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

June 27, 2014

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. 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 joint or conditional 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,

\footnote{To simplify the exposition in the rest of this paper, we will use the term “household” to refer to any heterogeneous agent.}
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. To use this 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. The method also allows us to compute the marginal likelihood of the data under the model with little additional computation.

We derive the 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 this method 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.

The method 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.\textsuperscript{2} This feature 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, the method 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 steps of the algorithm, 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 our method should replace MCMC in all cases. It may not be practical for models with discrete parameters, a very large number of modes, or for which computing the log posterior density itself is difficult. The method 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

\textsuperscript{2}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.
models of the kind marketing researchers will encounter can be estimated well using our method, at least relative to the effort involved in using MCMC. Like MCMC and other non-MCMC methods, our method 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 presents examples that demonstrate the accuracy of the method, including one for which MCMC is impractical. In Section 4, we explain when and why the method is scalable, and offer evidence to support that claim via a simulation study. In Section 5, we show how to use output from the method to estimate the marginal likelihood of the data. Finally, in Section 6, we discuss practical implementation issues, including limitations of the approach.

## 2 Method details

### 2.1 Theoretical basis

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

\[
\pi(\theta | y) = \frac{f(y|\theta)\pi(\theta)}{L(y)} = \frac{D(\theta, y)}{L(y)}
\]

where \( D(\theta, y) \) 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

\[
f(y|\theta) = \prod_{i=1}^{N} f_i(y_i | \beta_i, \alpha)
\]
where \( i \) indexes households. Each \( y_i \) is a vector of observed data, each \( \beta_i \) is a vector of heterogeneous parameters, and \( \alpha \) is a vector of homogeneous population-level parameters. The \( \beta_i \) are distributed across the population of households according to a mixing distribution \( \pi(\beta_i|\alpha) \), which also serves as the prior on each \( \beta_i \). The elements of \( \alpha \) may influence either the household-level data likelihoods, or the mixing distribution (or both). In this example, \( \theta \) includes all \( \beta_1 \ldots \beta_N \) and all elements of \( \alpha \). The prior itself can be factored as

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

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

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

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

\[
\pi(\theta|y) = \Phi(\theta|y) \cdot g(\theta) \cdot \frac{c_1}{c_2 \cdot L(y)}. \tag{5}
\]

An important restriction on the choice of \( g(\theta) \) is that the inequality \( 0 \leq \Phi(\theta|y) \leq 1 \) must hold, at least for any \( \theta \) with a non-negligible posterior density. We discuss this restriction, along with 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 \( p(u|\theta, y) = \frac{\pi(\theta|y)}{\Phi(\theta|y)} = \frac{c_1}{c_2 L(y)} g(\theta) \). Then construct a joint density of \( \theta|y \) and \( u|\theta, y \),
where

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

(6)

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

\[ p(\theta|y) = \frac{\pi(\theta|y)}{\Phi(\theta|y)} \int_0^{\Phi(\theta|y)} du = \pi(\theta|y) \]  

(7)

Simulating from \( p(\theta|y) \) is now equivalent to simulating from the target posterior \( \pi(\theta|y) \).

Using Equations 5 and 6, the marginal density of \( u|y \) is

\[ p(u|y) = \int_{\theta} \frac{\pi(\theta|y)}{\Phi(\theta|y)} \mathbb{1}[u < \Phi(\theta|y)] d\theta \]  

(8)

\[ = \frac{c_1}{c_2 L(y)} \int_{\theta} \mathbb{1}[u < \Phi(\theta|y)] g(\theta) d\theta \]  

(9)

\[ = \frac{c_1}{c_2 L(y)} q(u) \]  

(10)

where \( q(u) = \int_{\theta} \mathbb{1}[u < \Phi(\theta|y)] g(\theta) d\theta \). This \( q(u) \) function is the probability that any candidate draw from \( g(\theta) \) will satisfy \( \Phi(\theta|y) > u \). The sampler comes from recognizing that \( p(\theta, u|y) \) can be written differently from, but equivalently to, Equation 6.

\[ p(\theta, u|y) = p(\theta|u, y) \, p(u|y) \]  

(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

\[ p(\theta|u, y) = \frac{p(\theta, u|y)}{p(u|y)} \]  

(12)

\[ = \frac{c_1}{c_2 L(y) p(u|y)} \mathbb{1}[u < \Phi(\theta|y)] g(\theta) \]  

(13)

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 \( \theta \) 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 \( \hat{q}_v(v) \) 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, \( \hat{q}_v(v) \) is the proportion of samples that are strictly less than \( v \). As \( M \) becomes large, the empirical approximation becomes more accurate.

