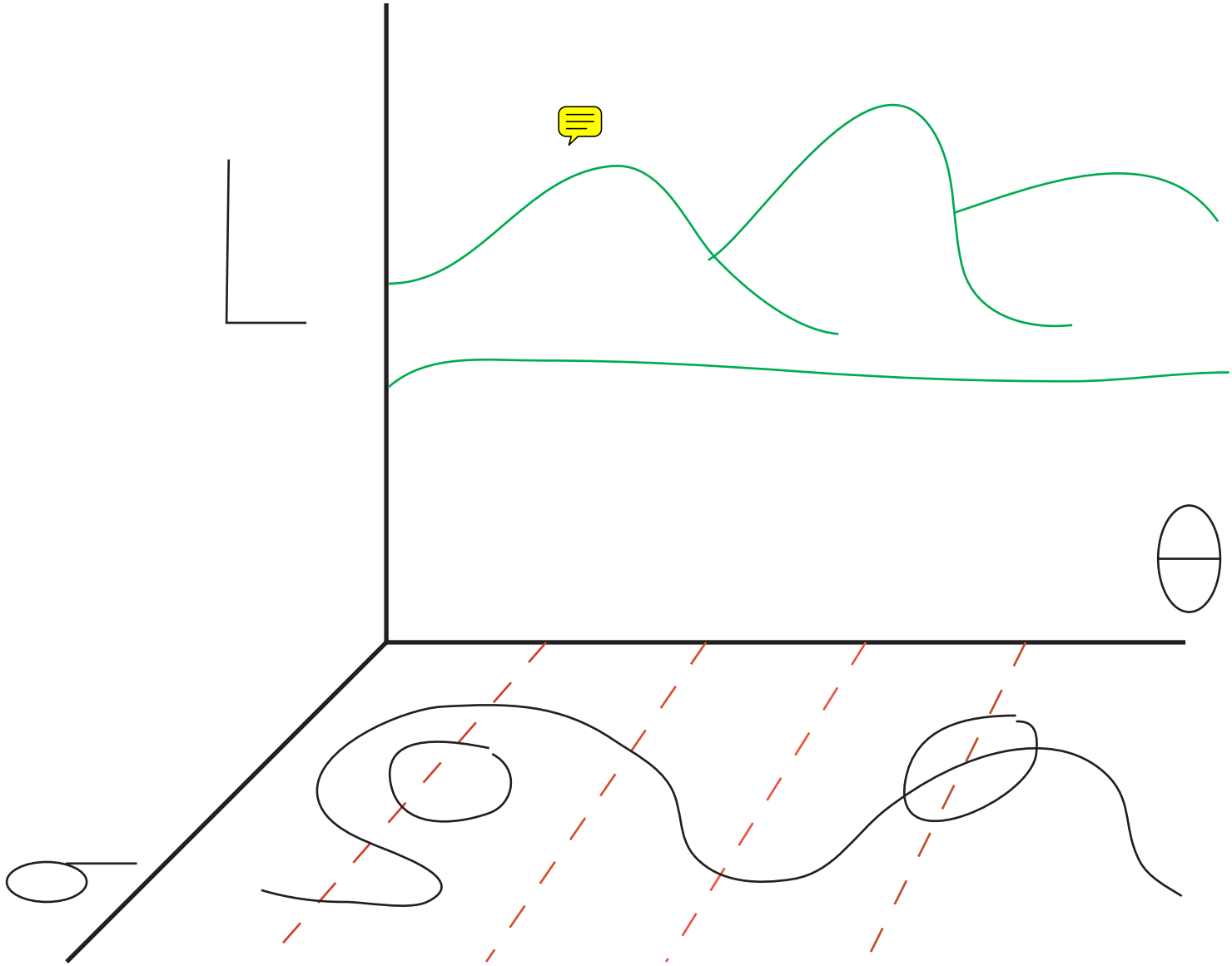
$$0 = G(\theta, \sigma)$$

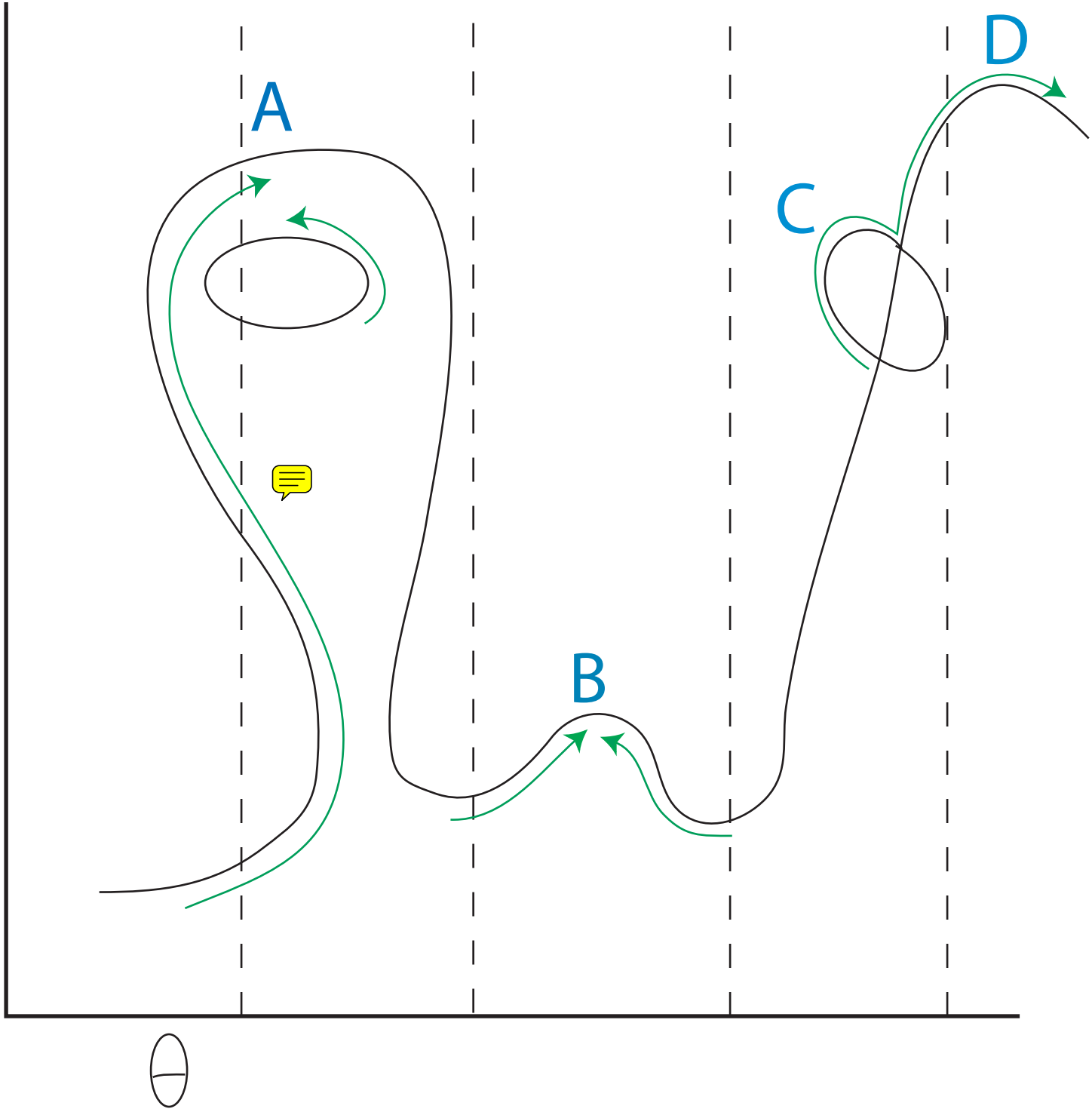
- Denote the augmented likelihood of a data set, X, by $\mathcal{L}(\theta, \sigma; X)$.
- Therefore, maximum likelihood is the constrained optimization problem

$$\begin{aligned} \max_{\sigma,\theta} \quad & \mathcal{L}(\theta, \sigma; X) \\ \text{s.t.} \quad & 0 = G(\theta, \sigma) \end{aligned}$$

- We do not require that equilibrium be defined as a solution to a fixed-point equation.
- We do not need to specify an algorithm for computing σ given θ; good solver is probably better.
- Gauss-Jacobi or Gauss-Seidel methods are often used in economics even though they are at best linearly convergent, whereas good solvers are at least superlinearly convergent locally (if not much better) and have better global properties than GJ and GS typically do.
- Using a direct optimization approach allows one to take advantage of the best available methods from the numerical analysis


A
B
C
D




A
B
C
D

JS Applied to Zurcher

- Timing for estimating three parameters (as in the Rust).

		cpu	Estimates			major	# obj	constr.
T	N	(secs)	RC	θ_1^c	θ_2^c	iter.	evals	evals
1,000	101	0.14	1.112	0.043	0.0029	66	72	72
1,000	201	0.31	1.140	0.055	0.0015	44	59	59
1,000	501	1.65	1.130	0.050	0.0019	58	68	68
1,000	1001	5.54	1.144	0.056	0.0013	58	94	94
10,000	101	0.24	1.236	0.056	0.0015	59	67	67
10,000	201	0.44	1.257	0.060	0.0010	59	67	67
10,000	501	0.88	1.252	0.058	0.0012	35	45	45
10,000	1001	3.47	1.256	0.060	0.0010	39	52	52

- Rust did a two-stage procedure, estimating transition parameters in first stage. We do full ML

		cpu	Estimates					major	# obj	# constr
data	states	(secs)	RC	θ_1^c	θ_2^c	θ_1^p	θ_2^p	iters	evals	evals
1,000	101	0.50	1.107	0.039	0.0030	0.723	0.262	111	137	137
1,000	201	1.13	1.140	0.055	0.0015	0.364	0.600	109	120	120
1,000	501	3.37	1.129	0.050	0.0019	0.339	0.612	115	127	127
1,000	1001	7.56	1.144	0.056	0.0014	0.360	0.608	84	116	116
10,000	101	0.50	1.236	0.052	0.0016	0.694	0.284	76	91	91
10,000	201	0.86	1.257	0.060	0.0010	0.367	0.593	85	97	97
10,000	501	2.73	1.252	0.058	0.0012	0.349	0.596	83	98	98
10,000	1001	19.12	1.256	0.060	0.0010	0.370	0.586	166	182	182

- Problem is solved very quickly.
- Timing is nearly linear in the number of states for modest grid size.
- The likelihood function, the constraints, and their derivatives are evaluated only 45-182 times in this example.
- In contrast, the Bellman operator in NFXP (the constraints here) is evaluated hundreds of times in NFXP.

