Scalable Rejection Sampling for Bayesian Hierarchical Models

Michael Braun
Cox School of Business
Southern Methodist University
Dallas, TX 75275
braunm@smu.edu

Paul Damien
McCombs School of Business
University of Texas at Austin
Austin, TX 78712
paul.damien@mccombs.utexas.edu

November 22, 2013

Abstract

We develop a new method to sample from posterior distributions in Bayesian hierarchical models, as commonly used in marketing research, without using Markov chain Monte Carlo. This method, which is a variant of rejection sampling ideas, is generally applicable to high-dimensional models involving large data sets. Samples are independent, so they can be collected in parallel, and we do not need to be concerned with issues like chain convergence and autocorrelation. The method is scalable under the assumption that heterogeneous units are conditionally independent, and it can also be used to compute marginal likelihoods.

1 Introduction

The application of Bayesian methods in marketing is well-established. Rossi and Allenby (2003) compiled a review of this work, including an annotated bibliography, and countless other papers that use Bayesian methods have been published in marketing journals in the decade since. One reason Bayesian methods appeal to marketing researchers is that they allow for a straightforward way to incorporate unobserved heterogeneity into a model, and estimate posterior estimates of individual-level latent characteristics. We often describe these models as "hierarchical" when latent parameters of sub-groups of individual agents, such as customers or households, are distributed according to characteristics of a larger population. (To simplify the exposition in the rest of this paper, we will use the term "household" to refer to any heterogeneous agent). In many cases, these sub-groups consist of a single household. With some notable exceptions (e.g., Yang and Allenby 2003), hierarchical models in marketing assume that households are "conditionally independent," so that the only dependence across units comes from their having common population-level parameters.

The marginal posterior densities of parameters of interest are typically not available in closed form. Instead, we simulate random samples from the marginal posteriors, and use the empirical distribution of the samples to estimate posterior moments and quantiles. The challenge is in determining the best way to generate these samples. Markov chain Monte Carlo (MCMC) has become perhaps the most popular simulation method for Bayesian computation because of its general applicability. The idea is to construct a Markov chain whose limiting distribution is the same as the "target" posterior density. Once the chain has converged to that density, we can treat each iteration as a posterior sample. Since the introduction of Gibbs sampling as an MCMC algorithm (Gelfand and Smith 1990), the Bayesian computational literature has exploded with numerous methods for generating valid and efficient MCMC algorithms. It would be difficult to list all of them here, so we refer the reader to Gelman et al. (2003); Chen, Shao, and Ibrahim (2000); Rossi, Allenby, and McCulloch (2005) and Brooks et al. (2010) and the hundreds of references therein.

However, MCMC has two main drawbacks that limit its broader use in practical settings with large datasets. First, although a proper MCMC algorithm will generate a Markov chain that converges to the target density *eventually*, there is no guarantee that such convergence will occur even after millions of iterations. Second, successive samples from an MCMC algorithm are autocorrelated, meaning that the effective sample size can be substantially less than the number of iterations. Thus, before starting the algorithm, there is no way to know how long it will take to converge to the posterior, and how many post-convergence iterations are required to conduct inference with an acceptable level of precision. The performance of MCMC along these dimensions is completely dependent on the structure of problem, the nature of the data, and the sampling algorithm that is chosen. Yet, as articulated by Papaspiliopoulos and Roberts (2008), "*to date, there has been little theoretical analysis linking the stability of the Gibbs sampler to the structure of hierarchical models.*"

There is now substantial interest in developing methods for estimating parameters in Bayesian hierarchical models that do not rely on MCMC. In this paper, we propose a new estimation method that generates independent samples from a target posterior density. We call this method Generalized Direct Sampling (GDS). In the GDS method, the only restrictions are that one must be able to compute the unnormalized log posterior of the parameters (or a good approximation of it); that the posterior distribution must be bounded from above over the parameter space; and that one is able to locate any local maxima of the log posterior function using available computing resources. One need not derive conditional posterior distributions (as with blockwise Gibbs sampling), and there are no conjugacy requirements. GDS also allows us to compute the marginal likelihood of the data under the model with little additional computation.

We derive the GDS algorithm in Section 2, but the basic idea is as follows. As a first step, we find the mode of the log posterior density, and construct a proposal density (typically a multivariate normal) with a mean at the posterior mode. We then repeatedly sample from the proposal density to estimate the marginal density of a scalar auxiliary parameter. Samples from this marginal density serve as thresholds for a rejection sampling step; the source of the thresholds is what distinguishes GDS from traditional rejection sampling. The final step is to repeatedly sample from the proposal until a threshold condition is reached, and retain that last sample as a draw from the target posterior.

GDS generates posterior samples that are independent, so they can be collected in parallel, dramatically reducing execution time. This is in contrast to MCMC for which the serial dependence of samples makes within-chain parallel execution impossible.[1] This feature of GDS allows the marketing researcher to exploit recent technological advances by which multiple processors are readily available in both personal computers and distributed computing clusters.

In addition, under the assumption that households are conditionally independent, GDS is scalable. For the purpose of this analysis, we call an algorithm "scalable" if the computational costs grow at most linearly in the number of households. This scalability comes from the conditional independence assumption under which the Hessian matrix of the log posterior density becomes increasingly sparse as the size of the dataset increases. By exploiting this sparsity, we find that certain tasks within the GDS steps, such as finding the posterior mode or sampling from a large-dimension multivariate normal distribution, are much faster than if the Hessian were dense.

We are not claiming that GDS should replace MCMC in all cases. GDS may not be

[1]Running multiple chains in parallel can reduce the number of required post-convergence iterations, but it does not avoid the need for each chain to converge independently.

practical for models with discrete parameters, a very large number of modes, or for which computing the log posterior density itself is difficult. GDS does not require that the model be hierarchical, or that the conditional independence assumption holds, but without those assumptions, it will not be as scalable. Nevertheless, we believe that many models of the kind marketing researchers will encounter can be estimated well with GDS, at least relative to the effort involved in using MCMC. Like MCMC and other non-MCMC methods, GDS is another useful algorithm in the researcher's and practitioner's toolkits.

In Section 2, we present the details of the algorithm, along with the theoretical basis for it, and compare it to other methods. Section 3 includes examples of GDS in action, including two small models to demonstrate the accuracy of the method, and a moderately-sized model for which MCMC is impractical. In Section 4, we explain when and why GDS is scalable, and offer evidence to support that claim via a simulation study. In Section 5, we show how to use GDS output to estimate the marginal likelihood of the data. Finally, in Section 6, we discuss practical issues that one should consider when implementing GDS, including limitations of the approach.

2 Generalized Direct Sampling: The Method

2.1 Theoretical basis

GDS is a variant on well-known importance sampling methods. The goal is to sample a parameter vector θ from a posterior density $\pi(\theta|y)$, where $\pi(\theta)$ is the prior on θ, $f(y|\theta)$ is the data likelihood conditional on θ, and $\mathcal{L}(y)$ is the marginal likelihood of the data. Therefore,

$$\pi(\theta|y) = \frac{f(y|\theta)\pi(\theta)}{\mathcal{L}(y)} = \frac{\mathcal{D}(\theta,y)}{\mathcal{L}(y)} \tag{1}$$

where $\mathcal{D}(\theta, y)$ is the joint density of the data and the parameters (of the unnormalized posterior density). In a marketing research context, under the conditional independence assumption, the likelihood can be factored as

$$f(y|\theta) = \prod_{i=1}^{N} f_i\left(y_i|\beta_i, \alpha\right) \tag{2}$$

where i indexes households. Each y_i is a vector of observed data, each β_i is a vector of heterogeneous parameters, and α is a vector of homogeneous population-level parameters. The β_i are distributed across the population of households according to a mixing distribution $\pi(\beta_i|\alpha)$, which also serves as the prior on each β_i. The elements of α may influence either the household-level data likelihoods, or the mixing distribution (or both). In this example, θ includes all $\beta_1 \ldots \beta_N$ and all elements of α. The prior itself can be factored as

$$\pi(\theta) = \prod_{i=1}^{N} \pi_i(\beta_i|\alpha) \times \pi(\alpha). \tag{3}$$

Let θ^* be the mode of $\mathcal{D}(\theta, y)$, which is also the mode of $\pi(\theta|y)$, since $\mathcal{L}(y)$ is a constant that does not depend on θ. One will probably use some kind of iterative numerical optimizer to find θ^*, such as a quasi-Newton line search or trust region algorithm. Define $c_1 = \mathcal{D}(\theta^*, y)$, and choose a proposal distribution $g(\theta)$ that also has its mode at θ^*. Define $c_2 = g(\theta^*)$, and define the function

$$\Phi(\theta|y) = \frac{f(y|\theta)\pi(\theta) \cdot c_2}{g(\theta) \cdot c_1} \tag{4}$$

Through substitution and rearranging terms, we can write the target posterior density as

$$\pi(\theta|y) = \Phi(\theta|y) \cdot g(\theta) \cdot \frac{c_1}{c_2 \cdot \mathcal{L}(y)} \tag{5}$$

An important restriction on the choice of $g(\theta)$ is that the inequality $0 < \Phi(\theta|y) \leq 1$ must hold, at least for any θ with a non-negligible posterior density. We discuss the choice of $g(\theta)$ in more detail a little later.

Next, let $u|\theta, y$ be an auxiliary variable that is distributed uniformly on $\left(0, \frac{\Phi(\theta|y)}{\pi(\theta|y)}\right)$, so that $p(u|\theta, y) = \frac{\pi(\theta|y)}{\Phi(\theta|y)} = \frac{c_1}{c_2\mathcal{L}(y)}g(\theta)$. Then construct a joint density of $\theta|y$ and $u|\theta, y$, where

$$p(\theta, u|y) = \frac{\pi(\theta|y)}{\Phi(\theta|y)}\mathbb{1}\left[u < \Phi(\theta|y)\right] \tag{6}$$

By integrating Equation 6 over u, the marginal density of $\theta|y$ is

$$p(\theta|y) = \frac{\pi(\theta|y)}{\Phi(\theta|y)} \int_0^{\Phi(\theta|y)} du = \pi(\theta|y) \tag{7}$$

Therefore, simulating from $p(\theta|y)$ is equivalent to simulating from the target posterior $\pi(\theta|y)$.