Because \( \hat{q}_v(v) \) is discrete, we can sample from a density proportional to \( q(u) \exp(-v) \) by partitioning the domain into \( M+1 \) segments with the 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

\[
\omega_i = \hat{q}_v(v) \left[ \exp(-v_i) - \exp(-v_{i+1}) \right],
\]

so we can sample an interval bounded by \( v_i \) and \( v_{i+1} \) from a multinomial density with
weights proportional to $\omega_i$. Once we have the $i$ that corresponds to that interval, we can sample the continuous $v$ by sampling $\epsilon$ from a standard exponential density, truncated on the right at $v_{i+1} - v_i$, and setting $v = v_i + \epsilon$. Thus, we can sample $v$ by first sampling $i$ with weight $\omega_i$, then sampling a standard uniform random variable $\eta$, and finally setting

$$v = v_i - \log [1 - \eta (1 - \exp(v_i - v_{i+1})].$$ (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 $\theta$ that meets this criterion, we save it as a valid sample from $\pi(\theta|y)$. The complete algorithm is summarized as Algorithm 1.

### 2.3 The proposal distribution

The only restriction on $g(\theta)$ is that the inequality $0 \leq \Phi(\theta|y) \leq 1$ must hold, at least for any $\theta$ with a non-negligible posterior density. Because $v > 0$, we must have $u < 1$. Thus, any $\theta$ for which $\Phi(\theta|y) > 1$ would always be accepted, no matter how small $\pi(\theta|Y)$ might be. By construction, $\Phi(\theta^*|y) = 1$, meaning that no candidate $\theta$ will have a higher acceptance probability than the $\theta$ with the highest posterior density. This is an intuitively appealing property.

In principle, it is up to the researcher to choose $g(\theta)$, and some choices may be more efficient than others. We have found that a multivariate normal (MVN) proposal distribution, with mean at $\theta^*$, works well for the kinds of continuous posterior densities that marketing researchers typically encounter. The MVN density, with a covariance matrix equal to the negative inverse Hessian of the log posterior at $\theta^*$, is an asymptotic approximation (specifically, a second-order Taylor series) 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.
Algorithm 1 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 $\hat{q}_v(v)$.
3: $\theta^* \leftarrow$ mode of $D(\theta, y)$
4: $c_1 \leftarrow D(\theta^*, y)$
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:  $\log \Phi(\theta_m|y) \leftarrow \log D(\theta_m, y) - \log g(\theta_m) - \log c_1 + \log c_2$.
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:     $\hat{q}_v(v_i) \leftarrow \sum_{j=1}^M 1[v_j < v_i]$.
23:     $\omega_i \leftarrow \hat{q}_v(v_i) \exp(-v_i) - \exp(-v_{i+1})$.
24: end for
25: for $r = 1$ to $R$ do
26:     Sample $j \sim \text{Multinomial}(\omega_1 \ldots \omega_M)$.
27:     Sample $\eta \sim \text{Uniform}(0,1)$.
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:         $p \leftarrow -\log \Phi(\theta_r|y)$.
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).
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 \( \theta \), and that the mode of \( g(\theta) \) is at \( \theta^* \).

We illustrate the idea of scaling the proposal density in Figure 1. The solid line (the same in all three panels) is a “target” posterior density. The dotted lines plot potential normal proposal densities, multiplied by the corresponding \( c_2/c_1 \) ratio. The covariance of the proposal density in the left panel is the negative inverse Hessian of the log posterior density. Samples from the left tail of the posterior distribution will have \( \Phi(\theta|y) > 1 \). In the middle and right panels, the covariance is the same as in the left panel, but multiplied by 1.4 and 1.8, respectively. As the covariance increases, more of the posterior support will have \( \Phi(\theta|y) \leq 1 \).

It is possible that \( g(\theta) \) could under-sample values of \( \theta \) in the tails of the posterior. However, if \( M \) is sufficiently large, and \( \Phi(\theta|y) \leq 1 \) for all \( M \) proposals, then it is unlikely that we would see \( \Phi(\theta|y) > 1 \) in the rejection sampling phase of the algorithm. If that were to happen, we can stop, rescale, and try again. Any values of \( \theta \) that we might miss would have such low posterior density that there would be little meaningful effect on inferences we might make from the output.

