CONSTRAINED OPTIMIZATION APPROACHES TO ESTIMATION OF STRUCTURAL MODELS: COMMENT*

FEDOR ISKHAKOV
UNIVERSITY OF NEW SOUTH WALES

JOHN RUST†
GEORGETOWN UNIVERSITY

BERTEL SCHJERNING
UNIVERSITY OF COPENHAGEN

JUNE 2014

Abstract

We revisit the comparison of mathematical programming with equilibrium constraints (MPEC) and nested fixed point (NFXP) algorithms for estimating structural dynamic models by Judd and Su (JS, 2012). They used an inefficient version, NFXP-SA, that relies on the method of successive approximations to solve the fixed point problem. We re-do their comparison using the more efficient version of NFXP that Rust (1987) used, NFXP-NK, which combines successive approximations and Newton-Kantorovich iterations to solve the fixed point problem. MPEC and NFXP-NK are similar in performance when the fixed point dimension and sample size are relatively small and the discount factor is not too close to 1. However for higher dimensional problems, or problems with large sample sizes, NFXP-NK outperforms MPEC by orders of magnitude. MPEC fails to converge with high probability as the fixed point dimension increases, or as the discount factor approaches 1.

KEYWORDS: Structural estimation, dynamic discrete choice, NFXP, MPEC, successive approximations, Newton-Kantorovich algorithm.

*We are grateful to Harry J. Paarsch and Che-Lin Su for helpful comments. An early version of this paper was presented by Bertel Schjerning at the ZICE2014 workshop at the University of Zurich. We are grateful to participants of the ZICE2014 workshop for helpful feedback that lead to the current draft of this comment. This paper is part of the IRUC research project financed by the Danish Council for Strategic Research (DSF). Financial support is gratefully acknowledged.

†Corresponding author, email: jr1393@georgetown.edu.

1 Introduction

In their paper "Constrained optimization approaches to estimation of structural models" Judd and Su (2012), hereafter JS, proposed a constrained optimization method for structural estimation of infinite horizon dynamic discrete choice models termed mathematical programming with equilibrium constraints (MPEC).

The standard approach for solving this type of problems is the *nested fixed point* (NFXP) algorithm proposed by Rust (1987). NFXP exploits the fact that the likelihood function depends on a value function from dynamic programming. This value function is in turn the unique fixed point to a contraction mapping, and under weak assumptions will be a smooth implicit function of the structural parameters. This implies that the likelihood can be maximized using a standard *unconstrained* quasi-Newton optimization algorithm. However, each time the likelihood is evaluated, NFXP must call a fixed point algorithm to compute the value function implied by the current parameter values.

In contrast to NFXP, the MPEC method does not need a specialized algorithm to compute the fixed point: it treats the problem of maximizing the likelihood function with respect to the K structural parameters plus N levels of the value function, where N is the number of grid points in the discretized state space. These values must satisfy the contraction fixed point constraint. Thus MPEC also implicitly solves the fixed point problem while searching for structural parameter values that maximize the likelihood, but using a general-purpose *constrained* optimization algorithm[1].

JS used the model of optimal replacement of bus engines of Rust (1987) to conduct a Monte Carlo study to compare the performance of the MPEC and NFXP algorithms. They found that MPEC outperformed NFXP in terms of CPU time by up to three orders of magnitude. The point of this comment is to note that JS used an inefficient version of the NFXP algorithm. JS used the method of *successive approximations* (SA) to solve the inner fixed point problem, so we call the version of NFXP they used NFXP-SA. But it is well known that successive approximations is an inefficient algorithm for computing fixed points of contraction mappings, especially when the modulus of the contraction (which

[1]The specific implementation JS use is KNITRO (see Byrd, Nocedal and Waltz 2006).

equals the discount factor in the underlying dynamic programming problem) is close to 1. We show that the poor performance of NFXP relative to MPEC is an artifact of JS's use of an inefficient version of NFXP — NFXP-SA.