Using Equations 5 and 6, the marginal density of $u|y$ is

$$\begin{aligned} p(u|y) &= \int_\theta \frac{\pi(\theta|y)}{\Phi(\theta|y)}\mathbb{1}\left[u < \Phi(\theta|y)\right] d\theta && (8)\\ &= \frac{c_1}{c_2\mathcal{L}(y)} \int_\theta \mathbb{1}\left[u < \Phi(\theta|y)\right] g(\theta)\, d\theta && (9)\\ &= \frac{c_1}{c_2\mathcal{L}(y)} q(u) && (10) \end{aligned}$$

where $q(u) = \int_\theta \mathbb{1}\left[u < \Phi(\theta|y)\right] g(\theta)\, d\theta$. This $q(u)$ function is the probability that any candidate draw from $g(\theta)$ will satisfy $\Phi(\theta|y) > u$.

The GDS sampler comes from recognizing that $p(\theta, u|y)$ can written differently from, but equivalently to, Equation 6.

$$p(\theta, u|y) = p(\theta|u, y)\ p(u|y) \tag{11}$$

The sampling method involves sampling a u from an approximation to $p(u|y)$, and then sampling from $p(\theta|u, y)$. Using the definitions in Equations 4, 5, and 6, we get

$$\begin{aligned} p(\theta|u, y) &= \frac{p(\theta, u|y)}{p(u|y)} && (12) \\ &= \frac{c_1}{c_2 \mathcal{L}(y) p(u|y)} \mathbb{1}[u < \Phi(\theta|y)]\ g(\theta) && (13) \end{aligned}$$

To sample *directly* from $p(\theta, u|y)$, one needs only to sample from $p(u|y)$ and then sample repeatedly from $g(\theta)$ until $\Phi(\theta|y) > u$. The samples of θ form the marginal distribution $p(\theta|y)$, and since sampling from $p(\theta|y)$ is equivalent to sampling from $\pi(\theta|y)$, they form an empirical estimate of the target posterior density.

2.2 Implementation

But how does one simulate from $p(u|y)$? In Equation 8, we see that $p(u|y)$ is proportional to the function $q(u)$. Walker et al. (2011) sample from a similar kind of density by first taking M proposal draws from the prior to construct an empirical approximation to $q(u)$, and then approximating that continuous density using Bernstein polynomials. However, in high-dimensional models, this approximation tends to be a poor one at the endpoints, even with an extremely large number of Bernstein polynomial components. It is for this reason that the largest number of parameters that Walker et al. tackle is 10.

Our approach is similar in that we effectively trace out an empirical approximation to $q(u)$ by repeatedly sampling from $g(\theta)$, and computing $\Phi(\theta|y)$ for each of those proposal draws. To avoid the endpoint problem in the Walker et al. method, we instead sample a transformed variable $v = -\log u$. Applying a change of variables, $q_v(v) = q(u)\exp(-v)$. With $q_v(v)$ denoting the "true" CDF of v, let $\widehat{q}_v(v)$ be the empirical CDF of v after taking M proposal draws from $g(\theta)$, and ordering the proposals $0 < v_1 < v_2 < \ldots < v_M < \infty$. To be clear, $\widehat{q}_v(v)$ is the proportion of samples that are *strictly* less than v.

Because $\widehat{q}_v(v)$ is discrete, we can sample from a density proportional to $q(u)\exp(-v)$ by partitioning the domain into $M+1$ segments with a break point of each partition at each v_i. The probability of sampling a new v that falls between v_i and v_{i+1} is now

$$\varpi_i = \widehat{q}_v(v)\left[\exp(-v_i) - \exp(-v_{i+1})\right], \tag{14}$$

so we can sample an interval bounded by v_i and v_{i+1} from a multinomial density with weights proportional to ϖ_i. Once we have the i that corresponds to that interval, we can sample the continuous v by sampling ϵ from a standard exponential density, truncated on the right at $v_{i+1} - v_i$, and setting $v = v_i + \epsilon$. A simplified way to sample v is to sample i with weight ϖ_i sample a standard uniform random variable η, and set

$$v = v_i - \log\left[1 - \eta\left(1 - \exp(v_i - v_{i+1})\right)\right]. \tag{15}$$

To sample R independent draws from the target posterior, we need R "threshold" draws of v. Then, for each v, we repeatedly sample from $g(\theta)$ until $-\log(\Phi(\theta|y)) < v$. Once we have a θ that meets this criterion, we save it as a valid sample from $\pi(\theta|y)$. The complete GDS algorithm is summarized as Algorithm 1.

2.3 The proposal distribution

The only restriction on $g(\theta)$ is that the inequality $0 < \Phi(\theta|y) \leq 1$ must hold, at least for any θ with a non-negligible posterior density. In principle, it is up to the researcher

Algorithm 1 The GDS Algorithm to collect R samples from $\pi(\theta|y)$

```
1: $R \leftarrow$ number of required samples from $\pi(\theta|y)$
2: $M \leftarrow$ number of proposal draws for estimating $\widehat{q}_v(v)$.
3: $\theta^* \leftarrow$ mode of $\mathcal{D}(\theta, y)$
4: $c_1 \leftarrow \mathcal{D}(\theta^*, y)$
5: FLAG$\leftarrow$ TRUE
6: while FLAG do
7:    Choose new proposal distribution $g(\theta)$
8:    FLAG$\leftarrow$FALSE
9:    $c_2 \leftarrow g(\theta^*)$.
10:   for $m := 1$ to $M$ do
11:      Sample $\theta_m \sim g(\theta)$.
12:      $\log\Phi(\theta_m|y) \leftarrow \log\mathcal{D}(\theta_m, y) - \log g(\theta_m) - \log c_1 + \log c_2$.
13:      $v_m = -\log\Phi(\theta_m|y)$
14:      if $\log\Phi(\theta_m|y) > 0$ then
15:         FLAG$\leftarrow$ TRUE
16:         break
17:      end if
18:   end for
19: end while
20: Reorder elements of $v$, so $0 < v_1 < v_2 < \ldots < v_M < \infty$. Define $v_{M+1} := \infty$
21: for $i := 1$ to $M$ do
22:    $\widehat{q}_v(v_i) \leftarrow \sum_{j=1}^{M} \mathbb{1}\left[v_j < v_i\right]$.
23:    $\varpi_i \leftarrow \widehat{q}_v(v_i)\left[\exp(-v_i) - \exp(-v_{i+1})\right]$.
24: end for
25: for $r = 1$ to $R$ do
26:    Sample $j \sim$ Multinomial$(\varpi_1 \ldots \varpi_M)$.
27:    Sample $\eta \sim$ Uniform(0,1).
28:    $v^* \leftarrow v_j - \log\left[1 - \eta\left(1 - \exp\left(v_j - v_{j+1}\right)\right)\right]$.
29:    $p \leftarrow 0$
30:    $n_r \leftarrow 0$. {Counter for number of proposals}
31:    while $p > v^*$ do
32:       Sample $\theta_r \sim g(\theta)$.
33:       $p \leftarrow -\log\Phi\left(\theta_r|y\right)$.
34:       $n_r \leftarrow n_r + 1$.
35:    end while
36: end for
37: return $\theta_1 \ldots \theta_R$ (plus $n_1 \ldots n_R$ and $v_1 \ldots v_M$ if computing a marginal likelihood).
```

to choose $g(\theta)$, and some choices may be more efficient than others. In all of our work using GDS, we have found that a multivariate normal (MVN) proposal distribution, with mean at θ^*, works well for the kinds of continuous posterior densities that marketing researchers typically encounter. Note that the MVN density, with a covariance matrix equal to the inverse Hessian of the log posterior at θ^*, is an asymptotic approximation to the posterior density itself (Carlin and Louis 2000, sec. 5.2). By multiplying that covariance matrix by a scaling constant s, we can derive a proposal distribution that has the general shape of the target posterior near its mode. That proposal distribution will be valid as long as s is large enough so that $\Phi(\theta|y)$ is between 0 and 1 for any plausible value of θ.

It is, of course, possible that $g(\theta)$ could under-sample values of θ in the tails of the posterior. If M is large enough, we should know if this would be the case before starting the rejection sampling phase of the algorithm since we would observe some θ for which $\log \Phi(\theta|y) > 0$. Heavier-tailed proposals, such as the multivariate-t distribution, can fail because of high kurtosis at the mode. If, during the rejection sampling phase of the algorithm, we propose a draw from which $\Phi(\theta|y) > 1$, we can stop, rescale, and try again. Although we believe that the potential cost from under-sampling the tails is dwarfed by GDS's relative computational advantage, we recognize that there may be some applications for which sampling extreme values of θ may be important. GDS may not be the best algorithm for those applications. Otherwise, there is nothing special about using the MVN for $g(\theta)$. It is straightforward to implement with manual adaptive selection of s. This is similar, in spirit, to the concept of tuning a Metropolis-Hastings algorithm.

2.4 Comparison to Other Methods

2.4.1 Rejection Sampling

At first glance, GDS looks quite similar to standard rejection sampling. With rejection sampling, one samples a threshold value from a standard uniform distribution ($p(u) = 1$), and then repeatedly samples from a proposal distribution $g(\theta)$ until $\pi(\theta|y)/g(\theta) \geq Ku$, where K is a positive constant. This is different than GDS for which the threshold values are sampled from a posterior $p(u|y)$, and K is specifically defined as the ratio of modal densities of the posterior to the proposal. The advantages that rejection sampling has over GDS are that the distribution of u is exact, and that the proposal density does not have to dominate the target density for all values of θ. However, for any model with more than a few dimensions, the critical ratio can be extremely small for even small deviations of θ away from the mode. Thus, the acceptance probabilities are virtually nil. With GDS, we accept a discrete approximation of $p(u|y)$ in exchange for higher acceptance probabilities.

2.4.2 Direct Sampling

Walker et al. (2011) introduced and demonstrated the merits of a non-MCMC approach called Direct Sampling (DS) for conducting Bayesian inference. As with GDS, DS removes the need to concern oneself with issues like chain convergence and autocorrelation. They also point out that their method generates independent samples from a target posterior distribution *in parallel*. Walker et al. also prove that the sample acceptance probabilities using DS are better than those from standard rejection algorithms. Put simply, for many common Bayesian models, they demonstrate improvement over MCMC in terms of efficiency, resource demands and ease of implementation.

