KATHOLIEKE UNIVERSITEIT LEUVEN

Vives
Vlaams Instituut voor Economie en Samenleving

Naamsestraat 61 - bus 3550
B-3000 Leuven - BELGIUM
Tel : 32-16-326661
vives@econ.kuleuven.be

VIVES 2012

DISCUSSION PAPER 35

Enhancing the Convergence Properties of the BLP (1995) Contraction Mapping

Jo Reynaerts
Ravi Varadhan
John C. Nash

Enhancing the Convergence Properties of the BLP (1995) Contraction Mapping

Jo Reynaerts*,[a] Ravi Varadhan[b] John C. Nash[c]

[a]*VIVES, LICOS and STORE, KU Leuven*
[b]*The Center on Aging and Health, Johns Hopkins University*
[c]*Telfer School of Management, University of Ottawa*

This version: 15 August 2012

Abstract. In the wake of Dubé, Fox and Su (2012), this paper (*i*) analyzes the consequences for the BLP Contraction Mapping (Berry, Levinsohn and Pakes, 1995) of setting the inner-loop convergence tolerance at the required level of $\epsilon_{\text{in}} = 10^{-14}$ to avoid propagation of approximation error from inner to outer loop, and (*ii*) proposes acceleration as a viable alternative within the confines of the Nested-Fixed Point paradigm (Rust, 1987) to enhance the convergence properties of the inner-loop BLP Contraction Mapping.

Drawing upon the equivalence between nonlinear rootfinding and fixed-point iteration, we introduce and compare two alternative methods specifically designed for handling large-scale nonlinear problems, in particular the derivative-free spectral algorithm for nonlinear equations (La Cruz *et al.*, 2006, DF-SANE), and the squared polynomial extrapolation method for fixed-point acceleration (Varadhan and Roland, 2008, SQUAREM). Running a Monte Carlo study with specific scenarios and with Newton-Raphson as a benchmark, we study the characteristics of these algorithms in terms of (*i*) speed, (*ii*) robustness, and (*iii*) quality of approximation.

Under the worst of circumstances we find that (*i*) SQUAREM is faster (up to more than five times as fast) and more robust (up to 14 percentage points better success rate) than BLP while attaining a comparable quality of approximation, and

*E-mail: Jo.Reynaerts@econ.kuleuven.be, rvaradhan@jhmi.edu and nashjc@uottawa.ca. We thank Klaus Desmet, Jan De Loecker, Laura Grigolon, Ken Judd, Joep Konings, Sanjog Misra, Benjamin Skrainka, Patrick Van Cayseele, Eddy Van de Voorde, Frank Verboven and Thijs Vandemoortele for comments and suggestions. This paper was presented at the University of Chicago/Argonne National Laboratory *Initiative for Computational Economics* (ICE) 2011, 18–29 July, Booth School of Business, University of Chicago; at the 1st LICOS Centre of Excellence "Governments and Markets" annual conference, 13 September 2011, Jodoigne, BE; at *Computational & Financial Econometrics* (CFE'10), 4th CSDA International Conference, 10–12 December, Senate House, University of London (London School of Economics/Queen Mary/Birckbeck), London, UK, and at *The R User Conference 2010* (useR! 2010), 20–23 July, The National Institute for Standards and Technology (NIST), Gaithersburg, Maryland. Financial support from the KU Leuven Research Fund (PF/10/003) is gratefully acknowledged.

(*ii*) also outperforms DF-SANE in nearly all scenarios. Eliminating averaging bias against more robust algorithms, a performance profile subsequently shows that (*iii*) SQUAREM is both faster, delivers the best quality approximation, and is more robust than BLP, DF-SANE and Newton-Raphson.

Key words: contraction mapping, fixed-point iteration, acceleration algorithms, (quasi-) Newton methods

JEL codes: C25, C63, C81, C87

1 Introduction

A number of real-world estimation problems require the solution of a large system of nonlinear equations. For some of these problems, fixed-point iteration schemes have been proposed, such as the Berry *et al.* (1995, BLP) Contraction Mapping used in estimating random coefficients logit models of demand for differentiated products. The BLP Contraction Mapping is a fixed-point problem in J products by T markets, where one must invert a demand system to uncover a vector $\boldsymbol{\delta} \in \mathbb{R}^{JT}$ that (*i*) equates predicted market shares with the actual observed market shares, and (*ii*) reflects the "mean utility" for each product $j = 1, \ldots, J$ in each market $t = 1, \ldots, T$.

Generally perceived as a nested fixed-point (NFXP) algorithm, the BLP method specifies a double-loop procedure to estimate the parameter vector $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\sigma})'$ that determines the distribution (means and standard deviations) of the random coefficients in the regression specification.[1] Whereas the "linear" utility parameter vector $\boldsymbol{\beta}$ can be estimated in a rather straightforward manner, the estimation of the "nonlinear" utility parameter vector $\boldsymbol{\sigma}$ relies on minimizing a Generalized Method of Moments (Hansen, 1982, GMM) objective function (outer loop) and an iterative scheme commonly referred to as the BLP Contraction Mapping (inner loop) which, for each value of $\boldsymbol{\sigma}$ generated by the outer loop, computes a new value for the "mean utility" vector $\boldsymbol{\delta}$ by inverting the nonlinear demand system

$$\hat{s}\left(\boldsymbol{\delta}, \boldsymbol{\sigma}\right) = S, \tag{1}$$

where $S \in \mathbb{R}_{+}^{JT}$ is the vector of observed market shares for products $j = 1, \ldots, J$ in markets $t = 1, \ldots, T$, and $\hat{s}(\cdot) \in \mathbb{R}_{+}^{JT}$ is the vector of theoretically predicted market shares. Stated otherwise, $\boldsymbol{\delta}$ is the vector that equates predicted market shares with observed market shares in equation (1). Summarized, the BLP Contraction Mapping computes $\boldsymbol{\delta}(\boldsymbol{\sigma}) = \hat{s}^{-1}(S; \boldsymbol{\sigma})$ using the following iterative scheme:

1. For each value of $\boldsymbol{\sigma}$, compute the next value for $\boldsymbol{\delta}$ as

$$\boldsymbol{\delta}^{h+1} = \boldsymbol{\delta}^{h} + \log(S) - \log\left(\hat{s}(\boldsymbol{\delta}^{h}, \boldsymbol{\sigma})\right), \tag{BLP}$$

   for $h = 0, 1, 2, \ldots$

2. Stop if $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\| \leq \epsilon_{\text{in}}$, where $\|\cdot\|$ can be either the L_2 or L_∞ norm and ϵ_{in} is the inner-loop tolerance level.

Berry *et al.* (1995) prove that the function underlying the right-hand side of equation (BLP) has a unique fixed point, and additionally, is a contraction with modulus less than one, implying that the fixed-point iteration is guaranteed to converge from any starting point. While this global convergence property is certainly appealing, the BLP Contraction Mapping converges only linearly and can prove to be a time-consuming procedure, especially in applications with a large number of observations (the size of $\boldsymbol{\delta}$ typically

[1] For the remainder of this paper, the terms *BLP algorithm*, *BLP method* or *BLP estimation* refer to the entire estimation procedure that includes both inner and outer loops, while *BLP Contraction Mapping* explicitly refers to the inner-loop computation, see Train (2009, Ch. 13) for a textbook treatment of the BLP algorithm.

exceeding 1000 observations and more). Figure 1 shows the average CPU time, average number of iterations, and success rate of the BLP Contraction Mapping as a function of sample size. For a fixed inner-loop tolerance level $\epsilon_{\text{in}} = 10^{-4}$, as the sample size increases from 1250 to 2500 and 5000 observations, the CPU time increases from approximately 3.51 to 5.69 and 11.00 seconds respectively when the outer loop is generating values for $\boldsymbol{\sigma}$ that are close to the solution of the outer-loop GMM minimization problem, represented by the left column of Figure 1. Moving left to right (from column 1 to 2 and 3), these values for $\boldsymbol{\sigma}$ are farther removed from the solution and negatively affect the performance of the BLP Contraction Mapping. For example, in the case where the $\boldsymbol{\sigma}$ values are very far from the truth (third column), the average CPU time needed to converge increases from 13.30, to 18.06 and 30.99 seconds respectively.

In order to speed up convergence, a common procedure among researchers is to relax the inner-loop convergence tolerance ϵ_{in} in regions where the minimization of the GMM objective function is far from the solution (with tolerance levels as low as 10^{-2}), and to gradually impose stricter criteria as the minimization procedure gets closer to the truth. However, concern has risen recently with respect to the numerical performance of the BLP algorithm, in particular (*i*) the trade-off between inner-loop numerical error and speed, and (*ii*) the propagation of the approximation error from the inner to the outer loop, which affects parameter estimation, see Knittel and Metaxoglou (2012) and especially Dubé, Fox and Su (2012). In order for the BLP estimator to be numerically correct (and the BLP estimation to converge to a global minimum), Dubé *et al.* (2012) show that the inner-loop tolerance must be set at a level of 10^{-14} with a corresponding outer-loop tolerance of 10^{-6}.

In this respect Figure 1 also displays the effects of tightening the tolerance level on the convergence properties of the BLP Contraction Mapping. Increasing the tolerance level naturally increases the time needed for BLP to converge; more important to note is the fact that the *success rate* (defined as the proportion of Monte Carlo replications where the BLP Contraction Mapping successfully converged) is unaffected only for $\boldsymbol{\sigma}$ values that are close to the solution (left column).[2] In other (more realistic) cases, the success rate drops significantly and even reaches a level as low as 8% for a tolerance level of 10^{-10} and a sample size of 5000. Even in this case we are still far away from the tolerance level of 10^{-14} advocated by Dubé *et al.* (2012) to assure precise estimation of the demand parameters. On the other hand, Kim and Park (2010) show that the inner-loop approximation error affects the outer-loop GMM estimation with the same order of magnitude and therefore does not propagate into the GMM estimation. In view of these conflicting results, the question arises therefore how well BLP fares under more demanding circumstances, and how it compares with other algorithms that can solve the fixed-point problem.

We therefore investigate the convergence properties of the BLP Contraction Mapping in terms of (*i*) speed (the time needed to converge), (*ii*) stability (the ability to converge

[2] As such, the success rate is not to be confused with the concept of "convergence rate" from numerical analysis where it refers to the speed at which a convergent sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ approaches its limit $L \equiv \lim_{k\to\infty} \mathbf{x}_k$, or in this case (and less formally), the speed with which an algorithm converges to the solution $\boldsymbol{\delta}^*$.

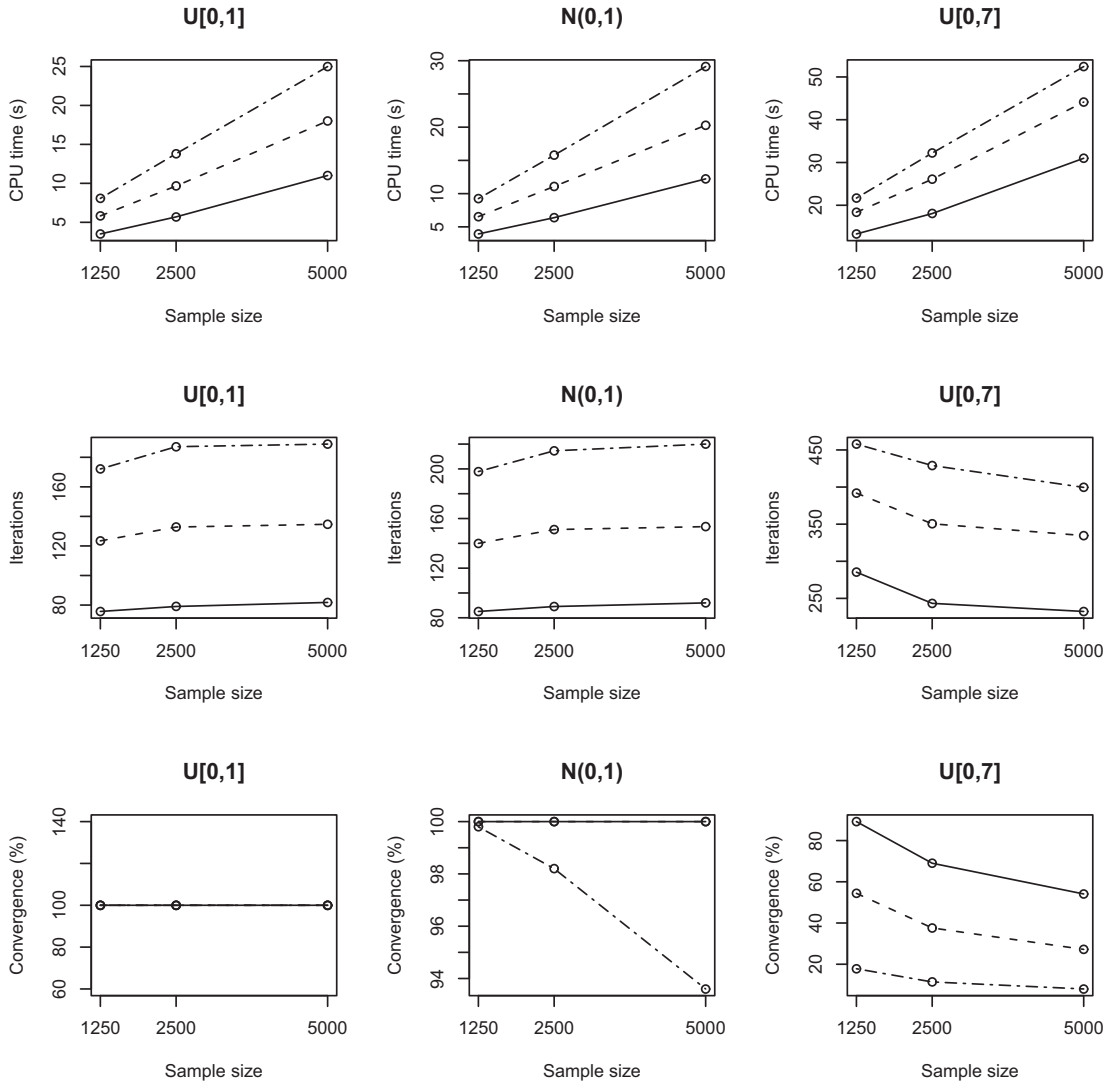

Figure 1: Convergence properties for the BLP Contraction Mapping. Solid $(-)$, dashed $(--)$ and dash-dotted $(-\cdot-)$ lines respectively represent results for inner-loop convergence tolerances $\epsilon_{\text{in}} \in \{10^{-4}, 10^{-7}, 10^{-10}\}$. From top to bottom, rows display the average CPU time in seconds, the average number of iterations, and the success rate as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from left to right respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0, 1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0, 1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value. The success rate is the percentage of times BLP succeeded in computing a numerical approximation to $\boldsymbol{\delta}$ within the maximum number of iterations allowed, set at 1500.

within a specified number of iterations), and (*iii*) quality (the approximation error) given a specified inner-loop convergence tolerance level and sample size, and compare these with results for alternative algorithms. To this end, we recast the fixed-point formulation of the BLP Contraction Mapping as a rootfinding problem, and propose two alternative algorithms, in particular the classic Newton-Raphson algorithm (used as a benchmark), and the derivative-free spectral algorithm for nonlinear equations (DF-SANE). A third algorithm, the squared polynomial extrapolation method for accelerating the Expectation-Maximization algorithm (SQUAREM), operates directly on the fixed-point formulation of the BLP Contraction Mapping. The latter two algorithms are specifically designed for solving large-scale nonlinear problems, an area of research that has witnessed a surge of interest recently from both mathematicians and statisticians. We find that within the confines of the NFXP approach, the SQUAREM algorithm is faster (up to more than five times faster) and more robust (up to 14 percentage points better success rate) than the BLP Contraction Mapping, and additionally, outperforms DF-SANE.

The natural question that arises is why one should not just forego the NFXP paradigm for the MPEC approach, as advocated by Dubé *et al.* (2012),[3] where the GMM objective function is minimized subject to the constraint that predicted market shares equal observed market shares. In response we offer two plausible arguments as to why our approach (NFXP acceleration) might be the better choice:

- From a theoretical point of view, the mathematical properties of the BLP Contraction Mapping are well-known and established, see Berry *et al.* (1995, App. I). They imply that the solution that comes out of the inner-loop computation is indeed a solution to equation (1). While apparently sidestepping the convergence difficulties the BLP Contraction Mapping experiences when operating under tighter tolerance levels as shown above, it is unclear whether the MPEC solution does the same, as
  - it remains to be verified whether the Karush-Kuhn-Tucker conditions are satisfied with MPEC, as large-scale constrained nonlinear optimization is a difficult undertaking
  - it is uncertain whether the implied $\boldsymbol{\delta}^*_{\text{MPEC}}$ from the MPEC approach satisfies the conditions of the BLP Contraction Mapping; more specifically, does the implied Jacobian matrix of the BLP Contraction Mapping at the MPEC solution for $\boldsymbol{\sigma}$ satisfy the sufficiency conditions (1–3) stated in the Berry *et al.* (1995) Contraction Mapping Theorem?
- From a modeling point of view, as discrete choice models become more complex with the introduction of *dynamics*, see e.g. Gowrisankaran and Rysman (2009) and Schiraldi (2011) where solving the dynamic programming problem by value function iteration introduces an additional inner loop, the need for techniques that reduce the computational burden involved will become increasingly important. A similar argument is valid when considering increasing the number of draws of fictitious consumers in order to reduce sampling error and sampling noise.

[3] MPEC is an acronym for Mathematical Programming with Equilibrium Constraints, see Su and Judd (2008) for an introduction to its use in economics.

While this paper devotes attention exclusively to the BLP Contraction Mapping (BLP), it interesting to note that the numerical properties of other parts of the BLP algorithm, such as the Monte Carlo-based smooth simulator (29) used to numerically compute the market share integral in (28), also have been scrutinized recently. In this respect, Skrainka and Judd (2011) show that polynomial-based quadrature rules for multidimensional numerical integration outperform number-theoretic quadrature (Monte Carlo) rules.[4]

The remainder of this paper is structured as follows: in Section 2 we reformulate the BLP Contraction Mapping as a nonlinear rootfinding problem and introduce alternative algorithms for computing a solution to equation (BLP). Section 3 documents the Monte Carlo study and the metrics used to measure convergence properties. Results are discussed in Section 4, while Section 5 contains an extensive numerical analysis of the convergence properties of the algorithms involved. Section 6 concludes and presents items for future research.

2 The BLP Contraction Mapping and its Relation to other Nonlinear Methods

The BLP Contraction Mapping (BLP) is in essence a fixed-point problem

$$g : \mathbb{R}^{J\times T} \to \mathbb{R}^{J\times T} : \boldsymbol{\delta} \mapsto \boldsymbol{\delta} = g(\boldsymbol{\delta}). \tag{2}$$

Mathematically, any fixed-point problem $\mathbf{x} = g(\mathbf{x})$ with $g : \mathbb{R}^n \to \mathbb{R}^n$ can be reformulated as a rootfinding problem $f(\mathbf{x}) = \mathbf{0}$ with $f : \mathbb{R}^n \to \mathbb{R}^n$, by defining $f(\mathbf{x}) = \mathbf{x} - g(\mathbf{x})$. The vector of mean utilities $\boldsymbol{\delta}$ computed using the BLP Contraction Mapping (BLP) can therefore also be computed as the solution to $f(\boldsymbol{\delta}) = \boldsymbol{\delta} - g(\boldsymbol{\delta}) = \mathbf{0}$, or $\log(\hat{s}(\boldsymbol{\delta}, \boldsymbol{\sigma})) - \log(S) = \mathbf{0}$, which is the rootfinding equivalent to (2). Note that this is an indirect approach; the direct approach solves the system of nonlinear equations (1) for $\boldsymbol{\delta}$. Although mathematically equivalent, the direct and indirect nonlinear rootfinding approach differ from a numerical computation perspective in that log-transforming each component of the system improves the scaling of the problem.[5] The result is a remarkable difference in performance when computing a root using finite-precision arithmetic: not only is the direct approach much slower to converge (approximately up to 60 times), it is also qualitatively inferior to the indirect approach, see Figure 2 for a graphical demonstration of the latter.

[4] We expressed our concerns with respect to the numerical instability of the smooth simulator in predicting market the shares $\hat{s}_{ijt}(\cdot)$ in a private communication with Benjamin Skrainka. Source of this instability is the occurrence of over- and underflow in the numerator and denominator of equation (31). One possible remedy for the problem is higher-precision representation of floats by computers, which can be achieved using low-level programming languages, see Skrainka (2011*b*) for an implementation in C++ .

[5] Other scaling options can also be considered, such as dividing both sides of equation (1) by $\max(S)$.

