L. Judd, Kenneth; Skrainka, Ben

Working Paper

High performance quadrature rules: How numerical integration affects a popular model of product differentiation

cemmap working paper, No. CWP03/11

Provided in Cooperation with:
The Institute for Fiscal Studies (IFS), London

Suggested Citation: L. Judd, Kenneth; Skrainka, Ben (2011) : High performance quadrature rules: How numerical integration affects a popular model of product differentiation, cemmap working paper, No. CWP03/11, Centre for Microdata Methods and Practice (cemmap), London, https://doi.org/10.1920/wp.cem.2011.0311

This Version is available at:
https://hdl.handle.net/10419/64785

cemmap
centre for microdata methods and practice

High performance quadrature rules: how numerical integration affects a popular model of product differentiation

Kenneth L. Judd
Benjamin S. Skrainka

The Institute for Fiscal Studies
Department of Economics, UCL

cemmap working paper CWP03/11

E·S·R·C ECONOMIC & SOCIAL RESEARCH COUNCIL
An ESRC Research Centre

High Performance Quadrature Rules: How Numerical Integration Affects a Popular Model of Product Differentiation

Benjamin S. Skrainka*
University College London
cemmap
b.skrainka@ucl.ac.uk

Kenneth L. Judd
Hoover Institution
NBER
judd@hoover.stanford.edu

February 1, 2011

Abstract

Efficient, accurate, multi-dimensional, numerical integration has become an important tool for approximating the integrals which arise in modern economic models built on unobserved heterogeneity, incomplete information, and uncertainty. This paper demonstrates that polynomial-based rules out-perform number-theoretic quadrature (Monte Carlo) rules both in terms of efficiency and accuracy. To show the impact a quadrature method can have on results, we examine the performance of these rules in the context of Berry, Levinsohn, and Pakes (1995)'s model of product differentiation, where Monte Carlo methods introduce considerable numerical error and instability into the computations. These problems include inaccurate point estimates, excessively tight standard errors, instability of the inner loop 'contraction' mapping for inverting market shares, and poor convergence of several state of the art solvers when computing point estimates. Both monomial rules and sparse grid methods lack these problems and provide a more accurate, cheaper method for quadrature. Finally, we demonstrate how researchers can easily utilize high quality, high dimensional quadrature rules in their own work.

Keywords: Numerical Integration, Monomial Rules, Gauss-Hermite Quadrature, Sparse Grid Integration, Monte Carlo Integration, pseudo-Monte Carlo, Product Differentiation, Econometrics, Random Coefficients, Discrete Choice.

*This paper would not have been possible without the help of Che-Lin Su and JP Dubé who shared their knowledge of BLP and their MATLAB^{TM}code. The authors were generously supported and hosted by Ian Foster and the Computation Institute at The University of Chicago and Argonne National Laboratory during Fall 2009. Benjamin S. Skrainka also gratefully acknowledges the financial support of the UK Economic and Social Research Council through a grant (RES-589-28-0001) to the ESRC Centre for Microdata Methods and Practice (CeMMAP). Please address questions or comments to b.skrainka@ucl.ac.uk.

1 Introduction

Efficient, accurate, multi-dimensional, numerical integration has become an important tool for approximating the integrals which arise in modern economic models built on unobserved heterogeneity, incomplete information, and uncertainty. Failure to compute these integrals quickly and accurately can prevent a problem from being numerically stable or computationally tractable, especially in higher dimensions. In this paper, we show that the twin goals of computational efficiency and accuracy can be achieved by using monomial rules instead of simulation and that these rules work well even in multiple dimensions. We support these claims by comparing monomial[1] rules to several popular methods of numerical integration – both polynomial-based (sparse grid integration (SGI), and Gaussian product rules) and Monte Carlo – and show that monomial rules are both more accurate and often an order of magnitude cheaper to compute for a variety of integrands, including low and high order polynomials as well as the market share integrals in Berry, Levinsohn, and Pakes (1995)'s 'industry standard' model of product differentiation (BLP hereafter). Furthermore, we also demonstrate how the use of Monte Carlo integration introduces numerical error and instability into the BLP model whereas polynomial-based methods do not. These numerical issues include inaccurate market share integrals, artificially small standard errors, instability of the inner loop 'contraction' mapping for inverting market shares, and the convergence of even state of the art solvers when computing point estimates.

A good quadrature rule delivers high accuracy at low computational cost. High accuracy comes from either using more points and/or choosing those points more cleverly. Cost depends on minimizing evaluations of the integrand – i.e. minimizing the number of nodes. A good numerical approximation to an integral should minimize the number of nodes while sacrificing as little accuracy as possible.[2] Fortunately, researchers now have access to a variety of high performance quadrature[3] methods – many of which have been available since the 1970s (Stroud, 1971) – one or more of which should suit the problem at hand. Monte Carlo methods are the primary option for very high dimensional problems, but for many multi-dimensional problems, the analyst can often obtain a cheaper, more accurate approximation by choosing a polynomial-based quadrature rule, such as monomial rules or sparse grid integration, – even for ten, 15, or more dimensions.[4] Because most integrals in economics are analytic, polynomial-based

[1] Monomials are the simplest possible basis for multidimensional polynomials. Each basis function is simply a product of the coordinates, each raised to some power. E.g., $x_1^3 x_2^2 x_5^1$. A formal definition follows below on page 10.

[2] We have also found that using a good quadrature rule in the computation of an objective function can significantly decrease the number of iterations a solver needs to converge to an optimum.

[3] Some authors (e.g. Cools, 2002) use quadrature to refer to one dimensional integrals and cubature to refer to integrals of dimension ≥ 2. We will always use quadrature to refer to any integration rule, regardless of the dimension.

[4] What actually constitutes a high dimensional problem will depend on the computing resources and numerically properties of the integral.

methods should provide accurate, efficient numerical approximations.

Our paper builds on Heiss and Winschel (2008) which showed that sparse grid integration outperforms simulation for likelihood estimation of a mixed logit model of multiple alternatives. However, we make several new contributions including that simulation introduces false local optima and excessively tight standard errors; that polynomial-based rules approximate both the level and derivatives of integrals better than pMC; that quadrature rules affect solver convergence; that polynomial rules outperform Monte Carlo for low to moderate degree monomials but both are poor for higher degrees; and, that monomial rules provide a lower cost alternative to SGI.

We illustrate how to use modern quadrature rules in the context of Berry, Levinsohn, and Pakes (1995)'s 'industry standard' model of product differentiation (BLP hereafter). Their paper develops an innovative method for studying both vertical and horizontal aspects of product differentiation by using a random coefficients multinomial logit with unobserved product-market characteristics.[5] But, the results of their model depend heavily on the numerical techniques used to approximate the market share integrals: any errors in computing these integrals – and, more importantly, the gradient of the the GMM objective function – have far reaching consequences, rippling through the model, affecting the point estimates, the standard errors, the convergence of the inner loop mapping, and even the convergence of the solver used to compute the GMM parameter estimates. To substantiate these claims, we generate multiple synthetic[6] data sets and then compute the product-market share integrals, $s_{jt}(\hat{\theta})$, at a variety of parameter values, $\hat{\theta}$, for several popular integration rules (Gaussian-Hermite product rule, monomial rule, sparse grids and pseudo-Monte Carlo). Although, the original BLP papers use importance sampling (Berry, Levinsohn, and Pakes, 1995, 2004), an informal survey of the BLP literature shows, with few exceptions (Conlon, 2010), most BLP practitioners (Nevo, 2000a,b, 2001) use pseudo-Monte Carlo integration without any variance reduction methods. Thus, errors from inaccurate (pMC[7]) quadrature rules could potentially affect much of the BLP literature.

In our integration experiments, monomial rules are superior to Monte Carlo integration. Sparse grid integration is also much more accurate and efficient than pMC. These rules are easy to implement and apply to a wide range of problems of moderate size. The benefits are obvious: more accurate computation at a lower computational burden, both crucial for obtaining convincing point estimates and standard errors.

In addition, the failure of a state-of-the-art solver such as KNITRO (Byrd, Nocedal, and Waltz, 2006) or SNOPT (Gill, Murray, and Saunders, 2002) often means that the Hessian is ill-conditioned or that a problem is numerically

[5] Another contribution of their paper is a GMM-based, nested fixed point algorithm to estimate the model.

[6] We will clarify exactly what 'pseudo-Monte Carlo' means below.

[7] We often refer to Monte Carlo rules as pseudo-Monte Carlo or pMC because of the pseudo random numbers used to generate these nodes. Quasi-Monte Carlo is an alternative, number-theoretic method. See 2.2 and Judd (1998).

unstable. The former usually indicates that a model is not well identified (numerically) because the mapping from data to parameters is nearly singular. Using an inferior quadrature rule, such as Monte-Carlo, can mask these problems because the noisiness of pMC creates false local basins of attraction where the solver converges, creating incorrect point estimates and excessively small standard errors. By combining a high-quality solver and quadrature rule a researcher can get early feedback that identification problems may be present.

Furthermore, the logit-class of models is prone to numerical instability because the exponential function quickly becomes large. Consequently, poorly implemented code will suffer from a variety of floating point exceptions, including overflow, underflow, and NaNs[8]. Typically, these problems will cause a good solver to abort. However, these problems will tempt some researchers to try different draws until they find a good set which avoids the problem regions of parameter space instead of addressing the underlying problem. Better to identify the source of the problem and correct it with robust code and proper box constraints for the solver.

We begin the paper by surveying current best practice for numerical integration, explaining the strengths and weaknesses of several popular methods for computing multi-dimensional integrals. In addition, we compute several metrics to illustrate the superiority of polynomial-based quadrature rules to simulation. Next, we briefly review the BLP model of product differentiation to establish a basis to understand how it performs under different quadrature rules. Then, we estimate the BLP model using monomial, sparse grid integration, and Monte Carlo methods. We examine the point estimates, standard errors, and convergence properties of the under these different rules to demonstrate that monomial rules provide correct point estimates and standard errors while simulation methods introduce false local optima into the GMM objective function, leading to incorrect point estimates and standard errors. Finally, we conclude.

2 Multi-Dimensional Numerical Integration

For more than four decades, researchers have had access to well-understood rules to compute multi-dimensional integrals on a variety of domains accurately and efficiently (Stroud, 1971). All quadrature methods approximate an integral as a weighted sum of the integrand evaluated at a finite set of well-specified points called nodes. I.e., a quadrature method approximates the integral

$$I[f] := \int_{\Omega} w(x) f(x) dx, \ \Omega \subset \mathbb{R}^d, \ w(x) \geq 0 \, \forall x \in \Omega$$

as

[8] NaN is computer-speak for 'not a number' and indicates that a floating point computation produced an undefined or unrepresented value such as ∞/∞, $\infty \cdot 0$, and $\infty - \infty$. NaNs are part of the IEEE-754 floating point standard. What happens when a program generates a NaN depends on the platform, typically either the process receives a signal to abort or the operating system silently handles the floating point exception and the computation produces the special floating point value NaN.

$$Q[f] := \sum_{k=1}^{R} w_k f(y_k), \; y_k \in \Omega,$$

where $w(x)$ is the weight function such as 1, $\exp(-x)$, or $\exp\left(-x^{'}x\right)$ depending on the problem. The region of integration, Ω, is also problem dependent. And, $\{w_k\}$ and $\{y_k\}$ are the quadrature weights and nodes, respectively. R refers to the number of nodes (or draws) and N to the number of replications. Thus, $N = 100$ and $R = 1,500$ means that we computed the integral 100 times using $1,500$ different draws each time.[9]

For example, a simple Monte Carlo rule would set $w_k = 1/R, \forall k$, and draw y_k from a suitable probability distribution, namely $w(x)$. Another common example is the mixed, or random-coefficients, logit with $\Omega = \mathbb{R}^d$, $f(x)$ the multinomial logit for some taste coefficient, and $w(x) = \exp\left(-x^{'}x\right)$. Then, assuming a pMC rule, y_k is drawn from the (Normal) distribution for the coefficients.[10]

The art of numerical integration lies in choosing these nodes and weights strategically so that the approximation achieves the desired accuracy with few points and, thus, minimal computational expense. A solution is said to be exact when $I[f] = Q[f]$: i.e., the approximation has no error. Many rules give an exact result for all polynomials below a certain degree. Because polynomials span the vector space of 'well-behaved' functions, any integral of a function which is smooth and differentiable – or better yet analytic – should have a good numerical approximation. More often, though, the approximation will not be exact. The quality of the approximation also depends on other properties of the integrand such as the presence of sharp peaks, kinks, high frequency oscillations, high curvature, symmetry, and the thickness of the tails all of which often lead to non-vanishing, high order terms in a Taylor series expansion. A good approximation, as well as minimizing error and the number of (expensive) function evaluations, should converge to the true value of the integral as the number of nodes goes to infinity Stroud (1971).

The two primary methods for choosing the quadrature nodes and weights are number theoretic and polynomial-based methods (Cools, 2002). The former refers to Monte Carlo (or simulation) methods whereas the later includes product rules based on the Gaussian quadrature family of methods as well as monomial rules and sparse grid integration. In general, polynomial-based methods are both more efficient and more accurate. Heiss and Winschel (2008) warn that polynomial-based methods poorly approximate functions with large flat

[9] This notation is based on Cools (2002).

[10] If you are integrating over a normal density $\tilde{w}(u) = (2\pi|\Omega|)^{-\frac{d}{2}} \exp\left(-\frac{1}{2}u^T\Sigma^{-1}u\right)$, you must perform a change of variables using the Cholesky decomposition $CC^{'} = 2\Sigma$ so $x = C^{-1}u$ produces yields the Gaussian weighting function $w(x) = \exp(-x'x)$. The convenience of this form becomes clear once you have a set of quadrature nodes $\{y_j\}$ and need to transform them for a specific problem. See 2.1.

regions or sharp peaks, and, by extension, regions with high frequency oscillations. The later two problems are also likely to affect MC methods as we show in 2.4. In the BLP example below, monomial rules have no trouble in the tails because of the Gaussian kernel. However, for very high dimensional integration MC rules may be the only option because the 'curse of dimensionality' makes even the most efficient polynomial rule intractable. MC methods can also be superior when integrating over irregularly shaped regions, unless there is a clever variable transform.[11] If the integrand has kinks, jumps, or other singularities more work is usually required, such as performing separate integrations on the different sides of the kink or using an adaptive rule. But, many economic applications have ten or fewer dimensions and well-behaved integrands (analytic, smooth, and bounded), making these problems well-suited for monomial rules.

2.1 A One-Dimensional Example

To illustrate these issues, consider a one-dimensional random coefficients multinomial logit (MNL) model. An agent i chooses the alternative $j \in J$ which yields the highest utility $U_{ij} = \alpha_i (\log y_i - \log p_j) + z_j^T \beta + \epsilon_{ij}$, where ϵ_{ij} follows a Type 1 Extreme Value distribution and the taste shock is a one dimensional random coefficient on price, $\alpha_i \sim N(\alpha, \sigma^2)$. Because of the distributional assumption on ϵ_{ij}, the market shares conditional on type α_i are[12]

$$s_{ij}(\alpha_i) = \frac{\exp\left[-\alpha_i \log p_j + z_j^T \beta\right]}{\sum_k \exp\left[-\alpha_i \log p_k + z_k^T \beta\right]}$$

(See Train (2009) for details.). Consequently, the total market share of good j is the just the expectation of the conditional market share integral for j:

$$\begin{aligned} s_j &= \int_\Omega s_{ij}(\alpha_i) f(\alpha_i) d\alpha_i \\ &= \int_{-\infty}^{\infty} s_{ij}(\alpha_i) \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}[\alpha_i - \bar{\alpha}]^2\right) d\alpha_i \\ &= \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} s_{ij}\left(\sqrt{2}\sigma u\right) \exp\left(-u^2\right) du \\ &\approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{R} w_k s_{ij}\left(\sqrt{2}\sigma y_k\right). \end{aligned}$$

[11] Genz (1993) uses a clever transformation to converted a bounded probit into integration over a hypercube.

[12] Note that the $\log(y_i)$ term cancels out of the market share expression, because of the well-known property of logit distributions that individual-specific characteristics drop out unless they are interacted with choice-specific attributes.

$\{w_k\}$ and $\{y_k\}$ are the R weights and nodes for a Gauss-Hermite quadrature rule. I performed a one-dimensional 'Cholesky' transformation to convert from the economic problem to the mathematical formula. We chose the Gauss-Hermite rule because it has the appropriate weighting function, $\exp\left(-x^2\right)$, and integration region, $\mathbb{R}$. This choice causes the Normal density to disappear from the sum used to approximate the integral.

In the following two sections, we survey the two main types of rules: Monte Carlo and polynomial-based.

2.2 Monte Carlo Integration

Monte Carlo integration is one of the most popular choices for numerical integration because it is easy to compute and conceptually simple.[13] This method computes the integral by taking draws from an appropriate distribution and may includes other techniques to increase accuracy and speed, such as importance sampling, Halton draws, and antithetic draws (Train, 2009). In its simplest form, simulation weights all nodes equally by setting the weights $\omega_k = {}^1/_R$, where $R = |\{y_k\}|$, and the nodes are drawn from a suitable distribution. The weight function is set to ${}^1/_R$ because the draws come from the corresponding distribution. Consequently, simulation is easy to implement and also works over irregular-shaped regions or with functions which are not smooth, even if MC methods do not always produce the most accurate approximations.