Resampling

- Resampling can often be used to generate standard errors
- We did 20 resamplings:
  - Five parameter estimation
  - 1000 data points
  - 1001 grid points in DP

Dataset	RC	theta_c1	theta_c2	theta_p1	theta_p2	theta_p3	cpu time	major iter	func evals
1	1.1442	0.0559	0.0014	0.3604	0.6076	0.032	7.53	84	116
2	1.3086	0.0343	0.0049	0.3684	0.5806	0.051	6.3	79	89
3	1.1776	0.0597	0	0.3654	0.5956	0.039	7.53	62	84
4	1.141	0.00956	0.006344	0.3764	0.5856	0.038	6.2	69	99
5	1.3139	0.0725	0	0.3864	0.5655	0.048	6.48	80	87
6	0.0963	0	0.0069	0.3654	0.5856	0.049	8.78	107	123
7	1.4962	0.094	0	0.3444	0.6016	0.054	12.55	82	95
8	1.284	0	0.0075	0.3764	0.5776	0.046	11.16	87	141
9	1.4792	0.0888	0	0.3834	0.5645	0.052	9.64	83	98
10	1.0454	0.03438	0.0007	0.3734	0.5916	0.035	12.08	138	157
11	1.4322	0.0745	0	0.3604	0.5866	0.053	9.17	77	98
12	1.3021	0.0788	0	0.3954	0.5695	0.035	7.78	68	82
13	1.1619	0.0425	0.002	0.3724	0.5886	0.039	12.99	157	174
14	1.0986	0	0.0085	0.3674	0.5866	0.046	6.4	73	101
15	1.1681	0.0507	0.00175	0.3704	0.5836	0.046	10.64	122	152
16	1.0635	0.0503	0.0001	0.3614	0.5916	0.047	10.26	119	142
17	1.1323	0.016	0.0038	0.3874	0.5756	0.037	14.56	134	149
18	1.2319	0	0.0082	0.3724	0.5916	0.036	7.05	81	116
19	1.1718	0	0.01015	0.3634	0.5816	0.0551	8.42	79	93
20	1.1047	0.0591	0	0.3714	0.5826	0.046	11.36	70	86
mean	1.1677	0.0411	0.0031	0.3711	0.5847	0.0442	9.344	92.55	114.1
std error	0.2844	0.0323	0.0036	0.0114	0.0109	0.0072	2.510448	27.04864	28.55263

Comparisons with NFXP

- We reduce time spent on solving DP
  - Important when DP is hard to solve
  - Less important as the cost of computing likelihood rises
- Closed-form solutions may hurt
  - Substituting out m variables from n squeezes all nonlinearities into the remaining $n - m$ dimensions.
  - Nonlinear elimination of variables reduces number of unknowns but may increase nonlinearity
  - Actually, it is often easier to solve large optimization problems!
  - In optimization, it is nonlinearity, not dimensionality, that makes a problem difficult.
- JS is far more flexible and easy to implementation.
  - Derivatives of both DP solution and likelihood are easier to compute
  - NFXP has a hard time doing analytic derivatives of DP step; uses finite differences
  - This approach encourages one to experiment with many solvers to find the best one

Comparison with Rust Implementation

- Ease of use
  - Rust used Gauss "because: 1. the GAUSS langauge is a high-level symbolic language which enables a nearly 1:1 translation of mathematical formulae into computer code. Matrix operations of GAUSS replace cumbersome do-loops of FORTRAN. 2. GAUSS has built-in linear algebra routines, no links to Lapack needed"
  - JS: AMPL is also easy to use. All solvers have access to linear algebra routines. AMPL does not have matrix notation, but its approach to matrices, tensors, and indexed sets is very flexible.
- Optimization Method
  - Rust: Outer iteration uses BHHH for a while then switches to BFGS, where the user chooses the switch point.
  - JS: Use solvers far superior to these methods.
- Derivatives
  - Rust: "The NFXP software computes the value of and its derivatives numerically in a subroutine. This implies that we can numerically compute and its derivatives for each trial value encountered in the course of the process of maximizing. In order to do this, we need a very efficient and accurate algorithm for computing the fixed point."
  - JS: Use true analytic derivatives. This is done automatically by AMPL, and is done efficiently using ideas from automatic differentiation.