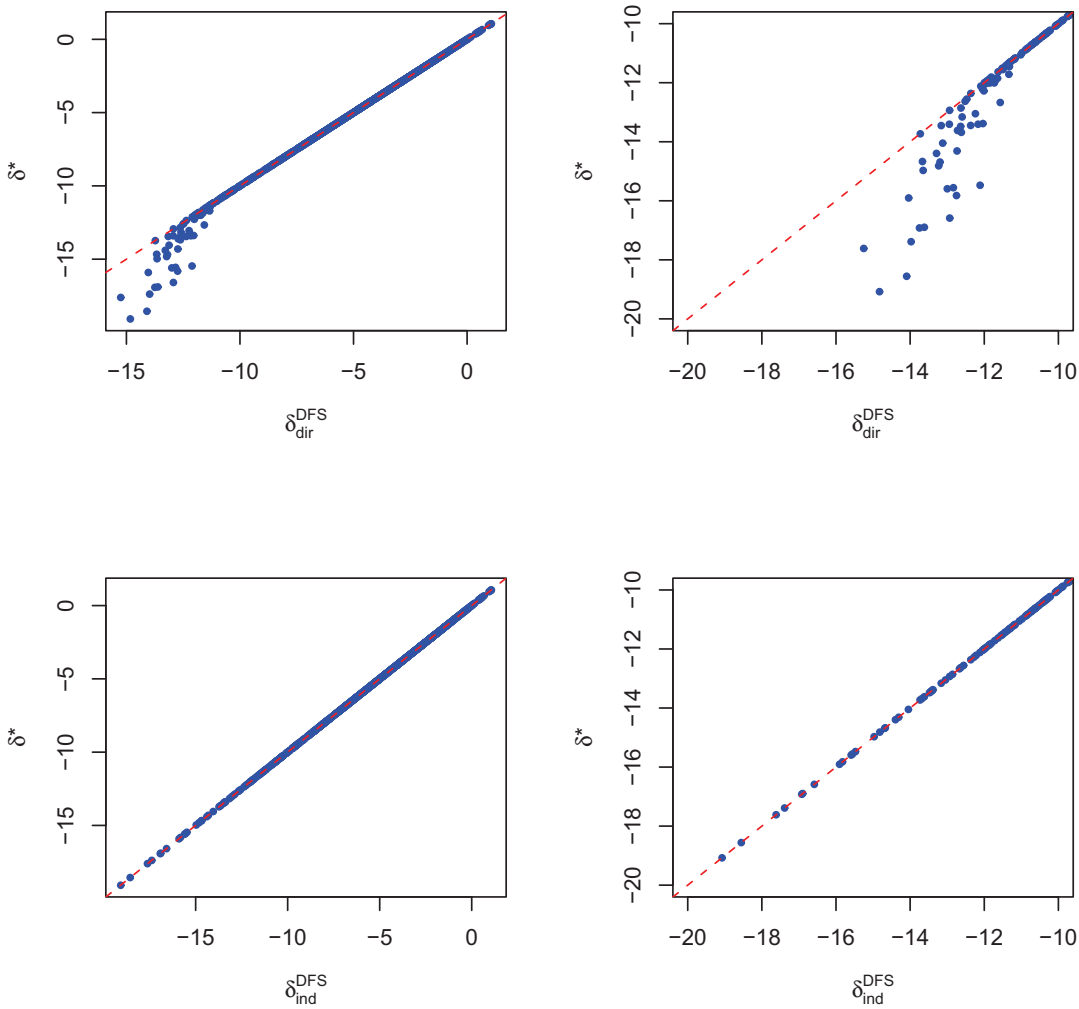

Figure 2: Numerical approximations to the true mean utility vector $\boldsymbol{\delta}^*$ with DF-SANE. Plotted against the "true mean" utility vector $\boldsymbol{\delta}^*$ and with the dashed 45° line representing an identical fit, top row shows solution $\boldsymbol{\delta}_{\text{dir}}^{\text{DFS}}$ to the nonlinear system of equations (1) (direct approach), while bottom row shows solution $\boldsymbol{\delta}_{\text{ind}}^{\text{DFS}}$ to the rootfinding equivalent (5) of the BLP Contraction Mapping (indirect approach). While offering a tighter fit for larger values of $\boldsymbol{\delta}$, the overall quality of approximation using the direct approach is badly affected by the large approximation errors for very small values of $\boldsymbol{\delta}$, as shown by the right column zooming in on the square $[-20,-10]^2$. Parameters of computation: sample size $JT = 1250$, convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$, $n_s = 100$ draws, start values $\boldsymbol{\delta}^0 = \mathbf{0}$ and $\boldsymbol{\sigma}^*$, with $\boldsymbol{\sigma}^* = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$ the "true" vector of standard deviations (nonlinear utility).

2.1 Newton-Raphson and Quasi-Newton Methods

In general, numerical rootfinding algorithms identify the solution $\mathbf{x}^*$ to the system of nonlinear equations f using an iterative scheme

$$\mathbf{x}^{h+1} = \mathbf{x}^h + \alpha^h \mathbf{s}^h, \tag{3}$$

for $h = 0, 1, 2, \ldots$, where $\mathbf{s}^h \in \mathbb{R}^n$ is the *search direction*, and $\alpha^h \in \mathbb{R}$ is the *step length*. Furthermore, a distinction can be made between algorithms that compute the step length using a *line search* technique, and those using a *trust region* method, see Dennis and Schnabel (1983, Section 6.4) for an introduction to the latter. In this section and in Section 2.2 we consider implementations of algorithms for systems of nonlinear equations relying on line search algorithms as a globalization strategy, where the step length is computed as

$$\alpha^h = \arg\min_{\alpha} f\left(\mathbf{x}^h + \alpha \mathbf{s}^h\right). \tag{4}$$

Hence, different numerical algorithms can be identified by the way they define and compute the search direction $\mathbf{s}^h$ and step length α^h.

2.1.1 Newton's method

One of the most elegant methods designed to find $\mathbf{x}^*$ such that $f(\mathbf{x}^*) = \mathbf{0}$, is Newton's method, also known as Newton-Raphson.[6] Newton's method to find the root of $f : \mathbb{R}^n \to \mathbb{R}^n$, where f is a nonlinear function with continuous partial derivatives, is to replace f with a first-order (linear) Taylor series approximation around an initial guess $\mathbf{x}^0$ for the root:

$$f(\mathbf{x}) \approx f\left(\mathbf{x}^0\right) + \mathbf{J}\left(\mathbf{x}^0\right)\left(\mathbf{x} - \mathbf{x}^0\right),$$

where $\mathbf{J} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ is the Jacobian matrix of f evaluated at $\mathbf{x}^0$. Substituting $f(\mathbf{x}) = \mathbf{0}$ then provides a first guess at the root

$$\mathbf{x}^1 = \mathbf{x}^0 - \mathbf{J}\left(\mathbf{x}^0\right)^{-1} f\left(\mathbf{x}^0\right),$$

and hints at the blueprint for the general Newton-Raphson iteration rule

$$\mathbf{x}^{h+1} = \mathbf{x}^h - \mathbf{J}\left(\mathbf{x}^h\right)^{-1} f\left(\mathbf{x}^h\right). \tag{NEW}$$

The iteration continues until some convergence criterion like $\left\|\mathbf{x}^{h+1} - \mathbf{x}^h\right\| \leq \epsilon$ or $\left\|f(\mathbf{x}^h)\right\| \leq \epsilon$ (or a combination of both) is satisfied, where ϵ is typically a very small positive number and $\|\cdot\|$ is a vector norm. Compared with (3) and (4), the simple unmodified Newton-Raphson method (NEW) uses step length $\alpha^h = 1$, and relies on the Jacobian matrix $\mathbf{J}$ to determine the search direction $\mathbf{s}^h = -\mathbf{J}\left(\mathbf{x}^h\right)^{-1} f\left(\mathbf{x}^h\right)$ at each iteration $h = 0, 1, 2, \ldots$ of the algorithm.

[6] See Dennis and Schnabel (1983, Ch. 5), Nash (1990, Ch. 15), Nocedal and Wright (2006, Ch. 11) or Judd (1998, Ch. 5) for detailed discussions.

2.1.2 The Jacobian of the BLP Contraction Mapping

Applying Newton-Raphson in the case of BLP requires an analytical expression for the Jacobian matrix $\mathbf{J}(\cdot)$. Given that any fixed-point problem $\mathbf{x} = g(\mathbf{x})$ can be reformulated as a rootfinding problem $f(\mathbf{x}) = \mathbf{0}$ by defining $f(\mathbf{x}) = \mathbf{x} - g(\mathbf{x})$, the rootfinding equivalent to equation (2) is

$$f : \mathbb{R}^{J\times T} \to \mathbb{R}^{J\times T} : \boldsymbol{\delta} \mapsto f(\boldsymbol{\delta}) = \mathbf{0}, \tag{5}$$

with $f(\boldsymbol{\delta}) = -\log(S) + \log\left(\hat{s}(\boldsymbol{\delta}, \boldsymbol{\sigma})\right)$. The corresponding Jacobian matrix $\mathbf{J}_f : \mathbb{R}^{JT} \to \mathbb{R}^{JT\times JT}$ is a block-diagonal matrix where the non-zero components are the partial derivatives $\partial \log(\hat{s}_{jt})/\partial\delta_{ml}$ computed as

$$\frac{\partial \log(\hat{s}_{jt})}{\partial \delta_{ml}} = \frac{1}{\hat{s}_{jt}} \frac{\partial \hat{s}_{jt}}{\partial \delta_{ml}} = \frac{1}{\hat{s}_{jt}} \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{\partial \hat{s}_{ijt}}{\partial \delta_{ml}}, \tag{6}$$

using the definition of the "smooth" simulator $\hat{s}_{jt}(\cdot) = n_s^{-1} \sum_{i=1}^{n_s} \hat{s}_{ijt}(\cdot)$, see equation (29), and where

$$\frac{\partial \hat{s}_{ijt}}{\partial \delta_{ml}} = \begin{cases} \hat{s}_{ijt} \cdot (1 - \hat{s}_{ijt}) & \text{if } m = j \text{ and } l = t \\ -\hat{s}_{ijt} \cdot \hat{s}_{imt} & \text{if } m \neq j \text{ and } l = t \\ 0 & \text{if } l \neq t. \end{cases} \tag{7}$$

The Jacobian matrix of first derivatives is defined as

$$\mathbf{J}_f(\boldsymbol{\delta}) = \left[\frac{\partial \log(\hat{s}_{jt})}{\partial \delta_{kt}}\right]_{\substack{j,k=1,\ldots,J \\ t=1,\ldots,T}} = \begin{bmatrix} \nabla \log\left(\hat{s}_{11}(\boldsymbol{\delta})\right)' \\ \nabla \log\left(\hat{s}_{21}(\boldsymbol{\delta})\right)' \\ \vdots \\ \nabla \log\left(\hat{s}_{JT}(\boldsymbol{\delta})\right)' \end{bmatrix}$$

where $\nabla \log(\hat{s}_{jt}) = (\partial \log(\hat{s}_{jt})/\partial\delta_{11}, \partial \log(\hat{s}_{jt})/\partial\delta_{21}, \ldots, \partial \log(\hat{s}_{jt})/\partial\delta_{JT})'$ is the gradient vector. The Jacobian matrix can thus be represented as a $JT \times JT$ block-diagonal matrix[7]

$$\mathbf{J}_f(\boldsymbol{\delta}) = \begin{bmatrix} \mathbf{J}_1(\boldsymbol{\delta}) & & & \\ & \mathbf{J}_2(\boldsymbol{\delta}) & & \\ & & \ddots & \\ & & & \mathbf{J}_T(\boldsymbol{\delta}) \end{bmatrix},$$

with typical $J \times J$ non-zero diagonal entry

$$\mathbf{J}_t(\boldsymbol{\delta}) = \left[\frac{\partial \log(\hat{s}_{jt})}{\partial \delta_{mt}}\right]_{m=1,\ldots,J}.$$

Note that the Jacobian matrix is well defined everywhere due to the use of the smooth simulator $\hat{s}(\cdot)$ which is differentiable in δ_{mt} for $m = 1, \ldots, J$ and $t = 1, \ldots, T$, see e.g. Nevo (2001).

[7] Our code for computing the Jacobian $\mathbf{J}_f(\boldsymbol{\delta})$ presently ignores the fact that it is a sparse matrix and that exploiting this property may result in more efficient computer code and faster computation times for the Newton-Raphson algorithm. This is left as an item for future research.

2.1.3 Properties of Newton-Raphson

Given a good starting value $\mathbf{x}^0$, Newton's method is known to converge quadratically, meaning that each iteration h doubles the number of accurate digits to the solution of $f(\mathbf{x}) = \mathbf{0}$. Despite this attractive property, Newton-Raphson has a number of drawbacks (Nocedal and Wright, 2006, pp. 275): first, it has a tendency to diverge if the initial guess is not sufficiently close to the root of $f(\mathbf{x}) = \mathbf{0}$, so it is very sensitive to the choice of start value. Second, the computation of the inverse in equation (NEW) is not only demanding in terms of computer memory and the number of arithmetic operations required, but can also be infeasible if at some point in the iteration process an ill-conditioned Jacobian matrix is computed. Additionally, fast convergence also relies on the user providing an analytical expression for the Jacobian which, in large problems, might prove difficult or even impossible to compute. If in that case a finite-differences approximation to the Jacobian is used, Newton's method is very slow to converge.

2.1.4 Quasi-Newton methods

As a remedy to (some of) these drawbacks, quasi-Newton methods replace the Jacobian matrix $\mathbf{J}$ with an approximation $\mathbf{A}$. Broyden's method, also known as the multidimensional secant method, replaces the Jacobian $\mathbf{J}\left(\mathbf{x}^h\right)$ with an approximation matrix $\mathbf{A}^h$, the computation of which only relies on the information contained in the successive evaluations of f. In particular, Broyden's method generates a sequence of vectors $\mathbf{x}^h$ and matrices $\mathbf{A}^h$ that approximate the root of f and the Jacobian J at the root, respectively, as

$$\mathbf{x}^{h+1} = \mathbf{x}^h - \mathbf{A}^h f\left(\mathbf{x}^h\right) \tag{8}$$

$$\mathbf{A}^{h+1} = \mathbf{A}^h + \frac{(\mathbf{y}^h - \mathbf{A}^h\mathbf{s}^h)(\mathbf{s}^h)'}{(\mathbf{s}^h)'\mathbf{s}^h}, \tag{9}$$

where $\mathbf{y}^h = f(\mathbf{x}^{h+1}) - f(\mathbf{x}^h)$, and $\mathbf{s}^h = -\mathbf{A}^h f(\mathbf{x}^h)$ is the iteration step. Quasi-Newton methods methods are attractive because they converge rapidly from any good starting value.[8]

However, while quasi-Newton methods have the advantage that no analytical Jacobian must be provided by the user, they need to solve a linear system of equations using the Jacobian or an approximation of it at each iteration, which can be prohibitively expensive for high-dimensional problems, as is the case here with the BLP Contraction Mapping.[9] We therefore turn attention to other algorithms that are better equipped to deal with such problems.

[8] The property of global convergence in numerical analysis refers to an algorithm's ability to converge to the solution (a fixed point or the root of a nonlinear system of equations) given any given starting point.

[9] A self-coded version of Broyden's method (8) was initially selected as an alternative in the Monte Carlo simulation. However, convergence turned out to be very slow and the algorithm was dropped for the remainder of the study.

2.1.5 Implementation in R

Appendices 8.1 and 8.2 respectively contain the R code used to compute the market shares and the BLP Contraction Mapping. Functioning as a benchmark against which to compare the performance of the algorithms introduced below, we code the Newton-Raphson algorithm exactly as in equation (NEW), see Appendices 8.4 and 8.5. Distinctive features are a step length $\alpha^h = 1$ for $h = 0, 1, 2, \ldots$, or the absence of a line search technique (such as backtracking or a line search satisfying the strong Wolfe conditions), but with the advantage of an analytical Jacobian matrix to enhance speed and accuracy.[10]

2.2 Spectral Algorithms for Nonlinear Rootfinding

As the number of equations in the system $f(\mathbf{x}) = \mathbf{0}$ increases, corresponding with increasing sample size in BLP, the Jacobian matrix requires increasing amounts of storage, negatively affecting the speed of Newton-Raphson. In such cases, and in cases where a(n) (analytical) Jacobian is simply unavailable, the Barzilai-Borwein (BB) spectral gradient method for solving large-scale optimization problems (Barzilai and Borwein, 1988; Raydan, 1997) is a viable alternative.

Modifying the iterative scheme (3), the BB spectral method for solving nonlinear systems of equations computes the step length α^h (now called the *spectral coefficient*) either as

$$\alpha^h = \frac{(\mathbf{s}^h)'\mathbf{s}^h}{(\mathbf{s}^h)'\mathbf{y}^h}, \tag{10}$$

for $h = 1, 2, \ldots$ with $\mathbf{s}^h = \mathbf{x}^h - \mathbf{x}^{h-1}$ and $\mathbf{y}^h = f(\mathbf{x}^h) - f(\mathbf{x}^{h-1})$, or as

$$\alpha^h = \frac{(\mathbf{s}^h)'\mathbf{y}^h}{(\mathbf{y}^h)'\mathbf{y}^h}. \tag{11}$$

Additionally, the search direction is simply the system evaluated at the current trial, $f(\mathbf{x}^h)$. The general iterative BB rule therefore is

$$\mathbf{x}^{h+1} = \mathbf{x}^h - \alpha^h f(\mathbf{x}^h), \tag{BB}$$

at iteration $h = 0, 1, 2, \ldots$ of the algorithm. Equation (BB) reveals that, relative to Newton-Raphson, no Jacobian or an approximation to it must be computed.[11]

Extending the BB method, the DF-SANE algorithm (La Cruz *et al.*, 2006) systematically uses $\pm f(\mathbf{x})$ as search direction and complements (BB) with a non-monotone line search technique to assure global convergence in the spirit of Grippo *et al.* (1986). This

[10] For an implementation of Newton-Raphson complemented with a trust region technique, see the **nleqslv** package (Hasselman, 2009). As the focus of this paper is on finding methods that can accelerate the BLP Contraction Mapping, using **nleqslv** proved to be too slow in comparison with the other alternatives and was therefore excluded from the Monte Carlo study.

[11] Note that spectral algorithms may be viewed as (quasi-)Newton methods by specifying the Jacobian as a constant matrix $\mathbf{J} = 1/\alpha^h$.

line search technique does not require any Jacobian computations either,[12] and can be written as

$$m(\mathbf{x}^{h+1}) \leq \max_{0 \leq j \leq M} m(\mathbf{x}^{h-j}) + \eta^h - \gamma(\alpha^h)^2 m(\mathbf{x}^h), \tag{12}$$

where $m(\mathbf{x}) = \|f(\mathbf{x})\|_2^2 = f(\mathbf{x})' f(\mathbf{x})$ is a merit function, γ is a small positive number usually fixed at 10^{-4}, and $M \in \mathbb{N}$ a parameter that specifies the allowable degree of non-monotonicity in the value of the merit function. The parameter $\eta^h > 0$ ensures that all the iterations are well-defined; it decreases with h and satisfies the summability condition $\sum_{h=0}^{\infty} \eta^h = \eta < \infty$. The last term in equation (12) is a forcing term that provides the theoretical condition sufficient for establishing global convergence, see La Cruz *et al.* (2006).

2.2.1 Implementation in R

Section 8.3 contains the code used to transform the BLP Contraction Mapping into a nonlinear rootfinding problem. Subsequently, the DF-SANE algorithm for solving equation (5) was implemented using the `dfsane` function from the **BB** package with all arguments set to their defaults, i.e. default BB step length (11), non-monotonicity parameter $M = 10$, and $\gamma = 10^{-4}$. For an overview of these defaults and those of the spectral line search technique (12), see Varadhan and Gilbert (2009) or simply enter `library(BB)` and `dfsane` in the R console.

2.3 Squared Polynomial Extrapolation Methods

Instead of reformulating the BLP Contraction Mapping as a system of nonlinear equations, another approach is to stick to the fixed-point formulation and enhance its convergence properties using acceleration algorithms. In a series of papers Roland and Varadhan (2005), Roland *et al.* (2007) and Varadhan and Roland (2008) propose a new class of methods, called SQUAREM, for accelerating the convergence of fixed-point iterations. Initially developed for accelerating the Expectation-Maximization (EM) algorithm in computational statistics and image reconstruction problems for positron emission tomography (PET) in medical imaging, the SQUAREM class employs an iteration scheme

$$\mathbf{x}^{h+1} = \mathbf{x}^h - 2\alpha^h \mathbf{r}^h + (\alpha^h)^2 \mathbf{v}^h, \tag{13}$$

where the step length is defined as

$$\alpha^h = \frac{(\mathbf{v}^h)' \mathbf{r}^h}{(\mathbf{v}^h)' \mathbf{v}^h}, \tag{14}$$

[12] Hence the acronym DF-SANE, or *derivative-free spectral algorithm for nonlinear equations*. DF-SANE is an improvement upon the SANE algorithm (La Cruz and Raydan, 2003) where the line search technique

$$m(\mathbf{x}^{h+1}) \leq \max_{0 \leq j \leq M} m(\mathbf{x}^{h-j}) + \gamma \alpha^h \nabla m(\mathbf{x}^h)' f(\mathbf{x}^h), \tag{GLL}$$

does involve the computation of the Jacobian $\nabla m(\mathbf{x}^h)$ and requires an additional function evaluation when computing the quadratic product $\nabla m(\mathbf{x}^h)' f(\mathbf{x}^h)$.

with residual $\mathbf{r}^h = g(\mathbf{x}^h) - \mathbf{x}^h$ and $\mathbf{v}^h = g(g(\mathbf{x}^h)) - 2g(\mathbf{x}^h) + \mathbf{x}^h$. In contrast with Newton-Raphson and DF-SANE, SQUAREM thus operates directly on the fixed-point formulation (2) of the BLP Contraction Mapping.

