Constrained Optimization Approaches to Estimation of Structural Models:

Mathematical programming with equilibrium constraints (MPEC)

Dynamic Programming and Structural Econometrics #7

Bertel Schjerning, University of Copenhagen

MPEC is used in multiple contexts

Single-Agent Dynamic Discrete Choice Models

- Rust (1987): Bus-Engine Replacement Problem
- Nested-Fixed Point Problem (NFXP)
- Su and Judd (2012): Constrained Optimization Approach

Random-Coefficients Logit Demand Models

- BLP (1995): Random-Coefficients Demand Estimation
- Nested-Fixed Point Problem (NFXP)
- Dube, Fox and Su (2012): Constrained Optimization Approach

Estimating Discrete-Choice Games of Incomplete Information

- Aguirregabiria and Mira (2007): NPL (Recursive 2-Step)
- Bajari, Benkard and Levin (2007): 2-Step
- Pakes, Ostrovsky and Berry (2007): 2-Step
- Pesendorfer and Schmidt-Dengler (2008): 2-Step
- Pesendorfer and Schmidt-Dengler (2010): comments on AM (2007)
- Kasahara and Shimotsu (2012): Modified NPL
- Su (2013), Egesdal, Lai and Su (2014): Constrained Optimization

Zurcher's Bus Engine Replacement Problem

- Choice set: Each bus comes in for repair once a month and Zurcher chooses between ordinary maintenance ($d_t = 0$) and overhaul/engine replacement ($d_t = 1$)
- State variables: Harold Zurcher observes:
  - x_t: mileage at time t since last engine overhaul
  - $\varepsilon_t = [\varepsilon_t(d_t = 0), \varepsilon_t(d_t = 1)]$: other state variable
- Utility function:

$$u(x_t, d, \theta_1) + \varepsilon_t(d_t) = \begin{cases} -RC - c(0, \theta_1) + \varepsilon_t(1) & \text{if } d_t = 1 \\ -c(x_t, \theta_1) + \varepsilon_t(0) & \text{if } d_t = 0 \end{cases} \quad (1)$$

- State variables process x_t (mileage since last replacement)

$$p(x_{t+1}|x_t, d_t, \theta_2) = \begin{cases} g(x_{t+1} - 0, \theta_2) & \text{if } d_t = 1 \\ g(x_{t+1} - x_t, \theta_2) & \text{if } d_t = 0 \end{cases} \quad (2)$$

- If engine is replaced, state of bus regenerates to $x_t = 0$.

Structural Estimation

Data: $(d_{i,t}, x_{i,t})$, $t = 1, ..., T_i$ and $i = 1, ..., n$

Likelihood function

$$\ell_i^f(\theta) = \sum_{t=2}^{T_i} log(P(d_{i,t}|x_{i,t}, \theta)) + \sum_{t=2}^{T_i} log\left(p(x_{i,t}|x_{i,t-1}, d_{i,t-1}, \theta_2)\right)$$

where

$$P(d|x, \theta) = \frac{\exp\{u(x, d, \theta_1) + \beta EV_\theta(x, d)\}}{\sum_{d' \in \{0,1\}} \{u(x, d', \theta_1) + \beta EV_\theta(x, d')\}}$$

and

$$\begin{aligned} EV_\theta(x, d) &= \Gamma_\theta(EV_\theta)(x, d) \\ &= \int_y \ln \left[\sum_{d' \in \{0,1\}} \exp[u(y, d'; \theta_1) + \beta EV_\theta(y, d')] \right] p(dy|x, d, \theta_2) \end{aligned}$$

The Nested Fixed Point Algorithm

NFXP solves the *unconstrained* optimization problem

$$\max_{\theta} L(\theta, EV_\theta)$$

Outer loop (Hill-climbing algorithm):

- Likelihood function $L(\theta, EV_\theta)$ is maximized w.r.t. θ
- Quasi-Newton algorithm: Usually BHHH, BFGS or a combination.
- Each evaluation of $L(\theta, EV_\theta)$ requires solution of EV_θ

Inner loop (fixed point algorithm):
The implicit function EV_θ defined by $EV_\theta = \Gamma(EV_\theta)$ is solved by:

- Successive Approximations (SA)
- Newton-Kantorovich (NK) Iterations

Mathematical Programming with Equilibrium Constraints

MPEC solves the *constrained* optimization problem

$$\max_{\theta, EV} L(\theta, EV) \text{ subject to } EV = \Gamma_\theta(EV)$$

using general-purpose constrained optimization solvers such as KNITRO

Su and Judd (Ecta 2012) considers two such implementations:

MPEC/AMPL:

- AMPL formulates problems and pass it to KNITRO.
- Automatic differentiation (Jacobian and Hessian)
- Sparsity patterns for Jacobian and Hessian

MPEC/MATLAB:

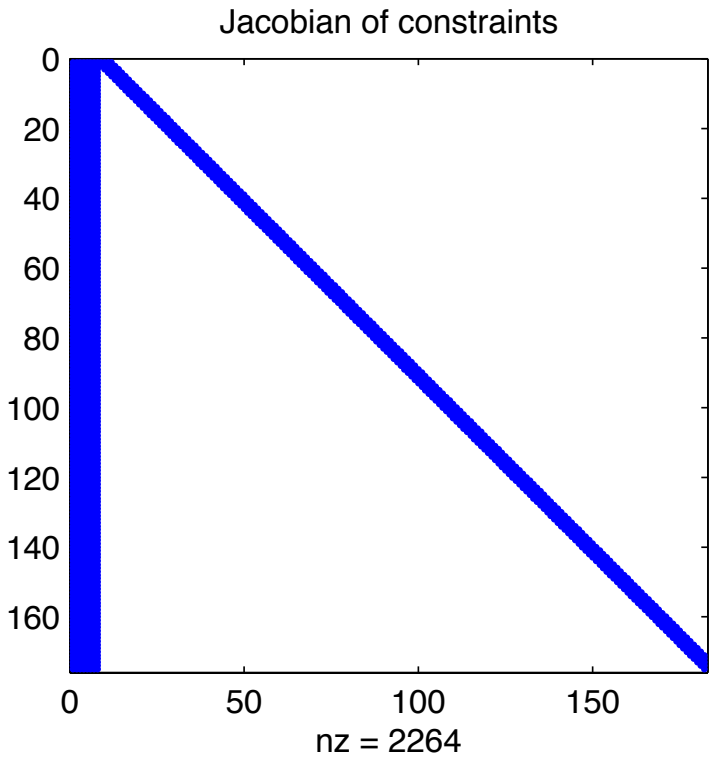
- User need to supply Jacobians, Hessian, and Sparsity Patterns
- Su and Judd do not supply analytical derivatives.
- ktrlink provides link between MATLAB and KNITRO solvers.

