Asynchronous Gibbs Sampling

Alexander Terenin aterenin@ucsc.edu
eBay, Inc. and Applied Mathematics and Statistics, University of California, Santa Cruz

Daniel Simpson d.simpson@bath.ac.uk
Mathematical Sciences, University of Bath (UK)

David Draper draper@ucsc.edu
Applied Mathematics and Statistics, University of California, Santa Cruz

April 26, 2016

Abstract

Gibbs sampling is a widely used Markov Chain Monte Carlo (MCMC) method for numerically approximating integrals of interest in Bayesian statistics and other mathematical sciences. It is widely believed that MCMC methods do not extend easily to parallel implementations, as their inherently sequential nature incurs a large synchronization cost. This means that new solutions are needed to bring Bayesian analysis fully into the era of large-scale computation. In this paper, we present a novel scheme – *Asynchronous Gibbs sampling* – that allows us to perform MCMC in a parallel fashion with no synchronization or locking, avoiding the typical performance bottlenecks of parallel algorithms. Our method is especially attractive in settings, such as hierarchical random-effects modeling in which each observation has its own random effect, where the problem dimension grows with the sample size. We present two variants: an exact algorithm, and an approximate algorithm with better scaling properties. We prove convergence of the exact algorithm under some basic regularity conditions, and discuss the proof for similar parallelization schemes for other iterative algorithms. We provide three examples that illustrate some of the algorithm's properties with respect to scaling, and one example that compares the exact and approximate algorithms. Because our hardware resources are bounded, we have not yet found a limit to the algorithm's scaling, and thus its true capabilities remain unknown.

Keywords: Bayesian mixed-effects models, Big Data, fixed-point algorithm, Gaussian Process regression, high-dimensional statistical modeling, Markov Chain Monte Carlo, parallel computing, synchronization.

1 Introduction

The Bayesian statistical paradigm has well-known optimality properties: given a full model specification (prior and sampling distributions for inference and prediction, plus action space and utility function for decision), a theorem due to Cox [9] – recently rigorized and extended by [51] – shows that the axiomatically-correct (i.e., Bayesian) use of conditional probability is the only approach to inference, prediction and decision that does not have at least the possibility of built-in information loss. Additionally, the paradigm offers extensive modeling flexibility, and an immediate way of determining what a statistical model assumes and what it is capable of learning from the data.

The e-commerce company eBay, Inc. is interested in using the Bayesian paradigm for purposes of inference and decision-making, and employs a number of Bayesian models as part of its operations. One of the main practical challenges in using the Bayesian paradigm on a modern industrial scale is that the standard approach to Bayesian computation for the past 25 years – Markov Chain Monte Carlo (MCMC) [22, 33] – does not scale well, either with data set size or with model complexity. This is especially of concern in e-commerce applications, where typical data set sizes range from $n = 1,000,000$ to $n = 10,000,000,000$.

In this paper we offer a new algorithm – *Asynchronous Gibbs sampling* – which removes synchronicity barriers that hamper the efficient implementation of most MCMC methods in a distributed environment. This is a crucial element in improving the behavior of MCMC samplers for complex models that fall within the current "Big Data/Big Models" paradigm: Asynchronous Gibbs is well-suited for models where the dimensionality of the parameter space increases with sample size. We show that Asynchronous Gibbs performs well on both illustrative test cases and a real large-scale Bayesian application.

The layout for the rest of the paper is as follows. In Section 2 we review the problem of sampling-based Bayesian inference, and Section 3 examines some alternatives to Asynchronous Gibbs for solving large-scale Bayesian problems. In Section 4 we introduce two variations of the Asynchronous Gibbs algorithm – exact and approximate – and discuss implementation. Section 5.1 proves, using a widely applicable meta-theorem, that exact Asynchronous Gibbs converges, and this convergence is demonstrated on increasingly complex examples in Sections 6.1–6.3. In Section 6.4 we examine a setting in which approximate Asynchronous Gibbs can fail, and we introduce a diagnostic to help avoid such situations. The overall picture of Asynchronous Gibbs is evaluated in Section 7, where a number of future directions are also discussed.

2 The Problem

A Bayesian statistical model M has two components that arise out of the real-world problem in which it is employed. Let $\boldsymbol{x}$ denote the data and $\boldsymbol{\theta}$ denote the parameters of the model. To fully specify M, we must make assumptions regarding two components, the *likelihood* and the *prior*:

- *Likelihood*: $f(\boldsymbol{x} \mid \boldsymbol{\theta})$ describes what the distribution of the data would be *if we knew the parameters*.
- *Prior*: $\pi(\boldsymbol{\theta})$ describes our information (or lack thereof) about the parameters *external to the data*.