2.3.1 Implementation in R

Section 8.6 presents the single-update version of the BLP Contraction Mapping needed by the SQUAREM algorithm. We rely on the `squarem` function as provided by the **SQUAREM** package for solving (2) using the default step length

$$\alpha^h = \operatorname{sgn}\left((\mathbf{s}^h)'\mathbf{y}^h\right)\frac{\left\|\mathbf{s}^h\right\|}{\left\|\mathbf{y}^h\right\|}, \tag{15}$$

where

$$\operatorname{sgn}(x) = \begin{cases} \dfrac{x}{|x|} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

The solution to (2) is handled as a non-monotone problem by specifying the option `kr = Inf` which controls the amount of non-monotonicity in the fixed-point residual $\mathbf{r}^h$. For more details, see Varadhan (2011).

3 Monte Carlo Study

In order to assess the convergence properties of the BLP, Newton-Raphson, DF-SANE and SQUAREM algorithms, respectively indexed as

$$a \in \mathcal{A} = \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\},$$

we ran $R = 1000$ Monte Carlo replications for each of the following three simulation scenarios referred to as "good," "bad" an "ugly," and computed a numerical approximation to the mean utility vector $\boldsymbol{\delta}$. Each of these scenarios represents a possible situation for the value of the $\boldsymbol{\sigma}$ vector produced by minimizing the outer-loop GMM objective function (32).

Scenario "Good" The values for $\boldsymbol{\sigma}$ computed by the optimization routine and passed on to the inner loop, are reasonably close to the true parameter values $\boldsymbol{\sigma}^* = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$. We simulate this by drawing from the standard uniform distribution, or $\boldsymbol{\sigma} \sim \mathcal{U}[0, 1]$.

Scenario "Bad" By drawing from the standard normal distribution, this case represents a situation in which $\boldsymbol{\sigma}$ has (absolute) values that are close to true solution but where some (or all) elements of the vector have the wrong sign, i.e. are negative.

Scenario "Ugly" This scenario represents situations in which the outer loop is generating values for $\boldsymbol{\sigma}$ that are very far from the solution. Hence we draw from the uniform distribution over the $[0, 7]$ interval, or $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$.

Additionally, to show the effect of sample size on convergence properties, we run the previous scenarios for three different sample sizes, simply by increasing the number of markets T from 50 to 100 and 200, thus accounting for sample sizes of 1250, 2500 and 5000 observations respectively.

Finally, to show the extent to which the various algorithms are sensitive to starting values, all of the previous scenarios are simulated using two different start values for the $\boldsymbol{\delta}$ vector, with $\boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t})$ being the "analytical" delta as proposed by Berry (1994), and $\boldsymbol{\delta}^0 = \mathbf{0}$ the null vector often used as a start value in numerical algorithms.[13]

3.0.2 Interpretation of the Monte Carlo Scenarios

By taking 1000 replications of the previous scenarios, the interpretation is that each replication represents an iteration in the outer-loop minimization of the GMM objective function (32) that generates a specific value for the $\boldsymbol{\sigma}$ vector. By drawing from different distributions, an environment is created in which the outer loop is having more and more difficulties in computing a value for $\boldsymbol{\sigma}$ that is close to the solution.

Another (chronological) point of view is that the outer loop starts of somewhere in a region of the parameter space that resembles the "ugly" scenario. With the passing of each iteration, the outer loop gradually finds values for $\boldsymbol{\sigma}$ that belong to the "bad" case, and as the estimation procedure continues, finally moves on to values that are close to the solution of the optimization problem.

3.1 Monte Carlo Parameters and Evaluation Metrics

In running the $r = 1, \ldots, R$ Monte Carlo replications, convergence is declared when the stopping rule

$$\left\| \boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h} \right\|_2 \leq \epsilon_{\text{in}} \tag{16}$$

is satisfied. In (16), $\|\cdot\|_2$ is the Euclidean distance or L_2 norm, $\boldsymbol{\delta}^{a,h}$ the numerical approximation computed by algorithm $a \in \mathcal{A}$ at iteration $h = 0, 1, 2, \ldots$, and $\epsilon_{\text{in}} = 10^{-7}$ the inner-loop tolerance level. To keep algorithms from continuing without achieving convergence, the maximum number of iterations is set at `max.it` $= 1500$.

In evaluating the algorithms, we use the following criteria which measure both performance in terms of convergence (`iter`, `time`, `conv`), and quality of the approximation (L_2, L_∞, MSE, variance, bias and MAE):

`iter` Number of iterations taken by algorithm a to converge.

`time` CPU time in seconds taken by algorithm a. Specifically, this measure corresponds with the CPU time charged for the execution of user instructions of the corresponding algorithm, being the first element of the numeric vector returned by the `system.time()` function in R .

[13] Here s_{0t} refers to the market share of the "outside" good, see Section 7 for an explanation.

`conv` Percentage of Monte Carlo replications in which algorithm a succesfully converged, where convergence is defined as the ability of the algorithm to compute a numerical approximation to $\boldsymbol{\delta}^*$ within the maximum number of iterations allowed, `max.it` $= 1500$.

L_2 The Euclidean distance, or the statistic to measure the quality of the approximation to $\boldsymbol{\delta}^*$, defined as

$$\left\|\boldsymbol{\delta}^a - \boldsymbol{\delta}^*\right\|_2 := \sqrt{\sum_{i=1}^{J\times T}\left(\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*\right)^2}. \tag{17}$$

L_∞ The supremum norm as an alternative measure of the quality of approximation, defined as

$$\left\|\boldsymbol{\delta}^a - \boldsymbol{\delta}^*\right\|_\infty := \max_{i\in JT}\left|\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*\right|. \tag{18}$$

With respect to the L_2 and L_∞ norms assessing the quality of approximation, $\boldsymbol{\delta}^*$ denotes the "true" mean utility vector computed as $\boldsymbol{\delta}^* = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\xi}$ where $\mathbf{X}, \boldsymbol{\beta}$ and $\xi_{j,t} \in \boldsymbol{\xi}$ are generated using the Data Generating Process (DGP) described in Section 3.2, With the exception of the `conv` criterion which is averaged over all $R = 1000$ replications, the former metrics are subsequently reported as averages over those replications where all four algorithms successfully converged in order to guarantee a fair comparison between the algorithms.

In computing the mean squared error (MSE), its decomposition in variance and bias using the property $\text{MSE}\left(\boldsymbol{\delta}^a\right) = \text{Var}(\boldsymbol{\delta}) + \left(\text{Bias}\left(\boldsymbol{\delta}^a, \boldsymbol{\delta}^*\right)\right)^2$, and the mean absolute error (MAE) we draw attention to the fact that the former are all vectors of dimension $J \times T$ computed as averages over the Monte Carlo replications $r = 1, \ldots, R$. These metrics are summarized by taking the mean of these vectors.[14] In the following, calculations are executed component-wise:

MSE Mean squared error $\mathbb{E}\left[\left(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*\right)^2\right]$, computed as

$$\text{MSE}\left(\boldsymbol{\delta}^a\right) = \frac{1}{R}\sum_{r=1}^{R}\left(\boldsymbol{\delta}^{a(r)} - \boldsymbol{\delta}^*\right)^2. \tag{19}$$

Bias The bias, defined as $\mathbb{E}\left[\boldsymbol{\delta}^a - \boldsymbol{\delta}^*\right]$, is calculated as

$$\text{Bias}\left(\boldsymbol{\delta}^{a(r)}, \boldsymbol{\delta}^*\right) = \frac{1}{R}\sum_{r=1}^{R}\left(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*\right). \tag{20}$$

[14] Therefore, the actual numbers for e.g. MSE reported in Tables 1 to 18 are computed as

$$\text{MSE}\left(\boldsymbol{\delta}^a\right) = \frac{1}{JT}\sum_{i=1}^{JT}\left[\frac{1}{R}\sum_{r=1}^{R}\left(\boldsymbol{\delta}_i^{a(r)} - \boldsymbol{\delta}^*\right)^2\right].$$

Variance Defined as $\mathbb{E}\left[\left(\boldsymbol{\delta}^a - \bar{\boldsymbol{\delta}}\right)^2\right]$, the variance is calculated as

$$\text{Var}\left(\boldsymbol{\delta}^a\right) = \frac{1}{R}\sum_{r=1}^{R}\left(\boldsymbol{\delta}^{a(r)} - \bar{\boldsymbol{\delta}}\right)^2, \tag{21}$$

where $\bar{\boldsymbol{\delta}} = R^{-1}\sum_{r=1}^{R}\boldsymbol{\delta}^{a(r)}$ is the mean over the Monte Carlo replications.

Finally, we also compute the mean absolute error (MAE) as an alternative to the mean squared error that is less sensitive to outliers:

MAE Mean absolute error, computed as

$$\text{MAE}\left(\boldsymbol{\delta}^a\right) = \frac{1}{R}\sum_{r=1}^{R}\left|\boldsymbol{\delta}^{a(r)} - \boldsymbol{\delta}^*\right|. \tag{22}$$

3.2 Synthetic Data Set

We simulate the data set used in the Monte Carlo study following the DGP proposed by Dubé *et al.* (2012). Their synthetic data set consists of $T = 50$ independent markets, each with the same set of $J = 25$ products. Each product $j = 1, \ldots, J$ has $K = 3$ market-invariant observed product characteristics $x_{k,j}$, distributed as

$$\begin{bmatrix} x_{1,j} \\ x_{2,j} \\ x_{3,j} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & -0.8 & 0.3 \\ -0.8 & 1 & 0.3 \\ 0.3 & 0.3 & 1 \end{bmatrix}\right),$$

or, using a shorthand, $\mathbf{X} \sim \mathcal{N}(\mu_{\mathbf{X}}, \Sigma_{\mathbf{X}})$ where $\mathbf{X}$ is the matrix of observed product characteristics. Note that the product characteristics are correlated, as demonstrated by the non-zero off-diagonal elements of the variance-covariance matrix $\Sigma_{\mathbf{X}}$.

To assure variation in the data over markets, products also have a market-specific vertical (quality) characteristic $\xi_{j,t} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ and a market-specific price generated as

$$p_{j,t} = \left|0.5 \cdot \xi_{j,t} + e_{j,t} + 1.1\sum_{k=1}^{K} x_{k,j}\right|, \tag{23}$$

where $e_{j,t} \sim \mathcal{N}(0, 1)$ is a market-specific price shock. Note how equation (23) allows for explicit correlation between product price and unobserved product characteristics, eliciting the use of instruments for consistent estimation of demand parameters.

The random coefficients that reflect the heterogeneity in consumer preferences are captured by the $\boldsymbol{\beta}_i$ vector which in this case is of length five, reflecting differences in consumer tastes with respect to the intercept, the three product characteristics, and the price. Each random coefficient in $\boldsymbol{\beta}_i = \left(\beta_i^0, \beta_i^1, \beta_i^2, \beta_i^3, \beta_i^p\right)'$ is distributed independently normal with means

$$\mathbb{E}[\boldsymbol{\beta}_i] = (-1.0, 1.5, 1.5, 0.5, -3.0)',$$

and variances

$$\text{Var}[\boldsymbol{\beta}_i] = (0.5, 0.5, 0.5, 0.5, 0.2)'.$$

The former and especially the latter (or its square root) are respectively referred to as the "true" linear and nonlinear coefficients of utility that constitute the center of attention in the BLP estimation procedure, and are denoted as $\boldsymbol{\beta}^*$ and $\boldsymbol{\sigma}^*$ respectively.

Finally, to simulate the integral in the market share equation, we draw $n_s = 100$ fictitious consumers from the standard normal distribution for each product characteristic with a random coefficient. As in Dubé *et al.* (2012), these remain fixed in order to abstract away from the effects of simulation error.

4 Results

4.1 Response to Changing Monte Carlo Parameters

Before discussing the relative performance of the algorithms we first summarize the effect of changing the Monte Carlo simulation parameters on the performance of all four algorithms. In general, increasing the level of these parameters is in line with the a priori expectations that the convergence properties are negatively affected, as can be verified by the general patterns displayed in Figures 3 and 7.

Effect of sample size Increasing the sample size has a proportional effect on convergence properties in that doubling the number of observations nearly doubles computation times except for NEW where the effect is more than proportional, especially in the case where the null vector is used as a starting value for $\boldsymbol{\delta}$. The effect on the number of iterations is remarkably different in that it only seems to affect BLP in passing from 1250 to 2500 observations (with a less than proportional effect), whereas the effect on the other algorithms is very small. So, in terms of computational costs, DFS and SQM are less affected by increasing sample size.

The effect on the success rate appears less than proportional in the "'bad" scenario, and more than proportional in the "ugly" case. Important to note is that convergence is unaffected in regions of the parameter space for $\boldsymbol{\sigma}$ close to the minimum of the GMM objective function, i.e. the "good" scenario.

Effect of inner-loop start value $\boldsymbol{\delta}^0$ In general, using the null vector for $\boldsymbol{\delta} = \mathbf{0}$ instead of the analytical value *ceteris paribus* increases the time and number of iterations needed for the algorithms to converge, and this effect is more outspoken for NEW. The effect does seem to diminish though as the values for $\boldsymbol{\sigma}$ deteriorate in passing from the "good" to the "bad" and "ugly" scenario.

With respect to stability, the evidence is mixed in that the success rate stays the same, is enhanced or worsened for BLP, DFS and SQM. NEW on the other hand is always negatively affected by choosing an inferior-quality starting value for $\boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t})$.

Effect of outer-loop start value $\boldsymbol{\sigma}$ The values for $\boldsymbol{\sigma}$ are the driving force behind the convergence results for the algorithms. In passing from the "good" to the "bad" and "ugly" scenario, each algorithm is experiencing more difficulties in reconciling the

predicted theoretical market shares $\hat{s}_{jt}$ with their observed counterparts in reality S_{jt}.

In conclusion, where the general trends are as expected, it is primarily the value of $\boldsymbol{\sigma}$ coming from the outer-loop GMM minimization that drives the performances of the algorithms in finding an approximation to the true mean utility vector $\boldsymbol{\delta}^*$.

4.2 Relative Performance of the Algorithms

Whereas the differences between the CPU time, number of iterations, and success rate are easily discernible, the other evaluation metrics defined in Subsection 3.1 only differ at the 7th, 8th or even 9th decimal, meaning that the difference is less than 10^{-6}. Given the stopping rule (16), we therefore only report results with 7 significant digits.[15] Before going into details, we first provide a synopsis of the results reported in Tables 1 to 9 for simulations with an analytical delta as start value, and Tables 10 to 18 for simulations using the null vector.

- In terms of CPU time, both DFS and SQM are faster than BLP, independent of sample size and simulation scenario. These algorithms offer rates of acceleration that vary between 3.23 and 5.29 for DFS, and 3.34 and 5.22 for SQM.[16] NEW on the other hand, while faster under ideal circumstances, looses its advantage over BLP as the sample size increases.

- Additionally, SQM nearly always outperforms DFS in terms of CPU time and success rate. Of all three alternatives, SQM is nearly always faster and nearly always yields a better numerical approximation to $\boldsymbol{\delta}^*$.

- BLP nearly always produces the best numerical approximation to $\boldsymbol{\delta}^*$; only in a few cases did BLP perform worse in terms of L_2 or L_∞ when either SQM or DFS were better. Additionally, even though it sometimes has either larger bias or variance, BLP always manages to have smallest MSE.

- Finally, under the most demanding circumstances SQM produces better convergence properties than BLP (with up to 14 percentage points in terms of the success rate) and offers a considerable increase in speed (up to 5.22 times faster).

Due to the multitude of evaluation metrics, and in order to corroborate and emphasize the previous findings, we translate the results from Tables 1 to 18 into graphs. Figure 3 shows the relative performance of all algorithms in terms of CPU time, number of iterations and success rate. While some of the results are clearly displayed, some of the differences between SQM and DFS cannot be displayed properly by means of Figure 3. We therefore also display these separately in Figure 4. Although requiring more iterations, Figure 4 clearly shows that SQM is both faster and more stable than DFS.

[15] Tables containing evaluation metrics with 10 digits after the comma are available for consultation on the authors' website.

[16] Rates of acceleration for simulations with analytical and null delta respectively: $\gamma_{\text{DFS}} \in [3.23, 4.99]$ and $\gamma_{\text{DFS}} \in [3.25, 5.29]$, $\gamma_{\text{SQM}} \in [3.34, 5.22]$ and $\gamma_{\text{SQM}} \in [3.47, 5.19]$.

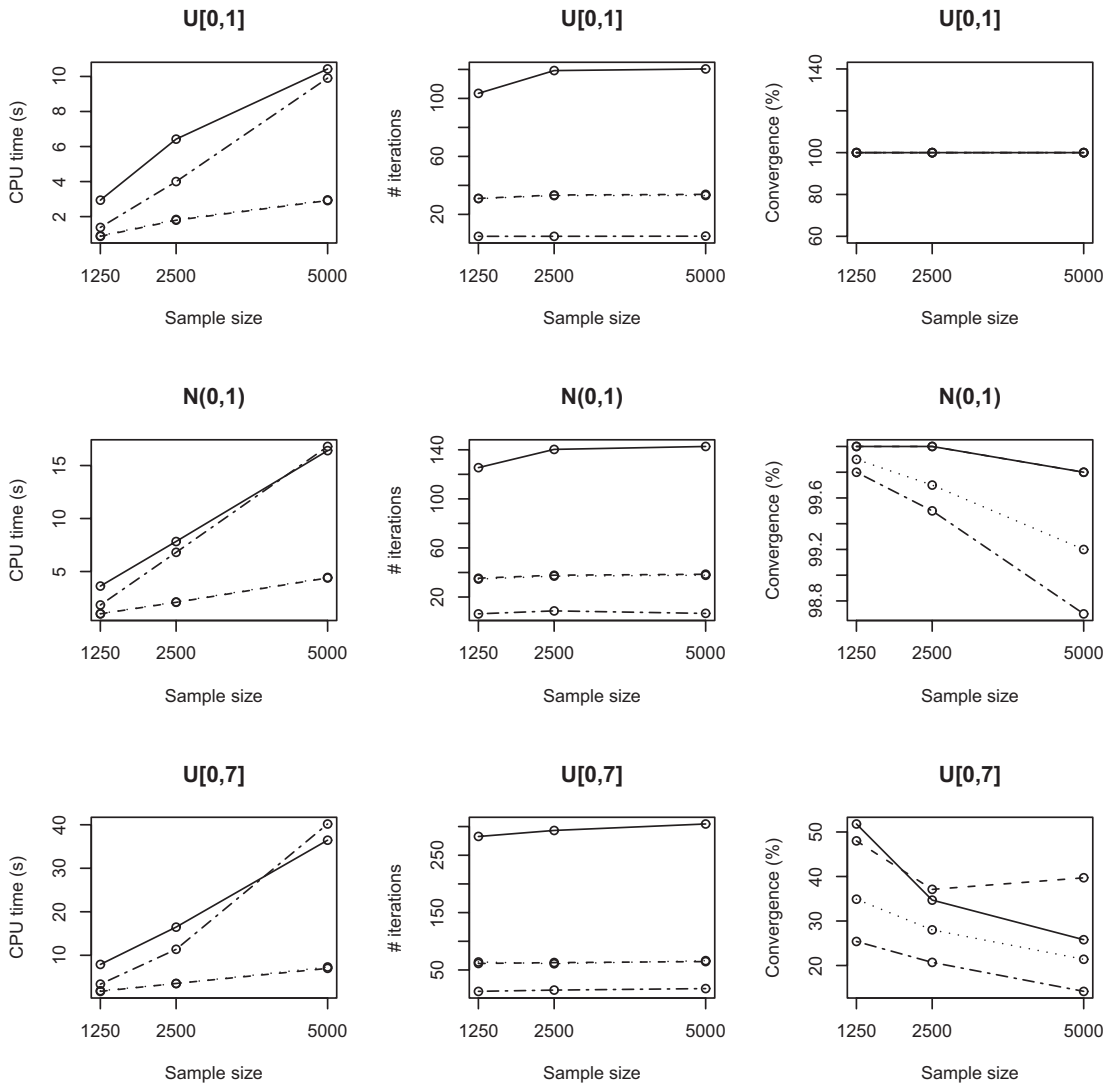

Figure 3: Convergence properties for BLP, NEW, DFS and SQM with analytical delta as start value. Solid $(-)$, dashed-dotted $(-\cdot-)$, dotted $(\cdots)$ and dashed $(--)$ lines respectively represent results for BLP, NEW, DFS and SQM. From left to right, columns display the average CPU time in seconds, the average number of iterations, and the success rate as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t})$ as start value. The success rate is the percentage of times the algorithm succeeded in computing a numerical approximation to $\boldsymbol{\delta}$ within the maximum number of iterations allowed, set at 1500.