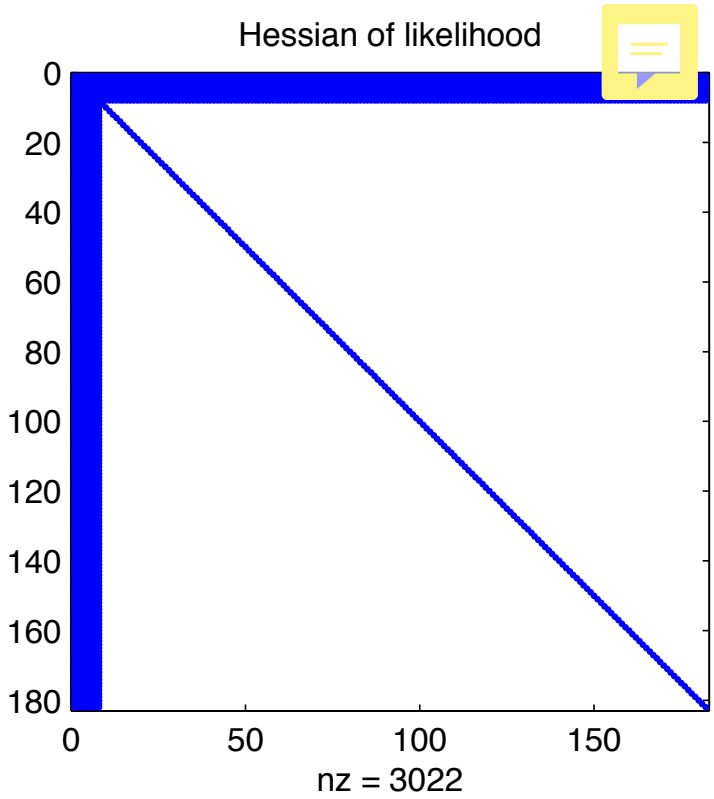
2nd order

Sparsity patterns for MPEC

Two key factors in efficient implementations:

- Provide analytic-derivatives (huge improvement in speed)
- Exploit sparsity pattern in constraint Jacobian (huge saving in memory requirement)





Zurcher's Bus Engine Replacement Problem

Discretize the mileage state space x into n grid points

$$\hat{X} = \{\hat{x}_1, ..., \hat{x}_n\} \text{ with } \hat{x}_1 = 0$$

Mileage transition probability: for $j = 1, ..., J$

$$p(x'|\hat{x}_k, d, \theta_2) = \begin{cases} Pr\{x' = \hat{x}_{k+j}|\theta_2\} = \theta_{2j} \text{ if } d = 0 \\ Pr\{x' = \hat{x}_{1+j}|\theta_2\} = \theta_{2j} \text{ if } d = 1 \end{cases}$$

Mileage in the next period x' can move up at most J grid points. J is determined by the distribution of mileage.

Choice-specific expected value function for $\hat{x} \in \hat{X}$

$$EV_\theta(\hat{x}, d) = \hat{\Gamma}_\theta(EV_\theta)(\hat{x}, d)$$

$$= \sum_j^J \ln \left[\sum_{d' \in D(y)} \exp[u(x', d'; \theta_1) + \beta EV_\theta(x', d')] \right] p(x'|\hat{x}, d, \theta_2)$$

Bellman equation in matrix form

The choice specific expected value function can be found as fixed point on the Bellman operator

$$EV(d) = \hat{\Gamma}(EV) = \Pi(d) * \ln\left[\sum_{d' \in D(y)} \exp[u(d') + \beta EV(d')]\right]$$

where

$EV(d) = [EV(1,d), .., EV(n,d)]$ and $u(d) = [u(1,d), .., u(n,d)]$

$\Pi(d)$ is a $n \times n$ state transition matrix conditional on decision d

Transition matrix for mileage is sparse

Transition matrix conditional on keeping engine

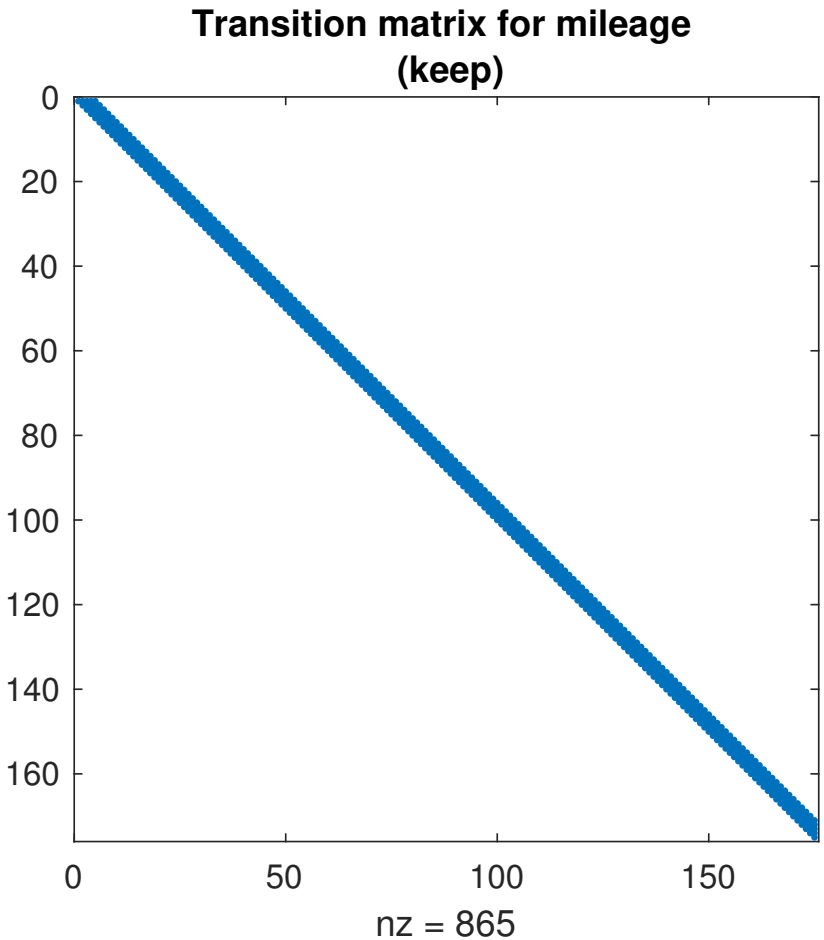
$$\Pi(d = \text{keep})_{nxn} = \begin{pmatrix} \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ 0 & & & & \pi_0 & \pi_1 & \pi_2 & 0 \\ 0 & & & & & \pi_0 & \pi_1 & \pi_2 \\ 0 & & & & & & \pi_0 & 1-\pi_0 \\ 0 & 0 & & & & & & 1 \end{pmatrix}$$

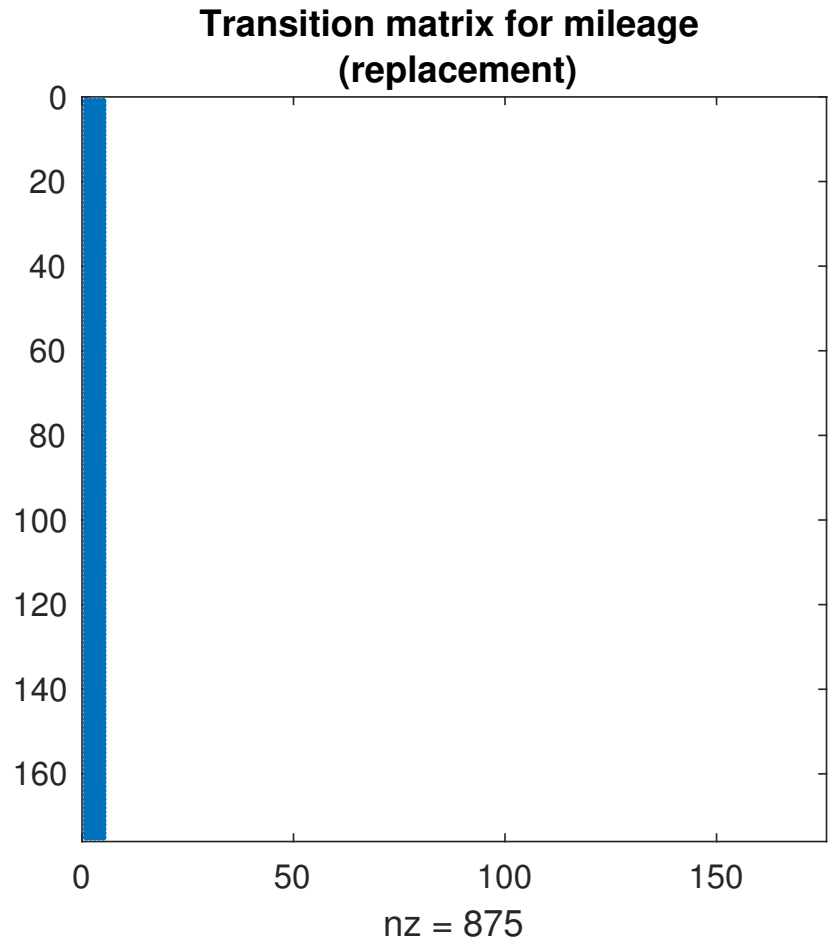
Transition matrix for mileage is sparse

Transition matrix conditional on replacing engine

$$\Pi(d = \text{replace})_{nxn} = \begin{pmatrix} \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \pi_0 & \pi_1 & \pi_2 & 0 & \cdot & \cdot & \cdot & 0 \end{pmatrix}$$