- Dynamic programming method
  - Rust: "Inner Fixed Point Algorithm. Contraction mapping fixed point (poly)algorithm. The algorithm combines contraction iterations with Newton-Kantorovich iterations to efficiently compute the functional fixed point." In Rust, contraction iterations are linearly convergent; quadratic convergence is achieved only at final stage.
  - JS: We use Newton-style methods that are globally faster than contraction mapping ideas. This is particularly important if β is close to 1, representing short, but realistic, time periods.

```
"NFXP.GPR: nested fixed point maximum likelihood algorithm";
"Version 8, October 2000. By John Rust, Yale University";

/* DISPLAY TOLERANCE AND INITIAL PARAMETER SETTINGS */

let fmt[6,3]="-*.*lf" 6 0
             "-*.*s" 8 8
             "*.*lf" 14 6
             "*.*lf" 14 6
             "*.*lf" 14 6
             "*.*lf" 14 6;
format /m1,/rz 12,8;

declare npk,dt,lnp,invp,c,dc,tr,ev,ev1,rtol,pr,cstp,pk,cdtp,dev,dtp,q11,q22;
declare dll0,h0;
declare modnum ?= 0;

if ((rows(modnum) == 1) or (modnum[1] == 0));
  "";
  "NFXP algorithm and model parameters have not been initialized";
  "Run SETUP.GPR to initialize algorithm and model settings";
  setup:
  #include "setup.gpr";
   if swj < 0;
   goto abort;
   endif;
endif;

{modnum,nvec,cstr}=display(modnum);
"fixed point dimension n=";; n;

if ss2 == 0; ss2=1; endif;

settings:

if modnum[4] == 0;
"beta value, 0<bet<1                  ";; bet; endif;
if modnum[3] == 1;
"transition probabilities";; p[1:1+modnum[5],1]'; endif;
"jacobian index, d (0<d<1)            ";; d;
"maximum number of lf evaluations     ";; maxlfe;
"maximum number of NK steps           ";; nstep;
"maximum stepsize                     ";; maxstp;
"maximum linesearch iterations        ";; maxls;
"minimum linesearch iterations        ";; minls;
"maximum contraction iterations       ";; maxcstp;
"minimum contraction iterations       ";; mincstp;
"switch to step size 1 when grad*direc<";; itol2;
"lf ratio to terminate line search    ";; itol1;
"initial stepsize                     ";; ss2;
"outer convergence tolerance          ";; ctol;
"inner convergence tolerance          ";; ltol;
"switch to Broyden when grad*direc <  ";; stol;
"check derivatives of lf? (1=yes, 0=no)";; drvchk;
"print hessian matrix?  (1=yes, 0=no) ";; hpr;
"print fixed point info? (1=yes, 0=no)";; pr;
"are these settings OK? (1=yes, 0=no) ";; swj=con(1,1);
```

```
if swj == 0;
   "Run  SETUP.GPR to change settings";
   goto setup;
endif;

if bet > 0;
 cstp=maxcstp;
 tcstp=cstp;
 ev=zeros(n,1);
 "set cstp=maxcstp when (q-q0) > : ";; qtol1';
 "set cstp=mincstp when (q-q0) < : ";; qtol';
endif;

"absolute parameter bounds  ";; qmax';
"initial parameter estimates";; q';

if modnum[3:4] == 2|1;
      "(implied beta value)";;  1/(1+exp(q[dm-modnum[5],1]));
elseif modnum[3:4] == 1|1;
      "(implied bet value)";; 1/(1+exp(q[dm,1]));
endif;

"number of rows read per pass";; t; "";

"plot hazard and value functions? (1=yes,0=no)   ";;
 grf=con(1,1); "";

 if grf; load npk; endif;


/* INITIALIZE VARIABLES USED IN PROGRAM */

ttt=hsec; i=1; ss1=0; nwt=0; ftol=1; ls=0; lfe=0; ll=0; idf0=2*stol;
m=floor(.5*n); q11=zeros(m,m+modnum[5]); q22=zeros(n-m,n-m);
q0=zeros(dm,1); lcstp=cstp; dm=rows(q); direc=ones(dm,1); dparm=direc;

if bet < .98; rtol=.0001; else; rtol=.0000001; endif;

if drvchk;

   "Enter delta value for numerical derivatives ";; delta=con(1,1);
   oll=0;

endif;

/* OPEN FILE BDT.DAT FOR READ  */

closeall; f1=0; open f1=bdt for read;
nr=rowsf(f1); swj=0; err=0;
"number of rows of data      "$+ftos(nr,"*.*lf",5,0);
 pass=(nr-(nr%t))/t; nr=nr%t;

/* BEGIN OUTER BHHH OPTIMIZATION ALGORITHM */

bhhhstart:
```

```
lfe=lfe+1;
if lfe >= maxlfe;
   "terminate at lf evaluation limit";;maxlfe;
    goto term;
endif;
if ls > 0; pr=0; endif;
if ls == 1; cstp=lcstp;  endif;

/* UPDATE COST FUNCTION c FOR CURRENT PARAMETER ESTIMATES */

function;
clear d1v, lp, dt;
if bet == 0;
   pk=1/(1+exp(c-tr-c[1,1]));
   goto core;
endif;
"";

/* ADAPTIVE DETERMINATION OF SWITCHPOINT FROM CONTRACTION TO
   Newton-Kantorovich iterations */

if sumc(abs(dparm) .> qtol1) > 0; tcstp=maxcstp; endif;

if  (abs(dparm) < qtol); tcstp=mincstp;
    if ftol < ltol; nstep=1;
       elseif ftol >= ltol; nstep=4;
    endif;
elseif ftol > ltol and nstep == 1;
    nstep=4;