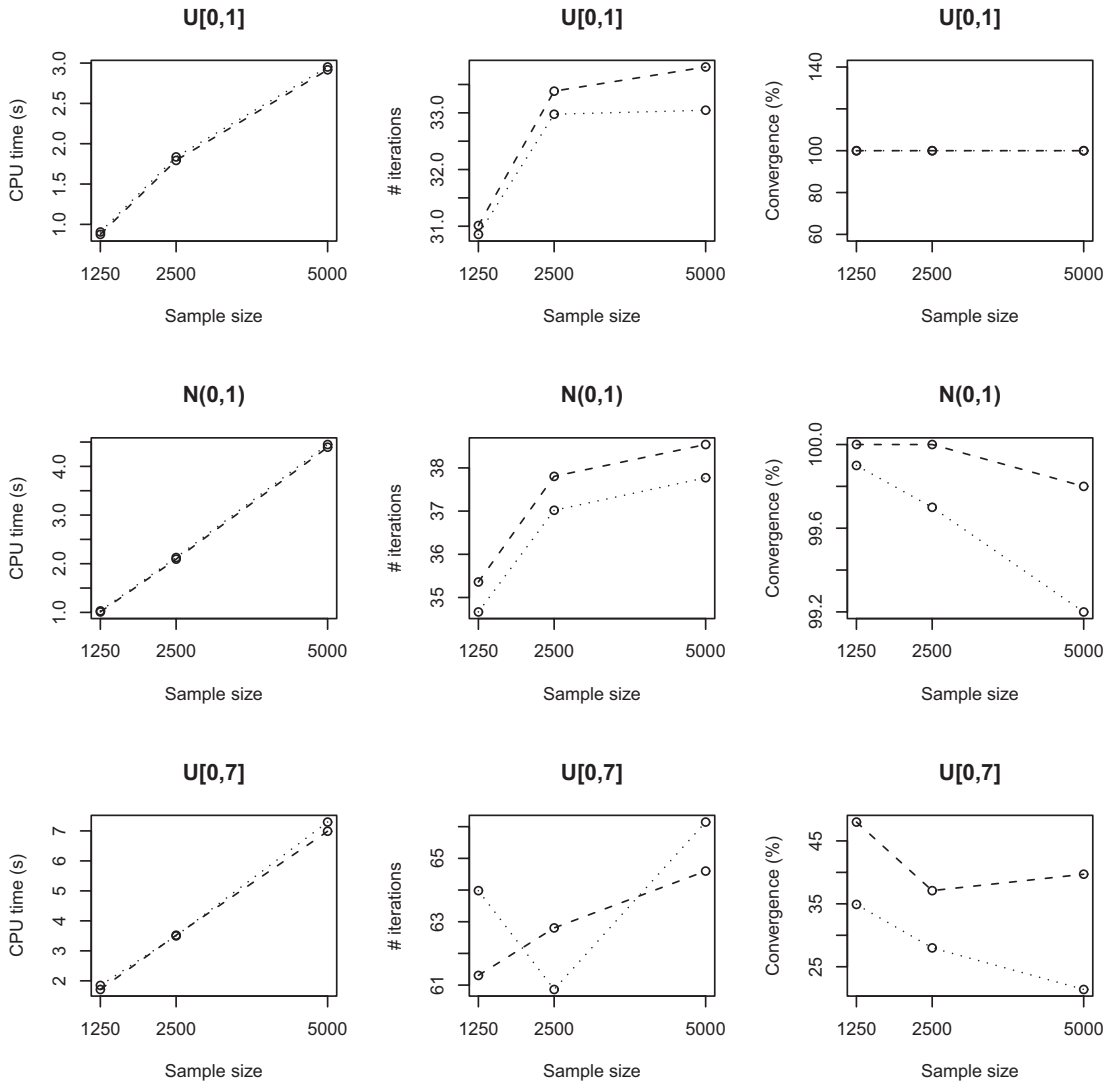

Figure 4: Convergence properties for DFS and SQM with analytical delta as start value. Dotted $(\cdots)$ and dashed $(--)$ lines respectively represent results for DFS and SQM. From left to right, columns display the average CPU time in seconds, the average number of iterations, and the success rate as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^{0} = \log(\hat{s}_{jt}(\cdot)/s_{0t})$ as start value. The success rate is the percentage of times the algorithm succeeded in computing a numerical approximation to $\boldsymbol{\delta}$ within the maximum number of iterations allowed, set at 1500.

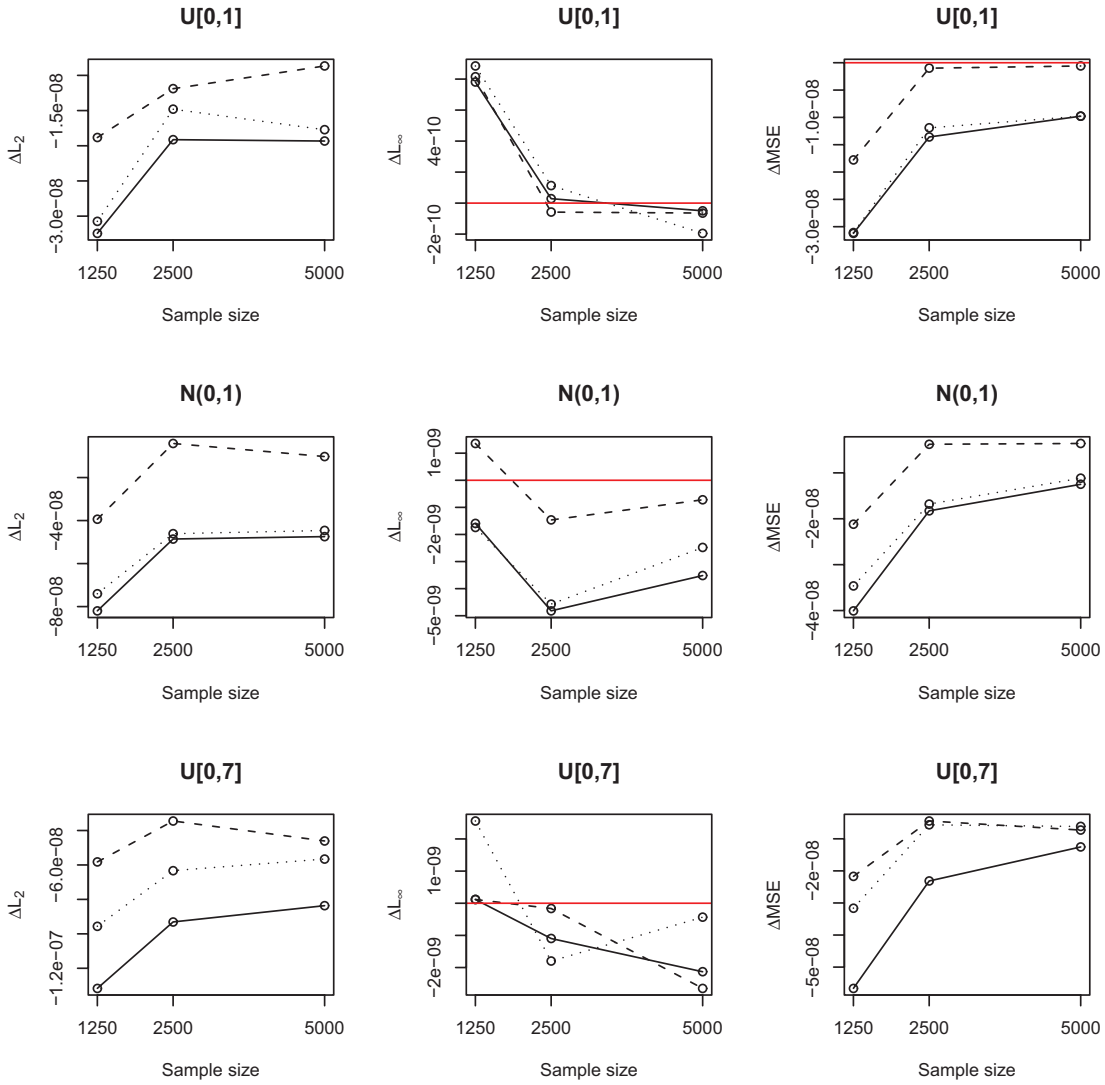

Figure 5: Quality of approximation for NEW, DFS and SQM with analytical delta as start value. From left to right, columns display the difference Δ between BLP and NEW (solid line, $-$), DFS (dotted line, $\cdots$) and SQM (dashed line, $--$) in the approximation error $\left\|\boldsymbol{\delta}^* - \boldsymbol{\delta}^a\right\|$ measured by L_2, L_∞ and MSE as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^h\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t})$ as start value.

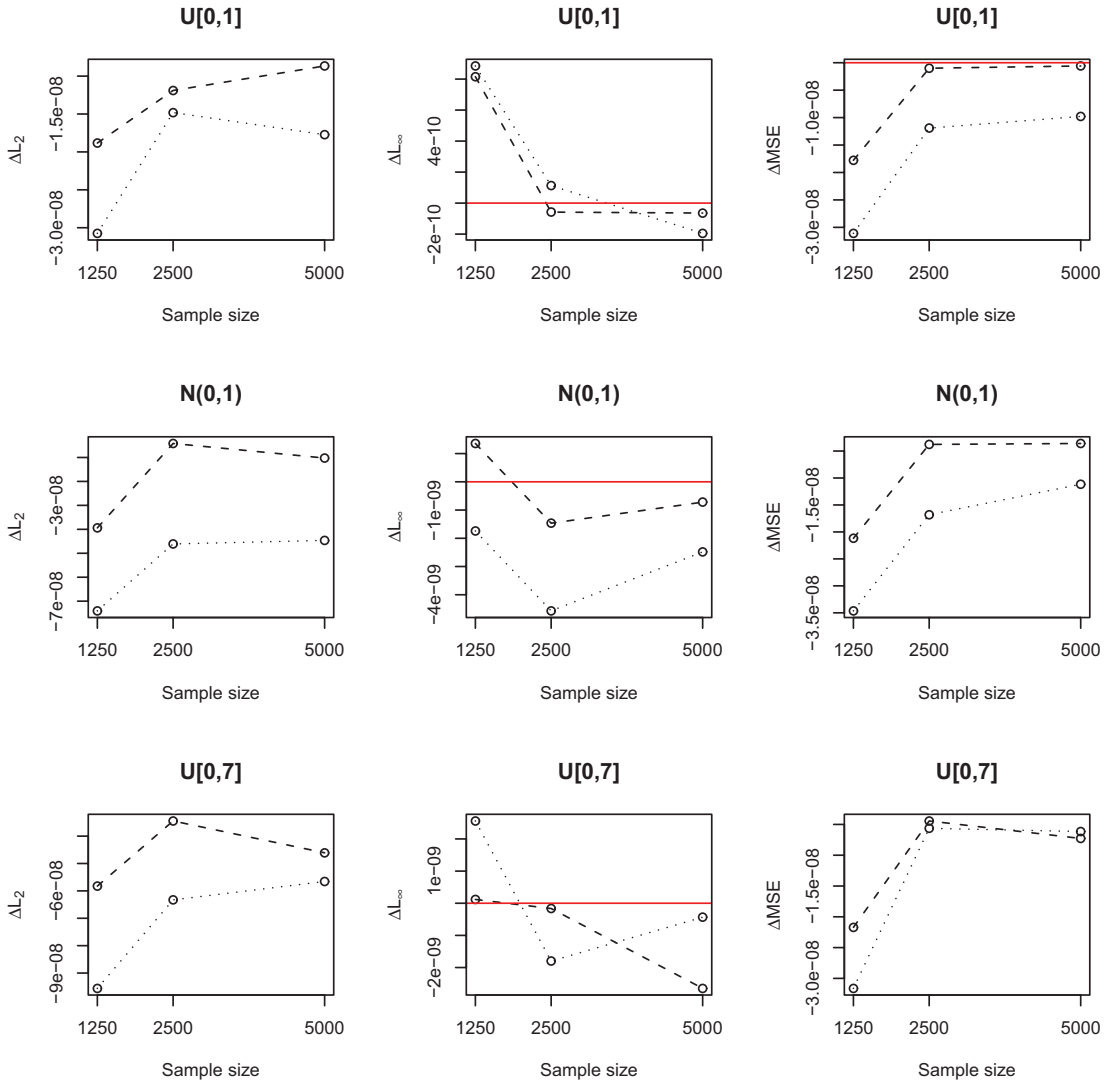

Figure 6: Quality of approximation for DFS and SQM with analytical delta as start value. From left to right, columns display the difference Δ between BLP and DFS (dotted line, $\cdots$) and SQM (dashed line, $--$) in the approximation error $\left\|\boldsymbol{\delta}^* - \boldsymbol{\delta}^a\right\|$ measured by L_2, L_∞ and MSE as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^h\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t})$ as start value.

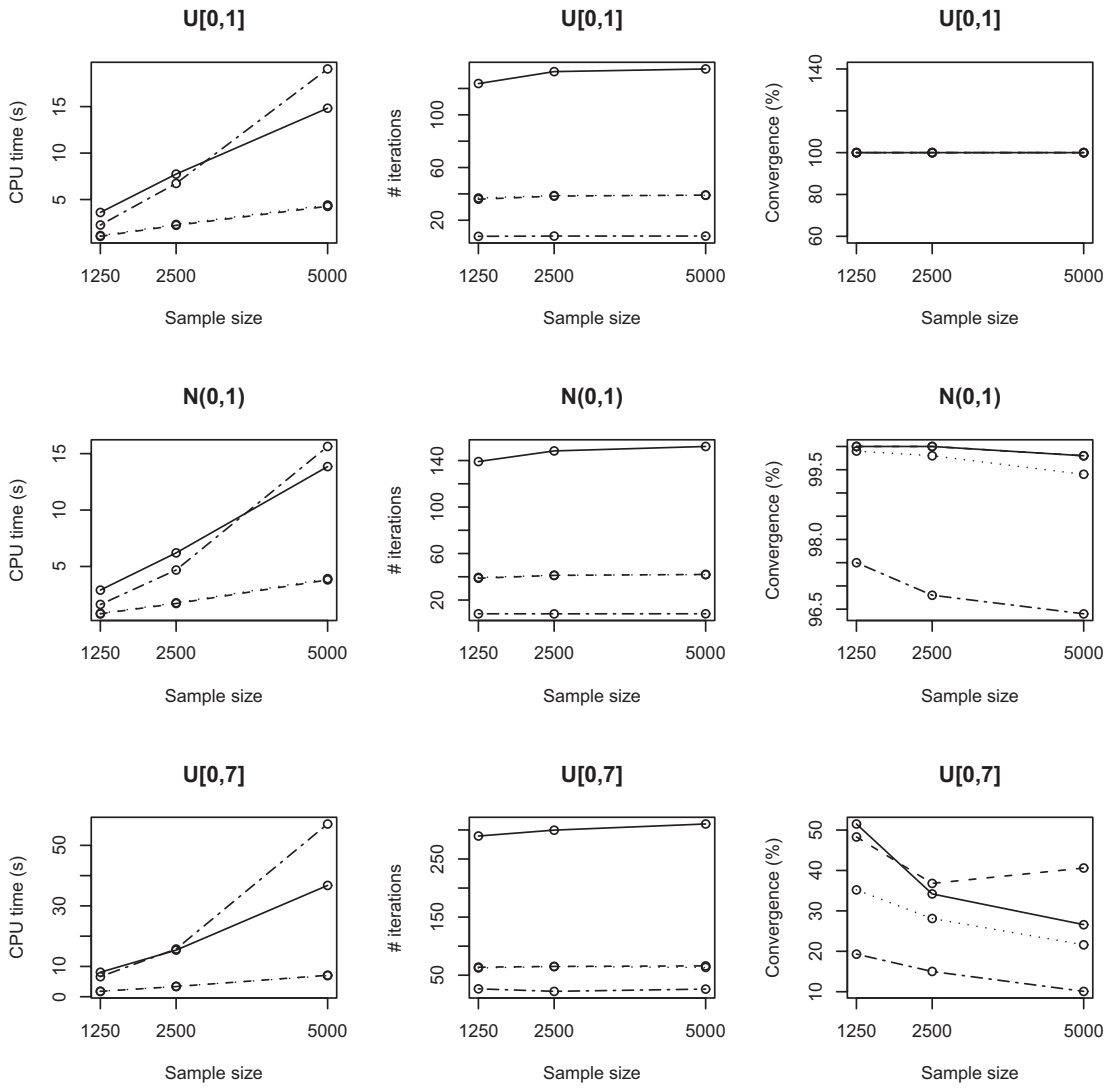

Figure 7: Convergence properties for BLP, NEW, DFS and SQM with null vector as start value. Solid (−), dashed-dotted (− · −), dotted (· · ·) and dashed (−−) lines respectively represent results for BLP, NEW, DFS and SQM. From left to right, columns display the average CPU time in seconds, the average number of iterations, and the success rate as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value. The success rate is the percentage of times the algorithm succeeded in computing a numerical approximation to $\boldsymbol{\delta}$ within the maximum number of iterations allowed, set at 1500.

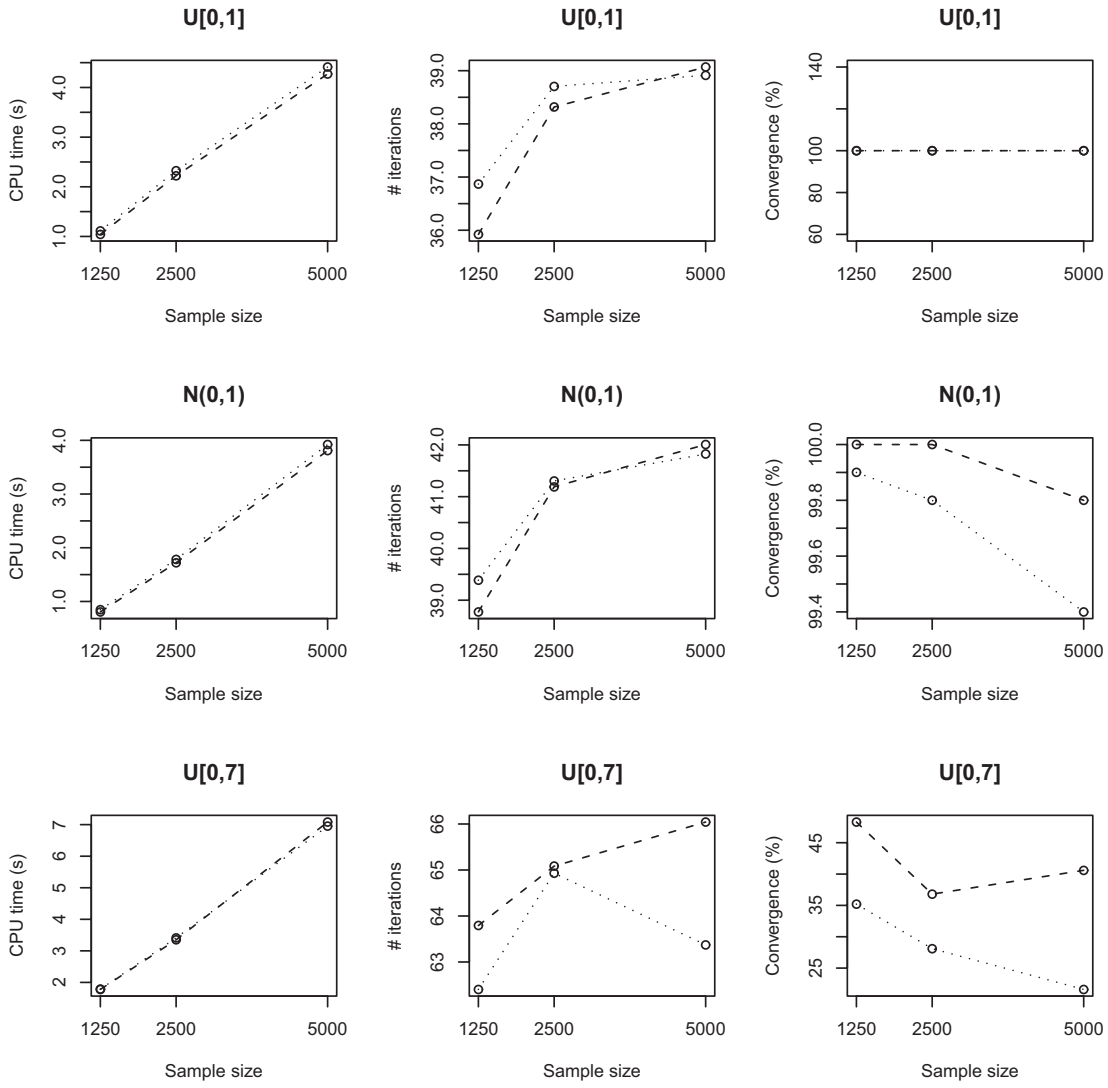

Figure 8: Convergence properties for DFS and SQM with null vector as start value. Dotted $(\cdots)$ and dashed $(--)$ lines respectively represent results for DFS and SQM. From left to right, columns display the average CPU time in seconds, the average number of iterations, and the success rate as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value. The success rate is the percentage of times the algorithm succeeded in computing a numerical approximation to $\boldsymbol{\delta}$ within the maximum number of iterations allowed, set at 1500.