We believe that the potential cost from under-sampling the tails is dwarfed by our method’s relative computational advantage, we recognize that there may be some applications for which sampling extreme values of \( \theta \) may be important. In that case, this may not be the best estimation method 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. Heavier-tailed proposals, such as the multivariate-t distribution, can fail because of high kurtosis at the mode.

### 2.4 Comparison to Other Methods

#### 2.4.1 Rejection Sampling

At first glance, our method looks quite similar to standard rejection sampling. With rejection sampling, one repeatedly samples both a threshold value from a standard uniform distribution \( p(u) = 1 \), and a proposal from \( g(\theta) \) until \( \pi(\theta|y)/g(\theta) \geq Ku \), where \( K \) is a positive constant. This is different from our method, 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 our approach is that the distribution of \( u \) is exact, and that the proposal density does not have to dominate the target density for all values of \( \theta \). However, for any model with more than a few dimensions, the critical ratio can be extremely small for even small deviations of \( \theta \) away from the mode. Thus, the acceptance probabilities are virtually nil. In contrast, 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 for conducting Bayesian inference. Like our method, Direct Sampling removes the need to concern oneself with issues like chain convergence and autocorrelation, and generates independent samples from a target posterior distribution in parallel. Walker et al. also prove that the sample acceptance probabilities using Direct Sampling 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, Direct Sampling 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. Our method allows us to conduct full Bayesian inference on hierarchical models in high dimensions, with or without conjugacy, without MCMC.

Although we share some important features with Direct Sampling, our method differs in several respects. While Direct Sampling focuses on the shape of the data likelihood alone, we are concerned with the characteristics of the entire posterior density. Our method bypasses the need for Bernstein polynomial approximations, which are integral to the Direct Sampling algorithm. Finally, while Direct Sampling takes proposal draws from the prior (which may conflict with the data), our method 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 of our method 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 effi-
ciency 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 our method than 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 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 our method can be run in their entirety in parallel.

Another attractive feature of our method is that the model is fully specified by the log posterior, and possibly its gradient and Hessian. The tools that we discuss in Section 4 are components of a reusable infrastructure. Only the function that returns the log posterior changes from model to model. This feature is unlike a blockwise Gibbs sampler, for which we need to derive and implement a set of conditional densities for each model. A small change in the model specification can require a substantial change in the sampling strategy. For example, a change to a prior might mean that the sampler is no longer conditionally conjugate. So while it might be possible to construct a highly efficient Gibbs sampler for a particular model (e.g., through blocking, data augmentation, or parameter transformation), there can be considerable upfront investment in doing so.
3 Examples

We now provide three examples of our method in action. In the first example, we simulate data from a basic, conditionally conjugate model, and compare the marginal posterior distributions that were generated by our method with those from a Gibbs sampler. The second example is a non-conjugate hierarchical model of store-level retail sales. In that example, we compare estimates from our method with those from Hamiltonian Monte Carlo (HMC), using the Stan software package (Stan Development Team 2014). For these first two examples, the MCMC methods are efficient, so it is likely that they generate good estimates of the true posterior densities. Thus, we can use those estimates as benchmarks against which we can assess the accuracy of our method. The third example is a more complicated model for which MCMC performs poorly. We use this example to not only assess the accuracy of our method (for those parameters for which we think the MCMC estimates are reasonable), but to illustrate some of the computational problems that are inherent in MCMC methods. In all of our examples, we implemented our method using the bayesGDS package (Braun 2013a) for R.

3.1 Simulated data, conditionally conjugate model

In our first example, we simulated $T = 10$ observations for each of $N = 1500$ heterogeneous units. Each observation for unit $i = 1 \ldots N$ is a sample from a normal distribution, with mean $\theta_i$ and standard deviation $\sigma = 2$. The $\theta_i$ are normally-distributed across the population, with mean $\mu = -1$ and standard deviation $\tau = 3$. We place uniform priors on $\mu$, $\log \sigma$ and $\tau$. There are 1,503 parameters in this model.