However, DS suffers from some important shortcomings that limit its broad applicability. One is the failure to separate the specification of the prior from the specifics of the estimation algorithm. Another is an inability to generate accepted draws for even moderately sized problems; the largest number of parameters that Walker et al. consider is 10. GDS allows us to conduct full Bayesian inference on hierarchical models in high dimensions, with or without conjugacy, without MCMC.

Strictly speaking, GDS is not a generalization of the DS algorithm, but it does share some important features with DS. DS and GDS differ in the following respects.

1. While DS focuses on the shape of the data likelihood alone, GDS is concerned with the characteristics of the entire posterior density.

2. GDS bypasses the need for Bernstein polynomial approximations, which are integral to the DS algorithm.

3. While DS takes proposal draws from the prior (which may conflict with the data), GDS samples proposals from a separate density that is ideally a good approximation to the target posterior density itself.

2.4.3 Markov chain Monte Carlo

We have already mentioned the key advantage that GDS has over traditional MCMC: generating independent samples that can be collected in parallel. Thus, we do not need to be concerned at all with issues like autocorrelation, convergence of estimation chains, and so forth. Without delving into a discussion of all possible variations and improvements to MCMC that have been proposed in the last few decades, there have been some attempts to parallelize MCMC that deserve some mention. For a deeper analysis, see Suchard et al. (2010).

It is possible to run multiple independent MCMC chains that start from different starting points. Once all of those chains have converged to the posterior distribution, we can estimate the posterior by combining samples from all of the chains. The numerical efficiency of that approximation should be higher than if we used only one chain, because there should be no correlation between samples collected in different chains. However, each chain still needs to converge to the posterior independently, and only after that convergence could we start collecting samples. If it takes a long time for one chain to converge, it will take at least that long for all chains to converge. Thus, the potential for parallelization is much greater for GDS than it is for MCMC.

Another approach to parallelization is to exploit parallel processing power for individual steps in an algorithm. One example is a parallel implementation of a multivariate slice sampler (MSS), as in Tibbits, Haran, and Liechty (2010). The benefits of parallelizing the MSS come from parallel evaluation of the target density at each of the vertices of the multivariate slice, and from more efficient use of resources to execute linear algebra operations (e.g, Cholesky decompositions). But the MSS itself remains a Markovian algorithm, and thus will still generate dependent draws. Using parallel technology to generate a single draw from a distribution is not the same as generating all of the required draws themselves in parallel. The sampling steps of GDS can be run in their *entirety* in parallel.

3 Examples

We now provide some examples of GDS in action, and of some failures in MCMC. We start with a couple of small models simply to show that GDS generates the correct posterior distribution. Then, we run both MCMC and GDS on a moderately-sized problem of 2,015 parameters. In that example, we show that even an advanced MCMC algorithm cannot generate a sufficient number of samples in a reasonable amount of time, while GDS is a valid alternative with far lower computational expense. Then, we conduct a simulation study to illustrate the scalability of GDS.

3.1 Two simple examples

Our first demonstration model has the observed data being generated from a normal distribution with a mean of 100 and a standard deviation of 5. The prior on the mean is itself a normal density, with mean zero and standard deviation of 1000, while the prior on the standard deviation is uniform, from zero to 100. We simulated 20 observations from the model, and sampled from the posterior using both MCMC and GDS. Table 1 summarizes the posterior means, along with the 0.025 and 0.975 quantiles that were estimated from both methods. In addition, we report the frequentist point estimate of the parameters along with the endpoints of the 95% confidence interval. We see that the GDS intervals are closely aligned with the intervals from the other two methods.

	Mean			Std. Dev.		
Method	2.5%	50%	97.5%	2.5%	50%	97.5%
gds	98.2	100.7	103.2	13.5	25.8	52.3
CI	98.4	100.7	103.0	13.7	23.6	50.5
mcmc	98.4	100.7	103.0	13.7	24.5	50.4

Table 1: Frequentist and posterior intervals for parameters in first example.

For the second example, we again let the observed data come from a normal distribution with a mean of 100, but instead we fix the mean and sample τ, where the standard deviation is $1/\tau$. The prior on τ is a gamma distribution with a shape of 0.001 and scale of 1000. The posterior distribution has an analytic solution, namely gamma with a shape parameter of $n/2 + .001$, and a scale of $1000/(500 \sum_{i=1}^{n}((x-100)^2+1))$. In Figure 1, we plot histograms of the GDS samples for the posterior density of τ along with the true posterior density. Clearly, GDS is able to generate samples from the correct posterior distribution.

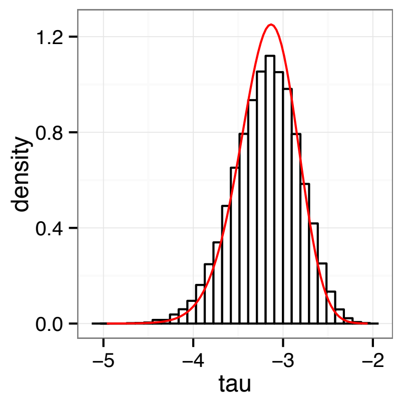


Figure 1: Histogram of GDS samples from posterior density of τ. The line represents the true posterior density.

3.2 Hierarchical model

In this section, we estimate parameters of a non-conjugate heterogeneous hierarchical model for a moderately-sized dataset, where some parameters are only weakly identified. The data are described in Manchanda, Rossi, and Chintagunta (2004), and are available in the **bayesm** package for R (Rossi. 2012). In this dataset, for each of 1,000 physicians, we observe weekly prescription counts for a single drug (y_{it}), weekly counts of sales visits from representatives of the drug manufacturer (x_{it}), and some time-invariant demographic information (z_i).. Although one could model the purchase data as conditional on the sales visits, Manchanda et al. argue that the rate of these contacts is determined endogenously, so that physicians who are expected to write more prescriptions, or who are more responsive to the sales visits, will receive visits more often.

To maintain our focus on the relative performance of MCMC and GDS, we will estimate the parameters for a simplified version of the endogenous Manchanda et al. model. We model y_{it} as a random variable that follows a negative binomial distribution with shape r and mean μ_{it}, and we model x_{it} as a Poisson-distributed random variable with mean η_i.

The expected number of prescriptions per week depends in the number of sales visits, so we let $\log \mu_{it} = \beta_{i0} + \beta_{i1} x_{it}$, where β_i is a vector of heterogeneous coefficients. We then model the contact rate so it depends on the physician-specific propensities to purchase. Thus, $\log \eta_i = \gamma_0 + \gamma_1 \beta_{i0} + \gamma_2 \beta_{i1}$. Next, define z_i as a vector of four physician-specific demographics (including an intercept), and define Δ as a 2×4 matrix of population-level coefficients. The mixing distribution for β (i.e., the prior on each β_i), is MVN with mean $\Delta' z_i$ and covariance V. We place weakly informative MVN priors on γ and the rows of Δ, an inverse Wishart prior on V, and a gamma prior on r. There are 2,015 distinct parameters to be estimated. This model differs from the one in Manchanda et al. 2004, in that η_i depends only on expected sales in the current period, and not the long-term trend. We made this change to make it easier to run the baseline MCMC algorithm.

As our baseline MCMC algorithm, we use Hamiltonian Monte Carlo (HMC, Duane et al. 1987), which uses the gradient of the log posterior to simulate a dynamic physical system. We refer the reader to Neal (2011) for a review of the details and theory that underlie HMC. We selected HMC mainly because it is known to generate successive samples that are less serially correlated than draws that one might sample using other methods like Metropolis-Hastings. We used the "double averaging" method to adaptively scale the step size (Hoffman and Gelman 2011), and we set the expected path length to 16.[2] HMC is a good benchmark for GDS because, like GDS, the model is specified only through a function that returns the log posterior, and not a series of conditional posteriors, as in Gibbs sampling. This similarity lets us use the same code for the log posterior that we use for GDS. In short, we implemented HMC so it can compete with GDS on a level playing field.

We ran four independent HMC chains for 700,000 iterations each, during a period of

[2]Shorter path lengths were less efficient, and longer ones frequently jumped so far from the regions of high posterior mass that the computation of the log posterior would underflow. We had the same problem with the No U-Turn Sampler (Hoffman et al. 2011). The HMC extensions in Girolami and Calderhead (2011) are inappropriate for this problem because the Hessian is not guaranteed to be positive definite for all parameter values.

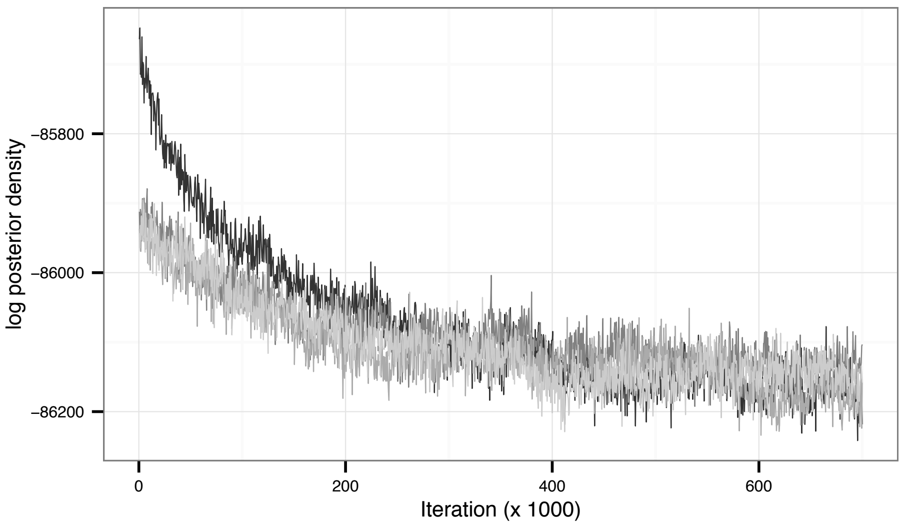


Figure 2: HMC traceplot of log posterior density for four chains. One chain was started at the mode, and the others started at random values. Every 500th iteration is plotted.

more than three weeks. Searching for the posterior mode is considered, in general, to be "good practice" for Bayesian inference, and especially with MCMC; see Step 1 of the "Recommended Strategy for Posterior Simulation" in Section 11.10 of Gelman et al. (2003). Since finding the mode of the log posterior is the first step of GDS anyway, so we initialized one chain there, and the other three at randomly-selected starting values. Figure 2 shows the trace plot of the log posterior density. The chains begin to approach each other only after about 500,000 iterations. In Figure 3, we present the trace plots of the population-level parameters. Some parameter chains appear to have converged to each other, with little autocorrelation, but others seem to make no progress at all. In Table 2, we report the effective sample sizes for estimates of the marginal posterior distributions of population-level parameters, using the final 100,000 samples of the HMC chains. For many of these parameters, it may require more than a million of additional iterations to achieve an effective sample size large enough to make inferences about these parameters.