We redo the JS Monte Carlo study using the more efficient version of NFXP that Rust (1987) employed. This version, which we abbreviate as NFXP-NK, uses a combination of successive approximations and the Newton-Kantorovich (NK) method to solve the inner fixed point problem and is significantly faster and more reliable, especially when the discount factor is close to 1. We demonstrate that NFXP-NK and MPEC are roughly equivalent in their numerical performance when the sample size and fixed point dimension are relatively small, and when the discount factor is not too close to 1. However as the discount factor approaches 1, NFXP-NK is significantly more stable and converged in every Monte Carlo trial, whereas MPEC failed to converge with probability approaching 1 as the discount factor approached 1. Furthermore, once the sample size or fixed point dimension is sufficiently large, NFXP is significantly faster than MPEC.

Newton-Kantorovich is the preferred method for computing contraction fixed points because it has guaranteed quadratic convergence rate in a domain of attraction of the fixed point. However the Newton-Kantorovich approach is only locally convergent, so the original design of the NFXP algorithm (Rust 1987, 2000) starts with successive approximations to ensure global convergence, and switches to Newton-Kantorovich iterations only when it detects that it is in a domain of attraction of the fixed point where the rapid quadratic convergence takes hold. This hybrid algorithm or "polyalgorithm" ensures that a highly accurate solution can be found after only a small number of iterations. In particular, the combination of these two approaches makes the performance of the NFXP-NK algorithm independent of the value of the discount factor β whereas the CPU times of the NFXP-SA algorithm steadily increase as $\beta \to 1$ as shown in Table 2 of JS (p. 2228).

We also show that NFXP-NK outperforms MPEC in larger problems. The intuition for this finding is that MPEC optimizes the likelihood over $N+K$ variables subject to N constraints, whereas NFXP performs an unconstrained optimization over the K structural parameters only. Nemirovsky and Yudin (1983) have proven that in the absence of "special

structure" for the optimization problem (e.g. concavity of the objective function), the worst case computational complexity of finding a global optimum of a smooth function is an exponential function of the total number of variables being optimized. NFXP-NK exploits the special structure of the inner fixed point problem and finds an approximate solution with a worst case complexity that is polynomial in N, namely $O(N^3)$, and this is *independent of of the discount factor* β as long as β is less than 1. Therefore the worst case complexity of optimizing the likelihood using NFXP-NK is exponential in K only, whereas for MPEC it is exponential in $N+K$. Our limited Monte Carlo experiments show that the ratio of MPEC CPU times to NFXP-NK CPU times does indeed increase exponentially in N and that NFXP is three orders of magnitude faster than MPEC when $N = 2000$. This finding is consistent with the worst case complexity bound of Nemirovsky and Yudin.

The rest of this comment is structured as follows. In section 2 we redo the original comparison of MPEC and NFXP of JS using NFXP-NK instead of NFXP-SA. Then we present Tables 2 and 3 to show the effect of a) increased sample size, and b) larger fixed point dimension on the relative performance of the two methods. Though we do not claim that the conclusions from this limited numerical comparison of the two methods in the bus engine replacement example necessarily applies to other problems, in the last section we discuss our own view of the overall strengths and weaknesses of the two estimation methods.

2 Results

Table 1 presents a comparison of CPU times for MPEC and NFXP that reproduces Table 2 of JS. This is a "small" problem — both in terms of the number of observations (50 buses followed for 120 months each, or 6,000 observations in total), and the fixed point dimension is relatively small ($N = 175$). While we have developed our own code to implement NFXP-NK, we have been using the exact same setup and computer code for MPEC that was used to produce the results of the original paper by JS.[2] We have replicated JS's results for

[2] JS have kindly provided well documented MPEC code for this problem via Che-Lin Su's website. Our NFXP code is available on request.