Transition matrix is sparse





Monte Carlo: Rust's Table X - Group 1,2, 3

- Fixed point dimension: $n = 175$
- Maintenance cost function: $c(x, \theta_1) = 0 : 001 * \theta_1 * x$
- Mileage transition: stay or move up at most $J = 4$ grid point
- True parameter values:
  - $\theta_1 = 2,457$
  - $RC = 11.726$
  - $(\theta_{21}, \theta_{22}, \theta_{23}, \theta_{24}) = (0.0937, 0.4475, 0.4459, 0.0127)$
- Solve for EV at the true parameter values
- Simulate 250 datasets of monthly data for 10 years and 50 buses

Is NFXP a dinosaur method?

Su and Judd (Econometrica, 2012)

TABLE II

NUMERICAL PERFORMANCE OF NFXP AND MPEC IN THE MONTE CARLO EXPERIMENTS[a]

β	Implementation	Runs Converged (out of 1250 runs)	CPU Time (in sec.)	# of Major Iter.	# of Func. Eval.	# of Contraction Mapping Iter.
0.975	MPEC/AMPL	1240	0.13	12.8	17.6	–
	MPEC/MATLAB	1247	7.90	53.0	62.0	–
	NFXP	998	24.60	55.9	189.4	134,748
0.980	MPEC/AMPL	1236	0.15	14.5	21.8	–
	MPEC/MATLAB	1241	8.10	57.4	70.6	–
	NFXP	1000	27.90	55.0	183.8	162,505
0.985	MPEC/AMPL	1235	0.13	13.2	19.7	–
	MPEC/MATLAB	1250	7.50	55.0	62.3	–
	NFXP	952	43.20	61.7	227.3	265,827
0.990	MPEC/AMPL	1161	0.19	18.3	42.2	–
	MPEC/MATLAB	1248	7.50	56.5	65.8	–
	NFXP	935	70.10	66.9	253.8	452,347
0.995	MPEC/AMPL	965	0.14	13.4	21.3	–
	MPEC/MATLAB	1246	7.90	59.6	70.7	–
	NFXP	950	111.60	58.8	214.7	748,487

[a]For each β, we use five starting points for each of the 250 replications. CPU time, number of major iterations, number of function evaluations and number of contraction mapping iterations are the averages for each run.

NFXP survival kit

Step 1: Read NFXP manual and print out NFXP pocket guide
Step 2: Solve for fixed point using Newton Iterations
Step 3: Recenter Bellman equation
Step 4: Provide analytical gradients of Bellman operator
Step 5: Provide analytical gradients of likelihood
Step 6: Use BHHH (outer product of gradients as hessian approx.)

STEP 1: NFXP documentation

References


Rust (1987): "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher" *Econometrica* 55-5, pp 999-1033.


Rust (2000): "Nested Fixed Point Algorithm Documentation Manual: Version 6"
`https://editorialexpress.com/jrust/nfxp.html`


Iskhakov, F. , J. Rust, B. Schjerning, L. Jinhyuk, and K. Seo (2015): "Constrained Optimization Approaches to Estimation of Structural Models : Comment." *Econometrica* 84-1, pp. 365-370.

Nested Fixed Point Algorithm

NFXP Documentation Manual version 6, (Rust 2000, page 18):

Formally, one can view the nested fixed point algorithm as solving the following constrained optimization problem:

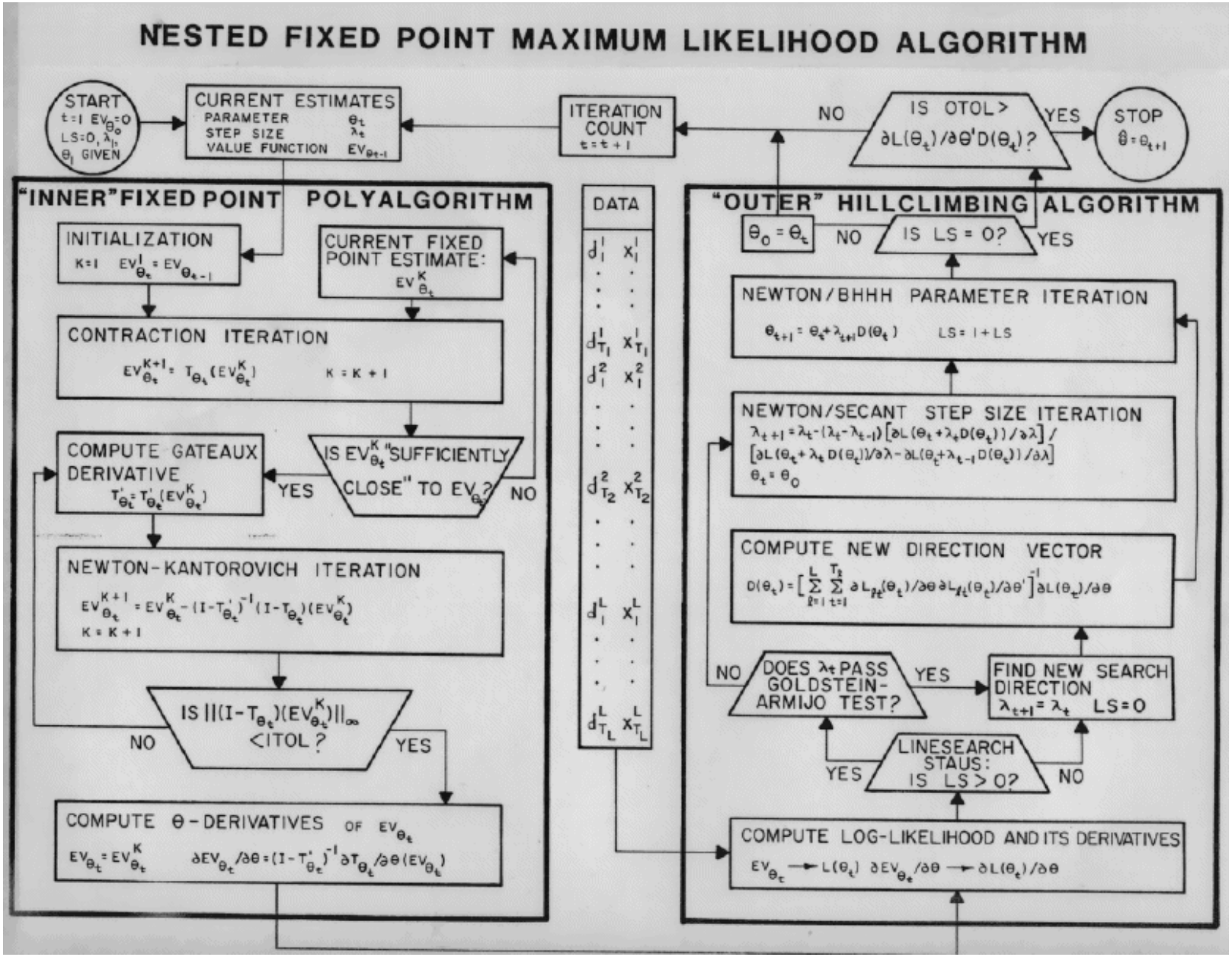
$$\max_{\theta, EV} L(\theta, EV) \text{ subject to } EV = \Gamma_\theta(EV) \tag{3}$$