The convergence problems are even worse when we consider that each of the 16 steps in the path length for iteration requires one evaluation for both the log posterior and

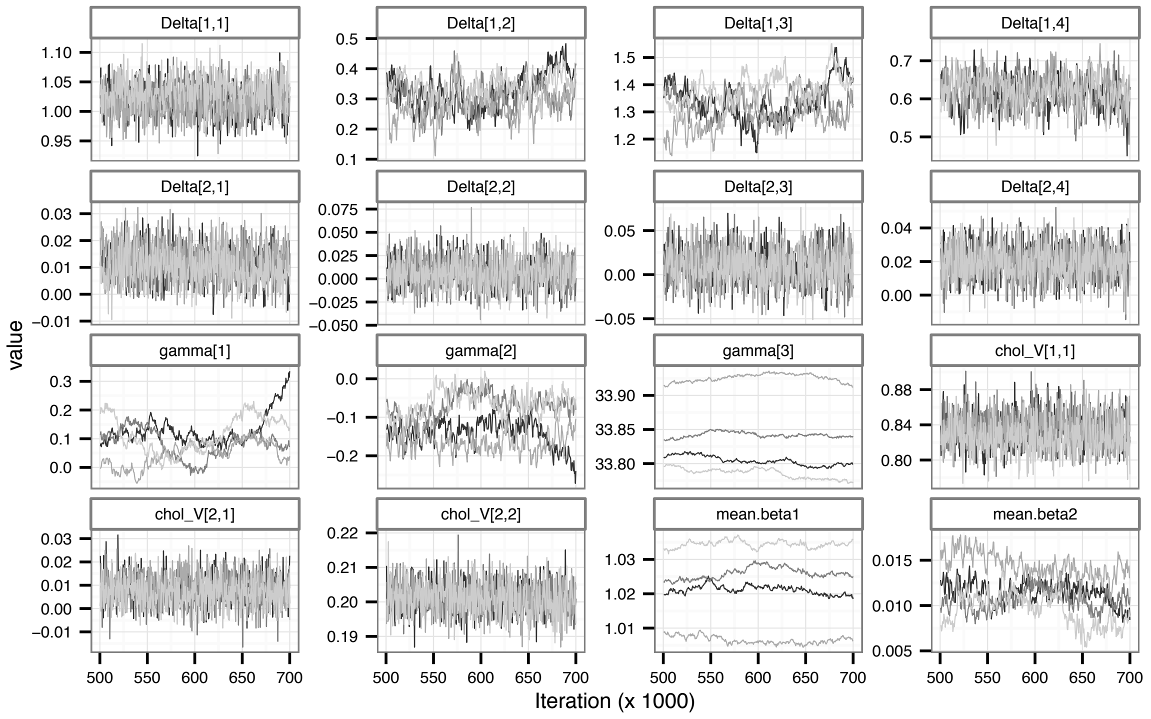


Figure 3: HMC traceplot for value of population-level parameters in hierarchical model example, starting at Iteration 500,000. Every 500th iteration is plotted.

its gradient. Using "reverse mode" automatic differentiation, the time to evaluate the gradient is no more than five times the time it takes to evaluate the log posterior, regardless of the number of variables (Griewank and Walther 2008). Therefore, each HMC iteration requires resources that equate to 96 evaluations of the posterior. In other words, the computational cost of 700,000 HMC iterations is equivalent to more than 67 million evaluations of the log posterior. And this assumes that 700,000 iterations were sufficient to collect enough samples from the true posterior.

So how much more efficient is GDS? We estimated the marginal density $q(v)$ by taking $M = 100,000$ proposal draws from an MVN distribution with the mean at the posterior mode, and the covariance matrix equal to the inverse of the Hessian at the mode, multiplied by a scaling factor of 1.3. This scaling factor is the smallest value that gave us 100,000 samples for which $0 < \Phi(\theta|y) \leq 1$. We then collected 300 *independent* samples

	Chain			
	1	2	3	4
$\Delta_{1,1}$	295	250	266	267
$\Delta_{1,2}$	16	24	27	29
$\Delta_{1,3}$	6	12	26	14
$\Delta_{1,4}$	51	43	43	60
$\Delta_{2,1}$	4703	5933	5981	1444
$\Delta_{2,2}$	507	573	526	538
$\Delta_{2,3}$	407	445	400	407
$\Delta_{2,4}$	3708	4797	3680	4926
γ_1	3	4	6	2
γ_2	10	33	22	22
γ_3	3	15	3	3
$\text{Chol}(V)_{1,1}$	553	513	500	511
$\text{Chol}(V)_{2,1}$	7844	18761	13796	9852
$\text{Chol}(V)_{2,2}$	530	523	517	477
$\text{mean}(\beta_{1i})$	6	5	12	12
$\text{mean}(\beta_{2i})$	21	22	28	10

Table 2: Effective sample sizes for estimates of marginal posterior distributions of population-level parameters, using the final 100,000 samples of HMC chains.

from the target posterior, *in parallel*, using 12 processing cores in a mid-2010 vintage Mac Pro. The median number of proposals required for each posterior sample was just under 40,000, the total number of likelihood evaluations was just under 18 million, and average acceptance rate was 1.7×10^{-5}. In absolute terms, these numbers may appear to be unfavorable to GDS. However, note that GDS can make better use of parallel computing infrastructure, leading to much shorter clock times. One could run multiple MCMC chains in parallel, but one would have to wait until all of the chains, individually, converge to the target posterior before sampling. Even then, there is no way to confirm that the chain has, in fact, converged.

We can draw inferences about the accuracy of GDS by comparing its estimated marginal densities to those that we get from HMC. Note that the HMC estimates are accurate only if all of the chains converge to the target density, and we have a large-enough effective sample size. This condition clearly does not hold, but we believe that it is sufficiently close for the majority of the population-level parameters for us to use HMC samples as a standard against which we can evaluate the GDS estimates. Figure 4 illustrates a comparison of the quantiles. We see that the densities for the elements of Δ and the Cholesky decomposition of V are extremely close. For other parameters, the convergence of the estimates is less clear. However, we can also see that the parameters for which the densities of HMC and GDS are not aligned are the same parameters for which there is high autocorrelation, and little movement, in the HMC chains. From these data, we infer that GDS matches HMC very well in terms of the marginal densities that it generates, with substantially less uncertainty and computational effort.

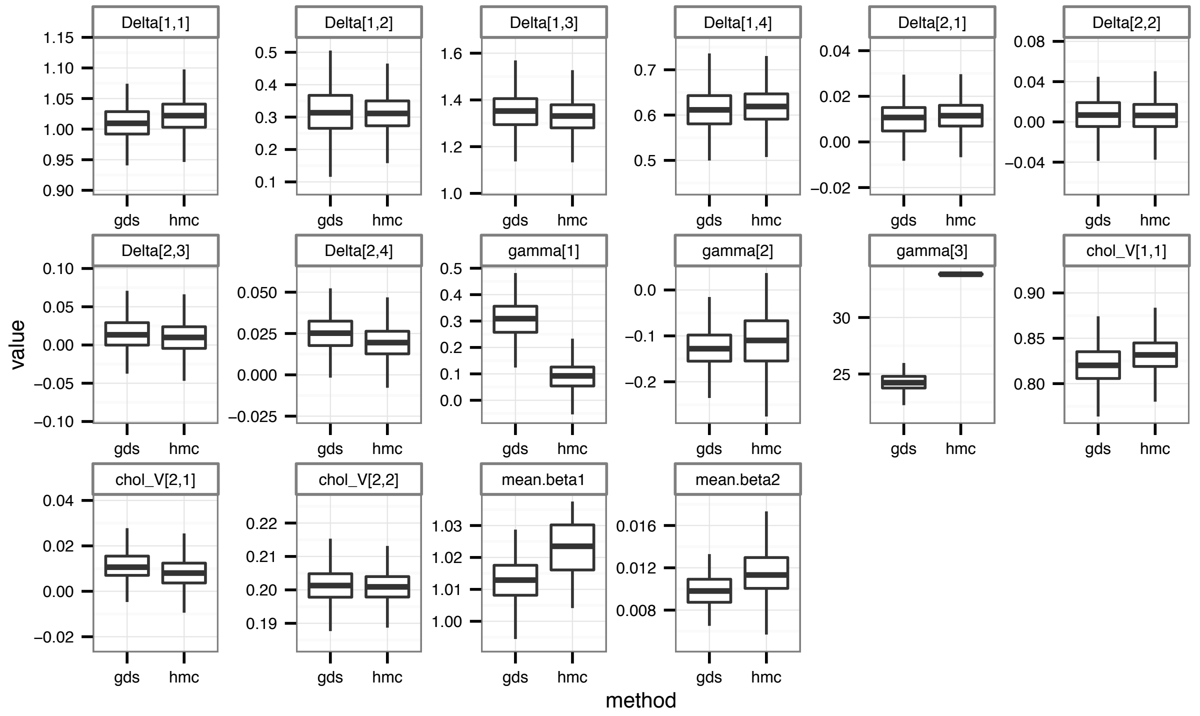


Figure 4: Estimated marginal distributions for population-level parameters using GDS and HMC.

4 Scalability and Sparsity

The ability for GDS to generate independent samples in parallel already makes it an attractive alternative to MCMC. In this section, we present an argument in favor of GDS's scalability. Our criterion for scalability is that the cost of running the algorithm grows close to linearly in the number of households. Our analysis considers the fundamental computational tasks involved in running GDS: computing the log posterior, its gradient and its Hessian; computing the Cholesky decomposition of the Hessian; and sampling from an MVN proposal distribution. We will show that scalability can be achieved because, under the conditional independence assumption, the Hessian of the log posterior is sparse, with a predictable sparsity pattern.