This model allows for a conditionally conjugate Gibbs sampler; the steps are described in Gelman et al. (2003, Section 11.7). In this case, the Gibbs sampler is sufficiently fast and efficient, so we have confidence that it does indeed sample from the correct posterior distributions. The Gibbs sampler was run for 2,000 iterations, including a 1,000-iteration
Figure 2: Quantiles of posterior densities from hierarchical model with conditional conjugacy, as estimated using our method, and with a Gibbs sampler.

burn-in. The process took about five minutes to complete. We then collected 360 independent samples using our method, after estimating \( q_v(v) \) with \( M = 70,000 \) proposals, and applying a scaling factor on the inverse Hessian of 1.02. To get those 360 proposals, we needed 381,507 proposals. Using a single core of a 2014-vintage Mac Pro, this process would take about 23 minutes. However, each draw can be collected in parallel. It took about five minutes to collect all 360 samples when using 12 processing cores. The mean number of proposals for each posterior sample was 1,060 (an acceptance rate of .0009), but the median was only 29. Ten of the 360 samples required more than 10,000 proposals.

Figure 2 shows the quantiles of samples in the estimated marginal posterior densities of the population-level parameters \( \mu, \tau^2, \) and \( \sigma^2 \), as well as the log of the unnormalized posterior density. We can see that both methods generate effectively the same estimated posterior distributions.

### 3.2 Hierarchical model without conditional conjugacy

In the second example, we model weekly sales of sliced cheese in 88 retail stores. The data are available in the `bayesm` package for R (Rossi. 2012), and were used in Boatwright, McCulloch, and Rossi (1999). Under this model, mean sales for store \( i \) in week \( t \) has a gamma distribution with mean \( \lambda_{it} \) and shape \( r_i \). The mean is a log-linear function of
price, and the percentage of “all category volume” on display in that week.

\[ \log \lambda_{it} = \beta_{i1} \log \text{PRICE}_{it} + \beta_{i2} \log \text{DISP}_{it} \]  \hspace{1cm} (16)

The prior on each \( r_i \) is a half-Cauchy distribution with a scale parameter of 5, and the prior on each \( \beta_i \) is MVN with mean \( \mu \) and covariance \( \Omega \). The hyperparameters \( \mu \) and \( \Omega \) have weakly informative MVN and inverse Wishart priors, respectively. There are 361 parameters in this model.

This model does not allow for a conditionally conjugate Gibbs sampler. Instead, we use Hamiltonian Monte Carlo (HMC, Duane et al. 1987; Neal 2011), which uses the gradient of the log posterior to simulate a dynamic physical system. 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 MCMC methods.

We implemented HMC using the Stan software package (Stan Development Team 2014). We ran 5 parallel chains for 800 iterations each, and discarding the first half of the draws. The chains appeared to display little autocorrelation, so we have confidence that the HMC samples form a good estimate of the true posterior distributions. We then estimated the model using our method with different numbers of proposal draws to estimate \( q_v(v) \), and different scale factors on the covariance of the proposal density. In Figure 3, we compare the estimates densities for elements of \( \mu \), the baseline and marginal effects on sales. We see that even with a relatively coarse estimate of \( q_v(v) \), and an overly diffuse proposal density, our method generates estimates of the posterior densities that are not only comparable with each other, but also comparable those generated by Stan. The acceptance rates across different runs using our method was .0003. It took about 1.4 seconds to sample and evaluate a block of 1,000 proposals on a single processing core.
3.3 Model with weakly identified parameters

Our third example concerns a non-conjugate heterogeneous hierarchical model, in which some parameters are only weakly identified. The data are described in Manchanda, Rossi, and Chintagunta (2004), and are available in the \textit{bayesm} package for \textit{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.

In this model, $y_{it}$ is a random variable that follows a negative binomial distribution with shape $r$ and mean $\mu_{it}$, and $x_{it}$ is a Poisson-distributed random variable with mean $\eta_i$. 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 $\beta_i$ is a vector of heterogeneous coefficients. We then model the contact rate so it depends on the physician-specific propensities to purchase, so $\log \eta_i = \gamma_0 + \gamma_1\beta_{i0} + \gamma_2\beta_{i1}$. Define $z_i$ as a vector of four physician-specific demographics...
(including an intercept), and define \( \Delta \) as a \( 2 \times 4 \) matrix of population-level coefficients. The mixing distribution for \( \beta \) (i.e., the prior on each \( \beta_i \)), is MVN with mean \( \Delta'z_i \) and covariance \( V \). We place weakly informative MVN priors on \( \gamma \) and the rows of \( \Delta \), an inverse Wishart prior on \( V \), and a gamma prior on \( r \). There are 2,015 distinct parameters to be estimated. This model differs slightly from the one in Manchanda et al. (2004) only in that \( \eta_i \) 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 before, our baseline estimation algorithm is HMC, but instead of using Stan, we use the “double averaging” method to adaptively scale the step size (Hoffman and Gelman 2011), and we set the expected path length to 16.\(^3\) By implementing HMC ourselves, we can use the same computer code to compute the log posterior, and its gradient, that we use with our method. This allows the two methods to compete on a level playing field.