Since the contraction mapping Γ *always has a unique fixed point, the constraint* $EV = \Gamma_\theta(EV)$ *implies that the fixed point* EV_θ *is an implicit function of* θ*. Thus, the constrained optimization problem (3) reduces to the unconstrained optimization problem*

$$\max_{\theta} L(\theta, EV_\theta) \tag{4}$$

where EV_θ *is the implicit function defined by* $EV_\theta = \Gamma(EV_\theta)$*.*

NFXP pocket guide



STEP 2: Newton-Kantorovich Iterations

- Problem: Find fixed point of the contraction mapping

$$EV = \Gamma(EV)$$

- Error bound on successive contraction iterations:
  $||EV_{k+1} - EV|| \leq \beta ||EV_k - EV||$
  linear convergence $\rightarrow$ slow when β close to 1
- Newton-Kantorovich:
  Solve $F = [I - \Gamma](EV_\theta) = 0$ using Newtons method
  $||EV_{k+1} - EV|| \leq A||EV_k - EV||^2$
  quadratic convergence around fixed point, EV

STEP 2: Newton-Kantorovich Iterations

Convert the problem of finding a fixed point $EV_\theta = \Gamma(EV_\theta)$ into the problem of finding a zero of the nonlinear operator $F_\theta(EV_\theta)$

$$F_\theta(EV_\theta) = (I - \Gamma_\theta)(EV_\theta) = 0$$

where I is the identity operator on B, and 0 is the zero element of B (i.e. the zero function).

Newton-Kantorovich iteration:

$$EV_{k+1} \quad = \quad EV_k - (I - \Gamma')^{-1}(I - \Gamma)(EV_k)$$

The nonlinear operator $F_\theta = I - \Gamma_\theta$ has a Fréchet derivative $I - \Gamma'_\theta$ which is a bounded linear operator on B with a bounded inverse.

The Fixed Point (poly) Algorithm

1. Successive contraction iterations
   (until EV is in domain of attraction)
2. Newton-Kantorovich (until convergence)

STEP 2: Newton-Kantorovich Iterations, $\beta = 0.9999$

Successive Approximations, VERY Slow

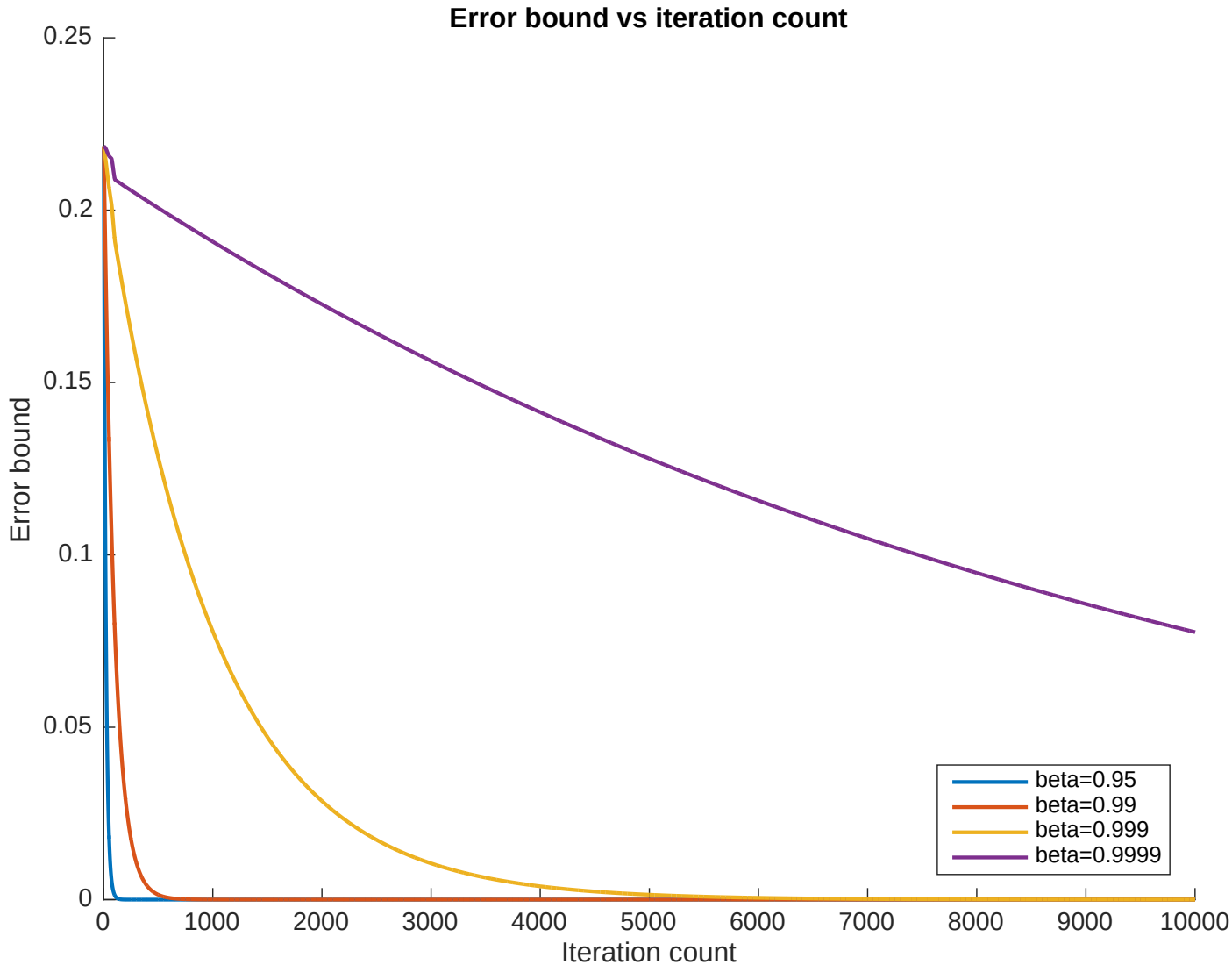
```
Begin contraction iterations
   j           tol         tol(j)/tol(j-1)
   1         0.24310300        0.24310300
   2         0.24307590        0.99988851
   3         0.24304810        0.99988564
   :              :                :
 9998        0.08185935        0.99990000
 9999        0.08185116        0.99990000
 10000       0.08184298        0.99990000
Elapsed time: 1.44752 (seconds)

Begin Newton-Kantorovich iterations
  nwt          tol
   1       9.09494702e-13
Elapsed time: 1.44843 (seconds)

Convergence achieved!
```