The Law of Large Numbers is used to justify MC rules: draw enough points and the result must converge to the 'truth' without bias. Unfortunately, accuracy only improves as $\sqrt{R}$ – so the number of nodes must be increased by a factor of 100 for each additional digit of accuracy. Consequently, a more sophisticated quadrature rule will usually outperform Monte Carlo for moderate-sized problems because adding well-chosen nodes improves the integral approximation more quickly than the same number of randomly-chosen points. In practice, Monte Carlo draws are generated using an algorithm for generating apparently random numbers, such as Mersenne twister (Matsumoto and Nishimura, 1998), which can pass the statistical tests associated with random numbers. These numbers are known as pseudo random and the corresponding Monte Carlo method is know as pseudo-Monte Carlo (pMC) integration. Because pseudo-random numbers are not truly random, the Law of Large Numbers only applies to theoretical discussions of MC methods based on true random numbers, not the pseudo-random implementations commonly used for numerical integration. Thus, researchers should be wary of using proofs which only hold for true random numbers and not for pseudo random numbers. A poor random number generator can compromise results. See Judd (1998) for further discussion of the potential pitfalls.

More sophisticated methods of taking draws – quasi-Monte Carlo methods, importance sampling, and antithetic draws – remedy some of the deficiencies of

[13] Certainly, it appears simple until faced with the implementation of a good source of 'random' numbers.

simple pMC. quasi-Monte Carlo (qMC) rules use a non-random algorithm which will not pass all of the statistical tests of randomness but instead provides better coverage of parameter space by constructing equidistributed nodes, resulting convergence which is often much faster than pMC methods. The weights, as in the case of pMC, are $w_j = 1/R$. In an earlier draft, we used a qMC quadrature rule with Niederreiter sequences[14] to estimate the BLP model. In theory, it should considerably out-performed a pMC rule. We chose a Niederreiter rule which produces good results for a variety of problems while retaining the simplicity of pMC rules (Judd, 1998). In practice, we found that using even $5,000$ nodes for the 5 dimensional integrals we consider below, qMC was not a significant improvement on pMC. Consequently, we do not discuss qMC further. Nevertheless, qMC is easy to implement and performs at least as well as pMC.

Another common mistake is to use the same set of draws for each integral. For example, in BLP, there are $J \times T$ market share integrals, where J is the number of products per market and T is the number of markets. Instead of taking $J \times T$ sets of draws ($J \times T \times R$ total draws), most researchers take only R draws and use the same R draws for each of the $J \times T$ integrals. By taking a new set of draws for each integral simulation errors will cancel to some extent and can considerably improve the quality of the point estimates because the individual integrals are no longer correlated (McFadden, 1989).

In summary, the basic problems of simulation remain regardless of the simulation rule: it is dirty and can produce inaccurate results, as Berry, Levinsohn, and Pakes (1995) point out:

> ... we are concerned about the variance due to simulation error. Section 6 develops variance reduction techniques that enable us to use relatively efficient simulation techniques for our problem. Even so, we found that with a reasonable number of simulation draws the contribution of the simulation error to the variance in our estimates (V3) is not negligible.

2.3 Polynomial-based Methods

We compare simulation to three multi-dimensional polynomial-based rules: Gaussian product rules, sparse grid integration, and monomial rules. Often these rules are said to be exact for degree d because they integrate any polynomial of degree d or less without error.[15] A common example in one-dimension is the Gaussian-family of rules: with R nodes they exactly integrate any polynomial of degree $2R - 1$ or less. Consequently, polynomial rules require many fewer nodes than pMC, making them both parsimonious and highly accurate for most integrands. The higher the degree, the more accurate the approximation of the integral but the higher the cost because the integrand must be evaluated at

[14] The current industry standard is to use Halton draws and Kenneth Train's code, which is available on his website(Train, 1999).

[15] At least, theoretically. With finite precision of arithmetic there may be extremely small errors from truncation and round off.

more nodes. The actual choice of nodes and weights depends on the the rule and the weighting function in the integral. For smooth functions which are well approximated by polynomials – such as analytic functions – a good quadrature rule should always outperform simulation, except perhaps in extremely high dimensions where MC methods may be the only option. Like MC rules, polynomial approximations of integrals converge to the true value of the integral as the number of nodes approaches infinity, i.e. $\lim_{R\to\infty} Q[f] = I[f]$.

To use a Gaussian rule, simply determine which rule corresponds to the weighting function and parameter space of the integral in question. Then look up the nodes and weights in a table or use the appropriate algorithm to calculate them. See Judd (1998) for a description of all the common rules. Note: it is often necessary to use a change of variables, such as a Cholesky decomposition of a variance matrix.

2.3.1 Gaussian Product Rule

The Gaussian product rule uses a straight-forward method to construct nodes and weights: compute nodes by forming all possible tensor products of the nodes and weights which are associated one dimensional rule which is appropriate for the integral's domain and weighting function. I.e., each of the d-dimensional node z_k's individual coordinates are one of the one-dimensional nodes. The set of nodes, then, is all possible zs which are on the lattice formed from the Kronecker product of the one-dimensional nodes. See Figure 1 in Heiss and Winschel (2008). The weights are the product of the weights which correspond to the one-dimensional nodes. For example, consider a two-dimensional rule with one dimensional nodes and weights $\{y_1, y_2, y_3\}$ and $\{w_1, w_2, w_3\}$, respectively. Then the product rule has nodes $\mathcal{Y} = \{(y_1, y_1), (y_1, y_2), (y_1, y_3), \ldots, (y_3, y_3)\}$. The corresponding weights are $\mathcal{W} = \{w_1 \cdot w_1, w_1 \cdot w_2, w_1 \cdot w_3, \ldots, w_3 \cdot w_3\}$. And the approximation for the integral is $Q[f] = \sum_{k\in\mathcal{I}} \tilde{w}_k f(\tilde{y}_k)$, where $\mathcal{I}$ indexes $\mathcal{W}$ and $\mathcal{Y}$, and $\tilde{y}_k \in \mathcal{Y}$ and $\tilde{w}_k \in \mathcal{W}$.[16] See the example code in 4 for the actual algorithm.

Consequently, we must evaluate the function at R^d points to approximate a d-dimensional integral, which quickly becomes much larger than $10,000$, often a practical upper limit on the number of nodes which are feasible with current computer technology.[17] We use product of formulas which have the same number of nodes in each dimension – i.e. are exact for the same degree – so that we know roughly what degree polynomial can be integrated exactly (Cools, 1997). If the formulas are not exact to the same degree, then we know only upper

[16] To be more formal, consider a set of one-dimensional nodes and weights, $\{y_k, w_k\}_{k=1}^R$. The d-dimensional product rule is the set of nodes $z_k \in \left\{\times_{m=1}^d y_{i_m}\right\}$. Let $\mathcal{C}(z_k)$ be a function which returns an ordered list of the indexes $(i_1, i_2, \ldots, i_d)$ of the one-dimensional nodes forming the coordinates of the d-dimensional vector z_k. Then each node $z_k = (y_{i_1}, y_{i_2}, \ldots, y_{i_d})$ and has weight $w_{i_1} \cdot w_{i_2} \cdot \ldots \cdot w_{i_d}$, the product of the one-dimensional weights corresponding to the one dimensional nodes of z_k.

[17] Disclaimer: future readers should be mindful of the level of technology which was available at the time this paper was written.

and lower bounds on what polynomial will integrate exactly. One problem with product rules is that many extra terms will be integrated exactly because of the product between the one-dimensional bases. For example, consider a problem with three dimensions and five nodes per dimension. The one-dimensional Gaussian formula will be exact for all polynomials of degree $2*5-1=9$ but the corresponding product rule will also exactly compute higher-order terms such as $x_1^9 x_2^3 x_3^1$, where x_i is the variable for the i-th dimension. Thus, there is some indeterminacy about what will be exactly integrated by a product rule: this extra accuracy may be unnecessary and result in extra computational burden.

2.3.2 Sparse Grid Integration

We also consider sparse grid integration (SGI) which is closely related to the Gaussian product rules. SGI uses a subset of the nodes from the product rule and rescales the weights appropriately. The advantage of SGI is that it exploits symmetry so that it requires many fewer points, making it more efficient to compute with little or no loss in accuracy. In addition, the nodes and weights for higher levels of exactness are easier to compute for SGI than for monomial rules. We use a Konrad-Patterson rule for choosing nodes as explained in Heiss and Winschel (2008). Our experiments show that SGI is very competitive with monomial rules in many cases. However, when the lowest possible computational costs matter, the monomial rule is the best option because it delivers the highest accuracy with fewest nodes.

2.3.3 Monomial Rules

Monomial rules exploit symmetries even more effectively than SGI and provide very accurate approximations with surprisingly few nodes, even for moderate dimensions (Stroud, 1971; Cools, 2003). Formally, a monomial in $x \in \mathbb{R}^d$ is the product $\prod_{i=1}^{d} x_i^{\alpha_i}$ where $\alpha_i \in \mathbb{W}$ and $\mathbb{W} \equiv \{0, 1, 2, \ldots\}$. Thus, monomials are the simplest possible basis for the set of multi-dimensional polynomials. The total order is just the sum of the exponents $\sum_i \alpha_i$. $\mathbf{x}^{\alpha}$ is a compact notation which refers to the monomial $\prod_{i=1}^{d} x_i^{\alpha_i}$. Like the Gaussian rules, monomial rules are constructed so that they will exactly integrate all monomials less than or equal to some total order. Monomial rules are more efficient than Gaussian product rule in part because they do not exactly integrate any unnecessary higher order terms.

The performance gains from monomial rules are clear, but the cost comes in computing the rule's nodes and weights. Fortunately for researchers many efficient, accurate rules have already been computed for standard kernels and parameter spaces (Cools, 2003; Stroud, 1971). Thus, a practitioner only needs

to look up the appropriate monomial rule in a table [18,19] and can then compute the integral as the weighted sum of the integrand at the nodes. Unfortunately, if you need a rule which doesn't exist you will need the help of a specialist (Cools, 1997).

See Section 4 explains how to use the 983 node Stroud monomial rule 11-1 – which is exact for degree 11 polynomials in five dimensions – to compute the BLP market share integrals. For the BLP integrals, the Gaussian product rule required about ten times more nodes for insignificant gains in accuracy.

2.4 Precision and Accuracy

We now provide a quick comparison between all of these rules using the code we developed to validate that our implementation produced the 'same' answer as required by theory.[20] The polynomial rules correctly integrate all monomials less than or equal to their respective degrees and produce poor results for monomials of higher degree. However, we also performed these tests using 100 replications of a pMC rule with $R = 10,000$ draws which lead to a surprising result: pMC performed poorly for the low order monomials, with error increasing with the degree of the monomial, yet it also produced poor results for high order monomials where we expected it would outperform the polynomial rules. We conjecture that pMC works well only when the high-order terms in a Taylor series expansion are very small, something which is explicit in the construction of monomial rules. These results are summarized in Table 1 which shows the difference between the theoretical value and the value computed with each rule. The first three columns are the results for the Gaussian-Hermite Product rule with 3^5, 5^5, and 7^5 nodes – i.e., 3, 5, and 7 nodes in each of the 5 dimensions; next the Konrad-Patterson sparse grid rule which is exact for degree 11; then, the left and right versions[21] of rule 11-1 in Stroud (1971), also exact for degree 11; and, finally, the two right most columns show the mean absolute error and the standard error for the pMC rule with 100 replications. The monomials are listed by increasing degree. Note that the Gauss-Hermite product rules will exactly integrate any monomial rule as long as the coordinates in each dimension are raised to some power less than or equal to $2R - 1$ where R is the number of one dimensional nodes used in the tensor product. Sparse grid and the monomial rules are exact for any monomial whose degree is less than or equal

[18] Typically, a small amount of computation is required because the table will only provide each unique set of nodes and weights. A researcher must then calculate the appropriate (symmetric) permutations of the unique nodes to generate all possible nodes.

[19] A monomial rule may have several equivalent sets of nodes and weights because the system of equations used to compute the monomial rule may have multiple solutions. For example, Stroud rule 11-1 has two solutions, which we refer to as 'Left' and 'Right' after the two columns in the table which list the different solutions. The performance of these solutions will vary slightly based on the shape of the problem.

[20] The results should be the 'same' up to the limits of standard numerical errors such as truncation and round-off error.

[21] Often multiple monomial rules exist for a given domain, degree, and weight function because there multiple solutions to the systems of equations which is used to generate the rules. See Cools (1997) for an introduction.

to 11. For odd monomials, the difference in performance is even more stark: the polynomial rules are 0 to the limits of numerical precision whereas pMC has significant error, especially as the degree of the monomial increases. These results, in our opinion, considerably strengthen the case for using sparse grid or monomial rules because pMC is never better.[22]

2.4.1 Bias and Noise

When choosing which quadrature rule to use, a researcher should consider how it will affect their results. Simulation methods have become extremely popular because of their ease of use, especially for applied econometrics. However simulation, as discussed at length in Train (2009), can suffer from both bias as well as noise. The nature of the bias depends on the type of estimator: for Method of Simulated Moments (MSM) the bias is zero unlike Maximum Simulated Likelihood (MSL) and Maximum Simulated Score (MSS). The bias occurs because the bias term is linear only for MSM: consequently, Jensen's inequality shows that MSL and MSS must be biased. The noise term will approach zero asymptotically if R, the number of draws, approaches infinity faster than $\sqrt{N}$, the number of observations. Consequently, researchers who use MSL or MSS should remember to correct for this bias.

Polynomial-rules, on the other hand, only suffer from approximation error and that to a much lesser degree than Monte Carlo methods. Thus, the error is much smaller for these methods, making them much better suited for empirical and other problems than simulation. With polynomial rules, researchers can also consider more efficient econometric methods such as MLE instead of GMM.

[22] We also computed these tests for Halton draws generated by MATLAB R2010b's `qrandstream` facility which did not perform significantly better than pMC.

	GH 3^5	GH 5^5	GH 7^5	Sparse Grid	11-1 L	11-1 R	$\overline{\lvert pMC \rvert}$	$\sigma(pMC)$
1	−1.1e−15	−3.3e−15	−2.1e−15	6.7e−15	5.2e−12	−9.9e−14	2e−14	0
x_1^1	−2.1e−17	1e−15	1.3e−15	2.3e−17	7.1e−15	−3.6e−15	0.0075	0.0092
$x_1^1 x_2^1$	0	−1.7e−18	−1.7e−18	2.8e−17	0	−2.2e−16	0.008	0.01
$x_1^1 x_2^1 x_3^1 x_4^1 x_5^1$	0	0	0	0	0	0	0.0086	0.011
x_1^2	−4.4e−16	−2e−15	−1.1e−15	−5.2e−14	9.6e−13	−7.9e−14	0.012	0.014
x_1^4	0	−8.9e−15	4e−15	−1.5e−13	3.9e−13	7.5e−15	0.076	0.096
x_1^6	−6	−3.6e−14	−1.8e−14	−7.6e−13	4e−13	1.9e−13	0.76	0.94
$x_2^6 x_4^4$	−18	−9.9e−14	−2.3e−13	−2.2e−12	−3.1e−13	1.8e−13	7.4	9.4
x_1^{10}	−8.6e+02	−1.2e+02	−3.2e−12	−4.9e−11	2.9e−11	2.4e−11	1.8e+02	2.1e+02
$x_1^5 x_2^4 x_3^2$	5e−16	3.6e−15	−8.9e−15	−6.9e−16	1.8e−15	0	3.9	6.8
x_1^{12}	−1e+04	−3.7e+03	−4.4e−11	−5e−10	−7.2e+02	−7.2e+02	3.2e+03	3.9e+03
x_1^{13}	0	3.1e−11	2.8e−10	−1e−11	−2.9e−11	5.8e−11	1.3e+04	2e+04
x_1^{14}	−1.3e+05	−8.1e+04	−5e+03	−7.2e−09	−3.1e+04	−3.1e+04	6.2e+04	8.2e+04
x_1^{15}	9.9e−14	−3.3e−10	−1.2e−09	−5.8e−11	2.3e−10	−4.7e−10	2.4e+05	4.3e+05
x_1^{16}	−2e+06	−1.6e+06	−2.9e+05	−3.4e+04	−8.9e+05	−8.9e+05	1.3e+06	1.8e+06
$x_1^6 x_2^6 x_3^4 x_4^2 x_5^2$	−4.3e+02	−1.4e−12	−1.1e−12	−4.3e+02	1.6e+05	1.6e+05	8.5e+02	2.6e+03
$x_1^8 x_2^6 x_3^4 x_4^2 x_5^2$	−4e+03	−6.4e−12	−1.5e−11	−4e+03	1.8e+06	1.8e+06	6.7e+03	1.9e+04
$x_1^{10} x_2^5 x_3^4 x_4^2 x_5^2$	0	1.2e−13	−1.7e−13	0	5.8e−11	5.8e−11	1.9e+04	6.5e+04
$x_1^{10} x_2^{10} x_3^6 x_4^4 x_5^2$	−4e+07	−9.6e+06	−8.9e−08	−4e+07	3e+11	3e+11	7.4e+07	2.6e+08
$x_1^{16} x_2^{12} x_3^4 x_4^4 x_5^2$	−1.9e+11	−1.6e+11	−2.7e+10	−1.9e+11	4e+14	4e+14	2.1e+11	3.3e+11

Table 1: Monomial Integration Errors: Polynomial Rules vs. pMC ($R = 10,000$)

3 The Basics of BLP

We now quickly review the features and notation of Berry, Levinsohn, and Pakes (1995)'s model in order to examine how different quadrature rules affect estimation results. BLP has become one of the most popular structural models of product differentiation because it fits empirical data well by using a flexible form which combines both random coefficients and unobserved product-market characteristics, ξ_{jt}, enabling the model to explain consumers' tastes for both horizontal and vertical product differentiation. The model produces realistic substitution patterns:[23] the random coefficients can handle correlations between different choices, overcoming the Independence from Irrelevant Alternatives (IIA) problem that is a feature of logit models, and ξ_{jt} captures unobserved heterogeneity in product quality, preventing bias in parameter estimates from product traits which the econometrician cannot observe. Nevo (2000a) provides a detailed and accessible explanation of the model. BLP is now sufficiently established that the several recent textbooks (Train, 2009; Davis and Garcés, 2009) also cover it.