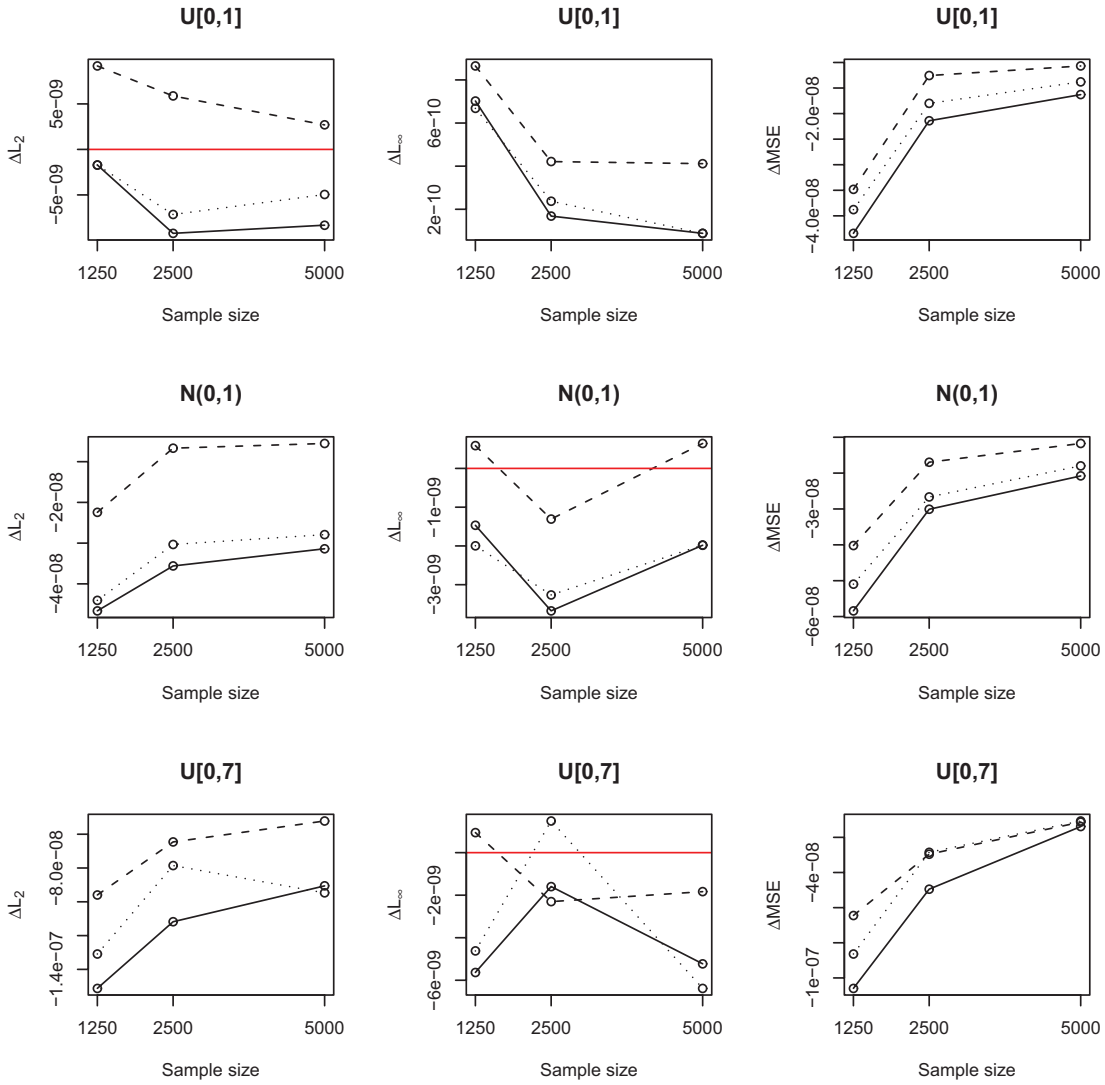

Figure 9: Quality of approximation for NEW, DFS and SQM with null vector as start value. From left to right, columns display the difference Δ between BLP and NEW (solid line, $-$), DFS (dotted line, $\cdots$) and SQM (dashed line, $--$) in the approximation error $\left\|\boldsymbol{\delta}^{*}-\boldsymbol{\delta}^{a}\right\|$ measured by L_2, L_∞ and MSE as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1}-\boldsymbol{\delta}^{h}\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^{0} = \mathbf{0}$ as start value.

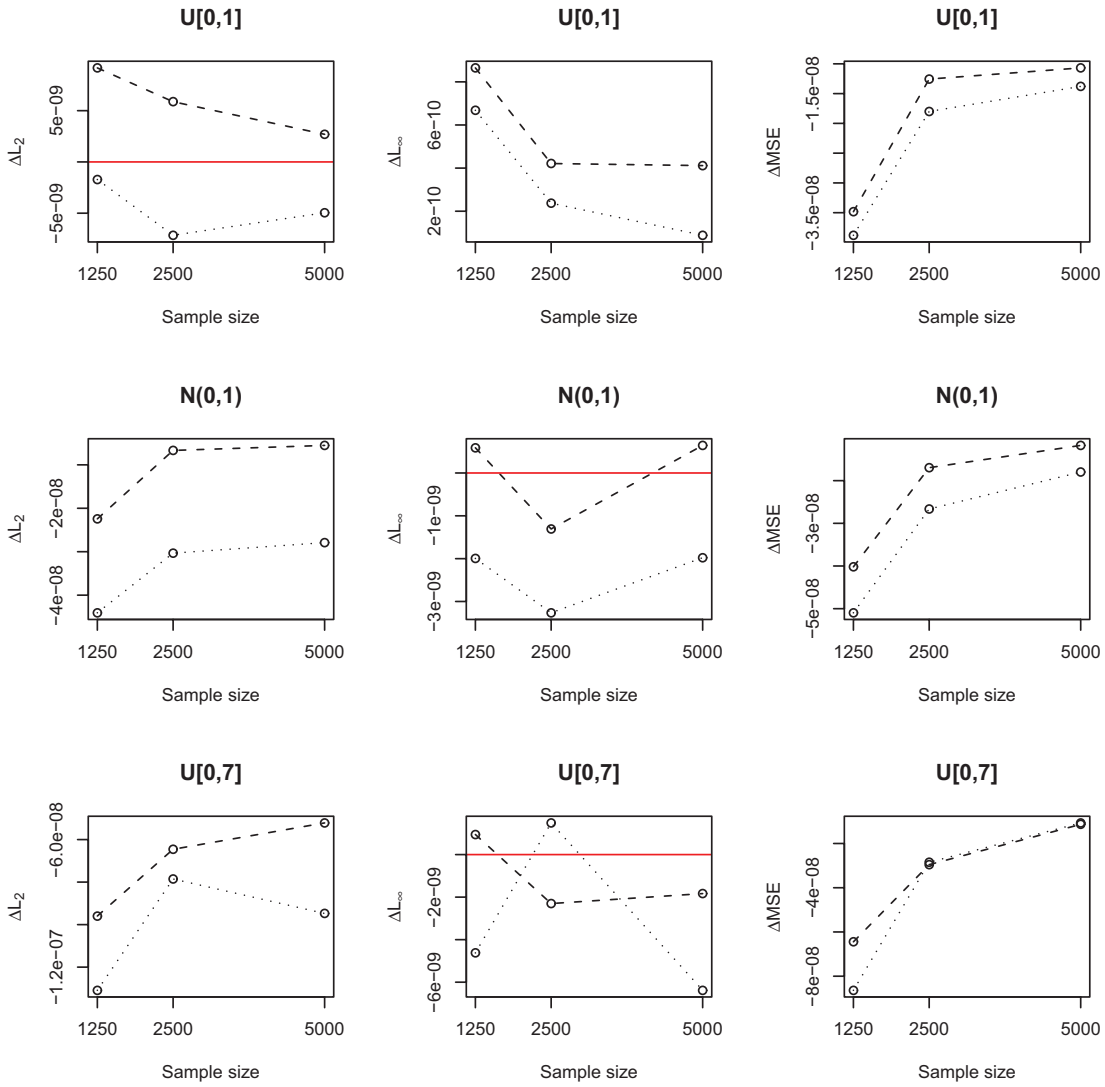

Figure 10: Quality of approximation for DFS and SQM with null vector as start value. From left to right, columns display the difference Δ between BLP and DFS (dotted line, $\cdots$) and SQM (dashed line, $--$) in the approximation error $\left\|\boldsymbol{\delta}^* - \boldsymbol{\delta}^a\right\|$ measured by L_2, L_∞ and MSE as a function of sample size taken over $R = 1000$ Monte Carlo replications for simulation scenarios (from top to bottom respectively) "good" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,1]$), "bad" ($\boldsymbol{\sigma} \sim \mathcal{N}(0,1)$), and "ugly" ($\boldsymbol{\sigma} \sim \mathcal{U}[0,7]$) with stopping rule $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^h\right\|_2 \leq \epsilon_{\text{in}}$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value.

However, the very small differences in the quality of the respective approximation metrics $\{L_2, L_\infty, \text{MSE}, \text{MSE}, \text{Var}, \text{Bias}, \text{MAE}\}$ cannot be displayed graphically. A way around this is to plot e.g. the difference $\Delta m = m^{\text{BLP}} - m^{\hat{a}}$, where $m \in \{L_2, L_\infty, \text{MSE}\}$ and with $\hat{a} \in \{\text{NEW}, \text{DFS}, \text{SQM}\}$, as in Figure 5. The horizontal line $y = 0$ in this case functions as a reference with positive values indicating that the alternative $\hat{a}$ yields a better approximation to $\boldsymbol{\delta}^*$ than BLP, and vice versa for negative values. Figure 5 clearly shows that NEW almost always produces the worst quality approximation to $\boldsymbol{\delta}^*$, while Figure 6 presents evidence of the fact that SQM nearly always produces the best approximation.

Tables 10 to 18 are similarly summarized by Figures 7 to 10. Following the same patterns as described above, these graphs primarily show the effect of choosing the null vector as a start value on the performance of NEW.

5 Numerical Analysis

In order to explain why SQM is faster and more robust in computing the solution $\boldsymbol{\delta}^*$ to the BLP Contraction Mapping, and to provide a more comprehensive and consistent comparison between the algorithms, this Section presents a two-tier numerical convergence analysis consisting of

1. an analysis of the reduction in the residual error norm $\left\| \boldsymbol{\delta}^{(h+1)} - \boldsymbol{\delta}^{(h)} \right\|$ in both the *iteration* (h) and the *time* (s) domain, see Quarteroni *et al.* (2007), Roland and Varadhan (2005), Varadhan and Roland (2008), and Atkinson and Han (2009), and

2. a *performance profile* (Dolan and Moré, 2002) of the algorithms over the $R = 1000$ Monte Carlo replications under the most stringent conditions simulated in the Monte Carlo analysis, i.e. the "ugly" scenario with $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$, start value $\boldsymbol{\delta}^0 = \mathbf{0}$ and sample size $JT = 5000$.

5.1 Convergence Analysis

This first part serves to disentangle the numerical properties for all algorithms between convergence in the iteration and convergence in the time domain, as the subsequent analysis shows that the difference between them is quite substantial. In this respect, Figure 11 shows the trajectories in both domains for each algorithm towards convergence as specified by the Dubé *et al.* (2012) stopping criterion of $\left\| \boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h} \right\| \leq \epsilon_{\text{in}} = 10^{-14}$, starting from the null vector $\boldsymbol{\delta}^0 = \mathbf{0}$, a sample size of $JT = 5000$, and the true vector of nonlinear utility $\boldsymbol{\sigma} = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$.

Thinking in terms of the number of iterations can be deceiving in ranking algorithms, as the top row of the plot displays *linear* convergence for BLP, *superlinear* convergence for DFS and SQM, and *quadratic* convergence for NEW, giving reason to believe that the latter is the more superior algorithm, followed by SQM, DFS and BLP in decreasing order of performance.[17]

[17] Starting from the true vector of nonlinear utility $\boldsymbol{\sigma}$ and the null vector $\boldsymbol{\delta}^0 = \mathbf{0}$, NEW requires a mere 9 iterations to converge, versus 21, 66 and 242 iterations for SQM, DFS and BLP respectively.

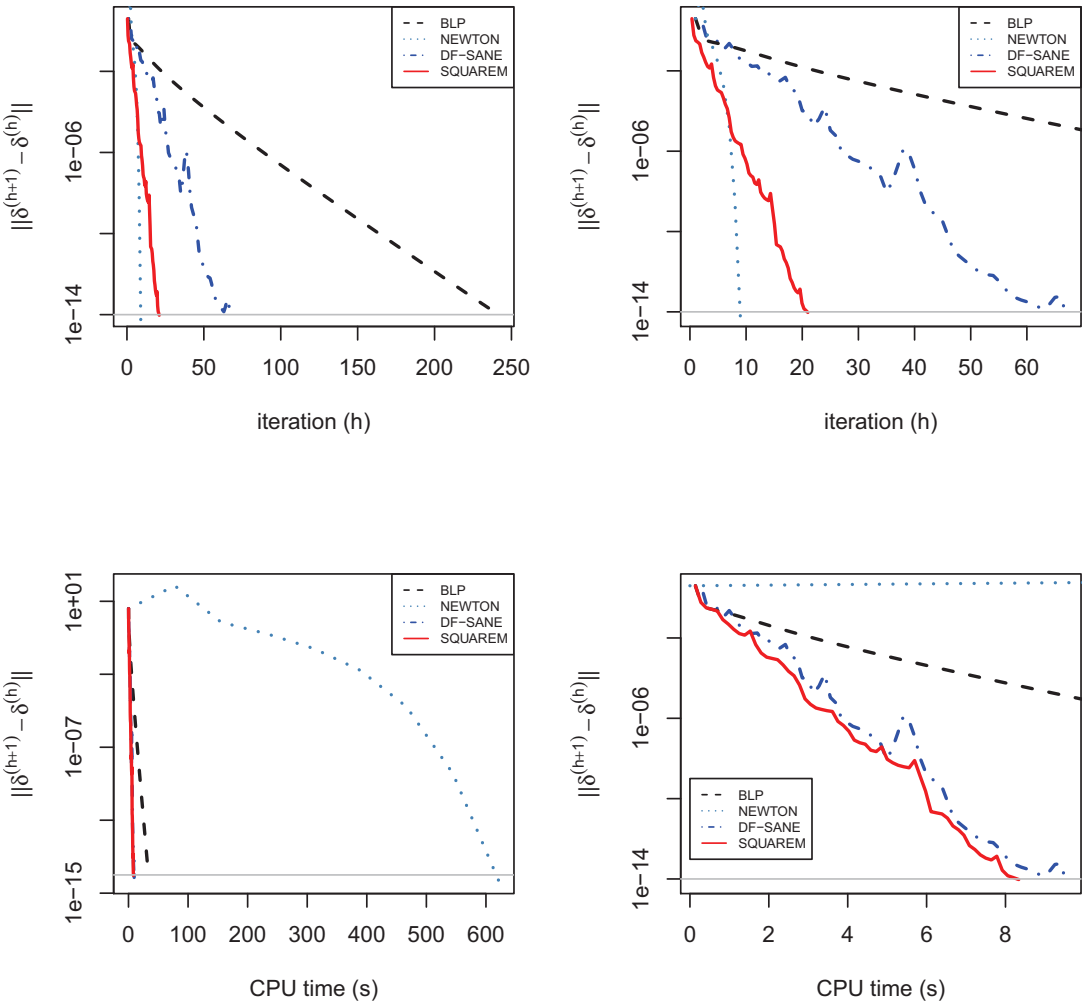

Figure 11: Convergence in the iteration (h) and time (s) domain. Top row shows convergence to the solution $\boldsymbol{\delta}^*$ of the BLP Contraction Mapping as the reduction in the residual error $\left\|\boldsymbol{\delta}^{h+1}-\boldsymbol{\delta}^{h}\right\|$ per iteration h of the respective algorithms (with the right column zooming in on the difference between DFS and SQM), while the bottom row displays convergence as the reduction in the residual error per unit of effective CPU time s. Parameters of computation: start vector $\boldsymbol{\delta}^0 = \mathbf{0}$, sample size $JT = 5000$, true value for the "nonlinear" utility vector $\boldsymbol{\sigma} = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$, and stopping criterion $\left\|\boldsymbol{\delta}^{h+1}-\boldsymbol{\delta}^{h}\right\| \leq \epsilon_{\text{in}} = 10^{-14}$.

However, if we plot convergence in the time domain, a different picture appears: preserving the pattern of quadratic convergence to the solution for NEW, the bottom row of Figure 11 displays the residual error reduction per unit of effective CPU time (in seconds), presenting the *true computational cost* of each algorithm.[18] Despite relying on an analytical Jacobian, the Newton-Raphson algorithm is hampered by the sheer size of the problem ($JT = 5000$), resulting in excessive CPU times when computing each of its 25 million entries at every iteration of the algorithm.[19]

Not having to compute first-derivative information, both DFS and SQM are better equipped to deal with large-scale nonlinear problems, as demonstrated by the substantial decrease in CPU time relative to NEW and BLP. To provide a clearer picture of the differences in performance between DFS and SQM, we also present Figures 12 and 13; in addition to displaying the nonmonotonic convergence trajectories characteristic for both algorithms, and their superior performance relative to BLP, it also shows that SQM is faster than DFS in achieving convergence under the required inner-loop convergence tolerance of 10^{-14} (Dubé *et al.*, 2012).

5.2 Performance Profile

To complete the comparison of the algorithms, the second part of our numerical analysis consists of a *performance profile* (Dolan and Moré, 2002). Stated briefly, a performance profile is a visual and statistical tool for benchmarking and comparing algorithms over a set of problems, and consists of the cumulative distribution function of a given performance metric. Denoting by $t_{p,a}$ the performance metric of algorithm a on problem $p \in \mathcal{P}$, the *performance ratio* is then defined by

$$r_{p,a} = \frac{t_{p,a}}{\min_{a \in \mathcal{A}}\{t_{p,a}\}}, \tag{24}$$

where $r_{p,a} = r_M$ if the algorithm fails to converge, and $r_M > r_{p,a}\ \forall\, p, a$. For the purposes of our analysis, given the difference between convergence in the iteration and time domain described above, we opt for the time metric as in Dolan and Moré (2002) to reflect the true computational cost.

For any given threshold $\tau \geq 1$, the overall performance of algorithm a is then given by

$$\rho_a(\tau) = \frac{1}{n_p} \#\left\{a \in \mathcal{A} : r_{p,a} \leq \tau\right\}, \tag{25}$$

where n_p is the number of problems considered, and $\#\{\cdot\}$ is the cardinality symbol. Therefore, $\rho_a(\tau)$ is the probability for algorithm $a \in \mathcal{A}$ that a performance ratio $r_{p,a}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio. The general idea is that algorithms with large probability $\rho_a(\tau)$ are to be preferred if the set of problems $\mathcal{P}$ is suitably large and representative of problems that are likely to occur in applications. Two extremes can be considered:

[18] The residual error reduction plot per function evaluation is nearly identical to Figure 13 and therefore omitted.

[19] As noted before, expressing the Jacobian as a $JT \times JT$ block-diagonal sparse matrix would reduce the computational burden of the Newton-Raphson algorithm substantially.

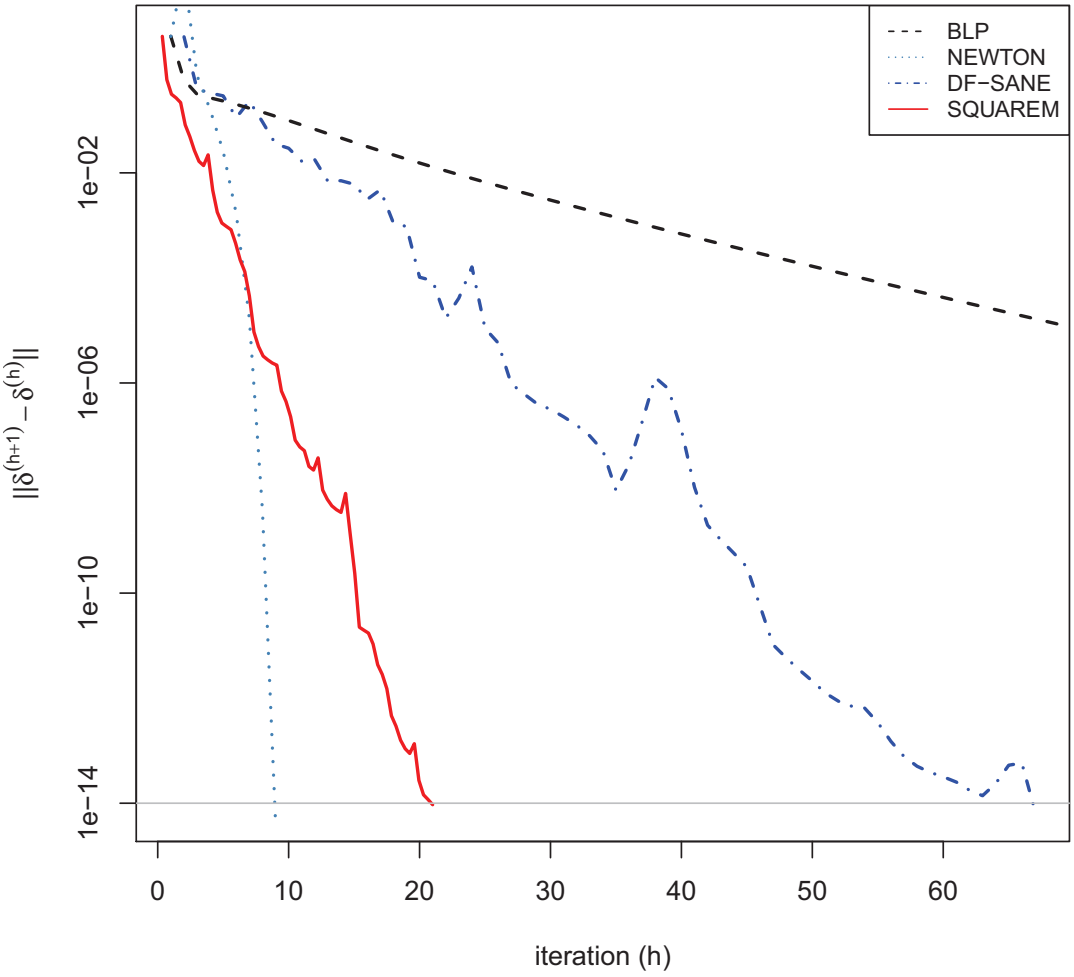

Figure 12: Convergence in the iteration (h) domain. Convergence to the solution $\boldsymbol{\delta}^*$ of the BLP Contraction Mapping is displayed as the reduction in the residual error $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|$ per iteration h of the respective algorithms, with quadratic convergence for NEW, superlinear nonmonotonic convergence for DFS and SQM, and linear convergence for BLP. Parameters of computation: start vector $\boldsymbol{\delta}^0 = \mathbf{0}$, sample size $JT = 5000$, true value for the "nonlinear" utility vector $\boldsymbol{\sigma} = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$, and stopping criterion $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\| \leq \epsilon_{\text{in}} = 10^{-14}$.