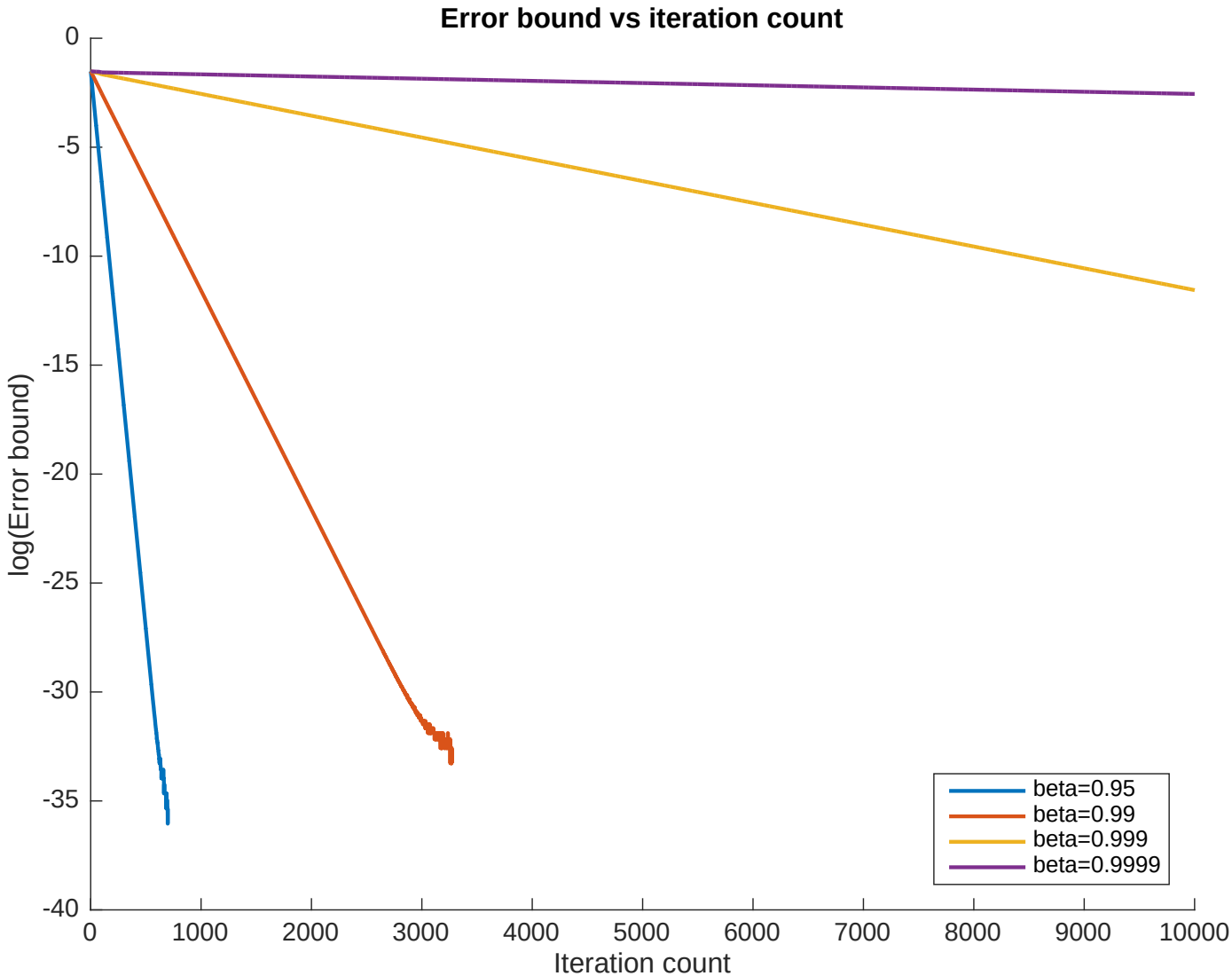
STEP 2: Newton-Kantorovich Iterations

Successive Approximations, VERY Slow



STEP 2: Newton-Kantorovich Iterations

Successive Approximations, Linear convergence



STEP 2: Newton-Kantorovich Iterations, $\beta = 0.9999$

Quadratic convergence!

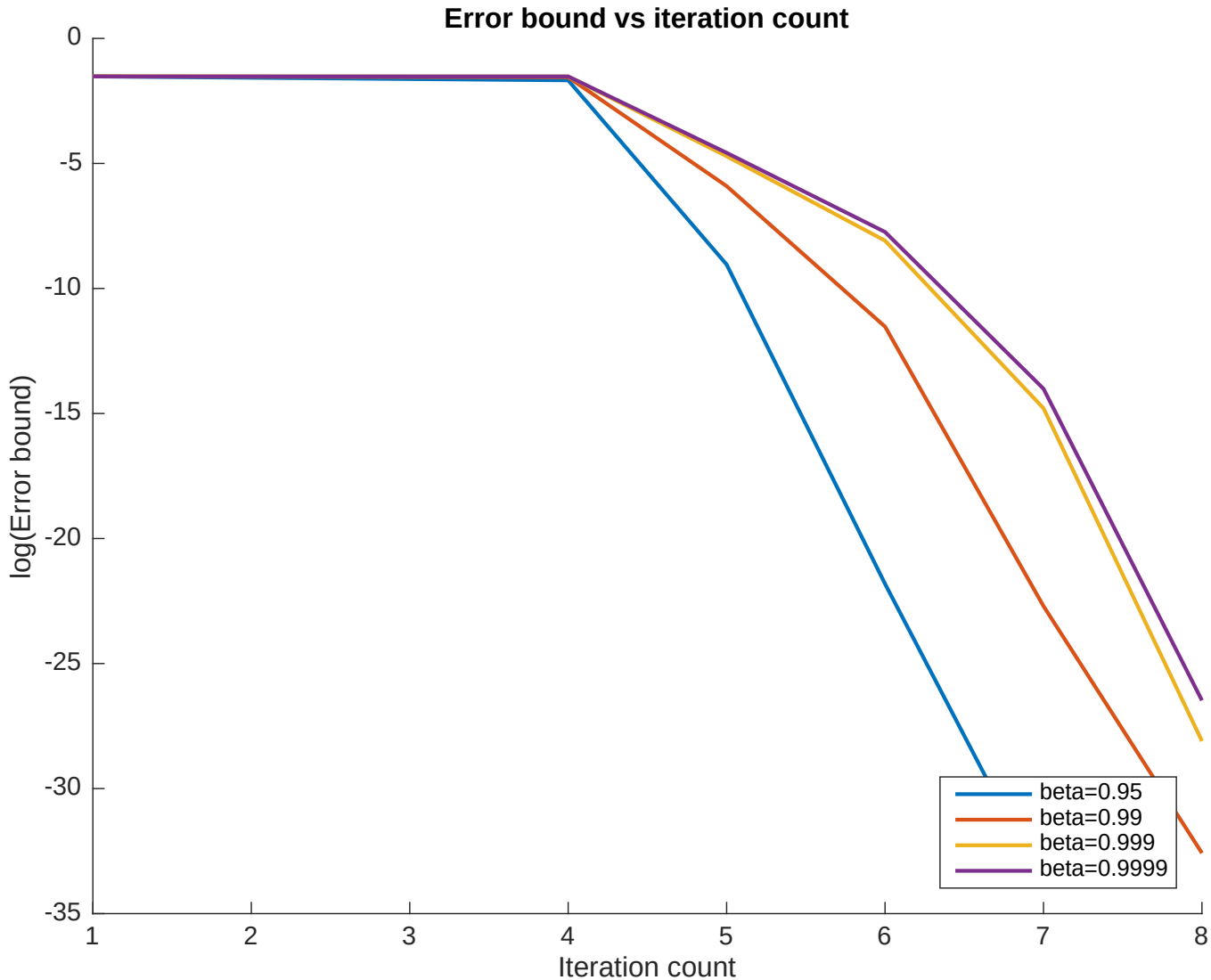
```
Begin contraction iterations
   j           tol         tol(j)/tol(j-1)
   1         0.21854635       0.21854635
   2         0.21852208       0.99988895
Elapsed time: 0.00056 (seconds)

Begin Newton-Kantorovich iterations
  nwt           tol
   1       1.03744352e-02
   2       4.40564315e-04
   3       8.45941486e-07
   4       3.63797881e-12
Elapsed time: 0.00326 (seconds)

Convergence achieved!
```

STEP 2: Newton-Kantorovich Iterations

NR: Quadratic convergence!