Throughout this paper, we base our notation on a simplified version of the notation in Dubé, Fox, and Su (2009). Thus, we consider T markets which each have J products plus an outside good. Each product $j \in J$ in market $t \in T$ has K characteristics x_{jt} and price p_{jt} as well as an unobserved, product-market shock, ξ_{jt}. The market could be a time period, as in the original BLP papers on automobiles, or a city, as in Nevo's papers on ready-to-eat breakfast cereal. The shock ξ_{jt} is observed by consumers and firms but not by the econometrician. This shock captures vertical aspects of product differentiation whereas the random coefficients model horizontal differentiation: all consumers value a larger ξ_{jt} but rank product characteristics differently according to their type. Lastly, y_i is consumer i's expenditure and drops out of the model because it is not interacted with any product-specific characteristics.

BLP assume consumers are rational, utility maximizers who choose the good which maximizes their utility. Let consumer i's utility from purchasing product j in market t be[24]

$$U_{ijt} = V_{ijt} + \epsilon_{ijt}$$

with

$$V_{ijt} = \alpha_i (y_i - p_{jt}) + x'_{jt}\beta_i + \xi_{jt}.$$

ϵ_{ijt} is an IID, Type I Extreme value shock, which leads to a simple closed form solution for market shares, conditional on consumer types, (α_i, β_i). In practice, y_i and p_{jt} are often the logarithm of the respective quantities, which ensures

[23] The substitution patterns will be incorrect if congestion in product space matters. See **Berry and Pakes (2007)** and, for an application where congestion matters, Nosko (2010).

[24] Some researchers specify $\log(y_i - p_{jt})$ instead of $(y_i - p_{jt})$ to capture income effects (Petrin, 2002).

that the utility is homogeneous of degree zero. Similarly, the utility of choosing the outside, 'no purchase' option ($j = 0$ by convention) is[25]

$$U_{i0t} \quad = \alpha_i y_i + \quad \epsilon_{i0t}.$$

The coefficients α_i and β_i are type-specific 'random coefficients' – i.e., they depend on a consumer's type and are drawn from some distribution in order to capture unobserved differences in consumers' tastes:[26]

$$\begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} \quad = \quad \begin{pmatrix} \bar{\alpha} \\ \bar{\beta} \end{pmatrix} + \Sigma \nu_i$$

where all consumers have the same mean taste preferences $\bar{\alpha}$ and $\bar{\beta}$. The unobserved taste shock ν_i is a $K+1$ column vector (because there are K product characteristics plus price) and has distribution $\nu_i \sim P_\nu$. The variance of the taste shock is Σ, a $(K+1) \times (K+1)$ matrix . P_ν is usually assumed to be multivariate normal. Following convention, we refer to the model's parameters as θ where $\theta = (\theta_1, \theta_2)$, $\theta_1 = \left(\bar{\alpha}, \bar{\beta}\right)$, the parameters for mean utility, and $\theta_2 = (\texttt{vec}\,(\Sigma))$, the parameters for the standard error of the random coefficients. Thus, θ refers to all of the parameters to be estimated.

It is convenient to partition the utility into the mean utility

$$\delta_{jt}\left(\xi_{jt}; \theta_1\right) \quad = \quad x_{jt}^{'} \bar{\beta} - \bar{\alpha} p_{jt} + \xi_{jt},$$

which is the constant utility that any consumer type gains from choosing product j in market t, regardless of type, and a type-specific preference shock

$$\mu_{ijt} \quad = \quad \left[-p_{jt} \; x_{jt}^{'}\right] \left(\Sigma \nu_i\right).$$

μ_{ijt} has mean zero and captures individual heterogeneity. μ_{ijt} is a scalar because $\left[-p_{jt} \; x_{jt}^{'}\right]$ is a $K+1$ row vector and $\Sigma\nu_i$ is a $K+1$ column vector. Some researchers permit $\alpha_i < 0$ which can produce a positive price coefficient for some consumer types. A possible solution is to assume that α_i is log-normally distributed. In other applications $\alpha_i < 0$ may make sense if price is a signal of quality.

Researchers typically assume that $\epsilon_{ijt} \sim$ Type I Extreme Value so the market shares, conditional on consumer type ν,[27] have a closed-form analytic solution, the multinomial logit.

[25] Berry, Levinsohn, and Pakes (1995) specify $U_{i0t} = \alpha_i y_i + \xi_{0t} + \sigma_{0t}\nu_{i0t} + \epsilon_{i0t}$.

[26] Some researchers also include demographic information in the random coefficients. We ignore demographics in order to focus on the numerical properties of the model.

[27] Note: the consumer type, ν, is scaled by Σ, the Cholesky decomposition of the variance matrix of the random coefficients.

$$s_{jt}\left(\delta\left(\xi;\theta_1\right)|\theta_1,\theta_2\right) = \frac{\exp\left[\delta_{jt}+\mu_{ijt}\left(\nu\right)\right]}{\sum_k \exp\left[\delta_{kt}+\mu_{ikt}\left(\nu\right)\right]}.$$

Then the unconditional share integrals are the just the expectation of the regular MNL choice probabilities with respect to ν:

$$s_{jt}\left(\delta\left(\xi\right)|\theta_1\right) = \int_{\mathbb{R}^{K+1}} \frac{\exp\left[\delta_{jt}+\mu_{ijt}\left(\nu\right)\right]}{\sum \exp\left[\delta_{kt}+\mu_{ikt}\left(\nu\right)\right]}\phi\left(\nu\right)d\nu.$$

Here $\phi\left(\nu\right)$ is the standard Normal probability density function. We restrict Σ to be diagonal as in the original BLP papers.[28] The random coefficients logit can in theory model any choice probabilities given a suitable mixing distribution (McFadden and Train, 2000; Train, 2009).[29] In practice, researchers choose a Normal distribution for the random coefficients because it is tractable. But, we don't know of any papers which actually test this assumption. Burda, Harding, and Hausman (2008) specify a more flexible mixing distribution; it may be possible to apply their method to test the performance of the assumption of logit + Normal. Nevertheless, logit + Normal should work well as long as the real-world mixing distribution is smooth and single-peaked because the tails will not contribute much to the integral.

Historically, a nested fixed point (NFP) algorithm based on Rust (1987) is used to estimate the model: : the outer loop computes the point estimates of $\hat{\theta}$ by minimizing a GMM objective function whose moments are constructed from ξ_{jt}; the inner loop solves the nonlinear system of equations equating predicted and observed shares for the mean utilities, $\delta_{jt}\left(\theta\right)$, and, hence, ξ_{jt}. The original implementation (Berry, Levinsohn, and Pakes, 1995; Nevo, 2000a) uses a contraction mapping to perform this inversion. Thus, the researcher codes an outer loop to solve the program

$$\hat{\theta} = \arg\max_{\theta}\left(Z'\xi\right)'W\left(Z'\xi\right)$$

and an inner loop to recover δ via a contraction mapping

$$\exp\left(\delta_{jt}^{n+1}\right) = \exp\left(\delta_{jt}^{n}\right)\times S_{jt}/s_{jt}\left(\delta_{jt}^{n};\theta_2\right),$$

where S_{jt} are the observed market shares, s_{jt} the predicted market shares, and δ_{jt}^{n} the n-th iterate in a sequence which hopefully converges to the true mean utilities. Given the mean utilities, the product market shock is simply

[28] Nevo (2001) estimates the off-diagonal elements. We expect that the advantages of monomial rules would be even more apparent when estimating a model with off-diagonal elements.

[29] McFadden and Train (2000) is a very general result and applies to elasticities and moments as well as choice probabilities. Consequently, the mixed logit can approximate general substitution patterns to arbitrary accuracy.

$$\xi_{jt} \quad = \quad \delta_{jt} - [-p_{jt}\ x_{jt}]\,\theta_1.$$

Berry, Levinsohn, and Pakes (1995) proves that this mapping is a contraction and Nevo (2000a) advocates using this exponential form to improve numerical performance by avoiding computing logarithms which are more costly than exponentiation.[30] In addition, Gandhi (2010) shows that the market share equations are invertible. Reynaerts, Varadhan, and Nash (2010) develop other methods for numerically inverting the market share equations which are faster and more robust as well as discussing some of the convergence problems of the contraction mapping.

Numerical integration affects both choice probabilities and the inversion of the market share equation. Consequently, numerical errors in computing integrals can propagate through both of these channels. With the above GMM specification, the gradient of the GMM objective function depends on the gradient of δ, which in turn depends on the gradient of the inverse of the market share equation, $s^{-1}(S;\theta)$. This provides a channel for numerical errors in computing not just the share integrals but also the gradient of the market share integrals to propagate, affecting both the point estimates and the standard errors. As discussed below, one big advantage of monomial rules over pMC is that they provide a more accurate approximation for both an integral and its gradient.

When estimating the BLP model below, we use the same moment conditions as Dubé, Fox, and Su (2009). These moment conditions depend on the product of the unobserved product-market shock, ξ, and a matrix of instruments. The matrix of instruments consists of various products of product attributes and a set of synthetic instrumental variables which correlated with price but not ξ. See Dubé, Fox, and Su (2009)'s code for details.

In this paper, we use Mathematical Programming with Equilibrium Constraints (MPEC) (Su and Judd, 2008) to estimate the BLP model because it is faster and more robust than NFP. MPEC relies on a modern, state of the art solver such as KNITRO or SNOPT, to solve the model in one optimization step using constraints:

$$\begin{aligned} \max_{\theta,\delta,\eta} \quad & \eta^{'} W \eta \\ \text{s.t.} \quad & s\left(\delta\left(\xi\right);\theta\right) = S \\ & \eta = Z^{'}\xi. \end{aligned}$$

By adding the extra variable η, we improve the sparseness pattern which makes the problem easier to solve and more stable numerically.[31] Furthermore, MPEC

[30] In simple heuristic tests, we find that the contraction mapping has poor convergence properties, fails to satisfy the sufficiency conditions of the Berry, Levinsohn, and Pakes (1995)'s theorem 10% of the time, and often has a contraction rate close to or exceeding 1.

[31] With modern solvers, the sparseness pattern and type of non-linearities are more important than the number of variables.

solves for the mean utilities implicitly via the constraint that observed market shares equal predicted market shares, increasing both speed and stability.

Besides nested fixed point and MPEC, there are several other estimation approaches including control functions (Petrin and Train, 2006), a two step procedure with maximum likelihood and instrumental variables (Train, 2009), and Bayesian (Jiang, Manchanda, and Rossi, 2009). Most of these methods exploit the fact that if you can estimate the mean utilities δ_{jt} then you can then recover the product-market shock ξ_{jt}.

The asymptotic and finite sample properties of BLP are still not well understood. Berry, Linton, and Pakes (2004) prove asymptotic normality assuming $J \to \infty$. Berry, Haile, and of Economic Research (2010) show the model is identified under the 'Large Support' assumption. Skrainka (2011) uses large scale simulations to characterize finite sample performance.

3.1 Example: Computing BLP Product-Market Shares

Given the above assumptions, the monomial (or Gauss-Hermite or sparse grid) approximation for the integral is

$$s_{jt} \approx \frac{1}{\pi^{(K+1)/2}} \sum_k \left\{ \frac{\exp\left[\delta_{jt} + \mu_{ijt}\left(\psi_k\right)\right]}{\sum_m \exp\left[\delta_{mt} + \mu_{imt}\left(\psi_k\right)\right]} \omega_k \right\}$$

where (ψ_k, ω_k) are the nodes and weights for an suitable quadrature rule with Gaussian kernel and $K+1$ is the dimension of ν_k.[32] The factor $\pi^{-(K+1)/2}$ comes from the normalization of the Normal density. The choice of monomial rule depends on the number of dimensions of the integral, desired level of exactness (accuracy), the domain of integration, and the mixing distribution a.k.a. weighting function.

For a Monte Carlo method, the approximation is

$$s_{jt} \approx \frac{1}{R} \sum_k \frac{\exp\left[\delta_{jt} + \mu_{ijt}\left(\psi_k\right)\right]}{\sum_m \exp\left[\delta_{mt} + \mu_{imt}\left(\psi_k\right)\right]}$$

for R nodes ψ_k drawn from the Normal distribution. Note that these two formula have the same structure: a weighted sum of the integrand evaluated at a set of nodes. For a Monte Carlo method, the weight $\omega_k \to 1/R$.

4 The Experiments: Simulation vs. Quadrature

We compare how pMC, monomial, Gaussian product, and sparse grid quadrature rules affect the computation of several key quantities in the BLP model. We compare how these rules perform when computing the market share integrals,

[32] $K+1$ for the K product characteristics plus price.

Parameter	Value
J	25
T	50
$\theta_{1\cdot} \equiv \left(\bar{\beta}', \bar{\alpha}\right)$	$\begin{pmatrix} 2 & 1.5 & 1.5 & 0.5 & -3 \end{pmatrix}'$
$\theta_{2\cdot} \equiv \texttt{diag}\left(\Sigma\right)^{1/2}$	$\begin{pmatrix} \sqrt{0.5} & \sqrt{0.5} & \sqrt{0.5} & \sqrt{0.5} & \sqrt{0.2} \end{pmatrix}'$
R	100

Table 2: Parameters Used to Generate Monte Carlo Data sets.

point estimates, standard errors, and the unobserved heterogeneity ξ_{jt}, all of which are critical components of the model. Of course, we use a state of the art solver (KNITRO or SNOPT) and algorithm (MPEC) for these experiments.

4.1 Experimental Setup

Our experiments use five different Monte Carlo data sets which we generated from unique seeds and the parameters shown in Table 2 using MATLAB$^{\text{TM}}$'s `rand` and `randn` functions. We use the same values as Dubé, Fox, and Su (2009) (DFS hereafter), except that we use fewer products and markets and chose different seeds.[33] We refer to these data sets via their seeds, which we label 1 to 5. To ensure that there is some noise in each data set, as in real-world data, we compute the 'observed' market share integrals using a pMC rule with $R = 100$ nodes.[34] Currently, these parameter values result in a market share of about 90% for the outside good, which seems reasonable for a differentiated, durable good such as an automobile. That many of the observed market shares are exceedingly small could lead to inaccuracies in the corresponding computed market shares because both types of quadrature rules can be unreliable in large regions of flatness. We only consider diagonal Σ to facilitate validation and to maintain consistency with the most BLP papers and DFS.

We focus on how each quadrature rule affects the point estimates – i.e. whether a state of the art solver and algorithm (MPEC) could consistently and efficiently find a unique global optimum. For each data set and quadrature rule, we compute the optimum for the following setups: (1) five randomly choose starts near the two-stage least squares (2SLS) logit estimate; (2) multiple starts taken about the average of the best point estimates for $\hat{\theta}$ for the 5^5 Gauss-Hermite product rule; and, (3) for pMC only, multiple Monte Carlo draws of nodes for the same starting value. In all cases, we compute standard errors using the standard GMM sandwich formula $\text{Var}\left(\hat{\theta}\right) = \left(\hat{G}'W\hat{G}\right)^{-1}\hat{G}'W\hat{\Lambda}W\hat{G}\left(\hat{G}'W\hat{G}\right)$ where $\hat{G}$ is the gradient of the moment con-

[33] Their code provided the starting point for the code which we developed to explore the impact of quadrature rules on BLP. We downloaded the code from JP Dubé's website (http://faculty.chicagobooth.edu/jean-pierre.dube/vita/MPEC%20code.htm), Fall 2009.

[34] For the rest of the paper, we use R to refer to the number of draws in a Monte Carlo simulation and N as the number of replications.

ditions, W the weighting matrix formed from the instruments, $\left(Z^{'}Z\right)^{-1}$, and $\hat{\Lambda}$ the covariance of the moment conditions, $\sum_{j\in J}\sum_{t\in T} z_{jt}z_{jt}^{'}\xi_{jt}^{2}$. In addition, we compare the level of the market share integrals calculated by the different rules. Future research should also examine how quadrature rules affect the approximation of the gradient and Hessian of the objective function, which are more important than the level of market shares in determining the point estimates and standard errors.

In our computations, we use the following numerical integration techniques: pseudo-Monte Carlo (pMC), Gaussian Hermite product rule, sparse grid integration (SGI) (Heiss and Winschel, 2008), and Stroud monomial rule 11-1 (Stroud, 1971). Because we have assumed that the mixing distribution of the random coefficients is Normal, we compute the pMC nodes by drawing between $1,000$ and $10,000$ nodes from a standard Normal distribution using MATLAB$^{\text{TM}}$'s `randn` function. We use the same draws for each market share integral, s_{jt}, as in DFS. Current 'best practice' seems to be $5,000$ points so $10,000$ nodes will enable us to put reasonable bounds on the accuracy of simulation-based BLP results. Berry, Levinsohn, and Pakes (1995) use pMC with importance sampling in an attempt to reduce variance, but importance sampling is just a non-linear change of variables and should not significantly improve the performance of pMC. Lastly, using different draws for each market share integral would improve the point estimates because the simulation errors tend to cancel out and improve the GMM estimates because the share equations are no longer correlated (McFadden, 1989).[35] However, taking separate draws for each integral requires more memory and could considerably increase computational burden through increase I/O costs.[36]

For polynomial-based rules, we use quadrature rules which are designed for a Gaussian kernel, $\exp^{-x^2}$, because the mixing distribution is Normal. Consequently, the correct one-dimensional rule to generate the nodes and weights for the multi-dimensional product rule is Gaussian-Hermite. The product rule consists of all Kronecker products of the one-dimensional nodes and the weights are the products of the corresponding one dimensional weights. The algorithm is:[37]

```
function [ Q_NODES, Q_WEIGHTS ] = GHQuadInit( nDim_, nNodes_ )
  % Get one-dimensional Gauss-Hermite nodes and weights
  tmp = gauher( nDim_ ) ;
```

[35] Quantifying the actual benefit of separate draws is an open research question and merits further investigation.