We ran four independent HMC chains for 700,000 iterations each, during a period of 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). Finding the mode of the log posterior is the first step of our method anyway, so we initialized one chain there, and the other three at randomly-selected starting values. Figure 4 is a trace plots of the log posterior density. The chains begin to approach each other only after about 500,000 iterations. The panels in Figure 5, are 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. Table 1, summarizes the effective sample sizes for estimates of the marginal posterior distributions of

\(^3\)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), whether using Stan, or coding the algorithm ourselves. 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.
population-level parameters, using the final 100,000 samples of the HMC chains. Many of these parameters may require more than a million additional iterations to achieve an effective sample size large enough to make reasonable inferences.

<table>
<thead>
<tr>
<th>Chain</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Chain</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Delta_{1,1})</td>
<td>295</td>
<td>250</td>
<td>266</td>
<td>267</td>
<td>(\gamma_1)</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>(\Delta_{1,2})</td>
<td>16</td>
<td>24</td>
<td>27</td>
<td>29</td>
<td>(\gamma_2)</td>
<td>10</td>
<td>33</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>(\Delta_{1,3})</td>
<td>6</td>
<td>12</td>
<td>26</td>
<td>14</td>
<td>(\gamma_3)</td>
<td>3</td>
<td>15</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>(\Delta_{1,4})</td>
<td>51</td>
<td>43</td>
<td>43</td>
<td>60</td>
<td>Chol((V))_{1,1}</td>
<td>553</td>
<td>513</td>
<td>500</td>
<td>511</td>
</tr>
<tr>
<td>(\Delta_{2,1})</td>
<td>4703</td>
<td>5933</td>
<td>5981</td>
<td>1444</td>
<td>Chol((V))_{2,1}</td>
<td>7844</td>
<td>18761</td>
<td>13796</td>
<td>9852</td>
</tr>
<tr>
<td>(\Delta_{2,2})</td>
<td>507</td>
<td>573</td>
<td>526</td>
<td>538</td>
<td>Chol((V))_{2,2}</td>
<td>530</td>
<td>523</td>
<td>517</td>
<td>477</td>
</tr>
<tr>
<td>(\Delta_{2,3})</td>
<td>407</td>
<td>445</td>
<td>400</td>
<td>407</td>
<td>mean((\beta_1))</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>(\Delta_{2,4})</td>
<td>3708</td>
<td>4797</td>
<td>3680</td>
<td>4926</td>
<td>mean((\beta_2))</td>
<td>21</td>
<td>22</td>
<td>28</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1: Effective sample sizes for estimates of marginal posterior distributions of population-level parameters, using the final 100,000 samples of HMC chains.

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 its gradient. Using “reverse mode” automatic differentiation, which we discuss more in Section 4.1, the time to evaluate the gradient is about 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 our sampling method? We estimated the marginal density $q_v(v)$ by taking $M = 100,000$ proposal draws from an MVN distribution with the mean at the posterior mode $\theta^*$, and the covariance matrix equal the inverse of the Hessian at the mode, multiplied by a scaling factor of $s = 1.3$. This value of $s$ is the smallest value for which $0 \leq \Phi(\theta|y) \leq 1$ for all 100,000 samples from $g(\theta)$. We then collected 300 independent samples in parallel from the target posterior $\pi(\theta|y)$. The median number of proposals required for each posterior sample was just under 38,000, the total number of likelihood evaluations was about 16.5 million, and average acceptance rate was $1.8 \times 10^{-5}$.
In absolute terms, the low acceptance rate appears to be unfavorable. However, the total run time is much lower than for MCMC. In our implementation (using a single core on a 2014-vintage Apple Mac Pro), the total time to compute the log posterior density of 1,000 proposal samples is about 8.7 seconds. The time to sample 1,000 proposals from an MVN distribution, and compute the MVN densities for each, is about .89 seconds. Therefore, to collect and evaluate 16.5 million proposal samples (to get 300 posterior draws) would take about 44 hours. But this is for a single processing core. Using all 12 cores on our Mac Pro, the sampling time is reduced to 220 minutes. We discuss the scalability of the component steps of the algorithm in Section 4.