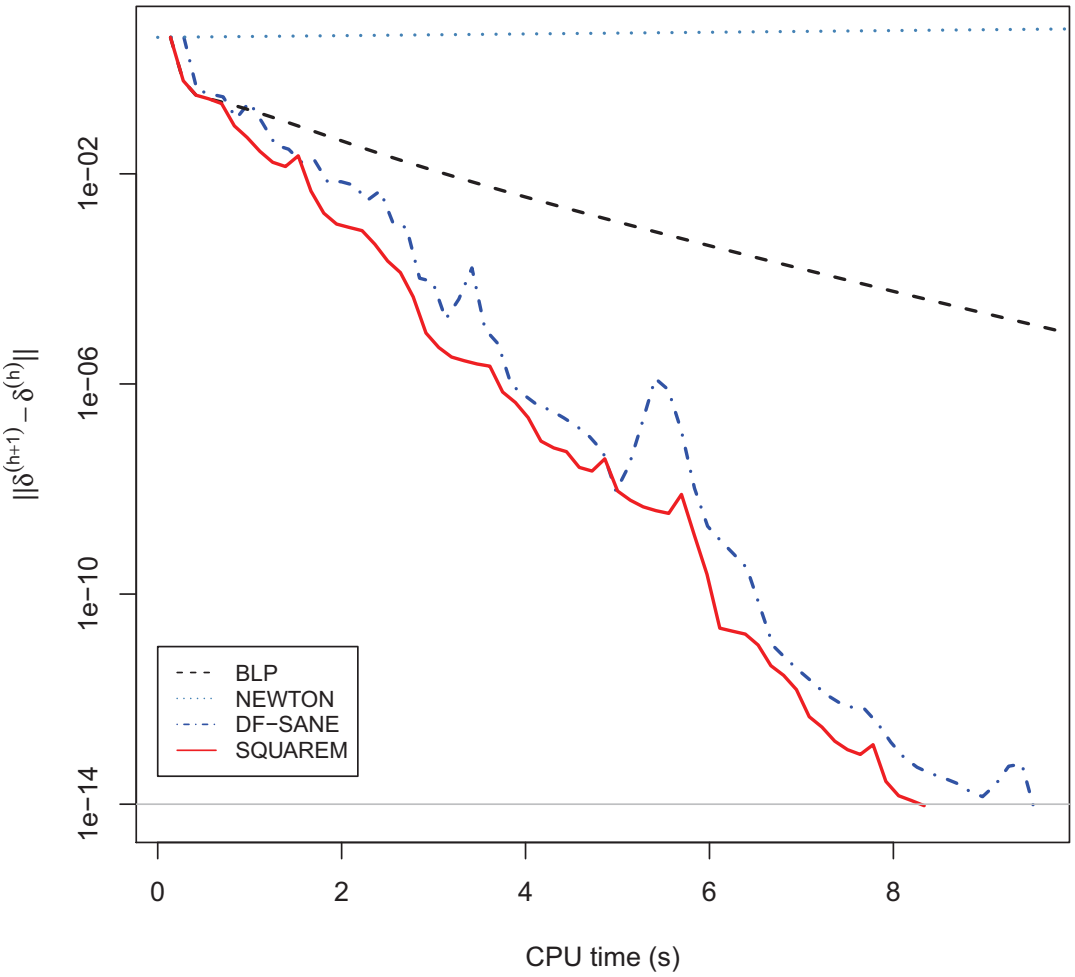

Figure 13: Convergence in the time (s) domain. Convergence to the solution $\boldsymbol{\delta}^*$ of the BLP Contraction Mapping is displayed as the reduction in the residual error $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\|$ per unit of effective CPU time s, indicating the superior performance of SQM. Parameters of computation: start vector $\boldsymbol{\delta}^0 = \mathbf{0}$, sample size $JT = 5000$, true value for the "nonlinear" utility vector $\boldsymbol{\sigma} = (\sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.5}, \sqrt{0.2})'$, and stopping criterion $\left\|\boldsymbol{\delta}^{h+1} - \boldsymbol{\delta}^{h}\right\| \leq \epsilon_{\text{in}} = 10^{-14}$.

1. $\rho_a(1)$, being the probability that a wins over all other algorithms (or the number of "wins"), and

2. $\rho_a(r_M)$, being the proportion of problems solved by a (or "robustness").

As Tables 9 and 18 show, under the most stringent conditions SQM proves the be (on average) the better algorithm over BLP both in terms of speed (up to more than five times as fast), and stability (up to 14 percentage points better success rate). However, with the exception of the `conv` success rate metric, the numbers appearing in Tables 1 to 18 are computed only over those replications in which all four algorithms converged. Although consistent, this practice biases results against more robust algorithms. Given the low numbers of success rate for NEW, DFS and BLP (respectively 10.10, 21.60 and 26.60%) in Table 18, a lot of information is lost in aggregating in this way, obfuscating the overall performance of the algorithms. Performance profiles eliminate such biases.

The results for the performance profiles of the four algorithms for the "ugly" Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0, 7]\}$ are presented in Figures 14 and 15. We take it that each Monte Carlo replication presents a problem for the algorithms to solve; the total number of problems therefore amounts to $n_p = R = 1000$. Figure 14 presents the performance profile step functions $\rho_a(\tau)$ over the first 100 Monte Carlo replications and for performance ratio's $\tau \in [1, 20]$. The following results stand out:

1. Both the robustness and superior performance of SQM in the time domain are clearly displayed by a $\rho_a(\tau)$ probability curve that lies nearly everywhere above those of the other algorithms

2. As measured by $\rho(1)$, in the "ugly" scenario, SQM is fastest in about 8% of the first 100 Monte Carlo replications, versus 10 % for DFS; BLP and NEW are never faster

3. There is a certain parameter region $\tau \in [1, 9]$ for which DFS is superior to BLP, and

4. BLP and NEW alternate for low parameter values of τ.

The general performance over the entire Monte Carlo analysis with $n_p = 1000$ and $\tau \in [1, 100]$ is given in Figure 15. From this picture it is clear that SQM has the most wins (with 9.9% vs. 8.2% for DFS), its probability curve $\rho_a(\tau)$ lies everywhere above those of its competitors, and its overall degree of robustness varies around 40%, corroborating the success rate reported in Table 18. This result also indicates that the higher average CPU time reported in Table 18 (7.08 seconds vs. 6.95 for DFS) is biased against the more robust SQM algorithm. It is also clear that BLP eventually overtakes NEW and DSF and ranks second when speed and robustness are combined.

We conclude with a performance profile for the supremum norm L_∞ as a measure of the quality of approximation. Following common practice in numerical analysis using Monte Carlo studies (Roland and Varadhan, 2005; Roland *et al.*, 2007; Varadhan and Roland, 2008), we use as a benchmark the solution $\tilde{\boldsymbol{\delta}}^*$ obtained using either BLP or SQM under the Dubé *et al.* (2012) convergence criterion of $\epsilon_{\text{in}} = 10^{-14}$, and subsequently compute the supremum norm $L_\infty = \left\| \boldsymbol{\delta}^a - \tilde{\boldsymbol{\delta}}^* \right\|_\infty$ as the performance metric $t_{p,a}$ for $a \in \mathcal{A}$

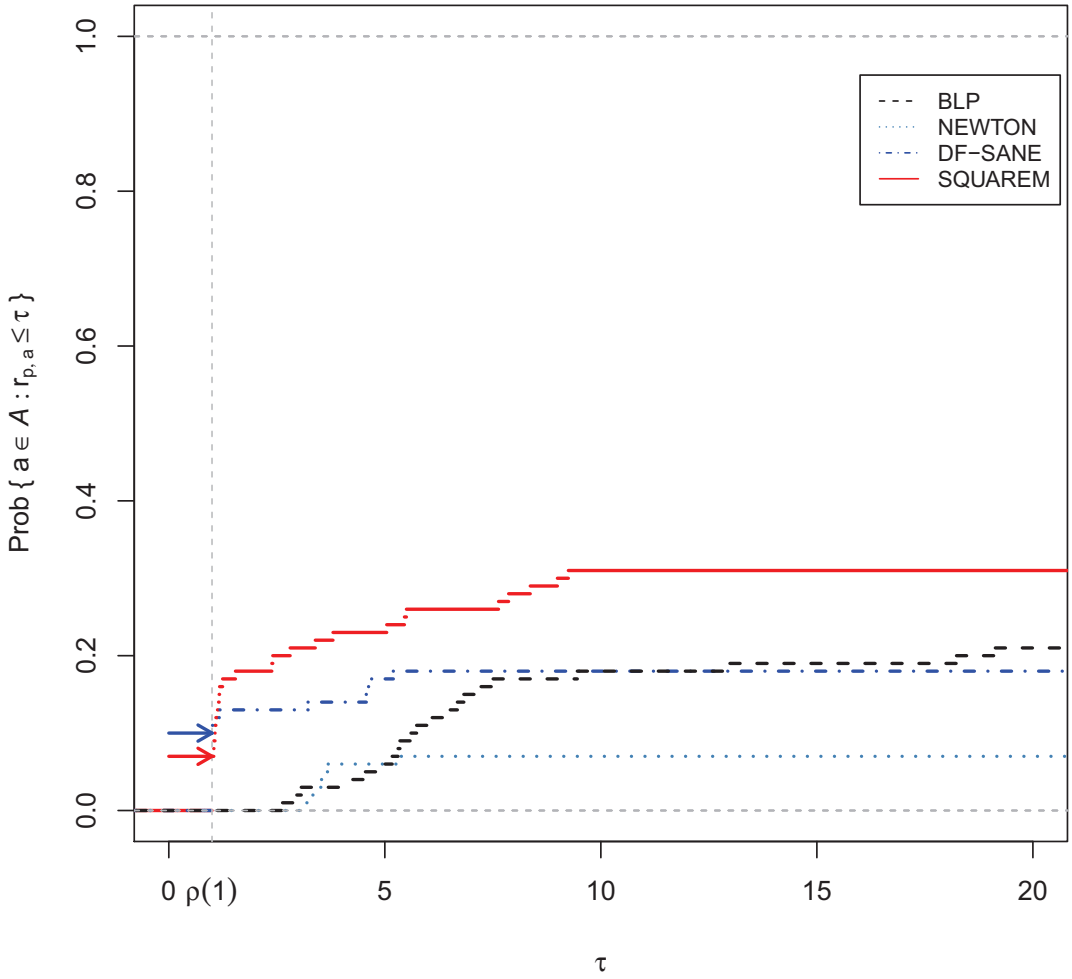

Figure 14: Performance profile $\rho_a(\tau)$ of CPU time over the first 100 replications of the $n_p = R = 1000$ Monte Carlo study for algorithms BLP, NEW, DFS, and SQM. For $\tau = 1, 2, \ldots, r_M$, the performance profile plots the proportion of problems algorithm a is able to solve within a factor τ of the best algorithm; $\rho_a(1)$ is the number of wins (indicated by arrows), and $\rho_a(r_M)$ is the measure for robustness. Parameters of computation: "ugly" scenario with $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$, sample size $JT = 5000$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value.

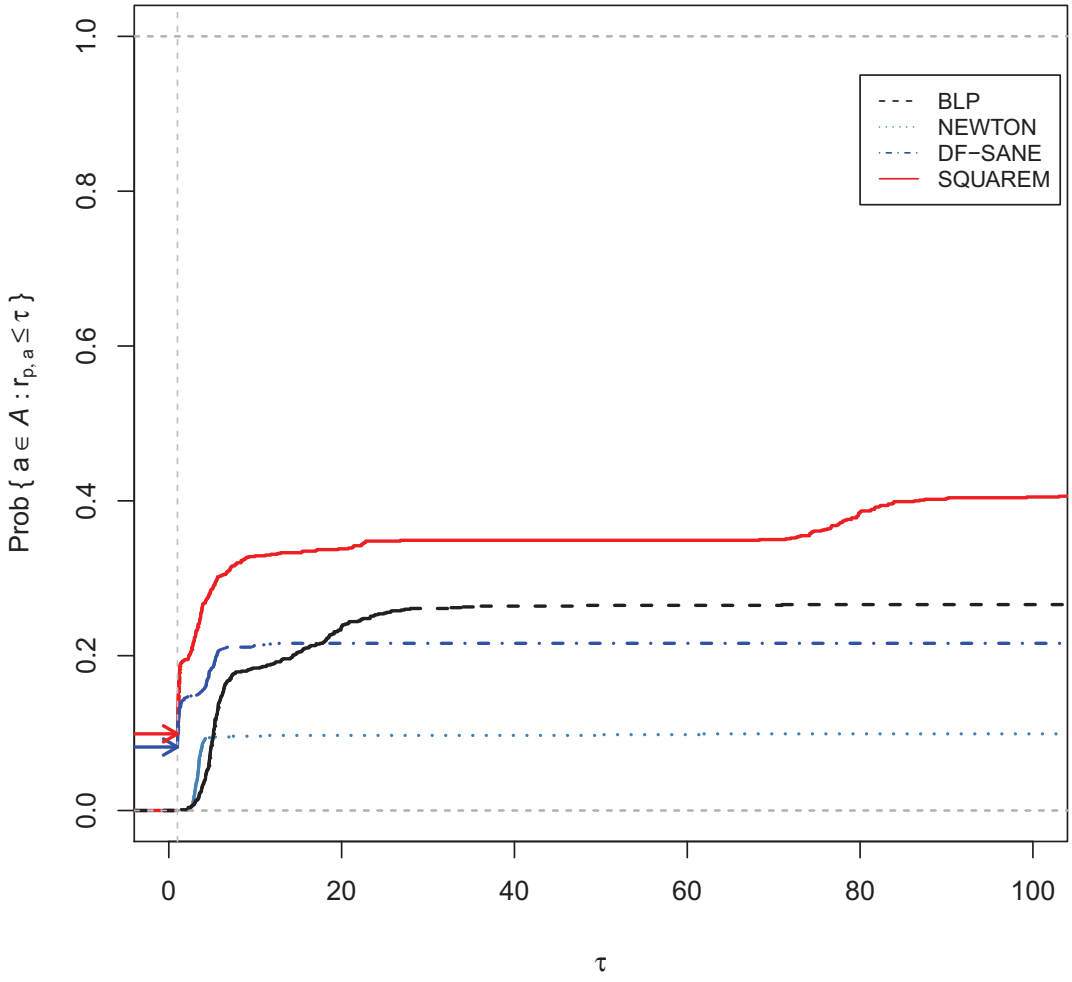

Figure 15: Performance profile $\rho_a(\tau)$ of CPU time over $n_p = R = 1000$ Monte Carlo replications for algorithms BLP BLP, NEW, DFS, and SQM. For $\tau = 1, 2, \ldots, r_M$, the performance profile plots the proportion of problems algorithm a is able to solve within a factor τ of the best algorithm; $\rho_a(1)$ is the number of wins (indicated by arrows), and $\rho_a(r_M)$ is the measure for robustness. Parameters of computation: "ugly" scenario with $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$, sample size $JT = 5000$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value.

and $p = 1, \ldots, n_p = R$. The change in benchmark combined with the robustness ingredient results in an entirely different conclusion than the one obtained from Tables 1 to 18, where BLP nearly always (on average) offered the best quality of approximation. Figures 16 (for the first 100 replications) and 17 (for the entire set of Monte Carlo replications) show clear evidence of SQM having the most wins (i.e. being closest to $\tilde{\boldsymbol{\delta}}^*$) and being the most robust algorithm (as measured by $\rho_a(r_M)$) compared with BLP, NEW and DFS. Also here BLP ranks second in terms of quality of approximation and robustness.[20]

6 Conclusion

In this paper we studied the convergence properties of the BLP (1995) Contraction Mapping in terms of CPU time, number of iterations, and success rate. As shown by a Monte Carlo study, the BLP (1995) Contraction Mapping is heavily influenced by both the sample size and the level of the inner-loop convergence tolerance. Combined with its linear rate of convergence, these properties ensure that in practice the BLP Contraction Mapping can prove to be a time-consuming procedure.

In addition, in light of recent findings with respect to the numerical properties of the BLP estimator, and the advice to tighten the inner-loop convergence tolerance to 10^{-14}, the BLP Contraction Mapping takes even more time to complete with each iteration of the outer-loop GMM minimization, and experiences substantial difficulties in achieving convergence. Proposing three alternative algorithms to accelerate the inner-loop fixed point iteration, we find that the squared polynomial extrapolation method (SQUAREM) offers significant increases in speed and stability over BLP while attaining a quality of approximation that only differs at the 7^{th}, 8^{th} or even 9^{th} decimal, and additionally outperforms the derivative-free spectral algorithm for nonlinear equations (DF-SANE). Eliminating bias against more robust algorithms arising from averaging over only those Monte Carlo replications where all four algorithms converged, a performance profile analysis of effective CPU times and quality of approximation shows that SQUAREM is both faster and delivers the best quality approximation, in addition to being more robust.

As to suggestions for further research, one item to explore might be the exploitation of the sparseness of the Jacobian of the Contraction Mapping to ensure additional increases in speed for NEW. Given that NEW only performs well for small sample sizes and in ideal conditions, and the fact that the quality of approximation is never better than either DFS and SQM however, it might be more useful to study the effect of using these alternative algorithms on the BLP estimates in the outer-loop GMM minimization using the blueprint of the Monte Carlo study documented in Section 3.

Other parameters that have not been addressed in this paper are the number of draws n_s of fictitious consumers, the total number of product characteristics K, and the number of product characteristics with random coefficients, $L \leq K$. Assessing the impact of increasing the number of draws n_s is a straightforward extension of the current Monte Carlo

[20] Over the first 100 replications, BLP, NEW, DFS and SQM have the better quality approximation in respectively 6, 1, 10 and 6% of the problems; overall, the ranking is SQM $\succ$ DFS $\succ$ BLP $\succ$ NEW with respectively 12.1, 8.3, 5.1 and 1.3% of the wins.

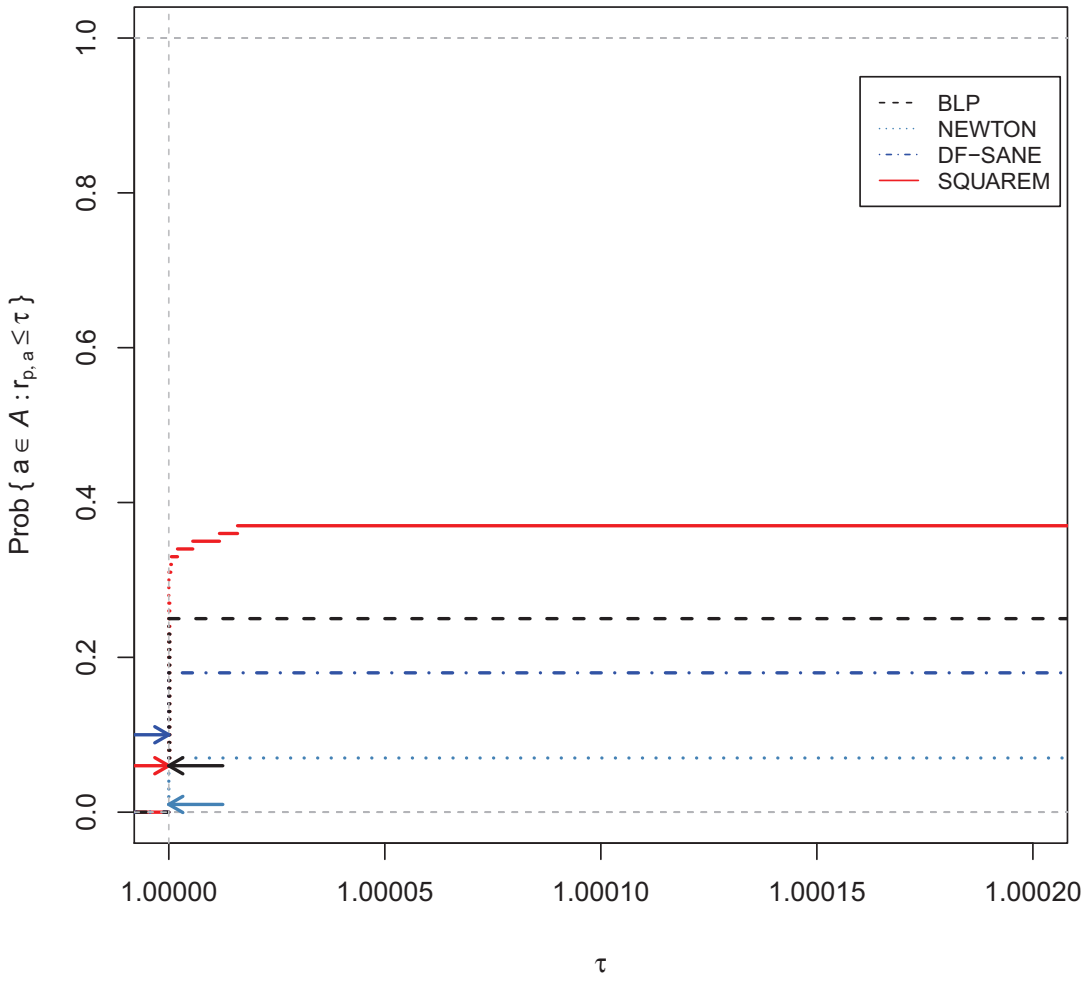

Figure 16: Performance profile $\rho_a(\tau)$ of the quality of approximation $\left\|\boldsymbol{\delta}^a - \tilde{\boldsymbol{\delta}}^*\right\|_\infty$ for the first 100 replications of the $n_p = R = 1000$ Monte Carlo replications for algorithms BLP, NEW, DFS, and SQM. For $\tau = 1, 2, \ldots, r_M$, the performance profile plots the proportion of problems algorithm a is able to solve within a factor τ of the best algorithm; $\rho_a(1)$ is the number of wins (indicated by arrows), and $\rho_a(r_M)$ is the measure for robustness. Parameters of computation: "ugly" scenario with $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$, sample size $JT = 5000$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value.