[36] Swapping occurs when a process's memory demands are greater than the physical RAM available, causing operating system's virtual memory facility to keep transferring memory between RAM and disk.

[37] `gauher` uses the algorithm in Press, Teukolsky, Vetterling, and Flannery (2007) to compute the Gaussian-Hermite nodes and weights. Many researchers mistrust Press, Teukolsky, Vetterling, and Flannery (2007), however we tested the nodes and weights on the relevant moments to verify that they were indeed correct.

```
% extract quadrature information for one dimension
vNodes = tmp( :, 1 ) ;
vWeights = tmp( :, 2 ) ;
% calculate three dimensional nodes and weights
% Q_WEIGHTS = kron( vWeights, kron( vWeights, vWeights ) ) ;
Q_WEIGHTS = vWeights ;
for ix = 2 : nDim_
  Q_WEIGHTS = kron( vWeights, Q_WEIGHTS ) ;
end
% Make sure that the right-most dimension (ixDim = nDim_) varies
% most quickly and the left-most (ixDim = 1) most slowly
Q_NODES = zeros( nDim_, nNodes_^nDim_ ) ;
for ixDim = 1 : nDim_
  Q_NODES( ixDim, : ) = kron( ones( nNodes_^(ixDim - 1), 1 ), ...
     kron( vNodes, ones( nNodes_^(nDim_ - ixDim), 1 ) ) ) ;
end

% Correct for Gaussian kernel versus normal density
Q_WEIGHTS = Q_WEIGHTS / ( pi ^ ( nDim_ / 2 ) ) ;
Q_NODES = Q_NODES * sqrt( 2 ) ;
```

Note that the Normal density requires renormalization of the nodes and weights because the Gaussian kernel, unlike the Normal density, lacks a factor of $1/2$ in the exponent as well as the factors of $\pi^{-1/2}$ used for normalization. We experimented with product rules for five dimensions[38] which used 3, 4, 5, 7, or 9 nodes in each dimension. We found that using more nodes than 7 in each dimension did not improve accuracy but greatly increased computational cost because of the curse of dimensionality: for a five dimensional shock the product rule with 7 nodes per dimension requires $7^5 = 16,807$ nodes to compute a share integral (whereas 9 nodes per dimension would require $9^5 = 59,049$.).

Sparse grid integration rules function similarly to product rules but exploit symmetry so that fewer points are required. We use the Konrad-Patterson algorithm for a Gaussian kernel, as described in Heiss and Winschel (2008), and compute nodes and weights for a five-dimensional problem which is exact for polynomials of total order 11 or less using their MATLAB™code.[39] We chose this configuration so that SGI is exact for the same total order as the monomial rule. For this level of accuracy, 993 nodes are required, a substantial improvement on the product rule and only ten more than the monomial rule. However even a small increase in accuracy requires a rapid increase in the number of nodes, e.g. exactness for total order 13 requires 2,033 nodes. See their paper for the details.

Lastly, we use Stroud (1971)'s monomial rule 11-1 for a Gaussian kernel.

[38] The dimension is five because the synthetic data has four product characteristics plus price.

[39] The code can be downloaded from http://www.sparse-grids.de/.

Stroud (1971) provides two solutions,[40] both of which provide comparable performance and integrate exactly all five-dimensional monomials of total order 11 or less using only 983 nodes. To implement the rule, we wrote a function which computed the nodes and weights from the data in Stroud's text. [41] This simply involves a lot of book-keeping to compute the correct permutations of the node elements using Stroud's data.

Now we briefly discuss our choice of the SNOPT solver, how we configured it, and numerical stability.

4.1.1 Solver Choice and Configuration

Because different solvers work better on different problems, we tried both the KNITRO and SNOPT solvers on BLP. Both solvers use efficient, modern algorithms: KNITRO supports active set and interior point algorithms (Byrd, Nocedal, and Waltz, 2006) whereas SNOPT uses a sequential quadratic programming (SQP) method (Gill, Murray, and Saunders, 2002). For details about these algorithms see Nocedal and Wright (2000). Although KNITRO out performs MATLAB's `fmincon` non-linear solver, it converged to an optimum much less frequently and quickly than SNOPT.[42] We suspect SNOPT outperforms KNITRO because the latest version has better support for rank deficient problems. Skrainka (2011) develops a C++ implementation of BLP which further improves the robustness of SNOPT when solving the BLP model by enabling SNOPT's LU rook pivoting option. This result is another indication of (near) rank deficiency and ill-conditioning. Consequently, if the objective function is nearly flat – i.e., poorly identified numerically – SNOPT should be more stable. In addition, interior point methods, such as those used by KNITRO, do not work well on nonconvex problems. SNOPT uses an SQP method which can handle the local nonconvexities caused by simulation for almost all of the datasets which we generated.

To get the most out of the solver, we fine-tuned the solver's options. In addition, for both solvers we specified the sparseness pattern and supplied hand-coded derivatives (gradient and Hessian of the objective function; Jacobian of the constraints) in order to increase numerical stability and performance. We also set box constraints to prevent the solver from searching bad regions of parameter space, as discussed below in 4.1.2. Lastly, we set the tolerance to $1e-6$, which is a typical outer loop tolerance for BLP.

4.1.2 Numerical Stability Considerations: Overflow and Underflow

During our initial experiments we soon became concerned that the BLP model, despite some support for identification (Berry, Linton, and Pakes, 2004; Berry,

[40] We label the two versions of rule 11-1 as 'Left' or 'Right', according to whether we use the set of nodes and weights in the left or right column of his Table $E_n^{r^2}$: 11-1 on pp. 322-323.

[41] Our monomial code is available at www.ucl.ac.uk/∼uctpbss/public/code/HighPerfQuad.

[42] Che-Lin Su reports that KNITRO is faster when you supply an analytic Hessian for the constraints.

Haile, and of Economic Research, 2010), was not identified – or at least not numerically identified given the limits of current computers. We found that SNOPT's error code EXIT=10 with INFORM=72 did not mean that the solver had failed to converge. Instead, SNOPT sets these flags when it encountered market shares which were indeterminate, i.e. the computations produced a NaN.[43,44] In some cases, the constraint that $\log S_{jt} = \log s_{jt}$, i.e. that the logs of the observed and calculated market shares are equal, diverged to $-\infty$ when the shares were nearly zero. This problem occurred because the market share calculation is numerically unstable when the utility from the chosen alternative is extremely large.

These problem frequently occurs with logit-based models because the exponential function diverges quickly to infinity for even moderately-sized arguments.[45] Consider the typical straight-forward implementation of the logit where $f(V;j) = \dfrac{\exp(V_j)}{\sum_k \exp(V_k)}$, for some vector of utilities, V, and choice j. This implementation is unstable because if $V_j \to \infty$ then $f(V;j) \to \dfrac{\infty}{\infty} \equiv \text{NaN}$. This situation usually occurs when evaluating quadrature nodes which are in the tail of the distribution and contribute little to the market share/choice probability. However, this formulation does allow one to compute a vector $w_k = \exp(V_k)$ and then compute choice probabilities from w, which greatly speeds up computation because it decreases the number of evaluations of $\exp(\cdot)$, which is an expensive function to compute. We found that the code was more than $10\times$ slower without this optimization on a 2.53 GHz dual-core MacBook Pro with 4 GB of 1067 MHz DDR3 RAM.[46]

By re-expressing the logit as the difference in utilities, $\tilde{f}(V;j) = \dfrac{1}{\sum_k \exp(V_k - V_j)}$, we can solve the stability problem. This specification is equivalent to $f(V;j)$ but much more stable: the difference in utilities are typically small whereas utility itself can be large and lead to overflow. The cost is that we are no longer able to work in terms of $w = \exp(V)$ and must perform more operations. See the code for details. This is a common example of the engineering need to trade-off speed versus robustness.

However, the BLP model uses an outside good with $V_0 = 0$ so the logit is

[43] NaNs result from trying to compute quantities which are undefined such as dividing by zero, $\infty \cdot 0$, $\dfrac{\infty}{\infty}$, and $\infty - \infty$.

[44] Many higher-level languages such as MATLAB treat these error conditions by setting a variable's value to Inf or NaN. However, the researcher must explicitly check for these conditions using `isinf()` and `isnan()`. In general, the CPU generates a floating point exception when these conditions occur. The operating system will then raise the signal `SIGFPE` to the process and the process can either catch the signal by installing a signal handler or ignore it, which is often the default.

[45] In some senses, the literature suffers from selection bias in that researchers who failed to write numerically stable code never publish so we never see these papers.

[46] Test runs on an 8 core Mac Pro with 32 GB of RAM were considerably faster although MATLAB used only two of the cores. Consequently, the bottleneck appears to be swapping and not CPU cycles.

now $f(V;j) = \dfrac{\exp(V_j)}{1+\sum_k \exp(V_k)}$ and, consequently, our trick no longer works. Instead, we use box constraints which are tight enough to prevent the solver from examining regions of parameter space which lead to overflow yet loose enough that we are unlikely to exclude the global optimum. Typically, the box constraints are $\pm 15 \times \left\|\hat{\theta}\right\|$ for θ_1 and θ_2 and $\pm 10^8$ for the other variables, δ and $g = Z'\xi$.

Nevertheless, this does not fully address the underlying problem of exceeding the limits of MATLAB's numerical precision. Recently, we developed a state of the art implementation of BLP in C++ which solves these issues (Skrainka, 2011).[47] This implementation uses MPEC, high performance quadrature rules, and a high quality solver (SNOPT). In addition the code uses the Eigen template library to perform linear algebra efficiently and work in higher precision arithmetic which has twice the precision of MATLAB.[48] Initial investigations show that higher precision completely solves these overflow and underflow problems. Skrainka's implementation also computes the same huge standard errors consistently for all polynomial rules, resolving the difficulty in reliably calculating standard errors which we report in 5.2.3: this problem is an artifact of the double precision arithmetic used by MATLAB.

Lastly, we start the solver at multiple different points which are randomly chosen about the average of the initial point estimates. If the solver converges to the same point for all starts, then it is likely to be the global optimum. On the other hand, if the solver converges to many different points, there are multiple local optima.

5 Results

The polynomial rules out-perform simulation in all respects: they produce more accurate results at much lower computational cost. Of all the rules, the Gauss-Hermite product rule with 7^5 nodes should be considered the 'gold standard' and serves as our benchmark for the other rules because it is exact for degree 13 monomials as well as many higher moments. We obtained broadly similar results for all five Monte Carlo data sets. However, all rules performed consistently well on data set 3. Estimates using data sets 4 and 5 varied considerably based on the quadrature choice, especially the standard errors. Data sets 1 and 2 performed between these two extremes.

We now discuss how the different rules affect computed market shares, point estimates, standard errors, and numerical identification.

[47] This code is available upon request.

[48] Namely, in C++ we code all floating point quantities as 'long double', which is a 16-byte floating point type whereas the default type, 'double', is 8 bytes. Double precision is also what MATLAB uses internally. Because Eigen is a template library, it defines 'generic' operations, making it easy to instantiate the necessary code for whatever type is appropriate. In our code, we use long double, however, it is possible to use higher or lower precision according to the demands of the problem.

5.1 Computation of Predicted Market Shares

One of the first computations we performed was to compute the predicted BLP market share integrals for $T = 50$ markets and $J = 25$ products with each quadrature rule. These results provided both a quick check that our code was performing correctly and a visual comparison of the rules. We computed the market share integrals for each data set at ten different parameter values near the MPEC point estimates, $\hat{\theta}^{MPEC}$. We selected these parameter values by first computing the GMM estimates using MPEC[49] and then computing an additional nine parameter values where θ is drawn from a Normal distribution with $\theta \sim N\left(\hat{\theta}_{MPEC}, \texttt{diag}\left[(0.25)^2 \left\|\hat{\theta}_{MPEC}\right\|\right]\right)$, i.e. the standard errors are 25% of the magnitude of the point estimates. These computations show a cloud of points for the $N = 100$ different pMC calculations of each integral ($R = 1,000$ draws) with the polynomial rules centered in the middle, as we would expect: pMC is unbiased so the polynomial results should be near the average of the Monte Carlo values. Figure 1 plots relative error of the market share integrals at $\hat{\theta}^{MPEC}$ versus mean market share. The relative error is with respect to the mean pMC market share, $\bar{s}_{jt}^{pMC} = \sum_{n=1}^{N} s_{jt}^{pMC(n)}$, where $s_{jt}^{pMC(n)}$ is the n-th replication of (j,t) market share integral computed using pMC. The green circles represent the pMC cloud of values calculated for different replications of a specific (j,t) product-market pair; the magenta pentagon the 7^5 node Gaussian-Hermite product rule; the red and blue triangles the 'Left' and 'Right' Stroud rules; and the yellow diamond the sparse grid integration rule. We only show this figure for data set 1 at $\hat{\theta}^{MPEC}$ because the story is essentially the same for other data sets[50]. These plots clearly show the simulation noise in the computation of market shares s_{jt}: the different MC share values form a green 'pMC cloud' in which the polynomial based rules are located in the center of the cloud. This is exactly where you would expect the true share value to be located because MC is unbiased as $N \to \infty$. Often, it was necessary to magnify the figures many times in order to detect any difference between the polynomial-based rules. This figure demonstrates the much higher accuracy of the monomial and sparse grid rules. An example of a close up for one market share integral is plotted in Figure 2: again, the noise in the pMC calculations and the consistency of the polynomial rules are clearly evident. However, the 'Right' Stroud rule did produce one share value which was far outside the MC cloud and also far from the values for the other polynomial rules. In addition, this rule also produced a negative share value, as discussed in 5.1.2, and was more likely to generate numerically undefined results during optimization.

[49] Historically, point estimates were computed with BLP's nested, fixed point algorithm (NFP). However, we use MPEC to compute our point estimates because it is much more robust and, in theory, both algorithms produce equivalent values at the global optimum (Su and Judd, 2008; Dubé, Fox, and Su, 2009).

[50] One exception is data set 3 which had one share integral whose value was well outside the Monte Carlo cloud when computed with Stroud rule 11-1 Right. The other is data set 5 which has extremely low variance for the integrals computed with pMC.

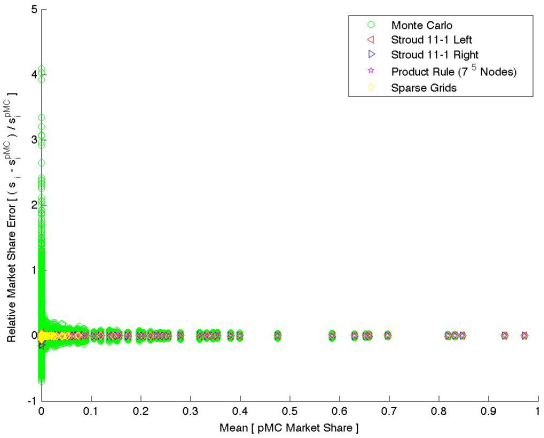

Figure 1: Relative Share Error for Different Quadrature Rules

Figure 1 shows the relative error of the product-market share integrals, s_{jt}, computed using different quadrature rules versus market share, s_{jt}. The relative error is calculated with respect to the mean pMC share value for each integral. The pMC rule is computed with $R = 1,000$ nodes and $N = 100$ replications. Because of simulation noise, these replications form a 'Monte Carlo' cloud about the values computed using the polynomial rules, which are located at the center of these clouds. Note: the error is greatest for shares with small standard errors because these integrals are the most difficult to compute correctly.

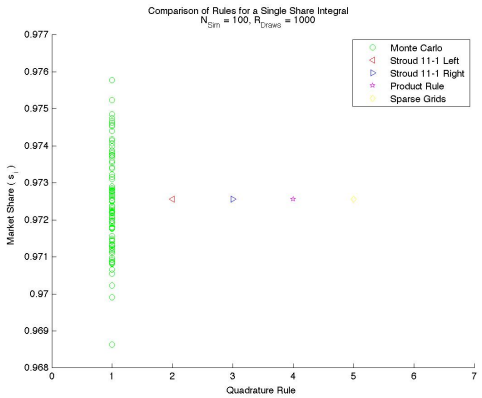

Figure 2: Close-Up of a Market Share Integral.

Figure 2 shows a close-up of the different computed values for a specific product-market share integral. Note how all the polynomial rules compute the exact same value to a very high degree of precision and are located at the center of the Monte Carlo cloud.

To quantify the performance of the different rules, we computed the maximum and average absolute deviation for the predicted share values from the product rule with 7^5 nodes at $\hat{\theta}^{MPEC}$ over all shares.[51] Although, it may not be the best representation of the 'truth', the Gaussian product rule is the most precise rule which we compute because of the additional, higher order terms. Consequently, we use it as our benchmark. Table 3 shows that pMC tends to have higher maximum absolute errors even for a large number of draws and that these errors are several orders of magnitude larger than the monomial and sparse grid rules. Note that even for large numbers of nodes such as $10,000$, which is a large number of draws by contemporary standards, pMC produces much less accurate results, despite using ten times more points. In addition, SGI should perform more like the product rule than the monomial rule because SGI uses a subset of the nodes in the Gaussian product rule, whereas the monomial rule uses entirely different nodes and weights. Furthermore, sparse grids set of nodes drops product rule nodes which are in the tail of the weight function and have little weight on them. Dropping them, if anything, should improve numerical stability because the extremal nodes can cause numerically indeterminate results in the logit.

The results in Table 3 only tell part of the story. The biggest differences between the Gauss-Hermite product rule with 7^5 nodes and the other quadrature rules occur for the largest share values. For larger shares an error of 10% or so appears as a huge maximum absolute error whereas the maximum absolute error for smaller shares may appear small even if a rule differs from the benchmark by several orders of magnitude because the share value is essentially zero . In these cases, relative error is a better measure of performance. Examining the histograms for the maximum absolute error shows that for the polynomial rules there are only a few integrals with significant differences from the benchmark 7^5 node product rule whereas for pMC there is a fat tail of shares which differ considerably.