STEP 2: When to switch to Newton-Kantorovich

Observations:

- $tol_k = \|EV_{k+1} - EV_k\| < \beta\|EV_k - EV\|$
- tol_k quickly slow down and declines very slowly for β close to 1
- Relative tolerance tol_{k+1}/tol_k approach β

When to switch to Newton-Kantorovich?

- Suppose that $EV_0 = EV + k$.
  (Initial EV_0 equals fixed point EV plus an arbitrary constant)
- Another successive approximation does not solve this:

$$\begin{aligned} tol_0 &= \|EV_0 - \Gamma(EV_0)\| = \|EV + k - \Gamma(EV + k)\| \\ &= \|EV + k - (EV + \beta k)\| = (1-\beta)k \\ tol_1 &= \|EV_1 - \Gamma(EV_1)\| = \|EV + \beta k - \Gamma(EV + \beta k)\| \\ &= \|EV + \beta k - (EV + \beta^2 k)\| = \beta(1-\beta)k \\ tol_1/tol_0 &= \beta \end{aligned}$$

- Newton will immediately "strip away" the irrelevant constant k
- Switch to Newton whenever tol_1/tol_0 is sufficiently close to β

STEP 3: Recenter to ensure numerical stability

Logit formulas must be reentered.

$$\begin{aligned} P_i &= \frac{\exp(V_i)}{\sum_{j \in D(y)} \exp(V_j)} \\ &= \frac{\exp(V_i - V_0)}{\sum_{j \in D(y)} \exp(V_j - V_0)} \end{aligned}$$

and "log-sum" must be recenteret too

$$\begin{aligned} EV_\theta &= \int_y \ln \sum_{j' \in D(y)} \exp(V_j) p(dy|x, d, \theta_2) \\ &= \int_y \left(V_0 + \ln \sum_{j' \in D(y)} \exp(V_j - V_0) \right) p(dy|x, d, \theta_2) \end{aligned}$$

If V_0 is chosen to be $V_0 = \max_j V_j$ we can avoid numerical instability due to overflow/underflow

STEP 4: Analytical Fréchet derivative of Bellman operator

Fréchet derivative

- For NK iteration we need Γ'

$$EV_{k+1} \quad = \quad EV_k - (I - \Gamma')^{-1}(I - \Gamma)(EV_k)$$

- In terms of its finite-dimensional approximation, Γ'_θ takes the form of an $N \times N$ matrix equal to the partial derivatives of the $N \times 1$ vector $\Gamma_\theta(EV_\theta)$ with respect to the $N \times 1$ vector EV_θ
- Γ'_θ is simply β times the transition probability matrix for the controlled process $\{d_t, x_t\}$
- Two lines of code in MATLAB

STEP 1-4: MATLAB implementation of Γ_θ and Γ'_θ

```
1   function [ev1, pk, dbellman_dev]=bellman_ev(ev, mp, u, P)
2     vK= u(:,1) + mp.beta*ev;       % Value off keep
3     vR= u(:,2) + mp.beta*ev(1);    % Value of replacing
4
5     % Need to recenter logsum by subtracting max(vK, vR)
6     maxV=max(vK, vR);
7     V=(maxV + log(exp(vK-maxV)  +  exp(vR-maxV)));
8     ev1=P{1}*V;
9
10    if nargout>1      % If requested, also compute choice probability
11      pk=1./(1+exp((vR-vK)));
12    end
13    if nargout>2 % compute Frechet derivative
14      dbellman_dev=mp.beta*(P{1}.*pk');
15      % Add additional term for derivative wrt Ev(1), since Ev(1) enter logsum f
16      dbellman_dev(:,1)=dbellman_dev(:,1)+mp.beta*P{1}*(1-pk);
17    end
18  end % end of zurcher.bellman_ev
```

Bellman operator can also be written in terms of the smoothed value function

Define the smoothed value function $V_\sigma(x) = \int V(x,\epsilon)g(\epsilon|x)d\epsilon$ where σ represents parameters that index the distribution of the $\epsilon's$.

Under our assumptions so far, the smoothed value function, V_σ is a fixed point on the mapping

$$V_\sigma = \hat{\Gamma}_\sigma(V_\sigma) = \ln\left[\sum_{d' \in D(y)} \exp[u(d') + \beta\Pi(d') * V_\sigma]\right]$$

where $V_\sigma = [V_\sigma(1), .., V_\sigma(n)]$ and $u(d) = [u(1,d), .., u(n,d)]$

Easy to implement to implement Fréchet derivative.

STEP 1-4: MATLAB implementation based on smoothed value function

```
function [V1, pk, dBellman_dV]=bellman_iv(V0, mp, u, P)
  vK= u(:,1) + mp.beta*P{1}*V0;   % Value of keeping
  vR= u(:,2) + mp.beta*P{2}*V0;   % Value of replacing

  maxV=max(vK, vR);
  V1=(maxV + log(exp(vK-maxV)  +  exp(vR-maxV)));

  % If requested, also compute choice probability from ev (initial input)
  if nargout>1
    pk=1./(1+exp((vR-vK)));
  end

  if nargout>2 % compute Frechet derivative
    dBellman_dV=mp.beta*(P{1}.*pk + P{2}.*(1-pk));
  end
end % end of zurcher.bellman_iv
```

STEP 5: Provide analytical gradients of likelihood

Gradient similar to the gradient for the conventional logit

$$\partial \ell_i^1(\theta)/\partial\theta = \sum_{j \in J} (y_{it}^j - P(j|x_{it}, \theta)) \partial(v_{it}^j)/\partial\theta$$

for binary replacement choice

$$\partial \ell_i^1(\theta)/\partial\theta = [d_{it} - P(d_{it}|x_{it}, \theta)] \times \partial(v_{repl.} - v_{keep})/\partial\theta$$

- Only thing that differs is the inner derivative of the choice specific value function that besides derivatives of current utility also includes $\partial EV_\theta/\partial\theta$ wrt. θ
- By the implicit function theorem we obtain

$$\partial EV_\theta/\partial\theta = [I - \Gamma'_\theta]^{-1} \partial\Gamma/\partial\theta'$$

- By-product of the N-K algorithm: $[I - \Gamma'_\theta]^{-1}$

STEP 5: MATLAB implementation of scores

```
cost=0.001*mp.c*mp.grid;
dc=0.001*mp.grid;

% step 1: compute derivative of contraction operator wrt. parameters
dbellman_dmp=zeros(mp.n,2);
dbellman_dmp(:, 1)=(1-pk)*(-1); % Derivative wrt. RC
dbellman_dmp(:, 2)=pk.*(-dc);   % Derivative wrt. c

% step 2: compute derivative of ev wrt. parameters
devdmp=F\dbellman_dmp;

% step 3: compute derivative of log-likelihood wrt. parameters
score=bsxfun(@times, (data.d-pxR), ...
    [-ones(N,1) dc(data.x,:)] + (devdmp(ones(N,1),:)-devdmp(data.x,:)));
```

STEP 6: BHHH

- Recall Newton-Raphson

$$\theta^{g+1} = \theta^{g} - \lambda \left(\Sigma_i H_i \left(\theta^{g} \right) \right)^{-1} \Sigma_i s_i \left(\theta^{g} \right)$$

- Berndt, Hall, Hall, and Hausman, (1974):
  Use *outer product of scores* as approx. to Hessian

$$\theta^{g+1} = \theta^{g} + \lambda \left(\Sigma_i s_i s_i' \right)^{-1} \Sigma_i s_i$$