4.1 Computing Log Posteriors, Gradients and Hessians

Under the conditional independence assumption, the log posterior density is the sum of the logs of Equations 2 and 3, with a heterogeneous component

$$\sum_{i}^{N} \log f_i\left(y_i|\beta_i,\alpha\right) + \log \pi_i(\beta_i|\alpha) \tag{16}$$

and a homogeneous component $\log \pi(\alpha)$. This homogeneous component is the hyperprior on the population-level parameters, so its computation does not depend on N, while each additional household adds another element to the summation in Equation 16. Therefore, computation of the log posterior grows linearly in N. In the subsequent text, let k be the number of elements in each β_i and let p be the number of elements in α.

There are two reasons why we might need to compute the gradient and Hessian of the log posterior, namely to use in a derivative-based optimization algorithm to find the posterior mode, and for estimating the precision matrix of an MVN proposal distribution.[3] Ideally, we would derive the gradient and Hessian analytically, and write code to estimate it efficiently. For complicated models, the required effort for coding analytic gradients may not be worthwhile. An alternative would be numerical approximation

[3]GDS does not require either derivative-based optimization or using MVN proposals, but these are most likely the best choices for differentiable, unimodal posteriors.

through finite differencing. The fastest, yet least accurate, method for finite differencing for gradients, using either forward or backward differences, requires $Nk + p + 1$ evaluations of the log posterior. Since the computational cost of the log posterior also grows linearly with N, computing the gradient this way will grow quadratically in N. The cost of estimating a Hessian using finite differencing grow even faster in N. Also, if the Hessian is estimated by taking finite differences of gradients, and those gradients themselves were finite-differences, the accumulated numerical error can be so large that the Hessian estimates are useless.

Instead, we can turn to automatic differentiation (AD, also sometimes known as algorithmic differentiation). A detailed explanation of AD is beyond the scope of this paper, so we refer the reader to Griewank et al. (2008), or Section 8.2 in Nocedal and Wright (2006). Put simply, AD treats a function as a composite of subfunctions, and computes derivatives by repeatedly applying the chain rule. In practical terms, AD involves coding the log posterior using a specialized numerical library that keeps track of the derivatives of these subfunctions. When we compile the function that computes the log posterior, the AD library will "automatically" generate additional functions that return the gradient, the Hessian, and even higher-order derivatives.[4]

The remarkable feature of AD is that computing the gradient of a scalar-valued function takes no more than five times as long as computing the log posterior, *regardless of the number of parameters* (Griewank et al. 2008, p. xii). If the cost of the log posterior grows linearly in N, so will the cost of the gradient.

In most statistical software packages, like R, the default storage mode for any matrix is in

[4]There are a number of established AD tools available for researchers to use for many different programming environments. For C++, we use **CppAD** (Bell 2013), although **ADOL-C** (Walther and Griewank 2012) is also popular. When using the R statistical platform (R Development Core Team 2012), we call our C++ functions through the **Rcpp** interface (Eddelbuettel and François 2011; Bates and Eddelbuettel 2013). Alternatively, R users (among others) can compute their log posterior using **AD Model Builder** (Fournier et al. 2012), and call the log posterior and gradient functions using the **R2admb** package (Bolker and Skaug 2012). Matlab users have access to **ADMAT** (Coleman and Verma 2000), among other options.

a “dense” format; each element in the matrix is stored explicitly, regardless of the value. For a model with n variables, this matrix consists of n^2 numbers, each consuming 8 bytes of memory at double precision. If we have a dataset in which $N = 10000$, $k = 5$ and p is small, the Hessian for this model with $50,000 + p$ variables will consume more than 20GB of RAM. Furthermore, the computational effort for matrix-vector multiplication is quadratic in the number of columns, and matrix-matrix multiplication is cubic. To the extent that either of these operations is used in the mode-finding or sampling steps, the computational effort will grow much faster than the size of the dataset. Since multiplying a triangular matrix is roughly one-sixth as expensive as multiplying a full matrix, we could gain some efficiency by working with the Cholesky decomposition of the Hessian instead. However, the complexity of the Cholesky decomposition algorithm itself will still be cubic in N (Golub and Van Loan 1996, Ch. 1).

The source of scalability for GDS is in the sparsity of the Hessian. If the vast majority of elements in a matrix are zero, we do not need to store them explicitly. Instead, we need to store only the non-zero values, the row numbers of those values, and the index of the values that begin each column.[5] Under the conditional independence assumption, the cross-partial derivatives between heterogeneous parameters across households are all zero. Thus, the Hessian becomes sparser as the size of the dataset increases.

To illustrate, suppose we have a hierarchical model with six households, two heterogeneous parameters per household, and two population-level parameters for a total of 14 parameters. Figure 5 is a schematic of the sparsity structure of the Hessian; the vertical lines are the non-zero elements, and the dots are the zeros. There are 196 elements in this matrix, but only 169 are non-zero, and only 76 values are unique. Although the savings in RAM is modest in this illustration, the efficiencies are must greater when we add more households. If we had 1,000 households, with $k = 3$ and $p = 9$, there would be 3009

[5]This storage scheme is known as the Compressed Sparse Column (CSC) format. This common format is used by the **Matrix** package in R, and the **Eigen** numerical library, but it is not the only way to store a sparse matrix.

```
 [1,] | | . . . . . . . . . . | |
 [2,] | | . . . . . . . . . . | |
 [3,] . . | | . . . . . . . . | |
 [4,] . . | | . . . . . . . . | |
 [5,] . . . . | | . . . . . . | |
 [6,] . . . . | | . . . . . . | |
 [7,] . . . . . . | | . . . . | |
 [8,] . . . . . . | | . . . . | |
 [9,] . . . . . . . . | | . . | |
[10,] . . . . . . . . | | . . | |
[11,] . . . . . . . . . . | | | |
[12,] . . . . . . . . . . | | | |
[13,] | | | | | | | | | | | | | |
[14,] | | | | | | | | | | | | | |
```

Figure 5: Example of sparsity pattern under conditional independence.

parameters, and more than 9 million elements in the Hessian, yet no more than 63,000 are non-zero, of which about 34,500 are unique. As we add households, the number of non-zero elements of the Hessian grows only linearly in N.

The cost of estimating a dense Hessian using AD grows linearly with the number of variables (Griewank et al. 2008). When the Hessian is sparse, with a pattern similar to Figure 5, we can estimate the Hessian so that the cost is only a multiple of the cost of computing the log posterior. We achieve this by using a graph coloring algorithm to partition the variables into q groups (or "colors" in the graph theory literature), such that a small change in the variable in one group does not affect the partial derivative of any other variable in the same group. This means we could perturb all of the variables in the same group at the same time, recompute the gradient, and after doing that for all groups, still be able to recover an estimate of the Hessian. Thus, the computational cost for computing the Hessian grows with the number of groups, not the number of parameters. Because the household-level parameters are conditionally independent, we do not need to add groups as we add households. For the Hessian sparsity pattern in Figure 5, we need only four groups: one for each of the heterogeneous parameters across all of the households, and one for each of the two population-level parameters. In the upcoming binary choice example in Section 4.4, for which $k = 3$, there are $\frac{1}{2}\left(k^2 + 5k\right) = 12$ groups, no matter how many households we have in the dataset.

Curtis, Powell, and Reid (1974) introduce the idea of reducing the number of evaluations to estimate sparse Jacobians. Powell and Toint (1979) describe how to partition variables into appropriate groups, and how to recover Hessian information through back-substitution. Coleman and Moré (1983) show that the task of grouping the variables amounts to a classic graph-coloring problem. Most AD software applies this general principle to computing sparse Hessians. Alternatively, R users can use the **sparseHessianFD** package (Braun 2013b) to efficiently estimate sparse Hessians through finite differences of the gradient, as long as the sparsity pattern is known in advance, and as long as the gradient was not itself estimated through finite differencing. This package is an interface to the algorithms in Coleman, Garbow, and Moré (1985b) and Coleman, Garbow, and Moré (1985a).

4.2 Finding the posterior mode

For simple models and small datasets, standard default algorithms (like the `optim` function in R) are sufficient for finding posterior modes and estimating Hessians. For larger problems, one should choose optimization tools more thoughtfully. For example, many of the R optimization algorithms default to finite differencing of gradients when a gradient function is not provided explicitly. Even if the user can provide the gradient, many algorithms will store a Hessian, or an approximation to it, densely. Neither feature is attractive when the number of households is large.

For this section, let's assume that the log posterior is twice-differentiable and unimodal.[6] There are two approaches that one can take. The first is to use a "limited memory"

[6]Neither assumption is required for GDS, but most marketing models satisfy them, and maintaining them simplifies our exposition.

optimization algorithm that approximates the curvature of the log posterior over successive iterations. Many algorithms are described in Nocedal et al. (2006), and are implemented in R using the default `optim` function, or contributed packages like `nloptwrap` (Borchers 2013). Once the algorithm finds the posterior mode, there remains the need to compute the Hessian exactly.

The second approach is to run a quasi-Newton algorithm, and compute the Hessian at each iteration explicitly, but store the Hessian in a sparse format. For example, the **trustOptim** package for R (Braun 2013d) implements a trust-region algorithm that exploits the sparsity of the Hessian. The user can supply a Hessian that is derived analytically, computed using AD, or estimated numerically using **sparseHessianFD**. Since memory requirements and matrix computation costs will grow only linearly in N, finding the posterior mode becomes feasible for large problems, compared to similar algorithms that ignore sparsity.

We should note that we cannot predict the time to convergence for general problems. Log posteriors with ridges or plateaus, or that require extensive computation themselves, may still take a long time to find the local optimum. Whether any mode-finding algorithm is "fast enough" depends on the specific application. However, if one optimization algorithm has difficulty finding a mode, another algorithm may do better.