Note that $\boldsymbol{\theta}$ and $\boldsymbol{x}$ are random variables – their precise form varies from problem to problem. In full generality they can be defined on $\mathbb{R}^k$ or just about any other well-behaved domain, even $\mathscr{F}_{\text{discrete}}$, the space of all discrete cumulative distribution functions (CDFs) on $\mathbb{R}$.

Once the two assumptions above are made, M is fully specified – the likelihood and prior lead to a unique posterior distribution:

- *Posterior*: $f(\boldsymbol{\theta} \mid \boldsymbol{x})$ describes *what we have learned* about the parameters from the data.

The posterior distribution can be found via Bayes' rule, which can be written in the following forms:

$$f(\boldsymbol{\theta} \mid \boldsymbol{x}) \propto f(\boldsymbol{x} \mid \boldsymbol{\theta})\pi(\boldsymbol{\theta}) \qquad\qquad f(\boldsymbol{\theta} \mid \boldsymbol{x}) = \frac{f(\boldsymbol{x} \mid \boldsymbol{\theta})\,\pi(\boldsymbol{\theta})}{\int_{\Omega_{\boldsymbol{\theta}}} f(\boldsymbol{x} \mid \boldsymbol{\theta})\,\pi(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}}\,, \tag{1}$$

in which $\Omega_{\boldsymbol{\theta}}$ is the support of $\boldsymbol{\theta}$. Note that the posterior distribution is a function of $\boldsymbol{\theta}$, whereas the integral in the right-hand expression in (1) is a function of $\boldsymbol{x}$, and is hence a normalizing constant, ensuring that

$f(\boldsymbol{\theta} \mid \boldsymbol{x})$ integrates to 1. In practice this integral is almost always intractable. Instead, if we are able to draw samples from $f(\boldsymbol{\theta} \mid \boldsymbol{x})$ without knowing the normalizing constant, we can approximate its corresponding CDF $F(\boldsymbol{\theta} \mid \boldsymbol{x})$ to arbitrary precision by computing the empirical CDF $\hat{F}$ based on those samples:

$$\hat{F}(\boldsymbol{\theta} \mid \boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} \mathrm{I}_{\{\boldsymbol{Z}_i^{(n)} \leq \theta_i \text{ for all } i\}} \text{ with } \boldsymbol{Z}^{(n)} \sim \boldsymbol{\theta} \mid \boldsymbol{x}\,. \tag{2}$$

If the $\boldsymbol{Z}^{(n)}$ are IID, this is a (standard) *Monte Carlo* estimator [32]. If $\boldsymbol{Z}^{(n)}$ are not IID, but instead form a Markov Chain with $f(\boldsymbol{\theta} \mid \boldsymbol{x})$ as its stationary distribution, this is a *Markov Chain Monte Carlo* (MCMC) estimator [22, 33]. Under mild regularity conditions, both estimators converge to the right answer.

One way to create such a Markov Chain is to specify the *full conditional* distributions and construct a (sequential-scan) *Gibbs sampler* [16, 17], which consists of repeated sequential sampling of the following full conditional distributions:

$$\theta_1^* \sim f(\theta_1 \mid \theta_2, .., \theta_k, \boldsymbol{x}) \qquad \theta_2^* \sim f(\theta_2 \mid \theta_1^*, \theta_3, .., \theta_k, \boldsymbol{x}) \qquad .. \qquad \theta_k^* \sim f(\theta_k \mid \theta_1^*, .., \theta_{k-1}^*, \boldsymbol{x})\,. \tag{3}$$

It has been shown [17] that under weak regularity conditions the Markov chain $\boldsymbol{\theta}^{(n)}$ has the posterior $f(\boldsymbol{\theta} \mid \boldsymbol{x})$ as its stationary distribution. It is also possible – and sometimes preferable [41] – to draw the samples from the full conditional distributions in random order, leading to a *random-scan Gibbs sampler* [30]. In either case, the key property is that of *synchronicity*: each sample is completed before the next begins.

Unfortunately, if the dimensionality of $\boldsymbol{\theta}$ is large, the Gibbs sampler scales poorly, for a variety of reasons:

(a) *Sheer size*: in large models there are too many full conditionals from which we must draw, in order to obtain enough samples from the full posterior distribution to be useful in any realistic time.

(b) *Inherently sequential nature*: to draw the next full conditional, the algorithm must know the values of *all of the most recent* full conditionals, and hence can only proceed one step at a time.

(c) *Centralization*: all of the data typically needs to be present on the computer running the algorithm – trying to do standard Gibbs sampling with data sets that are too big to fit on one machine, and instead live on *Hadoop* clusters or other forms of distributed storage, is impossible.