Tables 4 through 8 show the computational costs (seconds) of computing the point estimates for the different rules.[52] The CPU Time column refers to the total time the solver took to compute an optimum and hence depends on both the speed of the quadrature rule and how quickly the solver converged. We see that the most efficient polynomial rules – SGI and monomial rule 11-1 – are more than a factor of ten faster than the pMC rule with $R = 10,000$ draws as well as being more accurate. pMC with $R = 10,000$ draws and the Gauss-Hermite product rule with 7^5 nodes are both much slower then the monomial and sparse grid rules because they use many more nodes, which primarily determines the increase in computational costs. We discuss the other columns below in 5.2.

Increasing the number of simulation draws from 100 to $10,000$ does little to improve the accuracy of the integral because pMC convergence improves as

[51] I.e., the maximum absolute error is $\max_{j,t} \left[\left| s_{jt}\left(\hat{\theta}^{MPEC}\right) - s_{jt}^{Product\ Rule}\left(\hat{\theta}^{MPEC}\right) \right| \right]$.

[52] Quoted CPU times are for a 2.53 GHz Intel Core 2 Duo MacBook Pro running OS/X 10.6.4 in 64-bit mode with 4 GB of 1067 MHz DDR3 RAM, 6MB L2 cache, and 1.07 GHz bus.

Rule	Type	N_{nodes}	Max Abs Error	Ave Abs Error
Pseudo-Monte Carlo	Simple Random Draws	100	7.28782e-02	6.73236e-04
		1,000	2.59546e-02	2.19284e-04
		10,000	7.05101e-03	6.82600e-05
Product Rule	Gauss-Hermite	$3^5 = 243$	1.20663e-03	1.60235e-05
		$4^5 = 1,024$	2.51442e-04	2.51356e-06
		$5^5 = 3,125$	4.94445e-05	5.42722e-07
		$7^5 = 16,807$	0	0
Monomial Rule 11-1	Left Column	983	1.35871e-02	2.80393e-05
	Right Column	983	1.14304e-02	4.01983e-05
Sparse Grid	Konrad-Patterson ($K = 6$)	993	4.98042e-04	4.09252e-06

Table 3: Comparison of Integration Rules
The columns titled **Max Abs Error** and **Ave Abs Error** refer to the maximum and average absolute error observed for the market shares computed for each rule with respect to the benchmark Gaussian-Hermite product rule with 7 nodes in each of the 5 dimensions. The Monte Carlo rules use $N = 1$ replication. All values are computed at $\hat{\theta}_{MPEC}$ based on $R = 1,000$ draws.

$\sqrt{R}$. Because most of the products have very small market share – for the five synthetic data sets, about 90% of predicted shares are less than 0.01 – we conjecture that only a few products in each market determine the parameter values and that estimating these market shares correctly is necessary in order to obtain accurate point estimates for $\hat{\theta}$. The larger predicted market shares also have larger standard error, where standard error is computed over the N different pMC share replications . The small market shares have smaller errors not because they are calculated more accurately but because they are essentially zero. This effect becomes starker with more simulation draws. Another issue is that the parameter value used to compute the shares will affect which combinations of product and market produce the largest shares. Simple experiments show that 10% or more of shares can move into or out of the top decile of predicted share values.

When comparing the own-price elasticities computed with pMC ($R = 1,000$) and SGI, the values appear very similar (See Figure 3), with most of the difference in elasticities clumped at zero. But, most market share integrals are extremely close to zero. Consequently, we expect elasticties of small shares to be nearly same for both rules, based on the following argument. With linear utility and a simple logit, the own price elasticity is $e_{jt} = -\alpha p_{jt}(1 - s_{jt})$. If $s_{jt} \approx 0$ then $e_{jt} \approx -\alpha p_{jt}$ and the residual should be close to zero. Using this intuition for the mixed logit, even with random coefficients, if the market shares are small then the elasticities are likely to agree. For the larger product-market shares, the deviations in elasticity can be 10% or more, showing that pMC does not approximate the derivatives of the integrals as well as SGI. Results for the

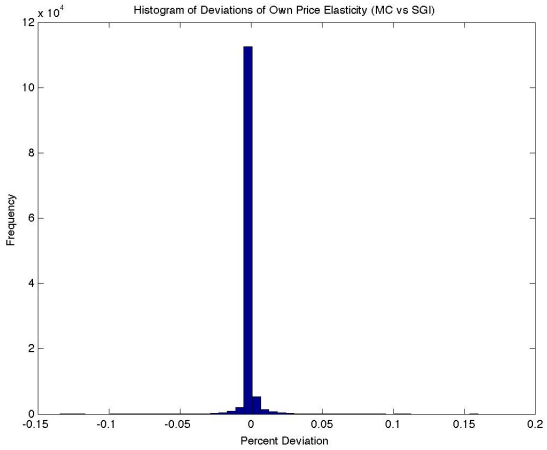

Figure 3: Comparison of Computed Own-Price Elasticities for Sparse Grid and pMC

Figure 3 shows the distribution of residuals which are the difference between the elasticities calculated with polynomial rules and the mean of the pMC share computations for $N = 100$ replications with $R = 1,000$ draws.

monomial rule are identical.

Clearly, the polynomial rules provide a much more accurate approximation of the level and gradient of the market share integrals than pMC. In addition, SGI and monomial rules are much more efficient for a given level of accuracy because they require far fewer nodes than pMC. Furthermore, computing the level of the market share integrals is less important than accurately computing the gradient and Hessian of the objective function and the Jacobian of the constraints because these derivatives determine the optimum – i.e. point estimates and standard errors. As we discuss in 5.2.1, the polynomial-based rules also outperform pMC when approximating higher order derivatives.

5.1.1 Simulation Error and Bias

Numerical integration is an approximation and like all approximations has error. The quality of the a quadrature rule depends on how quickly the rule converges as $R \to \infty$ and the bounds on its error. pMC rules converge as $R^{-1/2}$ (Judd, 1998). Consequently, you must increase the number of nodes R by a factor of 100 to gain an extra decimal place with pMC. For polynomial-based rules, if the Riemann–Stieltjes integral exists, the product rule will converge (Stroud, 1971). Multi-dimensional error bounds formulas do exist but they are sufficiently complicated that Stroud (1971) only states very simplified versions of the theorems. The key point is that the polynomial rules should converge and have much tighter error bounds than MC methods because their error depends

on higher order terms in a Taylor series expansion. Initially, we thought that pMC could out perform polynomial rules when high order terms of the Taylor series of the integrand did not vanish quickly. As we discussed in 2.4, simulation does not preform significantly better than polynomial rules for when these high order terms are significant. Consequently, polynomial rules should have less error for most problems.

Section 3 explained how integration errors can propagate through the model either via the choice probabilities or the gradient of δ (i.e. the inverse of the market shares, $s^{-1}(S;\theta)$. Error enters the share integrals from integrating over the distribution of the random coefficients which affect the BLP model via the mean zero, type-specific preference shock μ_{ijt}. Errors in computing this shock propagate through the model and are further distorted by the multinomial logit transformation which can be flat, concave, or convex depending on parameter values. From Jensen's inequality we know that the expectation of a concave (convex) function is more (less) than the function of the expectation. Consequently, simulation error percolates through the multinomial logit form to produce either positive or negative error. Two facts suggest that pMC causes much more error and bias than the monomial rule: (1) the expectation of μ_{ijt} with a pMC rule is on the order of 10^{-3} even with $R = 10,000$ draws about 10^{-17} for the monomial rule; and (2) the correlation coefficient of μ_{ijt} and the simulation error, $e_{jt} = s_{jt}^{MC} - s_{jt}^{Monomial}$, is about -0.2 conditional on $|e_{jt}| > 10^{-4}$.[53]

5.1.2 Negative Market Shares

Because some weights for monomial and sparse grid rules are negative, the approximation for a market share integral could be negative. However, this is only likely for extremely small shares where the polynomial approximation is poor in a region of parameter space where the integral is essential zero everywhere, i.e. flat. We only observed one negative value out of the $625,000$ integrals calculated.[54] This value was approximately -10^{-9} (i.e. effectively zero) and was occurred with the 'Right' version of the monomial rule.

5.1.3

5.2 Robustness of Point Estimates and Standard Errors

We computed the point estimates and standard errors for each synthetic data set at five starting values.[55] Tables 4-8 show f_k, the value of the GMM

[53] Here, $\text{mean}(\mu_{ijt}) \equiv \frac{1}{R}\sum_i \mu_{ijt}$.

[54] Five Monte Carlo data sets, each with $R = 100$ replications of the $J \times T = 1250$ share integrals results in $5 \times 100 \times 1,250 = 625,000$.

[55] Initially, we simply computed these optima for the same five starting values for each rule and data set. However, the solver often aborted with numerical problems. Imposing box constraints which were sufficiently loose to include a large region of parameter space yet rule out extremal regions of parameter space solved this problem for most quadrature rules and data sets. Many of these numerical problems are caused by floating point underflow/overflow. Ultimately, we resolved the problem by rewriting our code in C++ and using higher precision

objective function at the optimum, as well as the CPU time in seconds required for SNOPT to converge to the point estimate.[56] If the value of f_k is the same for all starts, then the solver has likely found a unique global optimum. For the most accurate rule, the Gauss-Hermite product rule with 7^5 nodes, the solver always finds the same f_k for each start. For SGI and the monomial rule, the solver always found the same optimum for every starting value and data set except for one start for data set 5. Furthermore, both SGI and the monomial rule had the same problematic starting value. pMC, however, typically finds two or three different optima for each data set, even when $R = 10,000$ draws, because Monte Carlo noise creates ripples in the surface of the objective function which generate spurious local maxima. In addition these tables show that sparse grid (993 nodes) and monomial rule (983 nodes) require the same amount of CPU time as pMC with $R = 1,000$ despite being more accurate than pMC with $R = 10,000$. Both of these polynomial rules are also a factor of ten faster than pMC with $R = 10,000$ draws.

The Point estimates[57] (Tables 9-13) also indicate that pMC rules cause false local maxima: by comparing $\hat{\theta}$ for different starts for a given data set, the estimates which have the same value for f_k agree to three or more digits while those with different f_k do not agree at all. On the other hand, the polynomial rules – with the exception of the data set 5's fifth start – agree to many decimal places. pMC point estimates also suffer from increased variation in $\hat{\theta}$, excessively tight standard errors (See 5.2.3), and confidence intervals which do not contain the point estimates from the polynomial rules. In general, the point estimates for $\hat{\theta}_1$ are more often significant than those for $\hat{\theta}_2$, the square root of the diagonal elements of the variance of the random coefficients. That $\hat{\theta}_2$ is not sharply identified could be because the number of markets, T, is 50. With more markets, we would observe more situations where there were similar characteristics but different choices because of variation in the taste shocks.

Furthermore, for each rule there are several data sets where the true data generating process (Table 2.) is not within the confidence interval formed from the point estimates and standard errors. For example, the true value of θ_{11} is 2, but the point estimates for data sets 2, 3, and 5 are never close for any of the rules. The point estimates for $\hat{\theta}_2$ are further from the 'truth' more often than those for $\hat{\theta}_1$. Consequently, the BLP model appears to suffer from finite sample bias. This problem could also be exacerbated because we estimate the model without a pricing equation. Increasing the number of markets, T, should improve the identification of θ_2 in particular because then we will observe more

arithmetic. See 4.1.2.

[56] Quoted CPU times are for a 2.53 GHz Intel Core 2 Duo MacBook Pro running OS/X 10.6.4 in 64-bit mode with 4 GB of 1067 MHz DDR3 RAM, 6MB L2 cache, and 1.07 GHz bus.

[57] Note: sometimes the solver finds negative values for θ_2, which is the square root of the diagonal elements of the variance matrix, Σ, for the random coefficients. In our code θ_2 acts as the scale on the quadrature nodes because we have assumed that Σ is diagonal. The symmetry of the Gaussian kernel means that only the magnitude of θ_2 matters, not the sign. To avoid this confusion, we report $\left|\hat{\theta}_2\right|$ for the point estimates of $\hat{\theta}_2$.

Dataset	EXIT	INFORM	f_k	CPU Time
1	0	1	33.23675	366.80
1	0	1	33.23675	753.15
1	0	1	33.85679	632.36
1	0	1	33.23681	687.31
1	0	1	38.53239	740.08
2	0	1	26.05084	457.01
2	0	1	24.60745	444.16
2	0	1	26.05084	526.40
2	0	1	26.05084	802.80
2	0	1	23.27163	855.66
3	0	1	19.76525	1071.80
3	0	1	19.76526	420.09
3	0	1	19.76524	783.48
3	0	1	19.76528	641.23
3	0	1	19.76524	635.87
4	0	1	28.19951	654.80
4	0	1	28.19951	1081.98
4	0	1	28.19951	820.40
4	0	1	28.19951	810.95
4	0	1	28.19951	796.42
5	0	1	203.50784	668.71
5	0	1	213.97591	503.92
5	0	1	203.50784	626.74
5	0	1	208.76144	489.06
5	0	1	208.76144	696.81

Table 4: Point Estimates: pMC with First 5 good starts and $R = 1,000$ draws

situations where agents with similar attributes make different decisions. It would also be useful to compare the GMM standard errors to bootstrap standard errors.

We now look at these issues in more detail in 5.2.1 and 5.2.3.

5.2.1 Impact of Quadrature on Optimization

One of encouraging result of our experiments is that the point estimates computed via MPEC + SNOPT for the polynomial-based rules are always the same for all starting values when the solver found a valid solution, unlike the pMC rules whose point estimates varied widely depending on the starting value. In general, SNOPT and KNITRO encountered numerical difficulties – typically an undefined computation for a conditional logit share of the form ∞/∞ – more often with the polynomial-based rules than pMC because the polynomial rules have better coverage of extreme areas of the parameter space, even though the

Dataset	EXIT	INFORM	f_k	CPU Time
1	0	1	34.75771	8035.81
1	0	1	34.75772	5744.15
1	0	1	34.75771	3308.05
1	0	1	33.71660	3341.52
1	0	1	34.75776	7782.77
2	0	1	23.33186	7666.86
2	0	1	23.13213	6793.78
2	0	1	22.66818	7161.72
2	0	1	23.24129	7053.06
2	0	1	23.76645	8901.79
3	0	1	21.64541	8376.50
3	0	1	21.58369	8265.87
3	10	72	294.95326	178.32
3	0	1	21.69790	6567.52
3	0	1	21.95653	7835.28
4	0	1	22.49406	7955.48
4	0	1	22.49407	5446.51
4	0	1	26.12617	6544.76
4	0	1	26.12617	7427.27
4	0	1	26.22725	6852.28
5	0	1	260.57447	5450.45
5	0	1	279.95232	6514.08
5	0	1	299.22156	5555.86
5	0	1	299.22156	5444.99
5	0	1	279.95232	4403.82

Table 5: Point Estimates: pMC with $R = 10,000$ draws

weights are quite small.[58] That the pMC point estimates vary widely depending on starting values indicates that the solver is finding false local maxima because of the inaccuracies of the pseudo-Monte Carlo approximation to the integral.

Another important issue is that approximating the share integrals is less important than accurately computing the gradient and Hessian of the GMM objective function and the Jacobian of the constraints which the solver uses to find a local optimum. The product rule and SGI affect solver convergence similarly, which is unsurprising because the SGI nodes, as mentioned in 2.3.2, are a subset of the product rule nodes. By omitting the nodes in the corners of the product rule lattice, SGI is less likely to evaluate the objective function, constraints, or the gradients at extremal points which produce NaNs and cause

[58] Dubé, Fox, and Su (2009) side-step these issues to some extent by using the MC draws which were used to generated their synthetic data to compute the market share integrals. Clearly, in a real world problem these shocks would not be observed by the econometrician. When I redraw these shocks, their code produces NaNs for some starting values. In addition, they use the same set of draws for each market share integral, s_{jt}.

Dataset	EXIT	INFORM	f_k	CPU Time
1	0	1	35.05646	7083.95
1	0	1	35.05639	9142.16
1	0	1	35.05644	4940.91
1	0	1	35.05639	6184.56
1	0	1	35.05651	5952.06
2	0	1	22.98928	15317.16
2	0	1	22.98929	14141.56
2	0	1	22.98927	14354.17
2	0	1	22.98928	9736.57
2	0	1	22.98928	10742.86
3	0	1	21.77869	7306.40
3	0	1	21.77873	6992.33
3	0	1	21.77872	5968.52
3	0	1	21.77869	5154.03
3	0	1	21.77870	6979.46
4	0	1	25.63232	7653.30
4	0	1	25.63232	6574.78
4	0	1	25.63232	8695.48
4	0	1	25.63232	6739.00
4	0	1	25.63232	9277.51
5	0	1	288.69920	6334.33
5	0	1	288.69920	7553.43
5	0	1	288.69920	7164.02
5	0	1	288.69920	8156.16
5	0	1	288.69920	5521.13

Table 6: Point Estimates: Gauss-Hermite with first 5 good starts and 7^5 nodes

the solver to abort.[59]

Note that increasing the number of draws to $R = 10,000$ does not significantly improve the optimum found by SNOPT with a pMC rule (Tables 9 and 10). Many of the point estimates and values of the optimum at the solution still vary considerably depending on the starting value even when the solver reports that it has found a local optimum. Clearly, even with $10,000$ draws, pMC still introduces spurious local optima.

In the MPEC formulation of BLP, quadrature only affects the problem via the constraint equating observed and predicted market shares. With simulation, this constraint will be noisier and have local areas where simulation errors make it possible to find a local optima. A key assumption of Gandhi (2010)'s proof that the market share equation is invertible is monotonicity which fails in this case. Furthermore, the optimizer adjusts parameters so that the spectrum of the mapping is less singular and has local basins of attraction. The different

[59] If SNOPT encounters a NaN it will abort with EXIT=10 and INFO=72.