Access to more processing nodes could reduce this time even further. It is the ability to collect posterior samples in parallel that gives our method its greatest advantage over MCMC methods. One could run multiple MCMC chains in parallel, but this involves waiting until all of the chains, individually, converge to the target posterior, before we could begin collecting samples for inference. Even then, there is no way to confirm that the chain has, in fact, converged.

We can draw inferences about the accuracy of our method by comparing the 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 it is sufficiently close for the majority of the population-level parameters for us to use HMC samples as a baseline standard. Figure 6 is a comparison of the quantiles. For the elements of \( \Delta \) and the Cholesky decomposition of \( V \), our method’s estimated distributions are close to the HMC estimates. For other parameters, the convergence of the estimates is less clear. However, the parameters for which the densities are not aligned are the same parameters for which there is high autocorrelation, and little movement, in the HMC chains. Thus, we infer that our method compares with HMC well in terms of the marginal densities that it generates, with substantially computational effort.
Figure 6: Estimated marginal distributions for population-level parameters using our method, and HMC.

4 Scalability and Sparsity

The ability for our method to generate independent samples in parallel already makes it an attractive alternative to MCMC. In this section, we present an argument in favor of our method’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: 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

\[ \sum_{i}^{N} \log f_i(y_i|\beta_i, \alpha) + \log \pi_i(\beta_i|\alpha) \]

and a homogeneous component \( \log \pi(\alpha) \). This homogeneous component is the hyper-prior 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 17. Therefore, computation of the log posterior grows linearly in \( N \). In the subsequent text, let \( k \) be the number of elements in each \( \beta_i \) and let \( p \) be the number of elements in \( \alpha \). Using the notation from Section 2, \( \theta \) is a vector that concatenates all of the \( \beta_i \) together, along with \( \alpha \).

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.\(^4\) 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 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.

\(^4\)We do not require either derivative-based optimization or using MVN proposals, but these are most likely the best choices for differentiable, unimodal posteriors.
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.\(^5\)

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 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 eight bytes of memory at double precision. If we have a dataset in which \(N = 10000\), \(k = 5\) and \(p\) is relatively 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

\(^5\)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. We call our C++ functions from R (R Development Core Team 2014) using the Rcpp package (Eddelbuettel and François 2011; Bates and Eddelbuettel 2013). CppAD is also available for Python. Matlab users have access to ADMAT (Coleman and Verma 2000), among other options.
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).

For our purposes, the source of scalability 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.\(^6\) 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 7 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 much greater when we add more households. If we had 1,000 households, with \( k = 3 \) and \( p = 9 \), there would be 3009 parameters, and more than nine 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 7, 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 a small number of 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

\(^6\)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.
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 7, 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 \( \frac{1}{2} (k^2 + 5k) = 12 \) groups, no matter how many households we have in the dataset.

Curtis, Powell, and Reid (1974) introduce the idea of reducing the number of eval-
uations 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 prin-
ciple to computing sparse Hessians. Alternatively, R users can use the \texttt{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.\(^7\) There are two approaches that one can take. The first is to use a “limited memory” optimization algorithm that approximates the curvature of the log posterior over successive iterations. Several algorithms of this kind are described in Nocedal et al. (2006), and are available for many technical computing platforms. 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. 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.\(^7\)

\(^7\)Neither assumption is required, but most marketing models satisfy them, and maintaining them simplifies our exposition.
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($\theta^*, sH^{-1}$) distribution is straightforward. Let $\frac{1}{2} H = \Lambda\Lambda'$ 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 $\theta$ from an MVN, solve the triangular linear system $\Lambda'\theta = x$, and then add $\theta^*$. 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 $\Lambda$ 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 $\Lambda$ 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 (Davis 2006). The typical strategy is to first permute the rows and columns of $H$ to minimize the number of nonzero elements in $\Lambda$, 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 $\Lambda$. The time for this step grows with the sum of the squares of the number of nonzero elements in each column of $\Lambda$ (Davis 2006). Because each additional household adds $k$ columns to $\Lambda$, 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 $N$. Software for sparse Cholesky decompositions is widely available.