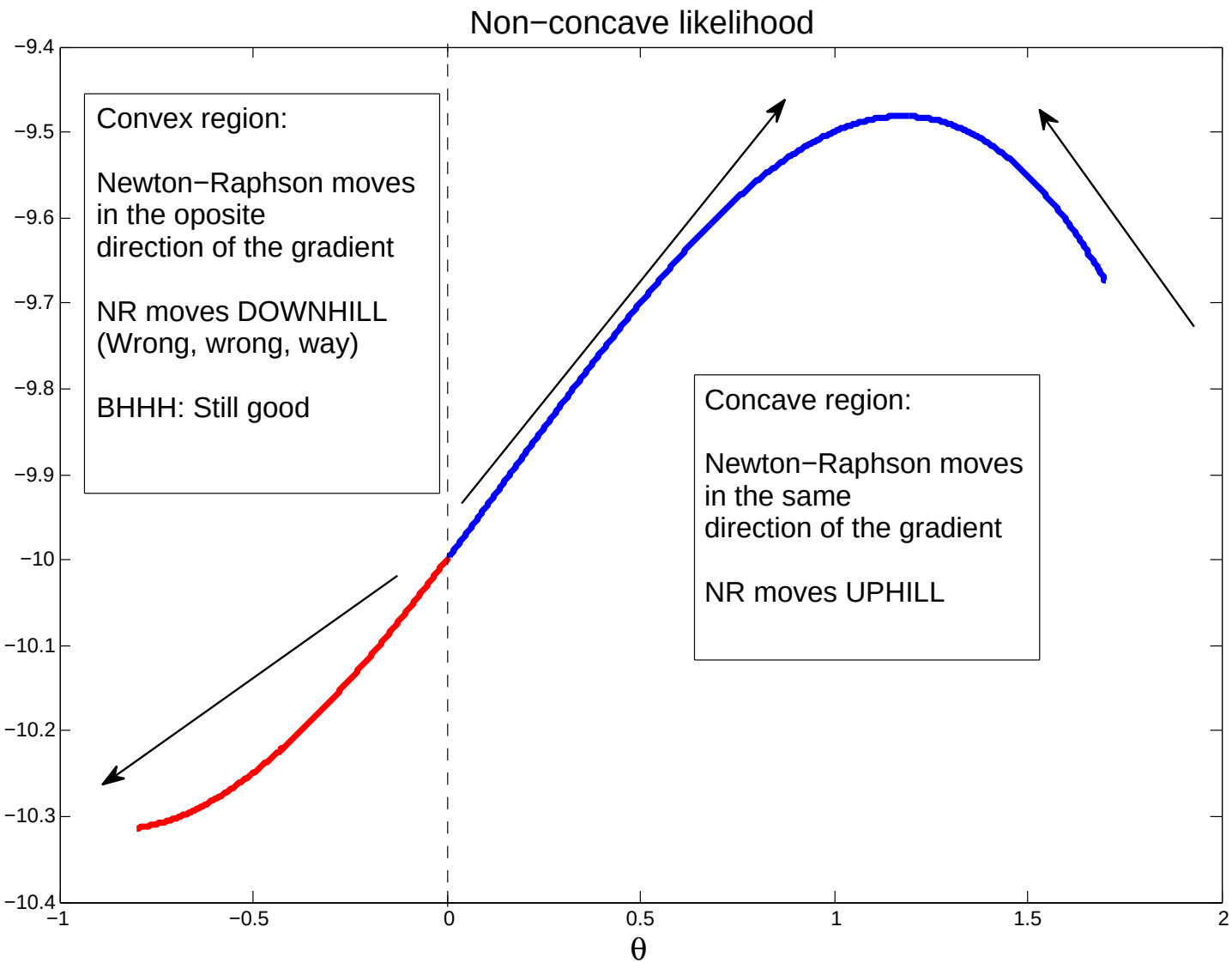
- Why is this valid? Information identity:

$$-E\left[H_i \left(\theta \right) \right] \quad = \quad E\left[s_i \left(\theta \right) s_i \left(\theta \right)' \right]$$

  (valid for MLE and CMLE)

STEP 6: BHHH

Some times linesearch may not help Newtons Method



STEP 6: BHHH

Advantages

- $\Sigma_i s_i s_i'$ is always positive definite
  I.e. it always moves uphill for λ small enough
- Does not rely on second derivatives

Disadvantages

- Only a good approximation
  - At the true parameters
  - for large N
  - for well specified models (in principle only valid for MLE)
- Only superlinear convergent - not quadratic

We can always use BHHH for first iterations and the switch to BFGS to update to get an even more accurate approximation to the hessian matrix as the iterations start to converge.

STEP 6: BHHH

Use BHHH!

Convergence!

β=0.9999

```
---------------------------------------------------------------------
***                    Convergence Achieved                       ***
---------------------------------------------------------------------
                            _
                            \`\
                            |= |
                           /-  ;.---.
                      _ __.'     (_____)
                       `         (______)
                      _'  ._ .'  (_____)
                       `         (____)
                      __`'------'`
Number of iterations: 9
grad*direc            0.00003
Log-likelihood    -276.74524

    Param.            Estimates          s.e.        t-stat
---------------------------------------------------------------------
    RC                  11.1525        0.9167       12.1655
    c                    2.3298        0.3288        7.0856
---------------------------------------------------------------------

Time to convergence is   0 min and 0.07 seconds
```