Dataset	EXIT	INFORM	f_k	CPU Time
1	0	1	35.27236	616.63
1	0	1	35.27217	549.22
1	0	1	35.27212	269.22
1	0	1	35.27216	414.71
1	0	1	35.27212	432.32
2	0	1	22.97539	980.88
2	0	1	22.97541	910.89
2	0	1	22.97539	724.68
2	0	1	22.97539	865.45
2	0	1	22.97540	1026.54
3	0	1	21.78433	433.50
3	0	1	21.78430	557.89
3	0	1	21.78432	610.45
3	0	1	21.78437	352.71
3	0	1	21.78434	604.79
4	0	1	25.59501	515.58
4	0	1	25.59501	388.67
4	0	1	25.59501	496.07
4	0	1	25.59501	439.85
4	0	1	25.59501	586.94
5	0	1	293.89029	494.45
5	0	1	293.89029	571.11
5	0	1	293.89029	481.82
5	0	1	293.89029	556.80
5	0	1	487.40742	6535.28

Table 7: Point Estimates: SGI with first 5 good starts and 993 nodes (exact for degree ≤ 11)

sizes of these basins affect how often solver finds them when searching for a local minimum of the GMM objective function. We found that SNOPT could often find an optimum when KNITRO would not converge because SNOPT uses a sequential quadratic programming method which is more robust than KNITRO's interior point method when the objective function or constraints are non-convex. In addition, SNOPT 7 was recently upgraded to handle rank deficient systems: we found that enabling LU rook pivoting in SNOPT, although a factor of two slower than the default algorithm, enabled the solver to find a valid solution more often. [60]

[60] We also found some evidence that a pMC rule may make the objective function more sensitive to variations in the parameters θ, but more research is required to resolve this issue definitively.

Dataset	EXIT	INFORM	f_k	CPU Time
1	0	1	34.63546	644.52
1	0	1	34.63550	578.42
1	0	1	34.63556	449.30
1	0	1	34.63552	294.48
1	0	1	34.63548	443.86
2	0	1	23.24928	1174.66
2	0	1	23.24928	660.97
2	0	1	23.24928	922.52
2	0	1	23.24928	1150.00
2	0	1	23.24928	1022.48
3	0	1	21.79928	688.71
3	0	1	21.79931	373.36
3	0	1	21.79926	669.28
3	0	1	21.79926	483.89
3	0	1	21.79926	573.57
4	0	1	24.72862	435.54
4	0	1	24.72862	587.55
4	0	1	24.72862	739.98
4	0	1	24.72862	613.63
4	0	1	24.72862	657.03
5	0	1	277.89463	441.45
5	0	1	278.03790	441.77
5	0	1	277.89463	548.75
5	0	1	277.89463	1134.53
5	0	1	278.03790	656.13

Table 8: Point Estimates: Monomial with First 5 Good Starts.

5.2.2 Differences in Objective Function Values

We were initially surprised to discover in Tables 4-8 that the objective function values, f_k, do not agree at the optimal point estimates. This occurs because the value of the objective function in the MPEC formulation is $g^{'}Wg$ where $g = Z^{'}\xi$ are the moment conditions. Using MPEC, we solve for g as part of the optimization program: consequently, differences in g across quadrature rules at the local optimum found by the solver produce different values of the objective function. Errors in computing ξ – whether numerical or due to model misspecification – accumulate and affect the optimal value of the moment conditions which in turn produce different values of the objective function. Initial investigations with a new C++ implementation using quad precision arithmetic and LU rook pivoting to increase solver stability appear to eliminate these differences so that only about 10 out 1302 values of the solver solution $(\theta_1, \theta_2, \delta, g)$ differ by more than 2%. However when using MATLAB, $\hat{\xi}$ varies considerably at the optima found by SNOPT even when $\hat{\theta}$ does not.

5.2.3 Simulation, Identification, and Standard Errors

When computing standard errors, we found that simulation – unlike the polynomial rules – produces excessively tight values and will lead researchers to think that some parameters are well identified when they are not. Examining the standard errors (Tables 9-13) shows that, in general, the pMC standard errors are much smaller than those computed with polynomial rules. For some data sets, such as data sets 4 and 5, the polynomial rules produce standard errors on the order of 10^4 or larger vs. pMC errors of 10^{-1} even with using $R = 10,000$ draws. For example, compare the results for $\hat{\theta}_{21}$ and $\hat{\theta}_{24}$ for data set 4 and $\hat{\theta}_{21}$ and $\hat{\theta}_{23}$ for data set 5. Standard errors computed using pMC show apparently tight identification when in fact the Hessian is ill-conditioned. Because pMC introduces spurious local optima and, concomitantly, pockets of higher curvature it produces standard errors which are too tight.[61] Consequently, pMC can mask poor identification and practitioners will miss an important diagnostic that the objective function is nearly flat because pMC does not approximate the gradient and Hessian well. In fact, as the order of differentiation increases, pMC performs increasingly poorly. Polynomial-based rules do not suffer from this problem because they approximate the level, gradient, and Hessian correctly: if a parameter had huge standard errors for one data set and rule, then it had huge the standard errors for all rules and starts. Nevertheless, the quadrature rules did not reliably detect large standard errors: the Gauss-Hermite product rule with 7^5 nodes detected 4 cases out of $10 \times 5 = 50$ parameters estimated;[62] SGI and the monomial rule found 3 of the 4 found by the product rule; and, pMC, even with $R = 10,000$ draws, failed to find any. Recently, we began replicated these results using the higher precision BLP implementation in Skrainka (2011). Our initial results show that when using higher precision arithmetic, all of the polynomial rules reliably compute the same large standard errors for the same data sets and parameters, θ. Even with this BLP implementation, the pMC rules still produce anomalously tight standard errors. Consequently, pMC quadrature rules will cause a downward bias in standard errors and mask (numerical) identification problems. Note, too, that because pMC produces standard errors which are too tight, pMC will not produce reliable standard errors with the bootstrap. Instead, polynomial-based rules are a better choice because of their increased accuracy and efficiency.

In BLP, the substitution patterns are 'diffuse' and all goods are substitutes for each other as opposed to the 'local' substitution patterns in pure characteristics models, where cross-price elasticities are non-zero for only a finite set of products (E.g., Berry and Pakes (2007); Shaked and Sutton (1982)). Consequently, the model is very sensitive to both sampling error in the observed market shares, S_{jt}, and simulation error in the computation of predicted market shares, s_{jt} (Berry, Linton, and Pakes, 2004). Particularly as J increases (And, Berry, Linton, and Pakes (2004) require $J \to \infty$ to prove that BLP is

[61] Just examine the formula for GMM standard errors which depends on the inverse of the Hessian.

[62] I.e., we estimated ten parameters, $\hat{\theta}$, for five starts for each rule and data set

consistent and asymptotically normal) small simulation or sampling errors considerably affect the value of ξ which is computed in the traditional NFP BLP implementations.

The small sample properties of GMM is another potential source of difficulty in estimating θ_2 parameters well. Altonji and Segal (1996) show that optimal minimum distance (OMD) estimators – i.e. GMM with the optimal weighting matrix – perform poorly in small sample estimates of the variance because the shocks which make the variance large also tend to increase the variance of the variance. Consequently, because θ_2 measures the standard error of the random coefficients it is probably estimated with downward bias. This correlation could also explain why $\text{Var}(\theta_2)$ is often surprisingly large: when estimation uses more accurate, polynomial rules is not masked by false correlation from simulation.

6 Conclusion

A head-to-head comparison of Monte Carlo and polynomial-based quadrature rules for approximating multi-dimensional integrals of moderate size shows that monomial rules provide superior accuracy at a computation cost which is at least an order of magnitude smaller. Monomial rules are marginally more difficult to implement than pMC, requiring a few well-defined permutations of the nodes and weights found in a table look-up. An even easier option is to use sparse grid integration which can generate a set of nodes and weights that provide comparable accuracy, often with only a few more nodes. An important area for future research is to develop monomial rules which explicit some of the common structure of economic problems such using functions which are smooth, bounded, and analytic.

When we applied these quadrature methods to BLP, it became clear that the choice of quadrature rule affects the model's results, including the computed value of product-market share integrals, the values of the point estimates, and the standard errors. In particular, pseudo-Monte Carlo rules produce very different point estimates for different starting values – even with very large numbers of draws – unlike the polynomial rules which always produce the same optimum. pMC rules also generate excessively tight standard errors potentially hiding an identification problem in the local basins of attraction created by the noisiness of Monte Carlo integration.

Using a high-quality quadrature rule, then, provides an easy way to improve the accuracy and efficiency of many numerical projects.

References

Altonji, J., and L. Segal (1996): "Small-sample bias in GMM estimation of covariance structures," Journal of Business & Economic Statistics, 14(3), 353–366.

Berry, S., P. Haile, and N. B. of Economic Research (2010):

Identification in differentiated products markets using market level data. National Bureau of Economic Research Cambridge, Mass., USA.

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

——— (2004): "Differentiated Products Demand Systems from a Combination of Micro and Macro Data: the New Car Market," Journal of Political Economy, 112(1), 68–105.

Berry, S., O. B. Linton, and A. Pakes (2004): "Limit Theorems for Estimating the Parameters of Differentiated Product Demand Systems," Review of Economic Studies, 71(3), 613–654.

Berry, S., and A. Pakes (2007): "The Pure Characteristics Demand Model," International Economic Review, 48(4), 1193–1225.

Burda, M., M. Harding, and J. Hausman (2008): "A Bayesian mixed logit–probit model for multinomial choice," Journal of Econometrics, 147(2), 232–246.

Byrd, R., J. Nocedal, and R. Waltz (2006): "Knitro: An Integrated Package for Nonlinear Optimization," Large-scale nonlinear optimization, pp. 35–59.

Conlon, C. T. (2010): "A Dynamic Model of Costs and Margins in the LCD TV Industry," .

Cools, R. (1997): "Constructing Cubature Formulae: the Science Behind the Art," Acta Numerica 1997, pp. 1–54.

——— (2002): "Advances in Multidimensional Integration," Journal of Computational and Applied Mathematics, 149(1), 1–12.

Cools, R. (2003): "An Encyclopaedia of Cubature Formulas," Journal of Complexity, 19(3), 445.

Davis, P., and E. Garcés (2009): Quantitative techniques for competition and antitrust analysis. Princeton University Press.

Dubé, J.-P. H., J. T. Fox, and C.-L. Su (2009): "Improving the Numerical Performance of BLP Static and Dynamic Discrete Choice Random Coefficients Demand Estimation," SSRN eLibrary.

Gandhi, A. (2010): "Inverting Demand in Product Differentiated Markets," Draft.

Genz, A. (1993): "Comparison of Methods for the Computation of Multivariate Normal Probabilities," Computing Science and Statistics, pp. 400–400.

Gill, P., W. Murray, and M. Saunders (2002): "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," SIAM Journal on Optimization, 12(4), 979–1006.

Heiss, F., and V. Winschel (2008): "Likelihood Approximation by Numerical Integration on Sparse Grids," Journal of Econometrics, 144(1), 62–80.

Jiang, R., P. Manchanda, and P. Rossi (2009): "Bayesian Analysis of Random Coefficient Logit Models Using Aggregate Data," Journal of Econometrics, 149(2), 136–148.

Judd, K. (1998): Numerical Methods in Economics. The MIT Press.

Matsumoto, M., and T. Nishimura (1998): "Mersenne twister," ACM transactions on Modeling and Computer Simulation, 8(1), 3–30.

McFadden, D. (1989): "A Method of Simulated Moments for Estimation of Discrete Response Models without Numerical Integration," Econometrica: Journal of the Econometric Society, pp. 995–1026.

McFadden, D., and K. Train (2000): "Mixed MNL Models for Discrete Response," Journal of Applied Econometrics, 15(5), 447–470.

Nevo, A. (2000a): "A Practitioner's Guide to Estimation of Random-Coefficients Logit Models of Demand," Journal of Economics & Management Strategy, 9(4), 513–548.

——— (2000b): "Mergers with differentiated products: The case of the ready-to-eat cereal industry," The RAND Journal of Economics, 31(3), 395–421.

——— (2001): "Measuring market power in the ready-to-eat cereal industry," Econometrica, pp. 307–342.

Nocedal, J., and S. Wright (2000): Numerical Optimization. Springer.

Nosko, C. (2010): "Competition and Quality Choice in the CPU Market," Manuscript, Harvard University.

Petrin, A. (2002): "Quantifying the benefits of new products: The case of the minivan," Journal of Political Economy, 110(4), 705–729.

Petrin, A., and K. Train (2006): "A control function approach to endogeneity in consumer choice models," Journal of Marketing Research.

Press, W., S. Teukolsky, W. Vetterling, and B. Flannery (2007): Numerical recipes: the art of scientific computing. Cambridge Univ Pr.

Reynaerts, J., R. Varadhan, and J. Nash (2010): "The Convergence Properties of the BLP (1995) Contraction Mapping and Alternative Algorithms in R," .

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

Shaked, A., and J. Sutton (1982): "Relaxing price competition through product differentiation," The Review of Economic Studies, 49(1), 3–13.

Skrainka, B. S. (2011): "Finite Sample Performance of a Popular Model of Product Differentiation," Working Paper.

Stroud, A. (1971): Approximate Calculation of Multiple Integrals. Prentice Hall.

Su, C.-L., and K. L. Judd (2008): "Constrained Optimization Approaches to Estimation of Structural Models," SSRN eLibrary.

Train, K. (1999): "Halton Sequences for Mixed Logit," Manuscript, Department of Economics, University of Berkeley.

Train, K. E. (2009): Discrete Choice Methods with Simulation. Cambridge University Press, "second edition" edn.

Data	INFORM	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{21}	θ_{22}	θ_{23}	θ_{24}	θ_{25}
1	1	1.0546	1.577	1.6490	0.09924	-2.594	1.0244	0.5553	0.5823	0.53126	0.2210
		(0.3494)	(0.1204)	(0.1337)	(0.09090)	(0.1465)	(0.4602)	(0.1198)	(0.09144)	(0.09601)	(0.06035)
1	1	1.0545	1.577	1.6489	0.09922	-2.594	1.0243	0.5553	0.5823	0.53125	0.2210
		(0.3494)	(0.1204)	(0.1337)	(0.09090)	(0.1465)	(0.4602)	(0.1198)	(0.09144)	(0.09602)	(0.06035)
1	1	0.9578	1.551	1.5827	0.11983	-2.551	1.0862	0.6529	0.3769	0.47002	0.2003
		(0.3249)	(0.1205)	(0.1260)	(0.09065)	(0.1332)	(0.4235)	(0.07828)	(0.1053)	(0.1095)	(0.05769)
1	1	1.0546	1.577	1.6489	0.09924	-2.594	1.0244	0.5553	0.5822	0.53122	0.2210
		(0.3494)	(0.1204)	(0.1337)	(0.09091)	(0.1465)	(0.4602)	(0.1198)	(0.09145)	(0.09604)	(0.06036)
1	1	0.9948	1.534	1.6011	0.07623	-2.564	0.9913	0.6167	0.4642	0.50064	0.2080
		(0.3667)	(0.1216)	(0.1282)	(0.09236)	(0.1457)	(0.4531)	(0.07804)	(0.08660)	(0.1206)	(0.06271)
2	1	0.5957	1.179	0.9327	0.25733	-2.311	0.6642	0.4303	0.8534	0.70568	0.1183
		(0.3597)	(0.1895)	(0.1781)	(0.1204)	(0.1278)	(0.7046)	(0.1577)	(0.06573)	(0.04496)	(0.09680)
2	1	0.7719	1.221	0.9475	0.26271	-2.396	0.7866	0.3935	0.8336	0.70480	0.1750
		(0.4212)	(0.1944)	(0.1788)	(0.1247)	(0.1807)	(0.6731)	(0.1573)	(0.07146)	(0.03602)	(0.1015)
2	1	0.5957	1.179	0.9328	0.25735	-2.311	0.6641	0.4303	0.8534	0.70568	0.1183
		(0.3597)	(0.1895)	(0.1781)	(0.1204)	(0.1278)	(0.7046)	(0.1577)	(0.06573)	(0.04496)	(0.09679)
2	1	0.5956	1.179	0.9327	0.25733	-2.311	0.6642	0.4303	0.8534	0.70568	0.1183
		(0.3597)	(0.1895)	(0.1781)	(0.1204)	(0.1278)	(0.7046)	(0.1577)	(0.06573)	(0.04496)	(0.09680)
2	1	0.8388	1.253	1.0184	0.26314	-2.439	0.8765	0.4185	0.8932	0.67489	0.1859
		(0.4100)	(0.1976)	(0.1862)	(0.1227)	(0.1733)	(0.5198)	(0.1464)	(0.06769)	(0.03969)	(0.08145)
3	1	1.3104	1.136	1.3428	0.66329	-2.437	0.6895	0.2544	0.9054	0.40733	0.1883
		(0.3115)	(0.1497)	(0.1706)	(0.1209)	(0.1270)	(0.7110)	(0.1604)	(0.04894)	(0.1104)	(0.05616)
3	1	1.3105	1.136	1.3428	0.66332	-2.437	0.6892	0.2544	0.9055	0.40733	0.1883
		(0.3115)	(0.1497)	(0.1706)	(0.1209)	(0.1270)	(0.7113)	(0.1604)	(0.04894)	(0.1104)	(0.05615)
3	1	1.3104	1.136	1.3428	0.66331	-2.437	0.6894	0.2544	0.9054	0.40732	0.1883
		(0.3115)	(0.1497)	(0.1706)	(0.1209)	(0.1270)	(0.7111)	(0.1604)	(0.04895)	(0.1104)	(0.05616)
3	1	1.3105	1.136	1.3428	0.66336	-2.437	0.6890	0.2544	0.9054	0.40730	0.1883
		(0.3115)	(0.1497)	(0.1706)	(0.1209)	(0.1270)	(0.7115)	(0.1605)	(0.04895)	(0.1105)	(0.05616)
3	1	1.3104	1.136	1.3428	0.66331	-2.437	0.6893	0.2544	0.9054	0.40731	0.1883
		(0.3115)	(0.1497)	(0.1706)	(0.1209)	(0.1270)	(0.7112)	(0.1604)	(0.04895)	(0.1104)	(0.05616)
4	1	1.6379	1.832	1.6840	0.26823	-3.021	0.7794	1.0241	0.6984	0.08211	0.6708
		(0.2955)	(0.2845)	(0.2787)	(0.2073)	(0.2661)	(1.070)	(0.1224)	(0.1609)	(0.3069)	(0.1024)
4	1	1.6379	1.832	1.6840	0.26823	-3.021	0.7794	1.0241	0.6984	0.08212	0.6708
		(0.2955)	(0.2845)	(0.2787)	(0.2073)	(0.2661)	(1.070)	(0.1224)	(0.1609)	(0.3069)	(0.1024)
4	1	1.6379	1.832	1.6840	0.26823	-3.021	0.7795	1.0241	0.6984	0.08213	0.6708
		(0.2955)	(0.2845)	(0.2787)	(0.2073)	(0.2661)	(1.070)	(0.1224)	(0.1609)	(0.3069)	(0.1024)
4	1	1.6379	1.832	1.6840	0.26822	-3.021	0.7794	1.0241	0.6984	0.08211	0.6708
		(0.2955)	(0.2845)	(0.2787)	(0.2073)	(0.2661)	(1.070)	(0.1224)	(0.1609)	(0.3069)	(0.1024)
4	1	1.6379	1.832	1.6840	0.26824	-3.021	0.7794	1.0241	0.6984	0.08212	0.6708
		(0.2955)	(0.2845)	(0.2787)	(0.2073)	(0.2661)	(1.070)	(0.1224)	(0.1609)	(0.3069)	(0.1024)
5	1	3.8847	2.941	1.7169	0.75841	-5.028	1.4322	1.2450	0.8917	1.21940	1.2161
		(0.7081)	(0.2525)	(0.2558)	(0.1694)	(0.5138)	(1.019)	(0.3340)	(0.1194)	(0.1908)	(0.1532)
5	1	0.1228	2.608	1.6139	0.24177	-2.575	1.5673	0.9745	0.7653	0.59988	0.4921
		(0.4817)	(0.1932)	(0.2031)	(0.1285)	(0.1604)	(0.6346)	(0.1237)	(0.1238)	(0.1533)	(0.04807)
5	1	3.8847	2.941	1.7169	0.75842	-5.028	1.4323	1.2450	0.8917	1.21941	1.2161
		(0.7081)	(0.2525)	(0.2558)	(0.1694)	(0.5138)	(1.019)	(0.3340)	(0.1194)	(0.1908)	(0.1532)
5	1	1.9287	2.693	1.5499	0.73407	-3.911	2.3156	1.4281	0.7147	1.01829	0.8986
		(0.6686)	(0.2155)	(0.2226)	(0.1501)	(0.3918)	(0.5851)	(0.1760)	(0.1347)	(0.1880)	(0.1172)
5	1	1.9287	2.693	1.5499	0.73408	-3.911	2.3156	1.4281	0.7147	1.01830	0.8986
		(0.6686)	(0.2155)	(0.2226)	(0.1501)	(0.3918)	(0.5851)	(0.1760)	(0.1347)	(0.1880)	(0.1172)