### 4.4 Scalability Test

Next, we provide some empirical evidence of scalability 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 week is $p_i$, where $\text{logit } p_i = \beta_i' x_i$, and $x_i$ is a vector of $k$ covariates. The distribution of $\beta_i$ across the population is MVN with mean $\bar{\beta}$ and covariance $\Sigma$. 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” $\Sigma$ is $0.1I$. We place weakly informative priors on both $\bar{\beta}$ and $\Sigma$.

In Figure 8, 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 9, 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 2 summarizes the acceptance rates and scale factors when generating 50 samples from the posterior for different values of $N$. Although there is a weak trend of increasing acceptance rates with $N$, we cannot say with any certainty that acceptance rates will always be either larger or smaller for larger datasets. The acceptance rate could be influenced by using different scale factors on the Hessian for the MVN proposal density. However, we expect higher acceptance rates as the target posterior density approaches an MVN asymptotically. Since none of the steps in the algorithm grows faster than linearly in $N$, we are confident in the scalability of the overall algorithm.
5 Estimating Marginal Likelihoods

Now, we turn to another advantage of our method: 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, $L(y)$ 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 consistent, stable and easy to compute. In fact, Lenk (2009) demonstrated that methods that 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 pseudo-bias arises because the convex hull of MCMC samples defines only a subset of the posterior support, while $L(y)$ 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.
Figure 9: 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)

in MCMC-based Bayesian statistics.

We estimate the marginal likelihood using quantities that we already 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, after substituting in Equation 8, we can express the expected marginal acceptance probability for any one posterior sample as

\[
\gamma = \int_0^1 q(u) p(u|y) \, du = \frac{c_1}{c_2 \mathcal{L}(y)} \int_0^1 q^2(u) \, du
\]  

(18)

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

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

(19)

The values for \( c_1 \) and \( c_2 \) are immediately available from the algorithm. A reasonable estimator of \( \gamma \) is \( \hat{\gamma} \), the inverse of the mean of the observed 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 \( \hat{q}_v(v) \). The empirical CDF of these draws is discrete, so we can partition the support of \( q_v(v) \) at \( v_1 \ldots v_M \).
Also, since \( \hat{q}_v(v) \) is the proportion of proposal draws less than \( v \), we have \( q_v(v_i) = i/M \). Therefore,

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

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

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

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

\[
L(y) \approx \frac{c_1}{M^2 c_2 \gamma} \sum_{i=1}^{M} (2i - 1) \exp(-v_i)
\]

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

\[
y_{it} \sim N(x_i' \beta, \sigma^2), \ i = 1 \ldots n, t = 1 \ldots T
\]

\[
\beta | \sigma \sim N(\beta_0, \sigma^2 V_0) \quad \sigma^2 \sim IG(r, \alpha)
\]

For this model, \( L(y) \) is a multivariate-t density (MVT), which we can compute analytically. This allows us to compare the estimates of \( L(y) \) 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 \( \beta \), plus \( \sigma \). The true intercept term is 5, and the remaining true \( \beta \) parameters are linearly spaced from \(-5\) to \(5\). In all cases, there are \( T = 25 \) observations per unit. Hyperpriors are set as \( r = 2, \alpha = 1, \beta_0 \) as a zero vector and \( V_0 = 0.2 \cdot I_k \).

For each dataset, we collected 250 samples from the posterior density, with differ-
ent numbers of proposal draws \((M = 1,000 \text{ or } 10,000)\), and different scale factors \((s = 0.5, 0.6, 0.7, \text{ or } 0.8)\) 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 3 presents the true log marginal likelihood (MVT), along with estimates using our method, the importance sampling method in Lenk (2009), and the harmonic mean estimator Newton et al. (1994). We also included the mean acceptance probabilities, and the standard deviations of the various estimates across the simulated datasets. We can see that our 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, while improving the approximation of \(p(u|y)\) by increasing the number of proposal draws offers negligible improvement. Note that the performance of our method is comparable to that of Lenk, but is much better than the harmonic mean estimator. Our 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, the inputs to the estimator of the marginal likelihood are intrinsically generated as the algorithm progresses. In contrast, the Lenk estimator requires an additional importance sampling run after the MCMC draws are collected.