endif;

if lfe > 1 and (ftol > ltol); tcstp=cstp+10; endif;

if nwt > 2 and swj >= cstp;
   tcstp=cstp+10;
elseif nwt > 2 and swj < cstp;
   rtol=.5*rtol;
elseif ftol < ltol and swj >= cstp;
   tcstp=cstp-10;
elseif ftol < ltol and swj < cstp;
   rtol=2*rtol;
endif;

cstp=tcstp;

if cstp > maxcstp; cstp=maxcstp; elseif cstp < mincstp; cstp=mincstp; endif;

/* UPDATE TRANSITION MATRIX P FOR CURRENT PARAMETER ESTIMATES */

if not(modnum[3] == 1 and i > 1);
   q11=eq11; q22=eq22;
endif;

/* CALL INNER FIXED POINT ALGORITHM */

ffxp;
```

```
if pr; ""; "return to outer BHHH algorithm"; endif;

/* CORE OF BHHH ALGORITHM: CALCULATION OF DIRECTION AND STEPSIZE */

core:

tt=hsec; h=zeros(dm,dm); dll=zeros(dm,1); ll=0; ii=1;

do until eof(f1);

   if ii > pass; nt=nr; else; nt=t; endif;

   /* read in a chunk of data into the matrix dt  */

   dt=readr(f1,t); ii=ii+1;

   /* EVALUATE LOGLIKELIHOOD FUNCTION AND DERIVATIVES for data in dt
      call eval if no lagged dependent variable, otherwise call eval1 */

   if modnum[7] == 0;
       eval;
   else;
       eval1;
   endif;


   /* CUMULATE MOMENT MATRIX                                    */

   if ls == 0;
       h=h+moment(d1v,1);
   endif;

endo;

y=seekr(f1,1);  /* reposition pointer at beginning of data file for
                next call to evaluate the likelihood */

   /* Conduct derivative check if drvchk = 1  */

   if drvchk;

      if (i == 1);

       q0=q;
         ndll=zeros(dm,1);
       oll=ll;
       "Numerical check of derivatives at the followin initial parameters";
         "    Parameter      Estimates     direction     gradient      t-stat";
         err=printfm(seqa(1,1,dm)~nvec~q~direc~dll~(q./(sqrt(diag(h)))),
                   ones(dm,1)~zeros(dm,1)~ones(dm,4),fmt);
         "";

         if modnum[3:4] == 2|1;
            "(implied bet value)";; 1/(1+exp(q[dm-modnum[5],1]));
         elseif modnum[3:4] == 1|1;
            "(implied bet value)";; 1/(1+exp(q[dm,1]));
         endif; "";
```

```
        "log likelihood ss=0 "$+ftos(ll,"*.*lg",12,8);;
        " grad*direc ";; idf0=dll'*direc; ftos(idf0,"*.*lg",12,8);
        "Iteration  "$+ftos(i,"*.*lf",3,0)$+" evaluation number ";;
        ftos(lfe,"*.*lf",3,0);

      elseif ((i > 1) and (i <= dm+1));

       q[i-1]=q0[i-1];
       ndll[i-1]=(ll-oll)/delta;

      else;

       "Comparison of analytical and numerical gradients with delta=";;
       ftos(delta,"*.*lg",12,9);
    "    Parameter        Value        analytic     numerical     difference";
        err=printfm(seqa(1,1,dm)~nvec~q~dll~ndll~abs(dll-ndll),
          ones(dm,1)~zeros(dm,1)~ones(dm,4),fmt);

       goto abort;

      endif;

      if (i <= dm);

          q[i]=q0[i]+delta;
          i=i+1;
        goto bhhhstart;

      endif;

   endif;

/* MEMORIZE NUMBER OF CONTRACTION STEPS, BRANCH TO LINE SEARCH  */

if ls == 1; lcstp=cstp; endif;

if ls > 0; goto lsrch; endif;

/* CALCULATE NEW DIRECTION VECTOR, CHECK FOR NUMERICAL ERRORS   */

if hpr > 0; "Cumulated outer product of first derivatives"; h; endif;


if idf0 > stol;  /* start out using BHHH  */

    h=invpd(h);
    h0=h;

else;            /* update h recursively using BFGS iterations */

   qk=(dll0-dll);
   rk=ss2*direc;
   h=h0;
   if rk'qk > 0;
       "using BFGS update for h ";
       h=(h-h*qk*qk'h/(qk'h*qk))*rk'qk/(qk'h*qk)+rk*(rk')/(rk'qk);
```

```
       /* Davidon-Fletcher-Powell update formula is
       h=h+rk*(rk')/(rk'rk)-h*qk*qk'h/(qk'h*qk);       */
   else;
      "BFGS failed: using last h";; rk'qk;
       h0=h;
   endif;

endif;

if hpr > 0; h; endif;

direc=h*dll;

if ss2 > 2; ss2=1; endif;

q0=q;

q=q0+ss2*direc;   /* update new parameter vector as previous value time
                  the stepsize ss2 times the direction vector */

tt=hsec-tt;

if abs(q) >= qmax;
   ss2=minc(.9*qmax./abs(q0+direc)); q=q0+ss2*direc;
   "parameter out of bounds ss=";; ss2;
endif;

/* PRINT OUT SUMMARY OF CURRENT ITERATION                      */

"";
"    Parameter estimates for cost function "$+cstr; "";
"    Parameter      Estimates     direction     gradient      t-stat";
     err=printfm(seqa(1,1,dm)~nvec~q~direc~dll~(q./(sqrt(diag(h)))),
       ones(dm,1)~zeros(dm,1)~ones(dm,4),fmt);
"";

if modnum[3:4] == 2|1;
   "(implied bet value)";; 1/(1+exp(q[dm-modnum[5],1]));
elseif modnum[3:4] == 1|1;