4.3 Sampling from an MVN distribution

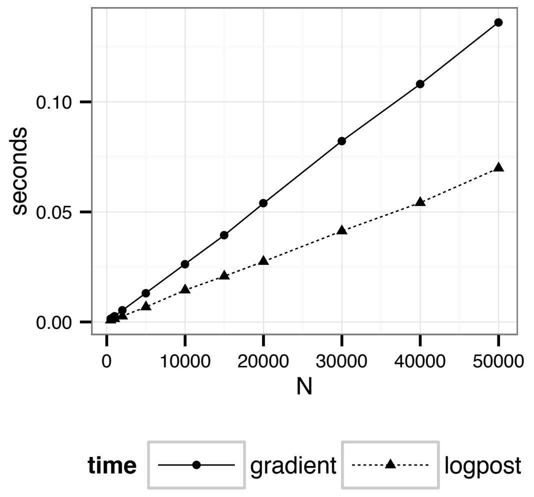
Once we find the posterior mode, and the Hessian at the mode, generating proposal samples from an MVN(θ^*, sH^{-1}) distribution is straightforward. Let $\frac{1}{s}H = \Lambda\Lambda'$ represent the Cholesky decomposition of the precision of the proposal, and let z be a vector of $Nk + p$ samples from a standard normal distribution. To sample θ from an MVN, solve the triangular linear system $\Lambda'\theta = x$, and then add θ^*. Since $E(z) = 0$, $E(\theta) = \theta^*$, and since $E(zz') = I$, $\text{cov}(\theta) = \Lambda'^{-1}\Lambda^{-1} = (\Lambda\Lambda')^{-1} = sH^{-1}$.

Because Λ is sparse, the costs of both solving the triangular system, and the premultiplication, grow linearly with the number of nonzero elements, which itself grows linearly in N (Davis 2006). If Λ were dense, then the cost of solving the triangular system would grow quadratically in N. Furthermore, computing the MVN density would involve premultiplying z by a triangular matrix, whose cost is cubic in N (Golub et al. 1996).

Computation of the Cholesky decomposition can also benefit from the sparsity of the Hessian. If H were dense $Nk + p$ square, symmetric matrix, then, holding k and p constant, the complexity order of the Cholesky decomposition is N^3 (Golub et al. 1996). There are a number of different algorithms that one can use for decomposing sparse Hessians, as discussed in Davis (2006). The typical strategy is to first permute the rows and columns of H to minimize the number of nonzero elements in Λ, and then compute the sparsity pattern. This part can be done just once. With the sparsity pattern in hand, the next step is to compute those nonzero elements in Λ. The time for this step grows with the sum of the squares of the number of nonzero elements in each column of Λ(Davis 2006). Because each additional household adds k columns to Λ, with an average of $p + \frac{1}{2}(k+1)$ nonzero elements per column, we can compute the sparse Cholesky decomposition in time that is linear in N. Most numerical libraries that support sparse matrices, including **Eigen** (for C++) and the **Matrix** package (for R), provide ready-to-use algorithms for sparse Cholesky decompositions.

4.4 Scalability Test

Next, we provide some empirical evidence of the scalability of GDS through a simulation study. For a hypothetical dataset with N households, let y_i be the number of times household i visits a store during a T week period. The probability of a visit in a single



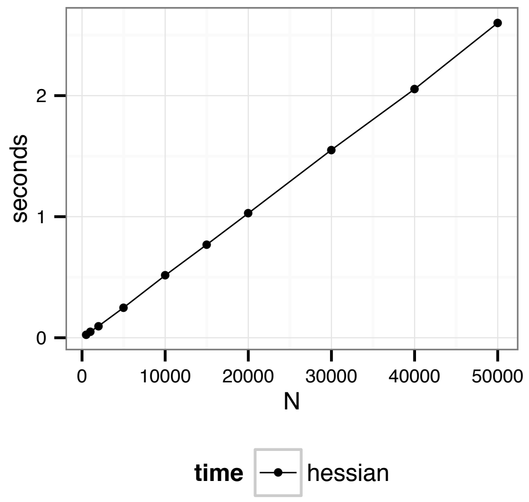


Figure 6: Average computation time for 100 evaluations of log posterior, gradient and Hessian.

week is p_i, where $\operatorname{logit} p_i = \beta_i' x_i$, and x_i is a vector of k covariates. The distribution of β_i across the population is MVN with mean $\bar{\beta}$ and covariance Σ. In all conditions of the test, we set $k = 3$ and $T = 52$, and vary the number of households by setting N to one of 10 discrete values from 500 to 50,000. The "true" values of $\bar{\beta}$ are -10, 0 and 10, and the "true" Σ is $0.1I$. We place weakly informative priors on both $\bar{\beta}$ and Σ.

In Figure 6, we plot the average time, across 100 replications, to compute the log posterior, the gradient, and the Hessian. As expected, each of these computations grows linearly in N. In Figure 7, we plot average times for the steps involved in sampling from an MVN: adding a vector to columns of a dense matrix, computing a sparse Cholesky decomposition, multiplying a sparse triangular matrix by a dense matrix, sampling standard normal random variates, and solving a sparse triangular linear system. Again, we see that the time for all of these steps is linear in N.

Table 3 summarizes the acceptance rates and scale factors when generating 50 samples from the posterior using GDS for different values of N. Although there is a weak trend of increasing acceptance rates with N,we really cannot say with any certainty that acceptance rates will always be either larger or smaller for larger datasets. This is because the

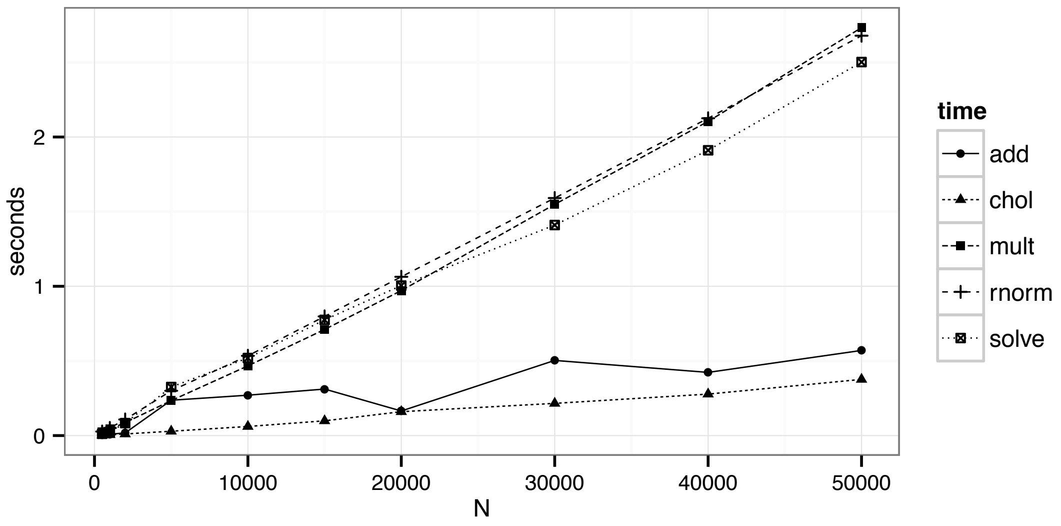


Figure 7: Computation time, averaged over 100 replications, for adding a vector to matrix columns (add); a sparse Cholesky decomposition (chol); multiplying a sparse triangular matrix by a dense matrix (mult); sampling standard normal random variates (rnorm); and solving a sparse triangular linear system (solve)

N	500	1,000	2,000	5,000	10,000	15,000	20,000	30,000	40,000	50,000
scale factor	1.22	1.16	1.10	1.08	1.04	1.03	1.03	1.03	1.03	1.02
acc. rate ($\times 10^{-5}$)	2.1	1.5	2.6	0.7	2.9	2.6	3.6	3.9	2.7	3.3

Table 3: GDS acceptance rates for binary choice simulation. For each condition, $k = 3$, so there are 6 population-level parameters and $3N$ heterogeneous parameters.

acceptance rate could be influenced by using different scale factors on the Hessian for the MVN proposal density. However, we would expect higher acceptance rates as the target posterior density approaches an MVN asymptotically. Thus, since none of the GDS steps grows faster than linearly in N, we are confident in the scalability of the overall GDS algorithm.

5 Estimating Marginal Likelihoods

Now, we turn to another advantage of GDS: the ability to generate accurate estimates of the marginal likelihood of the data with little additional computation. A number of researchers have proposed methods to approximate the marginal likelihood, $\mathcal{L}(y)$ from MCMC output (Gelfand and Dey 1994; Newton and Raftery 1994; Chib 1995; Raftery et al. 2007), but none has achieved universal acceptance as being unbiased, stable and easy to compute. In fact, Lenk (2009) demonstrated that methods which depend solely on samples from the posterior density could suffer from a "pseudo-bias," and he proposes an importance-sampling method to correct for it. This bias arises because the convex hull of MCMC samples defines only a subset of the *posterior* support, while $\mathcal{L}(y)$ is defined as an integral of the data likelihood over the *prior* distribution. Lenk demonstrates that his method dominates other popular methods, although with substantial computational effort. Thus, the estimation of the marginal likelihood remains a difficult problem in MCMC-based Bayesian statistics.

GDS allows us to estimate the marginal likelihood using quantities that we can collect during the course of the estimation procedure. Recall that $q(u)$ is the probability that, given a threshold value u, a proposal from $g(\theta)$ is accepted as a sample from $\pi(\theta|y)$. Therefore, one can express the expected marginal acceptance probability for any one posterior sample as

$$\gamma = \int_0^1 q(u)p(u|y)\, du. \tag{17}$$

Substituting in Equation 8,

$$\gamma = \frac{c_1}{c_2\mathcal{L}(y)} \int_0^1 q^2(u)\, du \tag{18}$$

Applying a change of variables so $v = -\log u$, and then rearranging terms,

$$\mathcal{L}(y) = -\frac{c_1}{c_2\gamma} \int_0^\infty q^2(v) \exp(-v)\, dv. \tag{19}$$

The values for c_1 and c_2 are immediately available from the GDS algorithm. One can estimate γ from the observed acceptance rate $\hat{\gamma}$ as the inverse of the mean of the average number of proposals per accepted sample.