(d) *Mixing*: the chain may be exploring the posterior distribution too slowly to produce useful results – this *mixing* rate is completely dependent on the individual problem and the precise form of the full conditional distributions.

In this article, we present an extension of Gibbs sampling that attacks problems (a–c) by running an equivalent computation fully in parallel on a cluster, while striving to keep problem (d) sufficiently under control to remain useful.

3 Previous work on Bayesian computation at scale

There are a number of approaches that aim to provide exact or highly accurate approximate Bayesian computation with large data sets. Alternatives to our approach include the following, in no particular order:

- *Variational Bayesian methods.* This approach, studied extensively by Jordan et al. [28], achieves approximate Bayesian computation at scale by doing the following:

(a) Define a space $\mathscr{M}$ of models in which computation with extremely large data sets is fast and exact.

(b) Find the closest model M^* in $\mathscr{M}$ to the actual model M, in a sense related to Kullback-Leibler discrepancy.

(c) Perform fast and exact posterior calculations with M^*.

(d) Use the results from (c) to approximate the (unknown) results from model M.

The quality of this approach naturally rests on the distance from M^* to M, on the scale of the integrals whose approximation is desired (a quantity that is not typically made precise by this method). All that is guaranteed is that M^* is the closest possible model to M in $\mathscr{M}$, not that this distance is small enough to ensure good approximations. Examples can be found in which this method is highly accurate, and other settings can be identified in which its performance is less satisfying.

- *Approximate Bayesian computation (ABC).* The idea here (e.g., Tavaré et al. [50]) is for the user of this method to identify a set of statistics that are regarded by the user as close to *sufficient* (in the usual statistical sense of sufficiency) – the method then in effect projects the problem under study into the space spanned by these pseudo-sufficient statistics, a space in which (by the nature of sufficiency) Bayesian calculations are extremely fast, and regards the results of these calculations as approximations to the exact calculations in the original problem. Thus the quality of this method rises and falls with the ingenuity of the user in identifying nearly-sufficient statistics – as with variational methods, examples of results with varying accuracy are relatively easy to find.

- *Consensus Monte Carlo.* In this approach, due to Scott et al. [45], the (large) data set is broken into small pieces (*shards*), with MCMC performed on each shard on a different processor in parallel and with the results averaged together to produce what is hoped is a good approximation to the (possibly intractable) MCMC calculations on the entire data set. The quality of this method depends on the problem-specific data-partitioning scheme and the method used to perform the averaging – examples with different degrees of accuracy can be constructed.

- *Subspace projection.* Approximate Bayesian computation with large data sets may be accomplished on a model-by-model basis – an example is given by Banerjee et al. [4] for Gaussian Process regression. Using an idea from compressive sensing, these authors linearly project the entire data set onto a lower-dimensional subspace in which calculations are much faster. This approach clearly depends on how much information is lost in the projection process – the authors use stochastic matrix approximation methods to arrive at "nearly optimal" projections. Examples can be constructed with both minimal and significant information loss.

- *Latent Dirichlet allocation.* Ihler and Newman [24] proposed a special case of our algorithm, applied to the topic of latent Dirichlet allocation (LDA). They derived a bound on the difference between the Monte Carlo output proposed by their sampling scheme and the output of an LDA-specific standard Gibbs sampler. They also tested the algorithm with a variety of hardware configurations and provided some numerical results on its scaling properties in terms of Monte Carlo iterations. However, they did not prove convergence, perform MCMC diagnostics, or analyze properties of the chain with respect to mixing, and their approach is entirely limited in scope to LDA.

- *Gaussian Approximate Asynchronous Gibbs.* Johnson et al. [27] have analyzed our approximate algorithm restricted to Gaussian targets. They have proved that if the Gaussian target's precision

matrix is diagonally dominant, approximate Asynchronous Gibbs converges to the correct mean. They have also analyzed some connections with parallel algorithms for solving linear systems. However, their approach is limited to the Gaussian case, and does not include any analysis of our exact algorithm.

We take no position in this paper on the positive and negative aspects of any of these methods when contrasted with our new algorithm or each other – comparisons of this type are left to future work.

4 Asynchronous Gibbs Sampling

Asynchronous Gibbs sampling is an algorithm for performing MCMC inference in parallel, without synchronization or locking. For notational simplicity, we omit all conditioning on the data, and henceforth assume it to be implicit. To understand the algorithm, we introduce the following concepts and notation:

- w_i: a worker that performs computations, such as a CPU core, or computer on a network.
- $\boldsymbol{\theta}$: a random variable representing the full distribution from which we wish to sample.
- $\boldsymbol{\theta_i}$: a subset of $\boldsymbol{\theta}$ that is assigned to worker w_i.
- $\theta_{ci} \mid \theta_{-ci}$: a full conditional random variable or block of random variables contained in $\boldsymbol{\theta_i}$. Full conditional in this context means the most recent values of all other variables in $\boldsymbol{\theta}$ that are *known to worker* w_i (which may not be the most recent values *known to the cluster as a whole*, due to network delays, packet loss, outages, and other potential scenarios).