   "(implied bet value)";; 1/(1+exp(q[dm,1]));
endif; "";

"log likelihood ss=0 "$+ftos(ll,"*.*lg",12,8);;
" grad*direc ";; idf0=dll'*direc; ftos(idf0,"*.*lg",12,8);
"Iteration  "$+ftos(i,"*.*lf",3,0)$+" evaluation number ";;
 ftos(lfe,"*.*lf",3,0);
"Cpu time to cumulate moment matrix "$+ftos(tt,"*.*lf",4,0); "";

i=i+1;  /* increment iteration counter */

if idf0 < ctol;
     goto term;
elseif idf0 < itol2;
     ls=0; ss2=1; lcstp=cstp;
else;
     ss0=0; ss1=ss2; df0=idf0; dll0=dll; lll=ll; ls=1; dparm=ss2*direc;
endif;
```

```
if grf;

library pgraph;
fonts("simplex complex microb simgrma");
_pdate="";
_ptitle="Estimated Hazard Functions: NFXP Iteration "$+ftos(i,"*.*lf",4,0);
_pylabel="Probability of Replacement";
_pltype=6|3|1|5;
_plegctl=2~4.6~1~5.1;
_plegstr="DP Model Hazard\000Non-Parametric Hazard";

#IFUNIX

      let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
      if (i == 2);
      wxy = WinOpenPQG(v,"Estimated Hazard Functions","XY");
      else;
       call WinClear(wxy);
      endif;
      call WinSetActive(wxy);

#ENDIF
xy(ogrid,(1-pk)~npk);
#IFUNIX
   call WinSetActive(1);
#ENDIF


_ptitle="Estimated Cost/Value Functions: NFXP Iteration "$+ftos(i,"*.*lf",4,0);
_pylabel="Normalized Cost/Value Functions";
_plegstr="c\000\204b\201EV\000c+\204b\201EV";
_pltype=1|3|6|5;
_plegctl=2~4.6~1~5;
#IFUNIX
      let v = 100 100 640 480 0 0 1 6 15 0 0 2 2;
      if (i == 2);
      wxz = WinOpenPQG(v,"Estimated Value Functions","XY");
      else;
       call WinClear(wxz);
      endif;
      call WinSetActive(wxz);
#ENDIF
xy(ogrid,(c-c[1])~(-bet*ev+bet*ev[1])~(c-bet*ev-c[1]+bet*ev[1]));
#IFUNIX
   call WinSetActive(1);
#ENDIF

endif;

goto bhhhstart;

/* LINESEARCH ALGORITHM BY SECANT METHOD            */

lsrch:

"linesearch";; ls;; print "derivative time";; hsec-tt;
```

```
"likelihood ss=";; ss2;; ll;
  df1=dll'*direc;
 "df1,df0";; df1;; df0;
  ss2=ss1-df1*(ss0-ss1)/(df0-df1);

if ss2 < 0;
  "negative step";; ss2; ss2=ss1; ls=0;
  q=q0+ss2*direc; dparm=zeros(dm,1); goto bhhhstart;
endif;

if ss2 > maxstp;
  "step exceeds maximum";; ss2;
   ss2=ss1; ls=0;
   q=q0+ss2*direc; dparm=zeros(dm,1); goto bhhhstart;
endif;

if ls > maxls;
  "line search limit ss=";; ss2;
   ls=0; q=q0+ss2*direc;
   dparm=(ss2-ss1)*direc; goto bhhhstart;
endif;

if (ll/lll > itol1 and ls > minls) or ss2 > maxstp;
  "last step ss=";; ss2;
   ls=0; q=q0+ss2*direc; dparm=(ss2-ss1)*direc;
   goto bhhhstart;
endif;

q=q0+ss2*direc; dparm=(ss2-ss1)*direc;
ls=ls+1; df0=df1;
lll=ll; ss0=ss1; ss1=ss2;

goto bhhhstart;

/* TERMINATION OF PROGRAM                                    */

term:

closeall;
""; "NFXP converged: total execution time";; hsec-ttt;

abort:

#IFUNIX
  call WinSetActive(1);
#ENDIF
```

```
/* FFXP.G: contraction mapping fixed point polyalgorithm
   Uses fast block elimination to perform Newton-Kantorovich iteration
   Version 3, October 2000. By John Rust, Yale University */

proc (0)=ffxp;

     local j,tt,nwt,tol,tol1,mm;

/* PHASE 1: CONTRACTION ITERATIONS */

      if pr;
      "   begin contraction iterations";
      endif;

      tt=hsec;
      nwt=0;
      j=0;
      tol1=1;

      ev=center(ev); /* Center value function for numerical stability */

begin:
      j=j+2;

      ev1=contract(ev);
      ev1=center(ev1);
      ev=contract(ev1);

      tol=maxc(abs(ev-ev1));
      mm=tol/tol1;
      tol1=tol;

      if pr == 1;
         tol;; mm;; j;
      endif;

      if j >= cstp and nstep == 1;
         swj=j;
         nwt=0;
         if pr == 0;
               tol;; mm;; j;
         endif;

       if pr;
         "   begin Newton-Kantorovich iterations at time";; hsec-tt;
       endif;

         ev=center(ev);
         ev1=contract(ev);
         goto fin;

      endif;

      if tol < .1 or j >= cstp;
         goto newt;
      endif;
```

```
      if j >= mincstp and mm > bet*bet-rtol;
         goto newt;
      endif;

      goto begin;

/* PHASE 2: NEWTON-KANTOROVICH ITERATIONS */

newt:
      if pr == 2;
           tol;; mm;; j;
      endif;
      nwt=1;
      swj=j;

      if pr;
      "   begin Newton-Kantorovich iterations at time";; hsec-tt;
      endif;

      ev=center(ev);
      ev1=contract(ev);

qmat:
      pk=1/(1+exp(c-bet*(d*ev+(1-d)*ev1)-tr-c[1,1]+
      bet*(d*ev[1,1]+(1-d)*ev1[1,1])));
      ev1=ev-partsolv((ev-ev1));
      tol=maxc(abs(ev-ev1));

      ev1=center(ev1);