Figure 17: Performance profile $\rho_a(\tau)$ of the quality of approximation $\left\| \boldsymbol{\delta}^a - \tilde{\boldsymbol{\delta}}^* \right\|_\infty$ over $n_p = R = 1000$ Monte Carlo replications for algorithms BLP, NEW, DFS, and SQM. For $\tau = 1, 2, \ldots, r_M$, the performance profile plots the proportion of problems algorithm a is able to solve within a factor τ of the best algorithm; $\rho_a(1)$ is the number of wins (indicated by arrows), and $\rho_a(r_M)$ is the measure for robustness. Parameters of computation: "ugly" scenario with $\boldsymbol{\sigma} \sim \mathcal{U}[0, 7]$, sample size $JT = 5000$, inner-loop convergence tolerance $\epsilon_{\text{in}} = 10^{-7}$ and $\boldsymbol{\delta}^0 = \mathbf{0}$ as start value.

study, as is the influence of the variance-covariance matrix Σ_X of the matrix of observed product characteristics, in particular the value of the variances σ_{ii}^2 and covariances σ_{ij}^2. We think it will be more interesting and relevant to the current difference in approaches (acceleration vs. MPEC) to see the effect of different combinations of L and K, i.e. to explore $\binom{K}{L}$ on the convergence characteristics of the algorithms.

Computational Details

All Monte Carlo replications were run in R version 2.9.1 (R Development Core Team, 2009) on a dedicated computing server equipped with a dual Intel® Xeon® X5550 processor (2.66 GHz clock speed, 8 MB cache, 6.40 GT/s Quad CPU, Turbo, Hyper Threading with 16 virtual CPUs, with 48 GB memory for 2 CPUs, and memory type DDR3-1066 MHz).[21] The main structure of the Monte Carlo study was coded by means of a `for` loop where in each replication $r = 1, \ldots, R$ a value for $\boldsymbol{\sigma}$ was drawn from the distribution corresponding to scenarios "good," "bad," or "ugly." Due to differences in the definition of the stopping rule between BLP, NEW, and DFS on the one hand ($\left\|\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}\right\|_2 / \sqrt{JT} \leq \epsilon_{\text{in}}$) and SQM on the other ($\left\|\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}\right\|_2 \leq \epsilon_{\text{in}}$), a uniform convergence criterion for all algorithms was assured by specifying the effective convergence tolerance for the former as $\epsilon_{\text{in}}/\sqrt{JT}$.

In serial execution, the time needed to complete the 1000 Monte Carlo replications in the "ugly" scenario with sample size $JT = 5000$ varied around four weeks or more, depending on whether the null vector or analytical $\boldsymbol{\delta}$ vector was selected as a starting value for the different algorithms. In an effort to reduce this time, we experimented with various R packages for parallel computing; we found the **snow** (Tierney *et al.*, 2010) and **snowfall** (Knaus, 2010) packages to be very versatile and flexible for this type of exercise, and were able to bring down the execution time to 3.5 and 4.5 days respectively.

The parallel version of our Monte Carlo study exploits the multicore structure of the faculty server and uses all of its 16 CPUs to parallelize the 1000 replications. The main difference with respect to the serial version is that the main `for` loop was recoded as a wrapper function `MC.sim()`. Additionally, to guarantee independent replications, the **rlecuyer** package (L'Ecuyer *et al.*, 2002; Sevcikova and Rossini, 2009) was used to generate a stream of core-independent random draws for $\boldsymbol{\sigma}$ in each of the scenarios.

References

Atkinson, K. and Han, W. (2009), *Theoretical Numerical Analysis. A Functional Analysis Framework*, 3rd edn., Texts in Applied Mathematics 39, Springer, New York. [26]

Barzilai, J. and Borwein, J.M. (1988), "Two-Point Step Size Gradient Methods," *IMA Journal of Numerical Analysis* 8(1), 141–148. [10]

Berry, S. (1994), "Estimating Discrete-Choice Models of Product Differentiation," *RAND Journal of Economics* 25(2), 242–262. [13]

[21] For an overview and description of the processor's technical specifications, see `http://ark.intel.com/Product.aspx?id=37106`.

Berry, S., Levinsohn, J. and Pakes, A. (1995), "Automobile Prices in Market Equilibrium," *Econometrica* 63(4), 841–890. [1, 4, 41]

Dennis, J.E. and Schnabel, R.B. (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall Series in Computational Mathematics, Prentice Hall, Upper Saddle River, New Jersey. [7]

Dolan, E.D. and Moré, J.J. (2002), "Benchmarking Optimization Software with Performance Profiles," *Mathematical Programming Series A* 91(2), 201–213. [26, 28]

Dubé, J.P., Fox, J.T. and Su, C.L. (2012), "Improving the Numerical Performance of BLP Static and Dynamic Discrete Choice Random Coefficients Demand Estimation," *Econometrica* (forthcoming). [1, 2, 4, 15, 16, 26, 28, 31]

Gowrisankaran, G. and Rysman, M. (2009), "Dynamics of Consumer Demand for Durable Goods," *NBER Working Paper 14737*, National Bureau of Economic Research, Cambridge, MA. [4]

Grippo, L., Lampariello, F. and Lucidi, S. (1986), "A Nonmonotone Line Search Technique for Newton's Method," *SIAM Journal on Numerical Analysis* 23(4), 707–716. [10]

Hansen, L.P. (1982), "Large Sample Properties of Generalized Method of Moments Estimators," *Econometrica* 50(4), 1029–1054. [1]

Hasselman, B. (2009), *nleqslv: Solve Systems of Nonlinear Equations*, URL `http://CRAN.R-project.org/package=nleqslv`, R package version 1.5. [10]

Judd, K.L. (1998), *Numerical Methods in Economics*, The MIT Press, Cambridge, MA. [7]

Kim, K.i. and Park, M. (2010), "Approximation Error in the Nested Fixed Point Algorithm for BLP Model Estimation," *Department of Economics Working Paper*, University of Minnesota. [2]

Knaus, J. (2010), *snowfall: Easier cluster computing (based on snow)*, URL `http://CRAN.R-project.org/package=snowfall`, R package version 1.84. [37]

Knittel, C.R. and Metaxoglou, K. (2012), "Estimation of Random Coefficient Demand Models: Challenges, Difficulties and Warnings," *The Review of Economics and Statistics* (forthcoming). [2]

La Cruz, W., Martínez, J.M. and Raydan, M. (2006), "Spectral Residual Method without Gradient Information for Solving Large-Scale Nonlinear Systems of Equations," *Mathematics of Computation* 75(255), 1429–1448. [1, 10, 11]

La Cruz, W. and Raydan, M. (2003), "Nonmonotone Spectral Methods for Large-Scale Nonlinear Systems," *Optimization Methods & Software* 18(5), 583–599. [11]

L'Ecuyer, P., Simard, R., Chen, E.J. and Kelton, W.D. (2002), "An Object-Oriented Random-Number Package with Many Long Streams and Substreams," *Operations Research* 50(6), 1073–1075. [37]

Nash, J.C. (1990), *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation*, Adam Hilger, Bristol–New York. [7]

Nevo, A. (2001), "Measuring Market Power in the Ready-to-Eat Cereal Industry," *Econometrica* 69(2), 307–342. [8]

Nocedal, J. and Wright, S.J. (2006), *Numerical Optimization*, 2nd edn., Springer, New York. [7, 9]

Quarteroni, A., Sacco, R. and Saleri, F. (2007), *Numerical Mathematics*, Texts in Applied Mathematics 37, Springer-Verlag, Berlin-Heidelberg. [26]

R Development Core Team (2009), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, URL `http://www.R-project.org`, ISBN 3-900051-07-0. [37]

Raydan, M. (1997), "The Barzilai and Borwein Gradient Method for the Large Scale Unconstrained Minimization Problem," *SIAM Journal on Optimization* 7(1), 26–33. [10]

Roland, C. and Varadhan, R. (2005), "New Iterative Schemes for Nonlinear Fixed Point Problems, With Applications to Problems with Bifurcations and Incomplete-Data Problems," *Applied Numerical Mathematics* 55(2), 215–226. [11, 26, 31]

Roland, C., Varadhan, R. and Frangakis, C.E. (2007), "Squared Polynomial Extrapolation Methods With Cycling: An Application to the Positron Emission Tomography Problem," *Numerical Algorithms* 44(2), 159–172. [11, 31]

Rust, J. (1987), "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher," *Econometrica* 55(5), 999–1033. [1]

Schiraldi, P. (2011), "Automobile Replacement: A Dynamic Structural Approach," *The RAND Journal of Economics* 42(2), 226–291. [4]

Sevcikova, H. and Rossini, T. (2009), *rlecuyer: R interface to RNG with multiple streams*, URL `http://CRAN.R-project.org/package=rlecuyer`, R package version 0.3-1. [37]

Skrainka, B.S. (2011*a*), *Discussion on the Numerical Instability of the BLP Formula for Computing Predicted Market Shares*, private communication. [5]

——— (2011*b*), "Finite Sample Performance of a Popular Model of Product Differentiation," *mimeo*, CEMMAP, University College London. [5]

Skrainka, B.S. and Judd, K.L. (2011), "High Performance Quadrature Rules: How Numerical Integration Affects a Popular Model of Product Differentiation," *mimeo*, CEMMAP, University College London and Hoover Institution, Stanford University. [5]

Su, C.L. and Judd, K.L. (2008), "Constrained Optimization Approaches to Estimation of Structural Models," *Center for Mathematical Studies in Economics and Management Science Discussion Paper 1460*, Northwestern University. [4]

Tierney, L., Rossini, A.J., Li, N. and Sevcikova, H. (2010), *snow: Simple Network of Workstations*, URL `http://CRAN.R-project.org/package=snow`, r package version 0.3-3. [37]

Train, K.E. (2009), *Discrete Choice Methods with Simulation*, 2nd edn., Cambridge University Press, Cambridge, UK. [1]

Varadhan, R. (2011), *SQUAREM: Squared extrapolation methods for accelerating fixed-point iterations*, URL http://CRAN.R-project.org/package=SQUAREM, R package version 2010.12-1. [12]

Varadhan, R. and Gilbert, P. (2009), "BB: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function," *Journal of Statistical Software* 32(4), 1–26, URL http://www.jstatsoft.org/v32/i04/. [11]

Varadhan, R. and Roland, C. (2008), "Simple and Globally Convergent Methods for Accelerating the Convergence of Any EM Algorithm," *Scandinavian Journal of Statistics* 35(2), 335–353. [1, 11, 26, 31]

7 Appendix: The BLP Algorithm

7.1 General Framework

In their seminal paper Berry *et al.* (1995, BLP) propose a method to estimate random coefficients logit models of demand for differentiated products. In addition to solving the curse of dimensionality traditionally associated with estimating such demand systems, the advantages of the BLP algorithm are that (*i*) it relies only on aggregate market data, (*ii*) incorporates consumer heterogeneity by specifying the regression coefficients as random variables, and (*iii*) accounts for price endogeneity by using instruments defined at the product characteristics' level.

In the BLP model, the utility an individual i derives from consuming product $j = 1, \ldots, J$ in market $t = 1, \ldots, T$ is defined as

$$u_{ijt} = \beta_i^0 + \mathbf{x}'_{jt}\boldsymbol{\beta}_i^{\mathbf{x}} - \beta_i^p p_{jt} + \xi_{jt} + \varepsilon_{ijt}, \tag{26}$$

where p_{jt} is the good's price, $\mathbf{x}_{jt}$ a $K \times 1$ vector of product characteristics, and ξ_{jt} a scalar representing a demand shock. The latter usually incorporates unobserved (to the researcher) product characteristics (such as quality). The error term ε_{ijt} is distributed as Type I Extreme Value. The BLP model allows for consumer heterogeneity by specifying β_i^0, $\boldsymbol{\beta}_i^{\mathbf{x}}$ and β_i^p as random coefficients with distribution $F_{\boldsymbol{\beta}}(\boldsymbol{\beta}; \theta)$. The aim of the model is to parametrically estimate the parameter vectors $\boldsymbol{\beta}$ and θ, where the latter in our case is restricted to the parameter vector $\boldsymbol{\sigma}$, see below.

Following the axiom of rationality, individual i will choose product j if $u_{ijt} \geq u_{ikt} \forall k \in J$, $k \neq j$. The market share for product j therefore equals

$$s_{jt}(\mathbf{x}_t, \mathbf{p}_t, \boldsymbol{\xi}_t; \theta) = \int_{\{\boldsymbol{\beta}_i, \varepsilon_i \,|\, u_{ij} \geq u_{ik} \forall k \neq j\}} \mathrm{d}F_{\boldsymbol{\beta}}(\boldsymbol{\beta}; \theta) \mathrm{d}F_{\epsilon}(\varepsilon). \tag{27}$$

where $\mathbf{x}_t = (\mathbf{x}'_{1t}, \ldots, \mathbf{x}'_{Jt})'$ is the vector of product characteristics for products $j = 1, \ldots, J$ in market t, $\mathbf{p}_t = (p_{1t}, \ldots, p_{Jt})'$ the corresponding vector of prices, and $\boldsymbol{\xi}_t = (\xi_{1t}, \ldots, \xi_{Jt})'$ the vector of market- and brand-specific demand shocks. Because the errors are assumed to follow a logit (or Type I Extreme Value) distribution, equation (27) can be written as

$$s_{jt}(\mathbf{x}_t, \mathbf{p}_t, \boldsymbol{\xi}_t; \theta) = \int_{\boldsymbol{\beta}} \frac{\exp\left(\beta_i^0 + \mathbf{x}'_{jt}\boldsymbol{\beta}_i^{\mathbf{x}} - \beta_i^p p_{jt} + \xi_{jt}\right)}{1 + \sum_{m=1}^{J} \exp\left(\beta_i^0 + \mathbf{x}'_{mt}\boldsymbol{\beta}_i^{\mathbf{x}} - \beta_i^p p_{mt} + \xi_{mt}\right)} \mathrm{d}F_{\boldsymbol{\beta}}(\boldsymbol{\beta}; \theta), \tag{28}$$

where the "1" in the denominator of equation (28) follows from the fact that the utility u_{i0t} of the "outside" good $j = 0$ is normalized to zero, yielding $\mathrm{e}^0 = 1$.[22]

The integral in equation (28) is subsequently simulated by drawing n_s fictitious consumers from the standard normal distribution for each product characteristic (including the constant term and the price) and computed using the following "smooth simulator" (sample average, plug-in estimator)

$$\hat{s}_{jt}(\cdot; \theta) = \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{\exp\left(\beta_i^0 + \mathbf{x}'_{jt}\boldsymbol{\beta}_i^{\mathbf{x}} - \beta_i^p p_{jt} + \xi_{jt}\right)}{1 + \sum_{m=1}^{J} \exp\left(\beta_i^0 + \mathbf{x}'_{mt}\boldsymbol{\beta}_i^{\mathbf{x}} - \beta_i^p p_{mt} + \xi_{mt}\right)}, \tag{29}$$

or simply as $\hat{s}_{jt}(\cdot) = n_s^{-1} \sum_{i=1}^{n_s} \hat{s}_{ijt}(\cdot)$, where $\hat{s}_{ijt}$ is the probability that individual i chooses alternative j in market t. Berry *et al.* (1995) use a normal distribution for the random coefficients

[22] The outside good embodies the option of not buying any of the products. Including it in the model allows for predicting the total demand under changed conditions.

$\beta_i^k \sim \mathcal{N}(\beta_k, \sigma_k^2)$. The draws for the consumer $\nu^i \sim \mathcal{N}(0,1)$, $i = 1, \ldots, n_s$ are then used to construct this distribution as $\beta_i^k = \beta^k + \sigma^k \nu^i$. Hence, it is common to decompose individual utility as

$$u_{ijt} = \delta_{jt} + \mu_{ijt}, \tag{30}$$

where $\delta_{jt} = \mathbf{x}'_{jt}\boldsymbol{\beta} + \xi_{jt}$ is the "linear" part of utility, and $\mu_{ijt} = \sum_{k=1}^{K} x_{jt}^k \sigma^k \nu_i^k$ is the "nonlinear" part of utility.

7.2 Econometric Procedure

The vector of parameters $\boldsymbol{\beta}$ (which contains the means of the distributions of the random coefficients, including constant term and price) can be consistently estimated with two-stage least squares (2SLS) using an appropriate vector of instruments Z_{jt} for the price p_{jt}. This instrument vector must satisfy two important conditions:

Validity $\text{Corr}(Z_t, p_{jt}) \neq 0$, meaning that the instrument explains (some of) the variation in prices.

Orthogonality $\text{Corr}(Z_t, \xi_{jt}) = 0$, meaning that the instruments must be uncorrelated with the unobserved product quality.

The latter condition is used as a moment condition $\mathbb{E}\left[Z_t \cdot \xi_{jt}\right] = 0$ in the GMM estimation of the vector of nonlinear utility parameters $\boldsymbol{\sigma} = (\sigma^0, \sigma^1, \ldots, \sigma^K, \sigma^p)'$, i.e. the vector of standard deviations of the distributions of individual tastes for the product characteristics, estimated in the outer loop. In general, the BLP procedure to estimate $\boldsymbol{\beta}$ and $\boldsymbol{\sigma}$ consists of the following steps:

1. Using the utility decomposition (30), compute the individual choice probability for product j in market t as
$$\hat{s}_{ijt}(\boldsymbol{\delta}, \boldsymbol{\sigma}) = \frac{\exp(\delta_{jt} + \mu_{ijt})}{1 + \sum_{m=1}^{J} \exp(\delta_{mt} + \mu_{imt})}. \tag{31}$$

2. Approximate the aggregate choice probability (or market share)
$$\hat{s}_{jt}(\boldsymbol{\delta}, \boldsymbol{\sigma}) = \int \hat{s}_{ijt}(\boldsymbol{\delta}, \boldsymbol{\sigma}) \, \mathrm{d}F_{\boldsymbol{\beta}}(\boldsymbol{\beta}; \theta),$$
using the smooth simulator (29), as
$$\hat{s}_{jt}(\boldsymbol{\delta}, \boldsymbol{\sigma}) = \frac{1}{n_s} \sum_{i=1}^{n_s} \hat{s}_{ijt}(\boldsymbol{\delta}, \boldsymbol{\sigma}).$$
This yields the $J \times T$ demand system in $J \times T$ unknowns
$$\hat{s}(\boldsymbol{\delta}, \boldsymbol{\sigma}) = S,$$
where S is the vector of observed market shares.

3. Solve for $\boldsymbol{\delta}$ by inverting the demand system using the BLP Contraction Mapping (BLP):
$$\boldsymbol{\delta}(\boldsymbol{\sigma}) = \hat{s}^{-1}(S; \boldsymbol{\sigma}).$$

4. Using $\boldsymbol{\delta}(\boldsymbol{\sigma}) = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\xi}$, construct the econometric error term $\boldsymbol{\xi}(\boldsymbol{\sigma}) = \boldsymbol{\delta}(\boldsymbol{\sigma}) - \mathbf{X}\boldsymbol{\beta}$.

5. Using the matrix of instruments $\mathbf{Z}$, find a consistent estimate of $\boldsymbol{\beta}$ using 2SLS:

$$\boldsymbol{\beta}_{2\text{SLS}} = (\mathbf{X}'\mathbf{Z}\mathbf{W}\mathbf{Z}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Z}\mathbf{W}\mathbf{Z}'\boldsymbol{\delta}(\boldsymbol{\sigma}),$$

   with weight matrix $\mathbf{W} = (\mathbf{Z}'\mathbf{Z})^{-1}$.