Table 1: MPEC versus NFXP-NK: sample size 6,000, fixed point dimension 175

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	1248	2.619	60.9	69.8		
0.985	1250	2.626	62.9	70.0		
0.995	1248	2.657	67.5	74.2		
0.999	1249	2.866	72.3	78.5		
0.9995	1237	2.864	75.5	85.7		
0.9999	104	3.904	171.6	609.5		
			MPEC-AMPL			
0.975	1141	0.068	15.9	43.3		
0.985	1141	0.074	13.3	23.6		
0.995	849	0.075	12.9	20.1		
0.999	1195	0.059	11.2	13.2		
0.9995	27	0.050	5.9	8.0		
0.9999	0	—	—-	—		
			NFXP-NK			
0.975	1250	0.075	11.2	13.7	213.5	33.3
0.985	1250	0.074	10.2	12.7	204.0	34.4
0.995	1250	0.076	9.4	12.0	192.7	36.6
0.999	1250	0.075	9.0	11.6	184.5	37.2
0.9995	1250	0.074	9.0	11.6	184.5	37.2
0.9999	1250	0.074	8.9	11.6	184.1	37.4

Notes: This table is a replication of Table 2 in Judd and Su (2012) with NFXP-SA replaced by NFXP-NK. For each β, five starting points were used for each of the 250 simulated samples. CPU time, number of major iterations, number of function evaluations and number of inner loop iterations are the averages over all runs. Inner loop iterations include both value function iterations and Newton-Kantorovich iterations. All statistics in Table 1 (and similarly in tables 2 and 3) pertain to the subset of runs that converged.

NFXP-SA, but since this version of NFXP is inefficient and is completely dominated by NFXP-NK, we only report results for the latter in Table 1 below.[3] To conserve space we omit the table of structural estimates (Table 1 of JS), which shows that in all converging runs the structural parameters of the model were accurately estimated by both methods.

[3]We used the same specification of the bus engine replacement model, including the same true parameter values, same sample size and fixed point dimension, and the same number of Monte Carlo replications as in Table 2 of JS. We fixed the stopping tolerance for the inner fixed point in the NFXP-NK algorithm at 10^{-13} as in Rust (1987), which is $1/1000$ of the stopping tolerance 10^{-10} that JS used for NFXP-SA. Similarly to JS, we estimated transition probabilities for mileage travelled by buses jointly with other parameters (replacement cost and maintenance cost) by maximizing the full likelihood function, following the partial likelihood optimization as described in Rust (1987, 2000). We used the BHHH algorithm on the outer loop of NFXP-NK and frequency based starting values for the transition probability parameters.

It is clear from Table 1 and Figure 1 that when the efficient NFXP-NK algorithm is used, the CPU times are comparable to those of MPEC-AMPL, the AMPL implementation of MPEC method utilizing first and second order analytic derivatives of the objective function. It takes both methods about 0.07 to 0.08 of a second to structurally estimate the bus engine replacement model. The MPEC-MATLAB implementation, which uses first order analytic derivatives only (and *ktrlink* function enabling KNITRO in Matlab), is about two orders of magnitude slower.

As the third panel of Table 1 shows, NFXP-NK uses far fewer successive approximation steps compared to the NFXP-SA results in Table 2 of JS, and this is the main reason why NFXP-NK is so much faster. For the highest discount factor $\beta = 0.995$ in Table 2 of JS, an average of nearly 750,000 successive approximation iterations were required, compared to just 184 for NFXP-NK. With an average of 37 Newton-Kantorovich iterations per estimation, the average number of both types of inner loop iterations for NFXP-NK is remarkably insensitive to the discount factor. It takes NFXP-NK an average of only 20 successive approximation steps and 4 Newton-Kantorovich iterations per function evaluation to compute a highly accurate fixed point (to a tolerance of 10^{-13}) when $\beta \geq 0.9995$.

Note the heavy right tail of the distribution of CPU times for MPEC in the left panel of Figure 1. This indicates that MPEC experiences a non-negligible set of "problematic" cases that take up to 2 to 3 times longer for it to find the optimum. When $\beta = 0.9995$ MPEC-AMPL converged in only 27 of the 1250 Monte Carlo trials.[4]

For all discount factors we tried, NFXP-NK converged in *every* case, and within a small range of CPU times, ranging from a minimum time of just under 0.05 seconds to a maximum time of 0.11 seconds. Overall, besides the similarity in run times of MPEC-AMPL and NFXP-NK, the most striking aspect of Table 1 is the high rate of convergence failures of both versions of MPEC. Our findings are thus completely contrary to the main conclusion of JS (2012) that "MPEC is significantly faster than NFXP" (p. 2228). We believe that their conclusion is an artifact of their use of the inefficient NFXP-SA algorithm, and we have shown that MPEC experiences a rapid increase in convergence failures when

[4]Contrary to all other runs, in this case the structural parameter estimates were also notably downward biased.