      ev=contract(ev1);

      if pr; tol;; endif;
      tol=maxc(abs(ev-ev1));
      if pr; tol;; endif;

      if tol < ltol or nwt > nstep;
           goto fin1;
      endif;

      nwt=nwt+1;

      pk=1/(1+exp(c-bet*(d*ev1+(1-d)*ev)-tr-c[1,1]+
      bet*(d*ev1[1,1]+(1-d)*ev[1,1])));
      ev=ev1-partsolv((ev1-ev));
      tol=maxc(abs(ev-ev1));

      ev=center(ev);
      ev1=contract(ev);

      if pr; tol;;  endif;
      tol=maxc(abs(ev-ev1));
      if pr; tol;;  endif;

      if tol < ltol or nwt > nstep;
            goto fin;
      else;
            nwt=nwt+1;
```

```
            goto qmat;
      endif;

/* PHASE 3: FINAL N-K STEP AND CALCULATION OF DERIVATIVES OF EV */

fin:
      nwt=nwt+1;
      pk=1/(1+exp(c-bet*(d*ev+(1-d)*ev1)-tr-c[1,1]+
      bet*(d*ev[1,1]+(1-d)*ev1[1,1])));

      if modnum[3] == 1;
           goto dv;
      endif;

      cdtp(ev);
dv:
      dev=(e(pk)-1)~(e(((dc[1,.]-dc).*pk))-dc[1,.]);
      if modnum[3:4] == 2|1;
           dev=dev~(-bet*(1-bet)*(ev[1,1]+e(((ev-ev[1,1]).*pk))))~
           dtp~(ev-ev1);
      elseif modnum[3:4] == 2|0;
           dev=dev~dtp~(ev-ev1);
      elseif modnum[3:4] == 1|0;
           dev=dev~(ev-ev1);
      elseif modnum[3:4] == 1|1;
           dev=dev~(-bet*(1-bet)*(ev[1,1]+e(((ev-ev[1,1]).*pk))))~
               (ev-ev1);
      endif;

      dev=partsolv(dev);
      ev1=ev-dev[.,dm-modnum[7]+1];
      tol=maxc(abs(ev-ev1));
      dev[.,1:dm-modnum[7]]=bet*dev[.,1:dm-modnum[7]];

      ev1=center(ev1);
      ev=contract(ev1);

      pk=1/(1+exp(c-bet*ev-tr-c[1,1]+bet*ev[1,1]));
      tol;; tol=maxc(abs(ev-ev1)); tol;

      goto term;

fin1:
      nwt=nwt+1;
      pk=1/(1+exp(c-bet*(d*ev1+(1-d)*ev)-tr-c[1,1]+
      bet*(d*ev1[1,1]+(1-d)*ev[1,1])));

      if modnum[3] == 1;
           goto dv1;
      endif;

      cdtp(ev1);
dv1:
      dev=(e(pk)-1)~(e(((dc[1,.]-dc).*pk))-dc[1,.]);

      if modnum[3:4] == 2|1;
            dev=dev~(-bet*(1-bet)*(ev1[1,1]+e(((ev1-ev1[1,1]).*pk))))~
```

```
            dtp~(ev1-ev);
      elseif modnum[3:4] == 2|0;
            dev=dev~dtp~(ev1-ev);
      elseif modnum[3:4] == 1|0;
            dev=dev~(ev1-ev);
      elseif modnum[3:4] == 1|1;
            dev=dev~(-bet*(1-bet)*(ev1[1,1]+e(((ev1-ev1[1,1]).*pk))))~
                (ev1-ev);
      endif;

      dev=partsolv(dev);
      ev=ev1-dev[.,dm-modnum[7]+1];
      tol=maxc(abs(ev-ev1));
      dev[.,1:dm-modnum[7]]=bet*dev[.,1:dm-modnum[7]];

      ev=center(ev);
      ev1=contract(ev);

      pk=1/(1+exp(c-bet*ev1-tr-c[1,1]+bet*ev1[1,1]));
      if pr; tol;; endif;
      tol=maxc(abs(ev-ev1));
      if pr; tol; endif;

      ev=ev1;
term:
      if pr; "   Fixed point time";; hsec-tt; endif;
endp;
```

```
#####  HAROLD ZURCHER BUS REPAIR EXAMPLE  #####
#
# An NLP method to compute maximum likelihood estimates of the
# Harold Zurcher bus problem in Rust, Econometrica, 1987.
# K. Judd and C.-L. Su,
# February 21, 2006. Current version: October 26, 2006
#
#
#####  SET UP THE PROBLEM  #####
#
#  Define the state space used in the dynamic programming part
param    N;   # number of states used in dynamic programming approximation
set X := 1..N;          # X is the index set of states
#
#  x[i] denotes state i; the set of states is a uniform grid on the interval [xmin, xmax]
param xmin := 0;
param xmax := 100;
param x {i in X} := xmin + (xmax-xmin)/(N-1)*(i-1);
#
#  Define and process the data
param nT;    # number of periods in data
set T := 1..nT;         # T is the vector of time indices
param Xt {T};         # Xt[t] is the true mileage at time t
param dt {T};         # decision at time t
#  The dynamic programming model in the estimation lives on a discrete state
#  Binning process: assign true mileage Xt[t] to the closest state in X
param xt {t in T} := ceil(Xt[t]/(xmax-xmin)*(N-1)+0.5);
#
#  Define "known" structural parameters
#  We fix beta since data cannot identify it
param beta;           # discount factor
#
```

```
#####  DECLARE STRUCTURAL PARAMETERS WE WILL ESTIMATE #####
#####  Parameters for cost function
#  c(x, thetaCost) = thetaCost[1]*x + thetaCost[2]*x^2
var thetaCost {1..2} >= 0;
#
#####  Parameters and definition of transition process
#  thetaProbs defines Markov chain
var thetaProbs {1..3} >= 0.00001;