6. Find an efficient estimate for $\boldsymbol{\beta}$ with GMM by recomputing the error term as

$$\boldsymbol{\xi}_{2\text{SLS}}(\boldsymbol{\sigma}) = \boldsymbol{\delta}(\boldsymbol{\sigma}) - \mathbf{X}\boldsymbol{\beta}_{2\text{SLS}},$$

   and construct the optimal weight matrix $\mathbf{W}^* = (\hat{\mathbf{Z}}'\hat{\mathbf{Z}})^{-1}$, where $\hat{\mathbf{Z}} = \mathbf{Z} \cdot \boldsymbol{\xi}_{2\text{SLS}}$, to find

$$\boldsymbol{\beta}_{\text{GMM}} = (\mathbf{X}'\mathbf{Z}\mathbf{W}^*\mathbf{Z}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Z}\mathbf{W}^*\mathbf{Z}'\boldsymbol{\delta}(\boldsymbol{\sigma}).$$

Step 6 concludes the estimation of the $\boldsymbol{\beta}$ (mean) vector of parameters. The estimation of $\boldsymbol{\sigma}$ involves using the following double-loop procedure:

Outer loop: Find $\boldsymbol{\sigma}$ by minimizing the GMM objective function

$$\min_{\boldsymbol{\sigma}} Q\big(\boldsymbol{\xi}(\boldsymbol{\sigma})\big) = \boldsymbol{\xi}(\boldsymbol{\sigma})'\mathbf{Z}\mathbf{W}\mathbf{Z}'\boldsymbol{\xi}(\boldsymbol{\sigma}), \tag{32}$$

   where $\boldsymbol{\xi}(\boldsymbol{\sigma})$ is computed repeatedly as in step 6 above.

Inner loop: For each value of $\boldsymbol{\sigma}$ from the outer loop, find $\boldsymbol{\delta}$ using the BLP Contraction Mapping.

Convergence is declared when the stopping rule

$$\left\| \boldsymbol{\sigma}^{h+1} - \boldsymbol{\sigma}^{h} \right\| \leq \epsilon_{\text{out}},$$

is satisfied, with $\epsilon_{\text{out}} = 10^{-6}$ the outer-loop convergence tolerance.

8 Appendix: R Code

8.1 Predicted Market Shares ("Smooth" Simulator)

```
predict.blp <- function(x, y) {

# Computes the individual choice probability \hat s_{ijt}

# INPUT
# x          : current value of delta vector
# y          : current value of sigma vector (from outer loop)

# OUTPUT
# ind.choice : matrix of individual choice probabilities for i = 1, ..., n.s
# choice     : aggregate market shares

    # Compute individual choice probability \hat s_{ijt}
    # for product j in market t by individual i = 1, ..., n.s:
    RC.in <- diag(y)
    MU.ind <- A %*% RC.in %*% V
    UT <- as.vector(x) + MU.ind
    UT <- exp(UT)
    UT2 <- UT

    for (i in 1:ncol(UT)) {
        UT2[,i] <- cumsum(UT[,i])
    }

    sum <- UT2[index, ]
    sum[2:nrow(sum),] <- diff(sum)

    # Compute \hat s_{ijt} = num/denom:
    num <- UT
    denom <- 1 + sum
    denom <- denom[rep(seq(nrow(denom)), each = J), ]
    if (any(abs(denom) < 1e-10)) {
        cat("denom failure in predict.blp()\n")
        print(denom)
        Error("denom failure")
    }

    ind.choice <- num/denom
    choice <- rowSums(ind.choice)/n.s

    if (any(is.nan(choice))) {
        cat("predict.blp() failure\n")
        cat("Numerical delta:\n")
        print(x)
        cat("Value of sigma:\n")
        print(y)
        Error("Failure in predict.blp()")
    }
    list(ind.choice, choice)
```

```
}
```

8.2 BLP (1995) Contraction Mapping

```
conmap.blp <- function(y, x.start = rep(0, J*T), tol.in = 1e-07, max.it = 1500) {

# Implements the BLP (1995) contraction mapping

# INPUT
# y           : current value of sigma
# x.start     : J*T zero vector as start value for delta (default); other
#               option is to use delta.0 (analytical delta)
# tol.in      : inner-loop convergence criterion
# max.it      : max. number of iterations consistent with SQUAREM and DF-SANE

# OUTPUT
# delta       : fixed point (FP)
# iter        : number of FP iterations
# convergence : 0 if iter < max.it, 1 otherwise

    sigma.in <- y
    delta.in <- x.start

    # Define local variables:
    obs.s <- s.jt
    it <- 0

    # Start BLP (1995) loop:
    repeat {
        cat("BLP iteration", it, "\n")
        pred.s <- predict.blp(delta.in, sigma.in)[[2]]
        delta <- delta.in + log(obs.s) - log(pred.s)
        if (sqrt(crossprod(delta - delta.in)/length(delta.in)) < tol.in) {
            break
        }
        delta.in <- delta
        it <- it + 1
        if (it > max.it) break
    }
    list(delta = delta, iter = it, convergence = as.numeric(it > max.it))
}
```

8.3 Contraction Mapping as a Nonlinear System of Equations

```
conmap.root <- function(x) {

# Reformulates the BLP (1995) Contraction Mapping
# (= fixed-point problem) as a rootfinding problem

# INPUT
# x : current value of delta vector
```

```
# OUTPUT
# g : system of nonlinear equations (vector of size J * M)

    n <- length(x)
    g <- rep(NA, n)
    obs.s <- s.jt
    sigma.in <- sigma.draw

    # Function for which we seek a root in \matbb{R}^{J*T}:
    g <- - log(obs.s) + log(predict.blp(x, sigma.in)[[2]])
    return(g)
}
```

8.4 Jacobian Matrix of BLP Contraction Mapping

```
conmap.jac <- function(x) {

# Computes the analytical Jacobian matrix for use
# in Newton-Raphson rootfinding

# INPUT
# x   : current value for delta vector
# y   : current value for sigma vector

# OUTPUT
# JAC : Jacobian matrix of BLP Contraction Mapping

    y <- sigma.draw
    JAC.in <- matrix(NA, J*T, J)
    JAC <- matrix(0, J*T, J*T)
    S.ind <- predict.blp(x, y)[[1]]
    s.pred <- predict.blp(x, y)[[2]]
    n <- 1

    for (i in 1:T) {
        T1 <- S.ind[n:index[i],]
        T2 <- T1 %*% t(T1)
        T3 <-   (diag(colSums(t(T1))) - T2)/n.s
        JAC.in[((i-1)*J + 1):(i*J),] <- T3
        n <- index[i] + 1
    }

    for (i in 1:T) {
        for (j in 1:J) {
            JAC[((i-1)*J + 1):(i*J), ((i-1)*J + 1):(i*J)] \
            <- JAC.in[((i-1)*J + 1):(i*J), ]
        }
    }
    JAC <- (1/s.pred) * JAC
}
```

8.5 Newton-Raphson Algorithm

```
NEW.blp <- function(conmap.root, x.start = rep(0, J*T), y.start,
                    tol.in = 1e-09, max.it = 1500) {

# Newton-Raphson method with analytical Jacobian
# matrix to compute numerical delta vector;
# NEW.blp() calls conmap.jac() to compute Jacobian

# INPUT
# conmap.root : function for which we seek x* s.t. F(x*) = 0
# x.start     : J*T zero vector (default) as start value for delta
#               (= root of BLP Contraction Mapping)
# y.start     : current value of sigma vector from
#               outer loop (remains fixed here)/not used in MC!
# tol.in      : inner-loop tolerance (default)
# max.it      : maximum number of iterations

# OUTPUT
# delta       : candidate solution x*
# iter        : # of NR iterations
# convergence : convergence = 0, failure = 1

    x.in <- x.start
    f.x <- conmap.root(x.in)
    it <- 0

    repeat {
        if (silent) cat("N-R iteration", it, "\n")
        A.k <- conmap.jac(x.in)
        s.k <- - solve(A.k, f.x)
        x <- x.in + s.k
        if (sqrt(crossprod(x - x.in)/length(x.in)) < tol.in) {
            break
        }
        f.x <- conmap.root(x)
        x.in <- x
        it <- it + 1
        if (it > max.it) break
    }
    list(delta = x, iter = it, convergence = as.integer(it > max.it))
}
```

8.6 BLP (1995) Single-Update Version for SQUAREM

```
blp.inner <- function(delta, sigma) {

# Computes a single update of the BLP (1995) contraction mapping.
# This single-update function is required by SQUAREM, see Varadhan and
# Roland (SJS, 2008), and Roland and Varadhan (ANM, 2005)

# INPUT
# delta : current value of delta vector
```

```
# sigma : current value of sigma vector

# OUTPUT
# delta : delta vector that equates observed with predicted market shares

    # make observed market shares a local variable
    obs.s <- s.jt

    # compute predicted shares at delta.in,
    # sigma remains fixed over iterations here!
    pred.s <- predict.blp(delta, sigma)[[2]]

    # BLP (1995) contraction mapping
    delta <- delta + log(obs.s) - log(pred.s)
    return(delta)
}
```

9 Appendix: Tables

Table 1: Monte Carlo results for scenario "good" with sample size 1250 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	103.49	4.87	30.86	31.01
`time`	2.94	1.38	0.91	0.88
`conv`	100.00	100.00	100.00	100.00
L_2	14.85319	14.85319	14.85319	14.85319
L_∞	2.414675	2.414675	2.414675	2.414675
MSE	0.5692279	0.5692279	0.5692279	0.5692279
Variance	0.1844951	0.1844951	0.1844951	0.1844951
Bias	0.008757138	0.008757135	0.008757135	0.008757136
MAE	0.2733417	0.2733417	0.2733417	0.2733417

[a] Monte Carlo scenario $\{JT = 1250, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 2: Monte Carlo results for scenario "good" with sample size 2500 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	119.21	4.88	32.98	33.38
`time`	6.42	4.00	1.84	1.79
`conv`	100.00	100.00	100.00	100.00
L_2	21.13954	21.13955	21.13955	21.13955
L_∞	2.420837	2.420837	2.420837	2.420837
MSE	0.664522	0.664522	0.664522	0.664522
Variance	0.1876329	0.1876329	0.1876329	0.1876329
Bias	0.005366165	0.005366163	0.005366163	0.005366165
MAE	0.2730573	0.2730573	0.2730573	0.2730573

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 3: Monte Carlo results for scenario "good" with sample size 5000 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	120.33	4.96	33.05	33.81
`time`	10.43	9.91	2.95	2.92
`conv`	100.00	100.00	100.00	100.00
L_2	29.92654	29.92654	29.92654	29.92654
L_∞	2.444621	2.444621	2.444621	2.444621
MSE	0.6499229	0.6499229	0.6499229	0.6499229
Variance	0.1875715	0.1875715	0.1875715	0.1875715
Bias	0.006717904	0.006717903	0.006717903	0.006717904
MAE	0.2729287	0.2729287	0.2729287	0.2729287

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 4: Monte Carlo results for scenario "bad" with sample size 1250 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	125.44	6.32	34.67	35.36
`time`	3.62	1.84	1.03	1.01
`conv`	100.00	99.80	99.90	100.00
L_2	54.70725	54.70725	54.70725	54.70725
L_∞	8.717141	8.717141	8.717141	8.717141
MSE	16.68179	16.68179	16.68179	16.68179
Variance	2.76229	2.76229	2.76229	2.76229
Bias	-0.7414045	-0.7414045	-0.7414045	-0.7414045
MAE	0.9683369	0.9683369	0.9683369	0.9683369

[a] Monte Carlo scenario $\{JT = 1250, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 5: Monte Carlo results for scenario "bad" with sample size 2500 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	140.32	8.72	37.02	37.81
`time`	7.84	6.80	2.13	2.09
`conv`	100.00	99.50	99.70	100.00
L_2	77.42671	77.42671	77.42671	77.42671
L_∞	8.823961	8.823961	8.823961	8.823961
MSE	16.78706	16.78706	16.78706	16.78706
Variance	2.816194	2.816194	2.816194	2.816194
Bias	-0.7289625	-0.7289625	-0.7289625	-0.7289625
MAE	0.9611717	0.9611717	0.9611717	0.9611717

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 6: Monte Carlo results for scenario "bad" with sample size 5000 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	142.71	6.64	37.77	38.54
`time`	16.40	16.80	4.45	4.39
`conv`	99.80	98.70	99.20	99.80
L_2	109.0806	109.0806	109.0806	109.0806
L_∞	8.800533	8.800533	8.800533	8.800533
MSE	16.76763	16.76763	16.76763	16.76763
Variance	2.77294	2.77294	2.77294	2.77294
Bias	-0.7274035	-0.7274035	-0.7274035	-0.7274035
MAE	0.9587865	0.9587865	0.9587865	0.9587865

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 7: Monte Carlo results for scenario "ugly" with sample size 1250 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	282.67	12.74	63.98	61.31
`time`	7.89	3.36	1.85	1.71
`conv`	51.80	25.40	34.90	48.00
L_2	229.272	229.272	229.272	229.272
L_∞	28.12225	28.12225	28.12225	28.12225
MSE	113.5542	113.5542	113.5542	113.5542
Variance	13.29962	13.29962	13.29962	13.29962
Bias	-3.806043	-3.806043	-3.806043	-3.806043
MAE	4.431439	4.431439	4.431439	4.431439

[a] Monte Carlo scenario $\{JT = 1250, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 8: Monte Carlo results for scenario "ugly" with sample size 2500 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	293.26	14.87	60.87	62.80
`time`	16.48	11.36	3.50	3.52
`conv`	34.70	20.70	28.00	37.10
L_2	311.0287	311.0287	311.0287	311.0287
L_∞	27.4599	27.4599	27.4599	27.4599
MSE	107.1148	107.1148	107.1148	107.1148
Variance	12.98328	12.98328	12.98328	12.98328
Bias	-3.472735	-3.472735	-3.472735	-3.472735
MAE	4.168831	4.168831	4.168831	4.168831

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 9: Monte Carlo results for scenario "ugly" with sample size 5000 and analytical delta as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	304.52	17.53	66.15	64.60
`time`	36.46	40.18	7.30	6.99
`conv`	25.80	14.20	21.40	39.70
L_2	405.9656	405.9656	405.9656	405.9656
L_∞	25.04918	25.04918	25.04918	25.04918
MSE	96.62911	96.62911	96.62911	96.62911
Variance	12.51685	12.51685	12.51685	12.51685
Bias	-3.127746	-3.127746	-3.127746	-3.127746
MAE	3.891679	3.891679	3.891679	3.891679

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \log(\hat{s}_{jt}(\cdot)/s_{0t}), \boldsymbol{\sigma} \sim \mathcal{U}[0,7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 10: Monte Carlo results for scenario "good" with sample size 1250 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	123.68	7.80	36.87	35.92
`time`	3.61	2.26	1.11	1.04
`conv`	100.00	100.00	100.00	100.00
L_2	14.85319	14.85319	14.85319	14.85319
L_∞	2.414675	2.414675	2.414675	2.414675
MSE	0.5692279	0.5692279	0.5692279	0.5692279
Variance	0.1844951	0.1844951	0.1844951	0.1844951
Bias	0.00875714	0.008757135	0.008757135	0.008757136
MAE	0.2733417	0.2733417	0.2733417	0.2733417

[a] Monte Carlo scenario $\{JT = 1250, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 11: Monte Carlo results for scenario "good" with sample size 2500 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	132.79	8.00	38.70	38.32
`time`	7.74	6.73	2.32	2.22
`conv`	100.00	100.00	100.00	100.00
L_2	21.13955	21.13955	21.13955	21.13954
L_∞	2.420837	2.420837	2.420837	2.420837
MSE	0.664522	0.664522	0.664522	0.664522
Variance	0.1876329	0.1876329	0.1876329	0.1876329
Bias	0.005366166	0.005366163	0.005366164	0.005366164
MAE	0.2730573	0.2730573	0.2730573	0.2730573

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 12: Monte Carlo results for scenario "good" with sample size 5000 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	134.80	8.00	38.91	39.07
`time`	14.83	19.05	4.41	4.27
`conv`	100.00	100.00	100.00	100.00
L_2	29.92654	29.92654	29.92654	29.92654
L_∞	2.444621	2.444621	2.444621	2.444621
MSE	0.6499229	0.6499229	0.6499229	0.6499229
Variance	0.1875715	0.1875715	0.1875715	0.1875715
Bias	0.006717905	0.006717903	0.006717903	0.006717904
MAE	0.2729287	0.2729287	0.2729287	0.2729287

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0,1]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 13: Monte Carlo results for scenario "bad" with sample size 1250 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	139.11	8.16	39.39	38.77
`time`	2.91	1.63	0.85	0.81
`conv`	100.00	97.50	99.90	100.00
L_2	53.27687	53.27687	53.27687	53.27687
L_∞	8.428415	8.428415	8.428415	8.428415
MSE	16.02431	16.02431	16.02431	16.02431
Variance	2.569361	2.569361	2.569361	2.569361
Bias	-0.7203199	-0.7203199	-0.7203199	-0.7203199
MAE	0.9453253	0.9453253	0.9453253	0.9453253

[a] Monte Carlo scenario $\{JT = 1250, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 14: Monte Carlo results for scenario "bad" with sample size 2500 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	148.25	8.14	41.30	41.19
`time`	6.20	4.69	1.78	1.72
`conv`	100.00	96.80	99.80	100.00
L_2	75.12597	75.12597	75.12597	75.12597
L_∞	8.510997	8.510997	8.510997	8.510997
MSE	16.0364	16.0364	16.0364	16.0364
Variance	2.602741	2.602741	2.602741	2.602741
Bias	-0.7048741	-0.7048741	-0.7048741	-0.7048741
MAE	0.9345858	0.9345858	0.9345858	0.9345858

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 15: Monte Carlo results for scenario "bad" with sample size 5000 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	152.09	8.19	41.82	42.01
`time`	13.85	15.63	3.92	3.81
`conv`	99.80	96.40	99.40	99.80
L_2	106.2255	106.2255	106.2255	106.2255
L_∞	8.530869	8.530869	8.530869	8.530869
MSE	16.08629	16.08629	16.08629	16.08629
Variance	2.580712	2.580712	2.580712	2.580712
Bias	-0.7068347	-0.7068347	-0.7068347	-0.7068347
MAE	0.9353301	0.9353301	0.9353301	0.9353301

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{N}(0,1)\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 16: Monte Carlo results for scenario "ugly" with sample size 1250 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	289.52	26.54	62.40	63.79
`time`	8.08	6.62	1.79	1.78
`conv`	51.50	19.30	35.20	48.30
L_2	197.3356	197.3356	197.3356	197.3356
L_∞	24.01191	24.01191	24.01191	24.01191
MSE	89.71689	89.71689	89.71689	89.71689
Variance	10.25683	10.25683	10.25683	10.25683
Bias	-3.147874	-3.147874	-3.147874	-3.147874
MAE	3.821096	3.821096	3.821096	3.821096

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0, 7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 17: Monte Carlo results for scenario "ugly" with sample size 2500 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	299.69	22.24	64.93	65.08
`time`	15.38	15.76	3.41	3.35
`conv`	34.20	15.00	28.10	36.80
L_2	265.4243	265.4243	265.4243	265.4243
L_∞	23.60009	23.60009	23.60009	23.60009
MSE	83.78952	83.78952	83.78952	83.78952
Variance	10.03398	10.03398	10.03398	10.03398
Bias	-2.797893	-2.797893	-2.797893	-2.797893
MAE	3.563659	3.563659	3.563659	3.563659

[a] Monte Carlo scenario $\{JT = 2500, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0, 7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP}, \text{NEW}, \text{DFS}, \text{SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.

Table 18: Monte Carlo results for scenario "ugly" with sample size 5000 and null vector as start value[a]

	BLP	NEW	DFS	SQM
`iter`[b]	310.24	26.09	63.37	66.04
`time`	36.76	57.06	6.95	7.08
`conv`	26.60	10.10	21.60	40.60
L_2	318.0137	318.0137	318.0137	318.0137
L_∞	20.35777	20.35777	20.35777	20.35777
MSE	64.66444	64.66444	64.66444	64.66444
Variance	7.918365	7.918365	7.918365	7.918365
Bias	-2.086484	-2.086484	-2.086484	-2.086484
MAE	3.052407	3.052407	3.052407	3.052407

[a] Monte Carlo scenario $\{JT = 5000, \boldsymbol{\delta}^0 = \mathbf{0}, \boldsymbol{\sigma} \sim \mathcal{U}[0, 7]\}$ with $R = 1000$ replications, inner-loop tolerance $\epsilon_{\text{in}} = 10^{-7}$ and stopping rule $||\boldsymbol{\delta}^{a,h+1} - \boldsymbol{\delta}^{a,h}||_2 \leq \epsilon_{\text{in}}$ for $a \in \{\text{BLP, NEW, DFS, SQM}\}$.

[b] Legend: with the exception of `conv`, all entries represent the average over replications where all four algorithms converged; `iter` and `time` respectively are the number of iterations and CPU time taken until convergence, `conv` is the percentage of replications in which algorithm a successfully converged (failure to converge was taken to be either the inability to compute a numerical approximation to the "true" mean utility vector $\boldsymbol{\delta}^*$, or failure to do so within the maximum number of iterations allowed, `max.it` $= 1500$); L_2 is the Euclidean distance, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_2 = ((\boldsymbol{\delta}^a - \boldsymbol{\delta}^*)'(\boldsymbol{\delta}^a - \boldsymbol{\delta}^*))^{1/2}$ and L_∞ denotes the supremum norm, $||\boldsymbol{\delta}^a - \boldsymbol{\delta}^*||_\infty = \max_{i \in JT} |\boldsymbol{\delta}_i^a - \boldsymbol{\delta}_i^*|$.