Table 2: MPEC versus NFXP-NK: sample size 60,000, fixed point dimension 175

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	1188	0.70	14.3	34.6		
0.985	1228	0.71	13.4	25.6		
0.995	794	0.85	12.5	19.1		
0.999	1236	0.61	11.4	13.3		
0.9995	0	—	—	—		
0.9999	0	—	—-	—		
NFXP-NK						
0.975	1250	0.16	8.1	11.3	150.3	29.0
0.985	1250	0.16	8.6	11.5	164.4	31.7
0.995	1250	0.16	9.1	11.9	175.8	36.2
0.999	1250	0.17	9.1	11.8	176.8	37.7
0.9995	1250	0.17	9.1	11.8	176.2	37.8
0.9999	1250	0.17	9.1	11.8	176.3	38.1

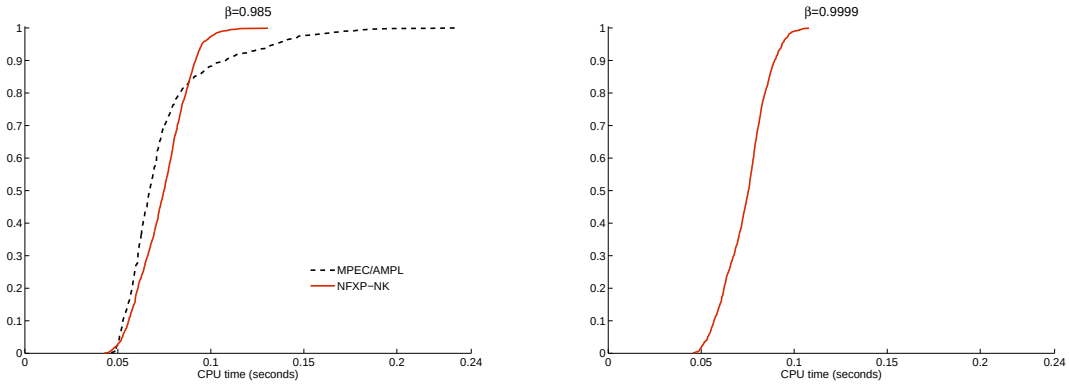
Notes: The experimental design in this Table is identical to that of Table 1 except that the sample size is 60,000.

β exceeds the highest value ($\beta = 0.995$) considered by JS.

Table 2 reports another Monte Carlo experiment we conducted to study the effect of a larger sample size on the relative run times for MPEC-AMPL and NFXP-NK (we do not consider MPEC-MATLAB further due to its run times being dominated by MPEC-AMPL). We kept the fixed point dimension small ($N = 175$) but increased the sample size to $60,000$ (500 buses followed for 120 months). Table 2 shows that NFXP-NK is uniformly 3.8 to 5.3 times faster than MPEC-AMPL for the various discount factors we considered. We see that as in Table 1, MPEC-AMPL "breaks down" for sufficiently high discount factors, and we were unable to get it converge in *any* of the 1250 Monte Carlo runs when $\beta \geq 0.9995$. The reason NFXP-NK dominates MPEC-AMPL in larger samples is related to the fact that unlike NFXP the likelihood as the function of observed data has to be calculated relatively more often during constrained maximization because the calculation of the fixed point is not separated from the evaluation of likelihood function.

Table 3 studies the effect of increasing the fixed point dimension N for a fixed sample size (6,000 observations) and for a fixed (relatively low) discount factor of $\beta = 0.975$, so

Figure 1: Distributions of CPU times for MPEC-AMPL and NFXP-NK



Notes: This figure plots the distribution of CPU times for 1250 runs of MPEC-AMPL and NFXP-NK. Distribution is computed based on converging runs only. See further description in Table 1.

that MPEC-AMPL's failures to converge are likely to be a result of the dimensionality of the fixed point rather than due to the discount factor being too close to 1. Table 3 reports the average CPU times to estimate the two key structural parameters (replacement cost and maintenance cost parameter) of the *partial likelihood function.* This is the second step of the consistent two step estimator described in Rust (1987) where the first step involves estimating the parameters of distribution of miles travelled per month on the bus using simple sample frequency (multinomial) estimators. We used 10 independent Monte Carlo samples to produce the results in Table 3.