What remains is estimating the integral in Equation 19, for which we use the same proposal draws that we already collected for estimating $\widehat{q}_v(v)$. The empirical CDF of these draws is discrete, so we can partition the support of $q_v(v)$ at $v_1 \dots v_M$. Also, since $\widehat{q}_v(v)$ is the proportion of proposal draws less than v, we have $q_v(v_i) = \frac{i}{M}$. Therefore,

$$\int_0^\infty q^2(v)\exp(-v)dv \approx \sum_{i=1}^{M}\int_{v_i}^{v_{i+1}}\left(\frac{i}{M}\right)^2\exp(-v_i)dv \tag{20}$$

$$= \frac{1}{M^2}\sum_{i=1}^{M} i^2\left[\exp(-v_i)-\exp(-v_{i+1})\right] \tag{21}$$

$$= \frac{1}{M^2}\sum_{i=1}^{M}(2i-1)\exp(-v_i) \tag{22}$$

Putting all of this together, we can estimate the marginal likelihood as

$$\mathcal{L}(y) \approx \frac{c_1}{M^2 c_2 \hat{\gamma}}\sum_{i=1}^{M}(2i-1)\exp(-v_i) \tag{23}$$

As a demonstration of the accuracy of this estimator, we use the same linear regression example that Lenk (2009) uses.

$$y_{it} \sim N(x_i'\beta, \sigma^2),\ i = 1\dots n, t = 1\dots T \tag{24}$$

$$\beta|\sigma \sim N(\beta_0, \sigma^2 V_0) \tag{25}$$

$$\sigma^2 \sim IG(r, \alpha) \tag{26}$$

For this model, $\mathcal{L}(y)$ is a multivariate-t density (MVT), which we can compute analytically. This allows us to compare the GDS estimates of $\mathcal{L}(y)$ with the "truth." To do this, we conducted a simulation study for simulated datasets of different numbers of observations $n \in \{200, 2000\}$ and numbers of covariates $k \in \{5, 25, 100\}$. For each n, k pair, we simulated 25 datasets. For each dataset, each vector x_i includes an intercept and k iid samples from a standard normal density. Thus, there are $k + 2$ parameters, corresponding to the elements of β, plus σ. The true intercept term is 5, and the remaining true β parameters are linearly spaced from -5 to 5. In all cases, there are $T = 25$ observations per unit. Hyperpriors are set as $r = 2$, $\alpha = 1$, β_0 as a zero vector and $V_0 = 0.2 \cdot I_k$.

For each dataset, we collected 250 draws from the posterior density using GDS, with different numbers of proposal draws ($M = 1,000$ or $10,000$), and scale factors ($s = 0.5, 0.6, 0.7,$ or $0.8\}$) on the Hessian (sH is the precision matrix of the MVN proposal density, and lower scale factors generate more diffuse proposals). We excluded the $s = 0.8$, $n = 200$ case because the proposal density was not sufficiently diffuse to ensure that $\Phi(\theta|y)$ was between 0 and 1 across the M proposal draws.

Table 4 presents the true log marginal likelihood (MVT), along with estimates using GDS, the importance sampling method in Lenk (2009), and the harmonic mean estimator Newton et al. (1994). We also included the mean GDS acceptance probabilities. We can see that the GDS estimates for the log marginal likelihood are remarkably close to the MVT densities, and are robust when we use different scale factors. Accuracy appears to be better for larger datasets than smaller ones; improving the approximation of $p(u)$ by increasing the number of proposal draws offers negligible improvement. Note that the performance of the GDS method is comparable to that of Lenk, but is much better than the harmonic mean estimator. The GDS method is similar to Lenk's in that it computes the probability that a proposal draw falls within the support of the posterior density. However, note that the inputs to the GDS estimator are intrinsically generated as the GDS algorithm progresses. In contrast, the Lenk estimator requires an additional importance sampling run after the MCMC draws are collected.

				MVT		GDS		Lenk		HME		Mean
k	n	M	scale	mean	sd	mean	sd	mean	sd	mean	sd	Acc %
5	200	1000	0.5	-309	6.6	-309	6.6	-311	6.8	-287	7.1	22.1
5	200	1000	0.6	-309	6.6	-309	6.7	-310	6.9	-287	6.9	40.5
5	200	1000	0.7	-309	6.6	-309	6.7	-310	6.5	-287	6.3	57.1
5	200	10000	0.5	-309	6.6	-309	6.6	-311	6.7	-287	6.7	24.0
5	200	10000	0.6	-309	6.6	-309	6.6	-310	7.5	-287	6.8	40.9
5	200	10000	0.7	-309	6.6	-309	6.7	-310	7.0	-287	7.1	55.2
5	2000	1000	0.5	-2866	46.2	-2865	46.3	-2868	46.2	-2836	46.2	22.1
5	2000	1000	0.6	-2866	46.2	-2866	46.2	-2868	45.7	-2836	45.5	37.8
5	2000	1000	0.7	-2866	46.2	-2866	46.3	-2867	45.9	-2836	45.9	49.6
5	2000	1000	0.8	-2866	46.2	-2866	46.2	-2867	46.3	-2835	46.3	64.6
5	2000	10000	0.5	-2866	46.2	-2866	46.4	-2867	46.7	-2836	46.9	25.3
5	2000	10000	0.6	-2866	46.2	-2866	46.2	-2867	45.8	-2836	46.3	36.3
5	2000	10000	0.7	-2866	46.2	-2866	46.4	-2867	46.0	-2836	46.3	51.4
5	2000	10000	0.8	-2866	46.2	-2866	46.2	-2867	46.5	-2835	46.3	72.0
25	200	1000	0.5	-387	8.1	-385	8.2	-391	7.6	-292	8.5	2.8
25	200	1000	0.6	-387	8.1	-386	8.1	-390	9.5	-292	8.8	8.1
25	200	1000	0.7	-387	8.1	-386	8.3	-390	8.0	-292	8.8	16.2
25	200	10000	0.5	-387	8.1	-385	8.5	-390	8.2	-292	8.4	1.7
25	200	10000	0.6	-387	8.1	-385	8.2	-390	8.9	-292	8.8	6.2
25	200	10000	0.7	-387	8.1	-386	8.2	-390	8.7	-292	9.1	20.0
25	2000	1000	0.5	-2990	28.7	-2989	28.8	-2994	28.3	-2865	28.8	2.7
25	2000	1000	0.6	-2990	28.7	-2989	28.7	-2993	28.4	-2864	29.0	4.6
25	2000	1000	0.7	-2990	28.7	-2989	28.9	-2991	30.0	-2864	29.5	15.4
25	2000	1000	0.8	-2990	28.7	-2990	28.7	-2992	29.6	-2864	29.4	43.1
25	2000	10000	0.5	-2990	28.7	-2988	29.2	-2992	28.5	-2864	28.9	.8
25	2000	10000	0.6	-2990	28.7	-2989	29.1	-2993	29.4	-2864	28.9	3.7
25	2000	10000	0.7	-2990	28.7	-2990	29.0	-2993	28.9	-2864	28.9	17.1
25	2000	10000	0.8	-2990	28.7	-2990	28.6	-2993	28.2	-2865	28.2	43.3
100	200	1000	0.5	-660	6.7	-661	6.5	-683	8.8	-292	9.2	.3
100	200	1000	0.6	-660	6.7	-660	6.6	-678	8.5	-286	9.0	.3
100	200	1000	0.7	-660	6.7	-659	7.1	-673	7.8	-282	8.0	.4
100	200	10000	0.5	-660	6.7	-659	6.9	-682	9.1	-288	10.4	.1
100	200	10000	0.6	-660	6.7	-660	5.7	-678	8.8	-286	8.9	.1
100	200	10000	0.7	-660	6.7	-658	6.7	-674	7.3	-282	8.4	.1
100	2000	1000	0.5	-3364	24.4	-3364	24.8	-3370	27.5	-2871	27.1	.3
100	2000	1000	0.6	-3364	24.4	-3362	24.6	-3369	24.3	-2868	25.3	.6
100	2000	1000	0.7	-3364	24.4	-3361	23.9	-3371	25.6	-2870	25.4	1.1
100	2000	1000	0.8	-3364	24.4	-3362	23.9	-3370	26.0	-2868	26.1	3.2
100	2000	10000	0.5	-3364	24.4	-3362	24.0	-3372	25.3	-2870	25.2	.1
100	2000	10000	0.6	-3364	24.4	-3360	24.9	-3368	25.3	-2867	25.4	.1
100	2000	10000	0.7	-3364	24.4	-3360	24.6	-3370	25.5	-2869	25.5	.4
100	2000	10000	0.8	-3364	24.4	-3362	24.5	-3367	24.3	-2867	24.4	3.0

Table 4: Results of simulation study for effectiveness of estimator for log marginal likelihood.

6 Discussion of practical considerations and limitations

To many marketing researchers who use Bayesian methods, having spent long work hours dealing with MCMC convergence and efficiency issues, the utility of an algorithm like GDS is appealing. GDS allows researchers to sample from a posterior density in parallel, without having to worry about whether an MCMC estimation chain has converged. If heterogeneous units (like households) are conditionally independent, then the sparsity of the Hessian of the log posterior lets us construct a sampling algorithm whose complexity grows only linearly in the number of units. We believe that GDS makes Bayesian inference more attractive to practitioners who might otherwise be put off by the inefficiencies of MCMC.

This is not to say that GDS is guaranteed to generate perfect samples from the target posterior distribution. One area of potential concern is that the empirical distribution $\widehat{q}_v(v)$ is only a discrete approximation to $q_v(v)$. That discretization could introduce some error into the estimate of the posterior density. This error can be reduced by increasing M (the number of proposal draws that we use to compute $\widehat{q}_v(v)$), at the expense of costlier computation of $\widehat{q}_v(v)$ and, possibly, lower acceptance rates. In our experience with GDS, we have not found this to be a problem, but some applications for which this may be an issue might exist.

Like many other methods that collect random samples from posterior distributions, the efficiency of GDS depends in part on a prudent selection of the proposal density $g(\theta)$. For the examples in this paper, we used a multivariate normal density that is centered at the posterior mode with a covariance matrix that is proportional to the inverse of the Hessian at the mode. One might then wonder if there is an optimal way to determine just how "scaled out" the proposal covariance needs to be. At this time, we think that trial and error is, quite frankly, the best alternative. For example, if we start with a small M

(say, 100 draws), and find that $\Phi(\theta|y) > 1$ for any of the M proposals, we have learned that the proposal density is not valid at little computational or real-time cost. We can then re-scale the proposal until $\Phi(\theta|y) < 1$, and then gradually increase M until we get a good approximation to $p(u)$. In our experience, even if an acceptance rate appears to be low (say, 0.0001), we can still collect draws in parallel, so the "clock time" remains much less than the time we spend trying to optimize selection of the proposal. This scaling phase evaluation in the GDS method is no different, in principle, than the tuning step in a Metropolis-Hasting algorithm.