Table 9: Point Estimates: pMC with First 5 good starts and $R = 1,000$ draws

Data	INFORM	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{21}	θ_{22}	θ_{23}	θ_{24}	θ_{25}
1	1	1.1427	1.5961	1.6348	0.08060	-2.607	1.00107	0.568936	0.5400	0.52351	0.2280
		(0.3011)	(0.1204)	(0.1260)	(0.09181)	(0.1185)	(0.4420)	(0.08408)	(0.08639)	(0.1115)	(0.04491)
1	1	1.1429	1.5962	1.6349	0.08061	-2.607	1.00077	0.568898	0.5401	0.52355	0.2280
		(0.2924)	(0.1192)	(0.1255)	(0.08935)	(0.1148)	(0.4597)	(0.08346)	(0.1001)	(0.09312)	(0.04868)
1	1	1.1429	1.5961	1.6349	0.08062	-2.607	1.00091	0.568914	0.5400	0.52354	0.2280
		(0.3278)	(0.1214)	(0.1296)	(0.09147)	(0.1353)	(0.4421)	(0.08909)	(0.09554)	(0.1115)	(0.04851)
1	1	1.1990	1.5906	1.6269	0.12819	-2.640	1.03337	0.613779	0.5108	0.50231	0.2434
		(0.2736)	(0.1199)	(0.1254)	(0.09026)	(0.1007)	(0.4469)	(0.08230)	(0.09002)	(0.1109)	(0.03897)
1	1	1.1428	1.5961	1.6348	0.08065	-2.607	1.00089	0.568973	0.5400	0.52354	0.2280
		(0.2781)	(0.1205)	(0.1264)	(0.09103)	(0.1078)	(0.4450)	(0.08310)	(0.09000)	(0.1100)	(0.04192)
2	1	0.6241	1.2062	1.0216	0.24133	-2.362	0.93166	0.374587	0.9034	0.71462	0.1442
		(0.3252)	(0.1928)	(0.1763)	(0.1199)	(0.1183)	(0.6059)	(0.1373)	(0.06899)	(0.03736)	(0.06459)
2	1	0.6640	1.2148	1.0039	0.24668	-2.375	0.96484	0.361393	0.8878	0.70441	0.1501
		(0.3252)	(0.1928)	(0.1763)	(0.1199)	(0.1183)	(0.6058)	(0.1373)	(0.06899)	(0.03736)	(0.06458)
2	1	0.7631	1.2212	0.9776	0.30353	-2.421	1.00189	0.403325	0.8816	0.69759	0.1685
		(0.3443)	(0.1944)	(0.1753)	(0.1222)	(0.1261)	(0.5700)	(0.1431)	(0.07513)	(0.03758)	(0.05836)
2	1	0.6603	1.2091	1.0048	0.24541	-2.376	0.95143	0.389741	0.8906	0.72306	0.1521
		(0.3370)	(0.1928)	(0.1763)	(0.1212)	(0.1245)	(0.6021)	(0.1416)	(0.06842)	(0.03572)	(0.06375)
2	1	0.7000	1.2057	1.0241	0.28422	-2.392	0.90694	0.400220	0.9065	0.70525	0.1626
		(0.3370)	(0.1928)	(0.1763)	(0.1212)	(0.1245)	(0.6022)	(0.1416)	(0.06842)	(0.03572)	(0.06373)
3	1	1.3384	1.1197	1.3602	0.70373	-2.432	0.53518	0.242688	0.8573	0.39160	0.1934
		(0.3173)	(0.1539)	(0.1649)	(0.1235)	(0.1298)	(0.6335)	(0.1424)	(0.04434)	(0.1097)	(0.05803)
3	1	1.2696	1.1225	1.3489	0.69837	-2.412	0.56292	0.278950	0.8384	0.39886	0.1824
		(0.3173)	(0.1539)	(0.1649)	(0.1235)	(0.1298)	(0.6336)	(0.1424)	(0.04434)	(0.1097)	(0.05803)
3	72	5.5266	0.9599	1.3286	0.98834	-4.695	0.04864	0.009567	0.2000	0.22681	0.8772
		(0.3177)	(0.1501)	(0.1634)	(0.1217)	(0.1285)	(0.8505)	(0.1510)	(0.04468)	(0.1048)	(0.06012)
3	1	1.2687	1.1307	1.3555	0.69063	-2.400	0.47493	0.284326	0.8512	0.40623	0.1730
		(9.844)	(4.984)	(5.185)	(3.981)	(3.441)	(9.323)	(1.500)	(4.053)	(2.119)	(1.143)
3	1	1.2232	1.1134	1.3305	0.69086	-2.406	0.68129	0.291198	0.8282	0.41123	0.1803
		(0.3169)	(0.1503)	(0.1632)	(0.1221)	(0.1356)	(0.7218)	(0.1247)	(0.04699)	(0.1032)	(0.06299)
4	1	1.8402	1.8307	1.7406	0.27790	-2.779	0.27336	0.814492	0.6361	0.04740	0.5035
		(0.2571)	(0.2980)	(0.2964)	(0.1870)	(0.1182)	(0.7049)	(0.09593)	(0.1270)	(0.3194)	(0.03201)
4	1	1.8403	1.8307	1.7405	0.27789	-2.779	0.27333	0.814509	0.6361	0.04740	0.5035
		(0.2571)	(0.2980)	(0.2964)	(0.1870)	(0.1182)	(0.7050)	(0.09593)	(0.1270)	(0.3194)	(0.03201)
4	1	1.7598	1.8252	1.7178	0.26449	-2.731	0.53581	0.898076	0.4578	0.08526	0.4867
		(0.2571)	(0.2980)	(0.2964)	(0.1870)	(0.1182)	(0.7049)	(0.09593)	(0.1270)	(0.3195)	(0.03201)
4	1	1.7598	1.8252	1.7178	0.26449	-2.731	0.53586	0.898080	0.4579	0.08527	0.4867
		(0.2571)	(0.2980)	(0.2964)	(0.1870)	(0.1182)	(0.7049)	(0.09593)	(0.1270)	(0.3195)	(0.03201)
4	1	1.7619	1.8359	1.7187	0.26380	-2.715	0.34868	0.892344	0.4476	0.10942	0.4802
		(0.2571)	(0.2980)	(0.2964)	(0.1870)	(0.1182)	(0.7051)	(0.09593)	(0.1270)	(0.3195)	(0.03201)
5	1	-0.2568	2.3403	1.1956	0.32084	-2.217	0.61026	1.101380	0.2332	0.56325	0.3884
		(0.4224)	(0.1931)	(0.1949)	(0.1170)	(0.1624)	(2.189)	(0.09419)	(0.2608)	(0.1058)	(0.04790)
5	1	0.3066	2.5059	1.2591	0.43423	-2.707	1.39306	1.193434	0.3408	0.52833	0.5531
		(0.5113)	(0.1932)	(0.1962)	(0.1176)	(0.1620)	(0.6929)	(0.08841)	(0.1647)	(0.1012)	(0.04609)
5	1	-0.7372	2.3060	1.1838	0.34910	-2.073	0.79228	0.992318	0.4481	0.77180	0.3472
		(0.4224)	(0.1931)	(0.1949)	(0.1170)	(0.1624)	(2.189)	(0.09419)	(0.2608)	(0.1058)	(0.04790)
5	1	-0.7372	2.3060	1.1838	0.34910	-2.073	0.79228	0.992316	0.4481	0.77180	0.3472
		(0.4224)	(0.1931)	(0.1949)	(0.1170)	(0.1624)	(2.189)	(0.09419)	(0.2608)	(0.1058)	(0.04790)
5	1	0.3066	2.5059	1.2591	0.43423	-2.707	1.39306	1.193433	0.3408	0.52833	0.5531
		(0.5113)	(0.1932)	(0.1962)	(0.1176)	(0.1620)	(0.6929)	(0.08841)	(0.1647)	(0.1012)	(0.04609)

Table 10: Point Estimates: pMC with $R = 10,000$ draws

Data	INFORM	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{21}	θ_{22}	θ_{23}	θ_{24}	θ_{25}
1	1	1.1304	1.596	1.6074	0.1013	-2.610	1.062e+00	0.5942	4.613e-01	5.178e-01	0.2307
		(0.3030)	(0.1197)	(0.1268)	(0.09048)	(0.1160)	(0.4376)	(0.08412)	(0.09132)	(0.1031)	(0.04396)
1	1	1.1304	1.596	1.6074	0.1014	-2.610	1.062e+00	0.5942	4.613e-01	5.178e-01	0.2306
		(0.3030)	(0.1197)	(0.1268)	(0.09049)	(0.1160)	(0.4376)	(0.08413)	(0.09131)	(0.1031)	(0.04396)
1	1	1.1304	1.596	1.6075	0.1013	-2.610	1.062e+00	0.5942	4.613e-01	5.178e-01	0.2306
		(0.3030)	(0.1197)	(0.1268)	(0.09050)	(0.1160)	(0.4377)	(0.08414)	(0.09134)	(0.1032)	(0.04398)
1	1	1.1306	1.596	1.6074	0.1013	-2.610	1.061e+00	0.5943	4.612e-01	5.178e-01	0.2306
		(0.3030)	(0.1197)	(0.1268)	(0.09049)	(0.1160)	(0.4377)	(0.08413)	(0.09133)	(0.1031)	(0.04397)
1	1	1.1305	1.596	1.6074	0.1013	-2.610	1.062e+00	0.5942	4.613e-01	5.178e-01	0.2306
		(0.3030)	(0.1197)	(0.1268)	(0.09051)	(0.1160)	(0.4377)	(0.08414)	(0.09134)	(0.1032)	(0.04398)
2	1	0.6704	1.201	0.9877	0.2705	-2.367	9.119e-01	0.3934	8.900e-01	6.823e-01	0.1440
		(0.3062)	(0.1918)	(0.1757)	(0.1212)	(0.09235)	(0.5672)	(0.1509)	(0.07302)	(0.04742)	(0.04844)
2	1	0.6705	1.201	0.9876	0.2705	-2.367	9.119e-01	0.3934	8.901e-01	6.823e-01	0.1440
		(0.3062)	(0.1918)	(0.1757)	(0.1212)	(0.09234)	(0.5672)	(0.1509)	(0.07302)	(0.04742)	(0.04844)
2	1	0.6704	1.201	0.9876	0.2705	-2.367	9.119e-01	0.3934	8.900e-01	6.823e-01	0.1439
		(0.3062)	(0.1918)	(0.1757)	(0.1212)	(0.09236)	(0.5672)	(0.1509)	(0.07302)	(0.04742)	(0.04846)
2	1	0.6705	1.201	0.9877	0.2705	-2.367	9.119e-01	0.3934	8.900e-01	6.823e-01	0.1440
		(0.3062)	(0.1918)	(0.1757)	(0.1212)	(0.09236)	(0.5673)	(0.1509)	(0.07303)	(0.04743)	(0.04845)
2	1	0.6704	1.201	0.9877	0.2705	-2.367	9.119e-01	0.3934	8.900e-01	6.823e-01	0.1439
		(0.3062)	(0.1918)	(0.1757)	(0.1212)	(0.09235)	(0.5673)	(0.1509)	(0.07302)	(0.04742)	(0.04845)
3	1	1.2715	1.116	1.3518	0.6985	-2.410	5.479e-01	0.2943	8.441e-01	4.045e-01	0.1785
		(0.3088)	(0.1511)	(0.1639)	(0.1213)	(0.1175)	(0.7818)	(0.1381)	(0.04395)	(0.1043)	(0.05457)
3	1	1.2715	1.116	1.3519	0.6985	-2.410	5.473e-01	0.2943	8.441e-01	4.044e-01	0.1785
		(0.3088)	(0.1511)	(0.1640)	(0.1213)	(0.1175)	(0.7826)	(0.1381)	(0.04395)	(0.1043)	(0.05459)
3	1	1.2714	1.116	1.3519	0.6984	-2.410	5.482e-01	0.2942	8.442e-01	4.045e-01	0.1786
		(0.3088)	(0.1511)	(0.1639)	(0.1212)	(0.1175)	(0.7814)	(0.1381)	(0.04395)	(0.1043)	(0.05456)
3	1	1.2715	1.116	1.3519	0.6984	-2.410	5.480e-01	0.2943	8.441e-01	4.045e-01	0.1785
		(0.3088)	(0.1511)	(0.1639)	(0.1212)	(0.1175)	(0.7816)	(0.1381)	(0.04395)	(0.1043)	(0.05456)
3	1	1.2715	1.117	1.3519	0.6984	-2.410	5.482e-01	0.2943	8.441e-01	4.045e-01	0.1786
		(0.3088)	(0.1511)	(0.1639)	(0.1212)	(0.1175)	(0.7813)	(0.1381)	(0.04395)	(0.1043)	(0.05455)
4	1	1.7607	1.851	1.7089	0.2778	-2.644	5.448e-07	0.8719	4.834e-01	1.553e-07	0.4462
		(0.2752)	(0.2991)	(0.2975)	(0.1918)	(0.1297)	(9.654E+05)	(0.09196)	(0.1107)	(2.315E+05)	(0.03429)
4	1	1.7607	1.851	1.7089	0.2778	-2.644	9.581e-05	0.8718	4.834e-01	2.261e-06	0.4462
		(0.2751)	(0.2991)	(0.2975)	(0.1918)	(0.1297)	(5489.)	(0.09196)	(0.1107)	(1.589E+04)	(0.03429)
4	1	1.7607	1.851	1.7089	0.2778	-2.644	1.092e-05	0.8718	4.834e-01	1.218e-07	0.4462
		(0.2752)	(0.2991)	(0.2975)	(0.1918)	(0.1297)	(4.818E+04)	(0.09196)	(0.1107)	(2.951E+05)	(0.03430)
4	1	1.7607	1.851	1.7089	0.2778	-2.644	7.800e-05	0.8718	4.835e-01	6.351e-07	0.4462
		(0.2751)	(0.2991)	(0.2975)	(0.1918)	(0.1297)	(6743.)	(0.09196)	(0.1107)	(5.658E+04)	(0.03429)
4	1	1.7607	1.851	1.7089	0.2778	-2.644	1.157e-06	0.8719	4.834e-01	2.014e-07	0.4462
		(0.2752)	(0.2991)	(0.2975)	(0.1918)	(0.1297)	(4.544E+05)	(0.09196)	(0.1107)	(1.784E+05)	(0.03429)
5	1	-0.5510	2.243	1.1337	0.3711	-2.066	1.250e-07	1.0547	1.578e-06	7.183e-01	0.3442
		(0.4717)	(0.2058)	(0.2005)	(0.1191)	(0.1520)	(5.628E+06)	(0.07972)	(4.830E+04)	(0.1011)	(0.04931)
5	1	-0.5510	2.243	1.1337	0.3711	-2.066	1.639e-07	1.0547	1.072e-06	7.183e-01	0.3442
		(0.4717)	(0.2058)	(0.2005)	(0.1191)	(0.1520)	(4.293E+06)	(0.07972)	(7.111E+04)	(0.1011)	(0.04931)
5	1	-0.5510	2.243	1.1337	0.3712	-2.066	1.819e-06	1.0547	5.292e-07	7.183e-01	0.3442
		(0.4717)	(0.2058)	(0.2005)	(0.1191)	(0.1520)	(3.868E+05)	(0.07972)	(1.440E+05)	(0.1011)	(0.04931)
5	1	-0.5510	2.243	1.1337	0.3712	-2.066	1.852e-07	1.0547	7.546e-07	7.183e-01	0.3442
		(0.4717)	(0.2058)	(0.2005)	(0.1191)	(0.1520)	(3.800E+06)	(0.07972)	(1.010E+05)	(0.1011)	(0.04931)
5	1	-0.5510	2.243	1.1337	0.3712	-2.066	3.086e-06	1.0547	2.230e-06	7.183e-01	0.3442
		(0.4717)	(0.2058)	(0.2005)	(0.1191)	(0.1520)	(2.280E+05)	(0.07972)	(3.417E+04)	(0.1011)	(0.04931)