In Table 3 we see a striking number of convergence failures for MPEC-AMPL as the fixed point dimension N increases. In some cases (i.e. where $N = 300, 400$ and 700), MPEC-AMPL failed to converge in *all* 10 Monte Carlo samples. NFXP-NK always converged, similar to what we observed in tables 1 and 2. We also see that for the subset of runs where MPEC-AMPL did converge, the CPU time it took to estimate the model increased dramatically with N, whereas the CPU time for NFXP-NK increased at a much slower rate. Recalling the discussion in the introduction, when we fix K and increase N, we expect that the CPU time for NFXP-NK to increase at a polynomial rate, with complexity bounded above by $O(N^3)$ rate (the upper bound on the time taken to solve a dense $N \times N$ system of linear equations, which is the most time consuming part of the Newton-Kantorovich

Table 3: Effect of fixed point dimension on MPEC and NFXP-NK: sample size 6,000, $\beta = 0.975$

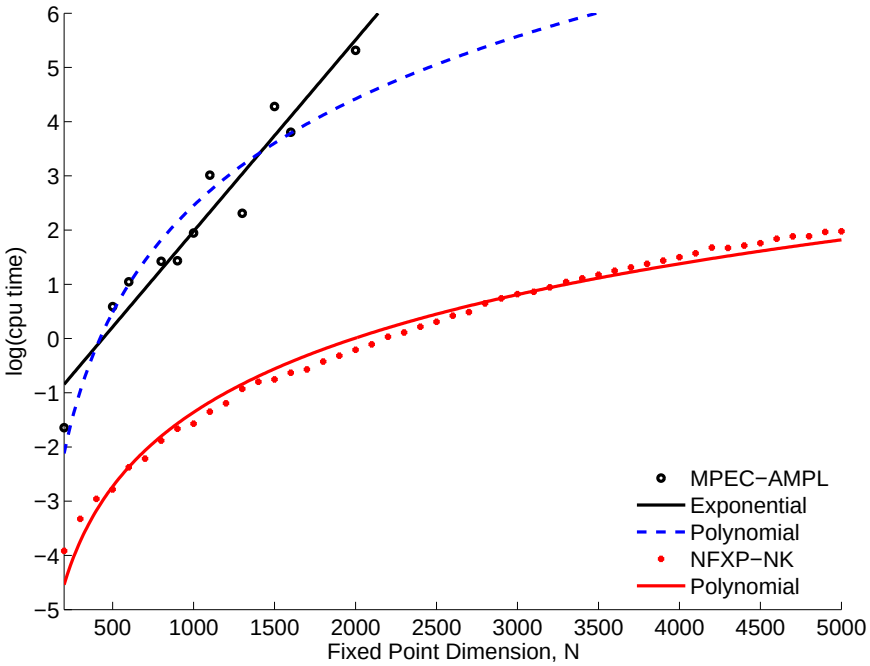
		MPEC-AMPL		NFXP-NK	
		Converged	CPU Time	Converged	CPU Time
N	M	(out of 10)	(in sec.)	(out of 10)	in sec.
200	7	4	0.19	10	0.03
300	10	0	—	10	0.03
400	13	0	—	10	0.05
500	16	6	1.80	10	0.06
600	19	4	2.85	10	0.09
700	22	0	—	10	0.11
800	25	1	4.15	10	0.15
900	28	3	4.20	10	0.18
1000	31	1	7.00	10	0.20
1500	46	1	72.14	10	0.44
2000	61	1	203.49	10	0.77

Notes: Monthly mileage is assumed to be distributed according to a censored normal, i.e. $\max(x_t, 0)$ where $x_t \sim \mathcal{N}(\mu, \sigma^2)$ and $\mu = 3.2889$ and $\sigma^2 = 1.4686$ is set to the mean monthly mileage for bus types 1,2,3 and 4 in Rust's data set. x_t is then discretized in N equally spaced. Parameters are identical to those from Table 1, although the slope parameter of the cost function is scaled with $N/175$ to preserve the same marginal cost of one additional mile. Transition probabilities smaller than 10^{-12} are rounded off at zero. M denotes the number of non-zero transition probability parameters, which defines the sparsity of the transition density matrix of mileage. We estimated the two structural parameters using partial MLE for 10 different datasets. We started both algorithms at the true value of the structural parameters and fixed the transition probability parameters using a first step frequency estimator.