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 alternative algorithm is appealing. Ours allows for sampling 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
<table>
<thead>
<tr>
<th>k</th>
<th>n</th>
<th>M</th>
<th>scale</th>
<th>MVT mean</th>
<th>sd</th>
<th>Ours mean</th>
<th>sd</th>
<th>Lenk mean</th>
<th>sd</th>
<th>HME mean</th>
<th>sd</th>
<th>Acc %</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>200</td>
<td>1000</td>
<td>0.5</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.6</td>
<td>-311.6</td>
<td>6.8</td>
<td>-287.7</td>
<td>7.1</td>
<td>22.1</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>1000</td>
<td>0.6</td>
<td>-309.6</td>
<td>6.7</td>
<td>-309.6</td>
<td>6.9</td>
<td>-310.6</td>
<td>6.5</td>
<td>-287.6</td>
<td>6.3</td>
<td>40.5</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>1000</td>
<td>0.7</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.7</td>
<td>-311.6</td>
<td>6.7</td>
<td>-287.6</td>
<td>6.7</td>
<td>57.1</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>1000</td>
<td>0.5</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>7.1</td>
<td>-311.6</td>
<td>7.1</td>
<td>-287.7</td>
<td>7.1</td>
<td>24.0</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>1000</td>
<td>0.6</td>
<td>-309.6</td>
<td>6.7</td>
<td>-309.6</td>
<td>6.8</td>
<td>-310.6</td>
<td>6.5</td>
<td>-287.6</td>
<td>6.3</td>
<td>40.9</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>1000</td>
<td>0.7</td>
<td>-309.6</td>
<td>6.5</td>
<td>-309.6</td>
<td>6.7</td>
<td>-311.6</td>
<td>7.0</td>
<td>-287.6</td>
<td>6.7</td>
<td>55.2</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>10000</td>
<td>0.5</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.7</td>
<td>-311.6</td>
<td>6.7</td>
<td>-287.6</td>
<td>6.7</td>
<td>24.0</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>10000</td>
<td>0.6</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.5</td>
<td>-310.6</td>
<td>6.3</td>
<td>-287.6</td>
<td>6.3</td>
<td>40.9</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>10000</td>
<td>0.7</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.8</td>
<td>-311.6</td>
<td>6.7</td>
<td>-287.6</td>
<td>6.7</td>
<td>55.2</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>10000</td>
<td>0.8</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.7</td>
<td>-311.6</td>
<td>6.7</td>
<td>-287.6</td>
<td>6.7</td>
<td>24.0</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>10000</td>
<td>0.9</td>
<td>-309.6</td>
<td>6.6</td>
<td>-309.6</td>
<td>6.5</td>
<td>-311.6</td>
<td>6.7</td>
<td>-287.6</td>
<td>6.7</td>
<td>40.9</td>
</tr>
</tbody>
</table>

Table 3: Results of simulation study for effectiveness of estimator for log marginal likelihood.
the Hessian of the log posterior lets us construct a sampling algorithm whose complexity grows only linearly in the number of units. This method makes Bayesian inference more attractive to practitioners who might otherwise be put off by the inefficiencies of MCMC.

This is not to say that our method is guaranteed to generate perfect samples from the target posterior distribution. One area of potential concern is that the empirical distribution $\hat{q}_v(v)$ 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 $\hat{q}_v(v)$), at the expense of costlier computation of $\hat{q}_v(v)$ and, possibly, lower acceptance rates. In our experience, and consistent with Figure 3, 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, its efficiency depends in part on a prudent selection of the proposal density $g(\theta)$. For the examples in this paper, we use a MVN 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 manual search is the best alternative. 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, with 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 $p(u)$. This is no different, in principle, than the tuning step in a Metropolis-Hasting algorithm. However, our method has the advantage that we can make these adjustments before the posterior sampling phase begins. In contrast, with adaptive MCMC methods an improperly tuned sampler might not be apparent until the chain has run for a substantial period of time. Also, even if an acceptance rate appears to be low, 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.

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 our method 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 opportunity for further research involves the case of multimodal posteriors. Our method 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 might 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 algorithm itself would remain 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. Even if the log posterior density is unimodal, one should take care that the mode-finding optimizer does not stop until reaching the optimum. For R, trustOptim Braun (2013d) is one such package, in that its stopping rule depends on the norm of the gradient being sufficiently close to zero.
For researchers who use R for statistical analysis, there are a number of packages that can help with implementation of our method. The bayesGDS package (Braun 2013a) includes functions to run the rejection sampling phase (lines 20-36 in Algorithm 1). This package also includes a function that estimates the log marginal likelihood from the output of the algorithm. 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


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.