There are many popular models, such as multinomial probit, for which the likelihood of the observed data is not available in closed form. When direct numerical approximations to these likelihoods (e.g., Monte Carlo integration) are not tractable, MCMC with data augmentation is a possible alternative. Recent advances in parallelization using graphical processing units (GPUs) might make numerical estimation of integrals more practical than it was even 10 years ago (Suchard et al. 2010). If so, then GDS could be a viable, efficient alternative to data augmentation in these kinds of models. Multiple imputation of missing data could suffer from the same kinds of problems, since a latent parameter, introduced for the data augmentation step, is only weakly identified on its own. If the number of missing data points is small, one could treat the representation of the missing data points as if they were parameters. But the implications of this require additional research.

Another consideration is the case of multimodal posteriors. GDS does require finding the global posterior mode, and all the models discussed in this paper have unimodal posterior distributions. When the posterior is multimodal, one could instead use a mixture of normals as the proposal distribution. The idea is to not only find the global mode, but any local ones as well, and center each mixture component at each of those local modes. The GDS algorithm itself remains unchanged, as long as the global posterior mode matches the global proposal mode. We recognize that finding all of the local modes could be a hard problem, and there is no guarantee that any optimization algorithm will find all local extrema in a reasonable amount of time. In practical terms, MCMC, offers no such guarantees either.

For researchers who use R for statistical analysis, there are a number of packages that can help with implementation of GDS. The **bayesGDS** package (Braun 2013a) includes functions to run the rejection sampling phase (lines 20-36 in Algorithm 1). To use these functions, the user will supply the $\log \Phi(\theta_m|y)$ for the M proposal draws, along with functions that compute the log posterior, sample from the proposal distribution, and compute the proposal density. This package also includes a function that estimates the log marginal likelihood from the GDS output. If the proposal distribution is MVN, and either the covariance or precision matrix is sparse, then one can use the **sparseMVN** (Braun 2013c) package to sample from the MVN by taking advantage of that sparsity. The **sparseHessianFD** (Braun 2013b) package estimates a sparse Hessian by taking finite differences of gradients of the function, as long as the user can supply the sparsity pattern (which should be the case under conditional independence). Finally, the **trustOptim** package (Braun 2013d) is a nonlinear optimization package that uses a sparse Hessian to include curvature information in the algorithm.

References

Bates, Douglas and Dirk Eddelbuettel (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. *Journal of Statistical Software* **52**(5), 1–24.

Bell, B.M. (2013). *CppAD: a package for C++ algorithmic differentiation*. Computational Infrastructure for Operations Research. URL: `www.coin-or.org/CppAD`.

Bolker, Ben and Hans Skaug (2012). *R2admb: ADMB to R interface functions*. R package version 0.7.5.3. URL: `cran.r-project.org/web/packages/R2admb`.

Borchers, Hans W (2013). *nloptwrap: Wrapper for Package nloptr*. R package version 0.5-1. URL: `cran.r-project.org/web/packages/nloptwrap`.

Braun, Michael (2013a). *bayesGDS: an R package for generalized direct sampling*. R package version 0.6.0. URL: `cran.r-package.org/web/packages/bayesGDS`.

Braun, Michael (2013b). *sparseHessianFD: an R package for estimating sparse Hessians*. R package version 0.1.1. URL: `cran.r-project.org/web/packages/sparseHessianFD`.

Braun, Michael (2013c). *sparseMVN: an R package for MVN sampling with sparse covariance and precision matrices.* R package version 0.1.0. URL: `cran.r-project.org/web/packages/sparseMVN`.

Braun, Michael (2013d). *trustOptim: an R package from optimization using trust regions*. R package version 0.8.2. URL: `cran.r-project.org/web/packages/trustOptim`.

Brooks, Steve, Andrew Gelman, Galin Jones, and Xiao-Li Meng, eds. (2010). *Handbook of Markov Chain Monte Carlo*. Boca Raton, Fla.: Chapman and Hall/CRC.

Carlin, Bradley P and Thomas A Louis (2000). *Bayes and Empirical Bayes Methods for Data Analysis*. 2nd. Boca Raton, Fla.: Chapman and Hall/CRC.

Chen, Ming-Hui, Qi-Man Shao, and Joseph G Ibrahim (2000). *Monte Carlo Methods in Bayesian Computation*. New York: Springer.

Chib, Siddhartha (1995). Marginal Likelihood from the Gibbs Output. *Journal of the American Statistical Association* **90**(432) Dec., 1313–1321.

Coleman, Thomas F, Burton S Garbow, and Jorge J Moré (1985a). Algorithm 636: FORTRAN Subroutines for Estimating Sparse Hessian Matrices. *ACM Transactions on Mathematical Software* **11**(4), 378.

Coleman, Thomas F, Burton S Garbow, and Jorge J Moré (1985b). Software for Estimating Sparse Hessian Matrices. *ACM Transactions on Mathematical Software* **11**(4) Dec., 363–377.

Coleman, Thomas F and Jorge J Moré (1983). Estimation of Sparse Jacobian Matrices and Graph Coloring Problems. *SIAM Journal on Numerical Analysis* **20**(1) Feb., 187–209.

Coleman, Thomas F and Arun Verma (2000). ADMIT-1: Automatic Differentiation and MATLAB Interface Toolbox. *ACM Transactions on Mathematical Software* **26**(1), 150–175.

Curtis, A R, M J D Powell, and J K Reid (1974). On the Estimation of Sparse Jacobian Matrices. *Journal of the Institute of Mathematics and its Applications* **13**, 117–119.

Davis, Timothy A (2006). *Direct Methods for Sparse Linear Systmes*. Philadelphia: Society for Industrial and Applied Mathematics.

Duane, Simon, A D Kennedy, Brian J Pendleton, and Duncan Roweth (1987). Hybrid Monte Carlo. English. *Physics Letters B* **195**(2) Sept., 216–222.

Eddelbuettel, Dirk and Romain François (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software* **40**(8), 1–18.

Fournier, David A, Hans J Skaug, Johnoel Ancheta, James Ianelli, Arni Magnusson, Mark N Maunder, Anders Nielsen, and John Sibert (2012). AD Model Builder: Using Automatic Differentiation for Statistical Inference of Highly Parameterized Complex Nonlinear Models. English. *Optimization Methods and Software* **27**(2) Apr., 233–249.

Gelfand, Alan E and Dipak K. Dey (1994). Bayesian Model Choice: Asymptotics and Exact Calculations. *Journal of the Royal Statistical Society, Series B* **56**(3), 501–514.

Gelfand, Alan E and Adrian F M Smith (1990). Sampling-Based Approaches to Calculating Marginal Densities. *Journal of the American Statistical Association* **85**(410) June, 398–409.

Gelman, Andrew, John B Carlin, Hal S Stern, and Donald B Rubin (2003). *Bayesian Data Analysis*. Boca Raton, Fla.: Chapman and Hall.

Girolami, Mark and Ben Calderhead (2011). Riemann Manifold Langevin and Hamiltonian Monte Carlo. *Journal of the Royal Statistical Society, Series B* **73**, 1–37.

Golub, Gene H and Charles F Van Loan (1996). *Matrix Computations*. 3rd. Johns Hopkins University Press.

Griewank, Andreas and Andrea Walther (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics.

Hoffman, Matthew D and Andrew Gelman (2011). The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *arXiv.org* Nov. arXiv: `1111.4246v1 [stat.CO]`.

Lenk, Peter (2009). Simulation Pseudo-Bias Correction to the Harmonic Mean Estimator of Integrated Likelihoods. *Journal of Computational and Graphical Statistics* **18**(4), 941–960.

Manchanda, Puneet, Peter E Rossi, and Pradeep K Chintagunta (2004). Response Modeling with Nonrandom Marketing-Mix Variable. *Journal of Marketing Research* **41**(4), 467–478.

Neal, Radford M (2011). MCMC Using Hamiltonian Dynamics. In: *Handbook of Markov Chain Monte Carlo*. Ed. by Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. Boca Raton, Fla.: CRC Press, 113–162.

Newton, Michael A and Adrian E Raftery (1994). Approximate Bayesian Inference with the Weighted Likelihood Bootstrap. *Journal of the Royal Statistical Society, Series B* **56**(1), 3–48.

Nocedal, Jorge and Stephen J Wright (2006). *Numerical Optimization*. Second edition. Springer.

Papaspiliopoulos, Omiros and Gareth Roberts (2008). Stability of the Gibbs Sampler for Bayesian Hierarchical Models. English. *The Annals of Statistics* **36**(1) Feb., 95–117.

Powell, M J D and Ph. L. Toint (1979). On the Estimation of Sparse Hessian Matrices. *SIAM Journal on Numerical Analysis* **16**(6) Dec., 1060–1074.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Raftery, Adrian E, Michael A Newton, Jaya M Satagopan, and Pavel N Krivitsky (2007). Estimating the Integrated Likelihood via Posterior Simulation Using the Harmonic Mean Identity. In: *Bayesian Statistics 8*. Ed. by J. M. Bernardo, M J Bayarri, J O Berger, A P Dawid, D Heckerman, and A F M Smith. Proceedings, 1–45.

Rossi., Peter (2012). *bayesm: Bayesian Inference for Marketing/Micro-econometrics*. R package version 2.2-5. URL: `cran.r-project.org/web/packages/bayesm`.

Rossi, Peter E and Greg M Allenby (2003). Bayesian Statistics and Marketing. *Marketing Science* **22**(3), 304–328.

Rossi, Peter E, Greg M Allenby, and Robert McCulloch (2005). *Bayesian Statistics and Marketing*. Chichester, UK: John Wiley and Sons.

Suchard, Marc A, Quanli Wang, Cliburn Chan, Jacob Frelinger, Andrew Cron, and Mike West (2010). Understanding GPU Programming for Statistical Computation: Studies in Massively Parallel Massive Mixtures. English. *Journal of Computational and Graphical Statistics* **19**(2) Jan., 419–438.

Tibbits, Matthew M, Murali Haran, and John C Liechty (2010). Parallel multivariate slice sampling. English. *Statistics and Computing* **21**(3) Apr., 415–430.

Walker, Stephen G, Purushottam W Laud, Daniel Zantedeschi, and Paul Damien (2011). Direct Sampling. *Journal of Computational and Graphical Statistics* **20**(3), 692–713.

Walther, Andrea and Andreas Griewank (2012). Getting Started with ADOL-C. In: *Combinatorial Scientific Computing*. Ed. by U Naumann and O Schenk. Chapman-Hall CRC Computational Science, 181–202.

Yang, Sha and Greg M Allenby (2003). Modeling Interdependent Consumer Preferences. *Journal of Marketing Research* **40**(3), 282–294.