iteration). However the worst case complexity theory of Nemirovsky and Yudin (1983) suggests that the CPU time for MPEC-AMPL should increase exponentially as a function of the total number of parameters being estimated, $N+2$.

Figure 2 plots the CPU times for NFXP-NK and MPEC-AMPL on a semi-log scale. If the NFXP-NK is a polynomial-time algorithm then its CPU times should conform to a regression line of the form $\log(T) = a + b \times \log(N) + \varepsilon$ where T denotes the CPU time to estimate the model. In particular, the regression estimate of the coefficient b should satisfy $b \leq 3$, corresponding to the $O(N^3)$ worst-case complexity bound discussed above. In fact the estimate is $b = 1.98$, substantially lower than 3. This is due to the use of sparse matrix algorithms that exploit the special structure of the transition probability matrix which determines the linear system of equations that must be solved to carry out the Newton-Kantorovich iterations. Due to the fact that there is an upper bound on the number of miles

Figure 2: CPU times for MPEC-AMPL and NFXP-NK as a function of fixed point dimension N



Notes: Average CPU time is computed based on converging runs only. Plotted polynomial and exponential regressions are $\log(T_{NFXP}) = -15.0 + 1.98 * \log(N)$ ($R^2 = 0.9866$), $\log(T_{AMPL}) = -1.55 + 0.003530 * N$ ($R^2 = 0.9336$) and $\log(T_{AMPL}) = -17.2 + 2.84 * \log(N)$ ($R^2 = 0.9115$). For experimental design, see note in Table 3.

that a bus can travel in any given month, the transition probability matrix for the odometer state variable in the bus engine replacement problem is *banded* and this special banded structure is exploited by the NFXP-NK algorithm to result in further speedups compared to "dense" linear algebra algorithms, e.g. Crout decomposition.

If the worst case CPU time of the MPEC-AMPL algorithm is exponential, as suggested by the theory of Nemirovsky and Yudin (1983), then roughly speaking, CPU times should conform to a regression line of the form $\log(T) = a + b \times N + \varepsilon$. There will be a curse of dimensionality (i.e. exponential increase in CPU times as N increases) if $b > 0$.

In Figure 2 we can see that NFXP-NK clearly dominates MPEC-AMPL for all fixed point dimensions considered, and the gap in their relative run times also increases with N. However, with the limited CPU time data for MPEC-AMPL algorithm due to convergence

failures, it is hard to make decisive conclusion about its computational complexity. We fitted both semi-log and log-log specifications on the available MPEC-AMPL time data, with estimation results given in the note of Figure 2. For the polynomial-time specification, we estimated $b = 2.84$ which lies below the theoretical upper bound of $b = 3$ for an $O(N^3)$ algorithm, though it is significantly higher than the estimated coefficient $b = 1.98$ for NFXP.[5] With $R^2 = 0.93$ the exponential specification provides a better fit to the CPU data for MPEC-AMPL than the polynomial specification ($R^2 = 0.91$). We conclude that our limited Monte Carlo results are consistent with the theoretical predictions of Nemirovksy and Yudin (1983) that MPEC-AMPL is subject to a "curse of dimensionality" as a function of the fixed point dimension N whereas NFXP-NK is a polynomial-time algorithm in N in the worst case.

In contrast, NFXP exploits the fact that the objective function is a likelihood function that is a smooth function of value functions, which are in turn smooth functions of the underlying structural parameters. It separates the task of computing the value function as the fixed point to a contraction mapping and the problem of maximizing the likelihood function into separate subproblems, and this enables it to maximize the likelihood function using traditional unconstrained quasi-Newton/gradient search algorithms such as the Berndt-Hall-Hall-Hausman (BHHH) algorithm over a relatively small number structural parameters K.