#
#  Define the Markov chain representing the changes in mileage on the x[i] grid.
#
#  The state increases by some amount in [0,JumpMax] where
#  JumpMax is the maximum increase in mileage in one period
#  JumpMax equals a fraction JumpRatio of the range [xmin,xmax]
param JumpRatio;
param JumpMax := (xmax-xmin) * JumpRatio;
#
#  We assume the jumps are independent of the current state
#  and the transition process has three pieces, each a uniform distribution
#  Define 1st break point for stepwise uniform distribution in mileage increase
param M1 := ceil(1/4*JumpMax/(xmax-xmin)*(N-1)+0.5);
#  Define 2nd break point for stepwise uniform distribution in mileage increase
param M2 := ceil(3/4*JumpMax/(xmax-xmin)*(N-1)+0.5);
#  Define end point for stepwise uniform distribution in mileage increase
param M := ceil(JumpMax/(xmax-xmin)*(N-1)+0.5);
#
#  Y is the vector of elements in transition rule
set Y := 1..M;
var TransProb {i in Y} =
        if      i <= M1                 then    thetaProbs[1]/M1
        else if i > M1 and i <= M2      then    thetaProbs[2]/(M2-M1)
        else    thetaProbs[3]/(M-M2);
#
#####  Scrap value parameter
var RC >= 0;
##### END OF STRUCTURAL VARIABLES  #####
#
```

```
#####  DECLARE EQUILIBRIUM CONSTRAINT VARIABLES #####
# The NLP approach requires us to solve equilibrium constraint variables
var EV {X};        # Value Function of each state
#####  END OF EQUILIBRIUM CONSTRAINT VARIABLES  #####
#
#####  DECLARE AUXILIARY VARIABLES  ######
#  Define auxiliary variables to economize on expressions
#  Cost[i] is the cost of regular maintenance at x[i].
var Cost {i in X} = sum {j in 1..2} thetaCost[j]*x[i]^(j);
#  CbEV[i] equals - Cost[i] + beta*EV[i];
#  this is the expected payoff at x[i] if regular maintenance is chosen
var CbEV {i in X} = - Cost[i] + beta*EV[i];
#  PayoffDiff[i] equals -CbEV[i] - RC + CbEV[1]; this is the difference in
#  expected payoff at x[i] between engine replacement and regular maintenance
var PayoffDiff {i in X} = -CbEV[i] - RC + CbEV[1];
#  ProbRegMaint[i] is 1/(1+exp(PayoffDiff[i])); this is the probability
#  of performing regular maintenance at state x[i];
var ProbRegMaint {i in X} = 1/(1+exp(PayoffDiff[i]));
#  BellmanViola[i] is the Bellman equation error at i;
var BellmanViola {i in 1..(N-M+1)} = sum {j in 0..(M-1)}
        log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1] - EV[i];

#####  END OF AUXILIARY VARIABLES  #####
#
```

```
#####  OBJECTIVE AND CONSTRAINT DEFINITIONS  #####
#
## Define objective function
##
maximize Likelihood:
#  The likelihood function contains two pieces
#  First: likelihood that engine is replaced given time t state
        sum {t in 2..nT} log(dt[t]*(1-ProbRegMaint[xt[t]]) + (1-dt[t])*ProbRegMaint[xt[t]])
#  Second: likelihood of the observed transition between t-1 and t
        + sum {t in 2..nT} log(dt[t-1]*(TransProb[xt[t]-1+1])
                                + (1-dt[t-1])*(TransProb[xt[t]-xt[t-1]+1]));

#
#  List the constraints, which are the Bellman equations
#
subject to
#
#  Bellman equation for states below N-M
        Bellman_1toNminusM {i in X: i <= N-(M-1)}:
                EV[i] = sum {j in 0..(M-1)}
                        log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1];
#
#  Bellman eqn for states above N-M, (adjust probabilities to keep state in [xmin, xmax])
        Bellman_LastM {i in X: i > N-(M-1) and i <= N-1}:
                EV[i] = (sum {j in 0..(N-i-1)}
                        log(exp(CbEV[i+j])+ exp(-RC + CbEV[1]))* TransProb[j+1])
                        + (1- sum {k in 0..(N-i-1)} TransProb[k+1]) * log(exp(CbEV[N])
                        + exp(-RC + CbEV[1]));
#
#  Bellman equation for state N
        Bellman_N:  EV[N] = log(exp(CbEV[N])+ exp(-RC + CbEV[1]));
#
#  The probability parameters in transition process must add to one
   Probability: sum {i in 1..3} thetaProbs[i] = 1;

#
#  Put bound on EV; should not bind, but helps keep algorithm within a priori bounds
   EVBound {i in X}: EV[i] <= 50;
#
##### END OF  OBJECTIVE AND CONSTRAINT DEFINITIONS  #####
```

```
#####   DEFINE THE PROBLEM   #####
#
# Name the problem
problem MPECZurcher:
#
# Choose the objective function
Likelihood,
#
# List the variables
EV, RC, thetaCost, thetaProbs, TransProb, Cost, CbEV, PayoffDiff, ProbRegMaint,
BellmanViola,
#
# List the constraints
Bellman_1toNminusM,
Bellman_LastM,
Bellman_N,
Probability,
EVBound;
#
#####################################
```