Table 11: Point Estimates: Gauss-Hermite with first 5 good starts and 7^5 nodes

Data	INFORM	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{21}	θ_{22}	θ_{23}	θ_{24}	θ_{25}
1	1	1.0960	1.595	1.6064	0.09740	-2.592	1.043e+00	0.5915	0.4620	5.214e-01	0.2236
		(0.2930)	(0.1196)	(0.1266)	(0.08999)	(0.1088)	(0.4373)	(0.08350)	(0.09019)	(0.1024)	(0.04203)
1	1	1.0958	1.595	1.6065	0.09747	-2.592	1.044e+00	0.5916	0.4619	5.214e-01	0.2235
		(0.2930)	(0.1196)	(0.1267)	(0.09001)	(0.1088)	(0.4371)	(0.08350)	(0.09022)	(0.1024)	(0.04204)
1	1	1.0959	1.595	1.6064	0.09745	-2.592	1.043e+00	0.5915	0.4619	5.214e-01	0.2235
		(0.2931)	(0.1196)	(0.1267)	(0.09003)	(0.1088)	(0.4373)	(0.08352)	(0.09025)	(0.1024)	(0.04206)
1	1	1.0958	1.595	1.6065	0.09744	-2.592	1.043e+00	0.5916	0.4619	5.214e-01	0.2235
		(0.2931)	(0.1196)	(0.1267)	(0.09001)	(0.1088)	(0.4372)	(0.08351)	(0.09022)	(0.1024)	(0.04205)
1	1	1.0958	1.595	1.6064	0.09749	-2.592	1.044e+00	0.5915	0.4619	5.214e-01	0.2235
		(0.2931)	(0.1196)	(0.1267)	(0.09003)	(0.1088)	(0.4372)	(0.08352)	(0.09024)	(0.1024)	(0.04207)
2	1	0.6712	1.200	0.9866	0.27148	-2.367	9.160e-01	0.3967	0.8891	6.792e-01	0.1442
		(0.3063)	(0.1919)	(0.1759)	(0.1209)	(0.09201)	(0.5634)	(0.1496)	(0.07312)	(0.04634)	(0.04810)
2	1	0.6712	1.200	0.9866	0.27147	-2.367	9.160e-01	0.3967	0.8891	6.792e-01	0.1442
		(0.3063)	(0.1919)	(0.1759)	(0.1208)	(0.09200)	(0.5634)	(0.1496)	(0.07312)	(0.04634)	(0.04810)
2	1	0.6712	1.200	0.9866	0.27150	-2.367	9.160e-01	0.3967	0.8891	6.792e-01	0.1442
		(0.3063)	(0.1919)	(0.1759)	(0.1209)	(0.09201)	(0.5634)	(0.1496)	(0.07312)	(0.04634)	(0.04810)
2	1	0.6712	1.200	0.9866	0.27149	-2.367	9.160e-01	0.3967	0.8891	6.792e-01	0.1442
		(0.3063)	(0.1919)	(0.1759)	(0.1209)	(0.09202)	(0.5634)	(0.1496)	(0.07312)	(0.04634)	(0.04811)
2	1	0.6712	1.200	0.9866	0.27149	-2.367	9.160e-01	0.3967	0.8891	6.792e-01	0.1442
		(0.3063)	(0.1919)	(0.1759)	(0.1208)	(0.09200)	(0.5634)	(0.1496)	(0.07312)	(0.04634)	(0.04809)
3	1	1.2691	1.116	1.3525	0.69822	-2.406	5.179e-01	0.2928	0.8440	4.056e-01	0.1770
		(0.3068)	(0.1508)	(0.1636)	(0.1211)	(0.1153)	(0.8268)	(0.1381)	(0.04395)	(0.1037)	(0.05372)
3	1	1.2693	1.116	1.3525	0.69836	-2.406	5.175e-01	0.2929	0.8440	4.055e-01	0.1769
		(0.3069)	(0.1508)	(0.1636)	(0.1211)	(0.1153)	(0.8275)	(0.1380)	(0.04396)	(0.1037)	(0.05374)
3	1	1.2691	1.116	1.3524	0.69829	-2.407	5.181e-01	0.2928	0.8440	4.056e-01	0.1770
		(0.3068)	(0.1508)	(0.1636)	(0.1211)	(0.1153)	(0.8265)	(0.1381)	(0.04395)	(0.1037)	(0.05371)
3	1	1.2690	1.116	1.3526	0.69823	-2.406	5.182e-01	0.2927	0.8440	4.056e-01	0.1770
		(0.3068)	(0.1508)	(0.1636)	(0.1211)	(0.1153)	(0.8263)	(0.1381)	(0.04395)	(0.1037)	(0.05371)
3	1	1.2691	1.116	1.3525	0.69826	-2.406	5.180e-01	0.2928	0.8440	4.056e-01	0.1770
		(0.3068)	(0.1508)	(0.1636)	(0.1211)	(0.1153)	(0.8266)	(0.1380)	(0.04395)	(0.1037)	(0.05371)
4	1	1.7555	1.843	1.7052	0.28229	-2.642	1.005e-06	0.8666	0.5007	1.449e-07	0.4468
		(0.2687)	(0.3005)	(0.2979)	(0.1899)	(0.1247)	(4.801E+05)	(0.09294)	(0.1156)	(2.516E+05)	(0.03291)
4	1	1.7554	1.843	1.7051	0.28229	-2.642	5.060e-05	0.8666	0.5007	6.030e-07	0.4468
		(0.2687)	(0.3005)	(0.2979)	(0.1899)	(0.1247)	(9535.)	(0.09294)	(0.1156)	(6.045E+04)	(0.03291)
4	1	1.7555	1.843	1.7052	0.28229	-2.642	4.291e-06	0.8666	0.5007	3.843e-07	0.4468
		(0.2687)	(0.3005)	(0.2979)	(0.1899)	(0.1247)	(1.125E+05)	(0.09294)	(0.1156)	(9.484E+04)	(0.03291)
4	1	1.7555	1.843	1.7052	0.28229	-2.642	3.661e-06	0.8666	0.5007	1.571e-06	0.4468
		(0.2687)	(0.3005)	(0.2979)	(0.1899)	(0.1247)	(1.318E+05)	(0.09294)	(0.1156)	(2.321E+04)	(0.03291)
4	1	1.7555	1.843	1.7052	0.28229	-2.642	1.007e-05	0.8666	0.5007	5.783e-07	0.4468
		(0.2687)	(0.3005)	(0.2979)	(0.1899)	(0.1247)	(4.793E+04)	(0.09294)	(0.1156)	(6.303E+04)	(0.03291)
5	1	-0.5386	2.263	1.1507	0.37200	-2.075	2.704e-06	1.0143	0.3028	7.257e-01	0.3472
		(0.4448)	(0.1969)	(0.1966)	(0.1182)	(0.1555)	(2.479E+05)	(0.08670)	(0.2649)	(0.1042)	(0.04988)
5	1	-0.5386	2.263	1.1507	0.37200	-2.075	3.862e-09	1.0143	0.3028	7.257e-01	0.3472
		(0.4448)	(0.1969)	(0.1966)	(0.1182)	(0.1555)	(1.736E+08)	(0.08670)	(0.2649)	(0.1042)	(0.04988)
5	1	-0.5386	2.263	1.1507	0.37200	-2.075	1.490e-06	1.0143	0.3028	7.257e-01	0.3472
		(0.4448)	(0.1969)	(0.1966)	(0.1182)	(0.1555)	(4.498E+05)	(0.08670)	(0.2649)	(0.1042)	(0.04988)
5	1	-0.5386	2.263	1.1507	0.37200	-2.075	4.880e-06	1.0143	0.3028	7.257e-01	0.3472
		(0.4448)	(0.1969)	(0.1966)	(0.1182)	(0.1555)	(1.374E+05)	(0.08670)	(0.2649)	(0.1042)	(0.04988)
5	1	-1.5668	2.804	1.3896	0.16099	-3.108	4.535e+00	0.9027	1.1143	1.209e+00	0.6888
		(0.3552)	(0.3670)	(0.3737)	(0.2443)	(0.08393)	(0.4089)	(0.2448)	(0.2046)	(0.1781)	(0.02397)

Table 12: Point Estimates: SGI with first 5 good starts and 993 nodes (exact for degree ≤ 11)

Data	INFORM	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{21}	θ_{22}	θ_{23}	θ_{24}	θ_{25}
1	1	1.1450	1.597	1.6074	0.1021	-2.618	1.058e+00	0.5974	0.4649	5.236e-01	0.2344
		(0.3107)	(0.1193)	(0.1266)	(0.09039)	(0.1209)	(0.4442)	(0.08484)	(0.09264)	(0.1033)	(0.04551)
1	1	1.1449	1.597	1.6074	0.1021	-2.618	1.058e+00	0.5974	0.4649	5.237e-01	0.2344
		(0.3107)	(0.1193)	(0.1266)	(0.09039)	(0.1209)	(0.4442)	(0.08484)	(0.09263)	(0.1032)	(0.04551)
1	1	1.1451	1.596	1.6073	0.1021	-2.618	1.058e+00	0.5975	0.4648	5.237e-01	0.2344
		(0.3107)	(0.1194)	(0.1266)	(0.09041)	(0.1209)	(0.4442)	(0.08484)	(0.09268)	(0.1033)	(0.04552)
1	1	1.1450	1.597	1.6073	0.1022	-2.618	1.058e+00	0.5974	0.4649	5.237e-01	0.2344
		(0.3106)	(0.1193)	(0.1266)	(0.09039)	(0.1209)	(0.4442)	(0.08484)	(0.09264)	(0.1032)	(0.04551)
1	1	1.1450	1.597	1.6074	0.1022	-2.618	1.058e+00	0.5974	0.4649	5.236e-01	0.2344
		(0.3107)	(0.1193)	(0.1266)	(0.09040)	(0.1209)	(0.4442)	(0.08484)	(0.09265)	(0.1033)	(0.04552)
2	1	0.7407	1.203	0.9848	0.2777	-2.375	7.587e-01	0.3400	0.9026	6.870e-01	0.1487
		(0.2769)	(0.1909)	(0.1754)	(0.1211)	(0.09287)	(0.6337)	(0.1723)	(0.07105)	(0.04652)	(0.04783)
2	1	0.7407	1.203	0.9848	0.2777	-2.375	7.587e-01	0.3400	0.9026	6.870e-01	0.1487
		(0.2769)	(0.1909)	(0.1754)	(0.1211)	(0.09287)	(0.6337)	(0.1723)	(0.07105)	(0.04652)	(0.04783)
2	1	0.7407	1.203	0.9848	0.2777	-2.375	7.587e-01	0.3400	0.9026	6.870e-01	0.1487
		(0.2769)	(0.1909)	(0.1754)	(0.1211)	(0.09287)	(0.6337)	(0.1723)	(0.07105)	(0.04652)	(0.04783)
2	1	0.7407	1.203	0.9848	0.2777	-2.375	7.587e-01	0.3400	0.9026	6.870e-01	0.1487
		(0.2769)	(0.1909)	(0.1754)	(0.1211)	(0.09287)	(0.6337)	(0.1723)	(0.07105)	(0.04652)	(0.04783)
2	1	0.7407	1.203	0.9848	0.2777	-2.375	7.588e-01	0.3400	0.9026	6.870e-01	0.1487
		(0.2769)	(0.1909)	(0.1754)	(0.1211)	(0.09287)	(0.6337)	(0.1723)	(0.07105)	(0.04652)	(0.04783)
3	1	1.2592	1.119	1.3545	0.6938	-2.406	5.522e-01	0.2969	0.8445	4.096e-01	0.1774
		(0.3125)	(0.1512)	(0.1640)	(0.1213)	(0.1207)	(0.7791)	(0.1367)	(0.04399)	(0.1030)	(0.05693)
3	1	1.2594	1.119	1.3545	0.6940	-2.406	5.511e-01	0.2970	0.8445	4.096e-01	0.1773
		(0.3125)	(0.1513)	(0.1640)	(0.1213)	(0.1208)	(0.7806)	(0.1367)	(0.04400)	(0.1031)	(0.05695)
3	1	1.2592	1.119	1.3545	0.6939	-2.406	5.520e-01	0.2969	0.8445	4.096e-01	0.1774
		(0.3125)	(0.1512)	(0.1640)	(0.1213)	(0.1208)	(0.7794)	(0.1367)	(0.04399)	(0.1031)	(0.05693)
3	1	1.2592	1.119	1.3545	0.6939	-2.406	5.518e-01	0.2969	0.8445	4.096e-01	0.1774
		(0.3125)	(0.1512)	(0.1640)	(0.1213)	(0.1208)	(0.7796)	(0.1367)	(0.04399)	(0.1031)	(0.05694)
3	1	1.2592	1.119	1.3545	0.6939	-2.406	5.518e-01	0.2969	0.8445	4.096e-01	0.1774
		(0.3125)	(0.1512)	(0.1640)	(0.1213)	(0.1208)	(0.7796)	(0.1367)	(0.04399)	(0.1030)	(0.05693)
4	1	1.8109	1.863	1.7192	0.2741	-2.708	2.421e-01	0.8711	0.5005	8.894e-07	0.4690
		(0.2819)	(0.2974)	(0.2946)	(0.1895)	(0.1510)	(2.351)	(0.09297)	(0.1068)	(3.940E+04)	(0.04176)
4	1	1.8109	1.863	1.7192	0.2740	-2.708	2.418e-01	0.8711	0.5005	2.742e-06	0.4690
		(0.2819)	(0.2974)	(0.2946)	(0.1895)	(0.1510)	(2.354)	(0.09297)	(0.1068)	(1.278E+04)	(0.04176)
4	1	1.8109	1.863	1.7192	0.2741	-2.708	2.421e-01	0.8711	0.5005	4.146e-06	0.4690
		(0.2819)	(0.2974)	(0.2946)	(0.1895)	(0.1510)	(2.351)	(0.09297)	(0.1068)	(8453.)	(0.04176)
4	1	1.8109	1.863	1.7192	0.2740	-2.708	2.423e-01	0.8711	0.5005	2.813e-06	0.4690
		(0.2819)	(0.2974)	(0.2946)	(0.1894)	(0.1510)	(2.349)	(0.09297)	(0.1068)	(1.246E+04)	(0.04176)
4	1	1.8109	1.863	1.7192	0.2741	-2.708	2.421e-01	0.8711	0.5005	1.145e-06	0.4690
		(0.2819)	(0.2974)	(0.2946)	(0.1895)	(0.1510)	(2.350)	(0.09297)	(0.1068)	(3.061E+04)	(0.04176)
5	1	-0.4660	2.258	1.1551	0.3580	-2.115	1.550e-06	1.0433	0.2330	7.726e-01	0.3592
		(0.5278)	(0.2026)	(0.1986)	(0.1179)	(0.1670)	(4.379E+05)	(0.07440)	(0.3373)	(0.1017)	(0.05318)
5	1	-0.4920	2.317	1.2141	0.3340	-2.123	1.174e-07	0.9852	0.5095	7.826e-01	0.3627
		(0.5487)	(0.1999)	(0.1967)	(0.1192)	(0.1653)	(5.881E+06)	(0.07564)	(0.1469)	(0.1019)	(0.05232)
5	1	-0.4660	2.258	1.1551	0.3580	-2.115	3.531e-07	1.0433	0.2331	7.726e-01	0.3592
		(0.5278)	(0.2026)	(0.1986)	(0.1179)	(0.1670)	(1.922E+06)	(0.07440)	(0.3372)	(0.1017)	(0.05318)
5	1	-0.4660	2.258	1.1551	0.3580	-2.115	1.848e-06	1.0433	0.2331	7.726e-01	0.3592
		(0.5278)	(0.2026)	(0.1986)	(0.1179)	(0.1670)	(3.671E+05)	(0.07440)	(0.3372)	(0.1017)	(0.05318)
5	1	-0.4920	2.317	1.2141	0.3340	-2.123	1.785e-06	0.9852	0.5095	7.826e-01	0.3627
		(0.5487)	(0.1999)	(0.1967)	(0.1192)	(0.1653)	(3.870E+05)	(0.07564)	(0.1469)	(0.1019)	(0.05232)

Table 13: Point Estimates: Monomial with First 5 Good Starts.