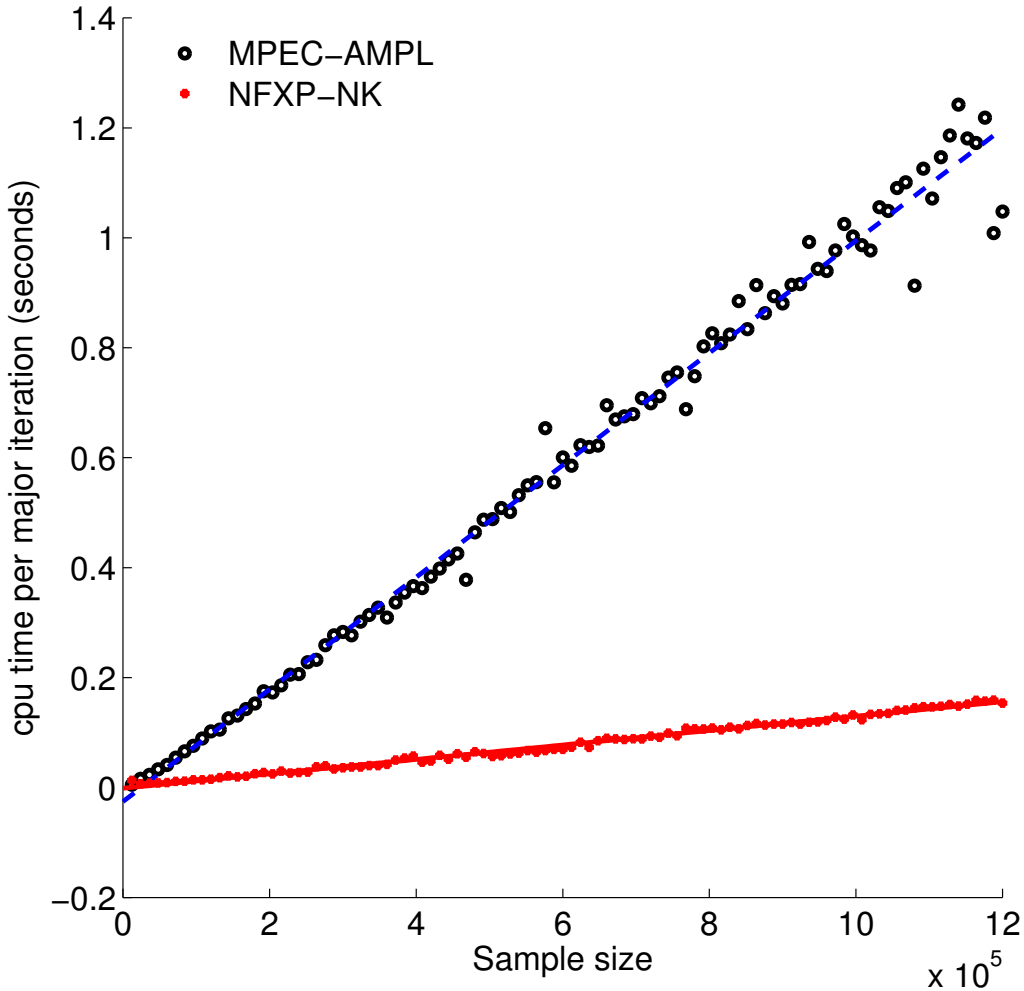
MPEC versus NFXP-NK: sample size 6,000

β	Converged (out of 1250)	CPU Time (in sec.)	# of Major Iter.	# of Func. Eval.	# of Bellm. Iter.	# of N-K Iter.
			MPEC-Matlab			
0.975	1247	1.677	60.9	69.9		
0.985	1249	1.648	62.9	70.1		
0.995	1249	1.783	67.4	74.0		
0.999	1249	1.849	72.2	78.4		
0.9995	1250	1.967	74.8	81.5		
0.9999	1248	2.117	79.7	87.5		
			MPEC-AMPL			
0.975	1246	0.054	9.3	12.1		
0.985	1217	0.078	16.1	44.1		
0.995	1206	0.080	17.4	49.3		
0.999	1248	0.055	9.9	12.6		
0.9995	1250	0.056	9.9	11.2		
0.9999	1249	0.060	11.1	13.1		
			NFXP-NK			
0.975	1250	0.068	11.4	13.9	155.7	51.3
0.985	1250	0.066	10.5	12.9	146.7	50.9
0.995	1250	0.069	9.9	12.6	145.5	55.1
0.999	1250	0.069	9.4	12.5	141.9	57.1
0.9995	1250	0.078	9.4	12.5	142.6	57.5
0.9999	1250	0.070	9.4	12.6	142.4	57.7

MPEC versus NFXP-NK: sample size 60,000

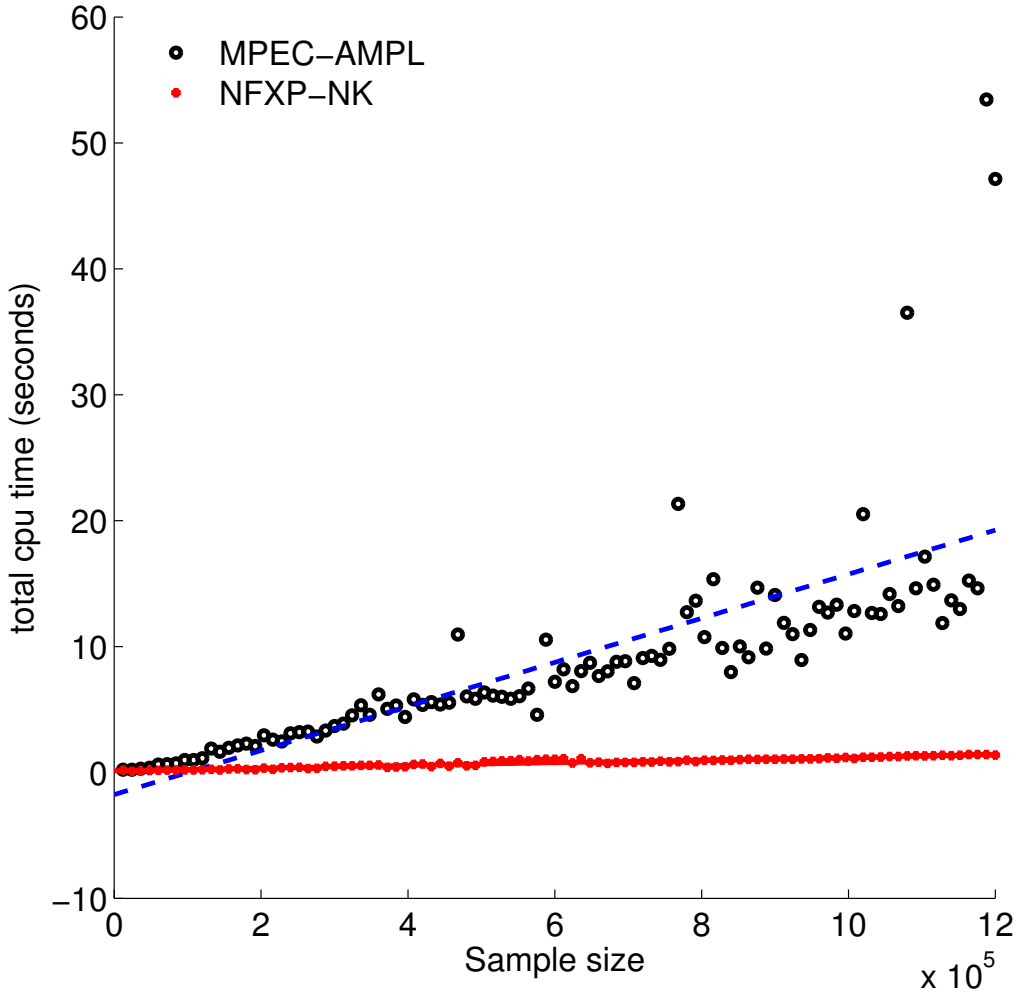
β	Converged (out of 1250)	CPU Time (in sec.)	# of Major Iter.	# of Func. Eval.	# of Bellm. Iter.	# of N-K Iter.
			MPEC-AMPL			
0.975	1247	0.53	9.2	11.7		
0.985	1226	0.76	13.9	32.6		
0.995	1219	0.74	14.2	30.7		
0.999	1249	0.56	9.5	11.1		
0.9995	1250	0.59	9.9	11.2		
0.9999	1250	0.63	11.0	12.7		
			NFXP-NK			
0.975	1250	0.15	8.2	11.3	113.7	43.7
0.985	1250	0.16	8.4	11.4	124.1	46.2
0.995	1250	0.16	9.4	12.1	133.6	52.7
0.999	1250	0.17	9.5	12.2	133.6	55.2
0.9995	1250	0.17	9.5	12.2	132.3	55.2
0.9999	1250	0.17	9.5	12.2	131.7	55.4

CPU time is linear sample size



$T_{NFXP} = 0.001 + 0.13x$ ($R^2 = 0.991$), $T_{MPEC} = -0.025 + 1.02x$ ($R^2 = 0.988$).

CPU time is linear sample size



$$T_{NFXP} = 0.129 + 1.07x \ (R^2 = 0.926) \ , \ T_{MPEC} = -1.760 + 17.51x \ (R^2 = 0.554).$$

Summary remarks

Su and Judd (Econometrica, 2012) used an inefficient version of NFXP

- that solely relies on the method of successive approximations to solve the fixed point problem.

Using the efficient version of NFXP proposed by Rust (1987) we find:

- MPEC and NFXP-NK are similar in performance when the sample size is relatively small.
- NFXP does not slow down as $\beta \to 1$

Desirable features of MPEC

- Ease of use by people who are not interested in devoting time to the special-purpose programming necessary to implement NFXP-NK.
- Can easily be implemented in the intuitive AMPL language.

Inference

- NFXP: Trivial to compute standard errors by inverting the Hessian from the unstrained likelihood (which is a by-product of NFXP).
- MPEC: Standard errors can be computed inverting the *bordered Hessian* Reich and Judd (2019): Develop simple and efficient approach to compute confidence intervals.

MPEC does not seem appropriate when estimating life cycle models