The JS Approach to games

- NFXP cannot be used to estimate data from games except for very special cases.
- Suppose that the game has parameters θ representing payoffs, probabilities, and whatever else is not observed directly by the econometrician.
- Let σ denote the equilibrium strategy given θ, and that σ is an equilibrium if and only if

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

  for some function G.
- Suppose that likelihood of a data set, x, is $\mathcal{L}(\theta, \sigma, X)$. Therefore, maximum likelihood is the problem

$$\begin{aligned} \max_{\sigma,\theta} \quad & \mathcal{L}(\theta, \sigma, X) \\ \text{s.t.} \quad & 0 = G(\sigma, \theta) \end{aligned}$$

- For each θ, NFXP would require one to find all the σ that solves $G(\sigma, \theta)$, compute the likelihood at each equilibrium σ, and report the max. Finding all equilibria is an intractable problem.
- In contrast, JS just sends the problem to good optimization solvers. Multiple equilibria may produce multiple local solutions, but that is a standard problem in maximum likelihood estimation, and would also be a problem for the NFXP approach.

JS Approach to Method of Moments

- Suppose you want to fit moments. E.g., likelihood may not exist in most of parameter space.
- Method then is

$$\begin{aligned} \min_{\sigma,\theta} \quad & \left\| M^{\text{model}}(\sigma,\theta) - M^{\text{data}}(X) \right\|^2 \\ \text{s.t.} \quad & 0 = G(\sigma,\theta) \end{aligned}$$

  - Could compute $M^{\text{model}}(\sigma,\theta)$ numerically via integral equation methods
  - Could simulate moments in $M^{\text{model}}(\sigma,\theta)$

JS Approach to GMM

- Efficient GMM estimation requires the use of the proper variance-covariance matrix W, which is a function of the estimates.
- Standard method iterates on W
- JS method is

$$\begin{aligned} \min_{\sigma,\theta,W} \quad & GMM\left(\sigma,\theta,W,X\right) \\ \text{s.t.} \quad & 0 = G\left(\sigma,\theta,W,X\right) \end{aligned}$$

Conclusion

- Structural estimation methods are far easier to construct if one includes the structural equations.
- The numerical algorithm advances of the past forty years (SQP, augmented Lagrangian, interior point, AD, MPCC) makes this tractable
- User-friendly interfaces (e.g., AMPL) makes this as easy to do as Stata, Gauss, and Matlab
- This approach makes structural estimation *really* accessible to a larger set of researchers.