Another major concern about MPEC is an alarming number of failures to converge. We conjecture that this is related to the much higher dimensionality of the optimization problem and MPEC's failure to fully exploit the structure of the dynamic programming problem. In contrast, NFXP-NK recognizes the fact that the objective function is a sum of individual likelihoods each of which is computed from the set of value functions that are smooth in the structural parameters. This enables maximization of the likelihood function using the globally convergent Berndt-Hall-Hall-Hausman (BHHH) algorithm over a relatively small number structural parameters K.

[5]Note that MPEC-AMPL *does* also exploit the sparsity of the banded transition probability matrix structure of the bus replacement problem, just as NFXP-NK does. Thus, the difference in CPU times in Figure 2 cannot be ascribed to NFXP-NK having an "unfair" advantage of exploiting the sparsity structure. Thus, the significantly higher b coefficient for the MPEC-AMPL data is likely due to other reasons other than failure to exploit sparsity.

The BHHH algorithm of Berndt et al. (1974) exploits the information identity to approximate the Hessian of the likelihood with the negative of the outer product of the scores. Since the Hessian approximation is always negative semi-definite, it ensures that BHHH will always move in the direction of the gradient (i.e. towards the maximum) even in convex areas of the likelihood function. Hence, beyond the advantage of avoiding computation of second order derivatives, BHHH has the major advantage of always moving uphill for small enough step size, and thus being globally convergent. In sum, the robustness of NFXP and its ability to handle larger problems comes from explicitly exploiting the structure of the problem, i.e. that the objective function is a sample sum over individual likelihoods and that the nested fixed point problem is a contraction mapping.

3 Conclusion

We have shown that the main conclusion of JS (2012), namely that "Monte Carlo results confirmed that MPEC is significantly faster than NFXP, particularly when the discount factor in the dynamic-programming model is close to 1." (p. 2228), appears to be an artifact of their use of an inefficient version of NFXP which we called NFXP-SA. We reran the comparison using the version of NFXP that Rust (1987) originally proposed, NFXP-NK, and found that MPEC-AMPL and NFXP-NK are about equally fast on the same test problem that JS considered. We also compared MPEC and NFXP for a) a larger range of discount factors β, b) different sample sizes, and c) different sizes for the fixed point dimension N. We have shown that NFXP-NK is significantly more robust and reliable than MPEC-AMPL, particularly when the discount factor of the dynamic programming problem, β, approaches 1. Further, we found that MPEC-AMPL may be significantly slower than NFXP for problems with large samples.

Yet, we believe that MPEC has many desirable features, the most important of which is ease of use by people who are not interested in devoting time to the special-purpose programming necessary to implement NFXP-NK. As long as the problem to be estimated is not too "big" (both in terms of the number of observations and the dimension N of the fixed point), and the discount factor is not too close to 1, our results indicate that MPEC is very

fast and competitive with NFXP-NK, and particularly in conjunction with the easy, intuitive AMPL language, it could save many users substantial programming time and enable them to structurally estimate many models of interest.

However for "heavy duty" applications where either there are many observations, or the dimension of the fixed point is sufficiently large, or where the discount factor is likely to be close to 1 (such as in problems with short time periods), our results indicate that the MPEC method may not be reliable and efficient in such cases, at least until better implementations are available. In these cases there may be no alternative but to incur the fixed costs to program the special-purpose fixed point algorithm that can exploit the special structure of the dynamic programming problem at hand.

References

[1] Byrd, R. H., J. Nocedal, and R. A. Waltz (2006): KNITRO: An Integrated Package for Non- linear Optimization, in *Large-Scale Nonlinear Optimization* ed. by G. di Pillo and M. Roma. New York: Springer, 3559.

[2] Judd and Su (2012): "Constrained optimization approaches to estimation of structural models" *Econometrica* 80-5 2213-2230.

[3] A. S. Nemirovsky and D. B. Yudin (1983) *Problem complexity and method efficiency in optimization* Wiley-Interscience. Translated by: E. R. Dawson.

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

[5] Rust (2000): "Nested Fixed Point Algorithm Documentation Manual: Version 6" `http://gemini.econ.umd.edu/jrust/nfxp.html`