Asynchronous Gibbs sampling then proceeds as follows.

(a) Provide all workers with initial values of the chain.

(b) For each worker, repeat the following in parallel without synchronization:

  (i) Select a variable or block θ_{ci} from $\boldsymbol{\theta_i}$ at random with constant probability $0 < p_{\theta_{ci}} < 1$, exactly as in random-scan Gibbs.

  (ii) Sample $\theta_{ci} \mid \theta_{-ci}$.

  (iii) Send the update θ_{ci} to all the other workers.

  (iv) Process all updates received from other workers and proceed to the next iteration.

(c) Stop when the chain has converged and sufficient monitoring iterations (subsequent to convergence) have been performed for the desired Monte Carlo accuracy, and then download the output from all the workers.

This is similar to standard random-scan Gibbs sampling, but with one primary difference: instead of sampling all the variables conditional on the *most recent values*, each worker is sampling within a subset of the variables conditional on the *most recent values that it knows about*. Note that in full generality, there are few restrictions on the way in which variables are sampled: Metropolis [33] and Metropolis-adjusted Langevin [42] steps are allowed, as are Slice Sampling [36] steps, and virtually all other techniques, as long as they can be used together to form a valid MCMC algorithm (see Section 5.1). Also note that not all variables need to be transmitted – it is only necessary that each worker is *able* to either sample or receive each variable. In practice, we have found it effective to partition and transmit latent variables that

correspond to data points, and sample – locally on every worker – variables located at the top level of a hierarchical model. We focus here on the case where all transmitted variables are sampled via Gibbs steps.

There are two ways in which updates received from other workers may be processed:

- Exact: accept updates with probability $\min\left\{1, \frac{f(\boldsymbol{\theta}_{\text{new}})f(\boldsymbol{\theta}_{\text{old}}|\boldsymbol{\theta}_{\text{sender}})}{f(\boldsymbol{\theta}_{\text{old}})f(\boldsymbol{\theta}_{\text{new}}|\boldsymbol{\theta}_{\text{sender}})}\right\}$ – see Section 5.1.
- Approximate: accept all updates.

The remarkable property we have observed is that, in many situations, the Metropolis-Hastings (MH) acceptance probability is close to 1 sufficiently often that the approximate algorithm yields the same numerical answer as the exact algorithm up to Monte Carlo noise. This allows us to ignore the MH correction while still obtaining good numerical results: intuitively, we're replacing the MH acceptance ratio with a biased estimator of it, namely 1. This type of approximation can sometimes be justified from a *Noisy Monte Carlo* perspective – see Alquier et al. [1].

In Section 5.1 we prove convergence for the exact algorithm. In Sections 6.1 - 6.3 we illustrate the approximate algorithm, which has much better scaling properties. In Section 6.4, we examine a case where the approximate algorithm can fail, and discuss when using it is appropriate.

If exactness is desired, the MH ratio can be computed quickly and inexpensively in a wide variety of problems – in Section 5.2, we show how it can be calculated in exchangeable latent variable models using only one data point. Note that the exact algorithm will not scale as well as the approximate algorithm, because even if calculating the MH ratio is fast, the algorithm's parallel nature will create a large number of such evaluations. Thus, we recommend running the approximate algorithm and collecting a random sample of the MH acceptance probabilities while the algorithm unfolds. This sample can be examined to see whether its distribution is sufficiently concentrated around 1. This gives a diagnostic check for whether the approximate algorithm is appropriate.

Our algorithm possesses a variety of positive characteristics with respect to parallelism.

- By relaxing the requirement of conditioning on the most recent value of each variable, we are able to completely eliminate all requirements for synchronization or locking, which makes our algorithm embarrassingly parallel with respect to the dimensionality of $\boldsymbol{\theta}$. This is of interest in a large class of Bayesian hierarchical models in which the number of full conditional distributions arising from $\boldsymbol{\theta}$ is greater than the number of data values, because each data point has its own latent variable that needs to be reconstructed.
- The algorithm is also fault-tolerant. Not all updates that are sent will be received by all other workers due to network traffic congestion and other types of failures. This does not adversely affect the algorithm – each worker simply continues its work. Even if a variable is dropped, at some point in the future it will be sampled again and sent again. Analogously, as long as workers write their output to disk periodically, our algorithm is tolerant of crashes: workers can be restarted with no loss, and the algorithm proceeds without interruption.
- In many practical situations, we find that the algorithm scales superlinearly with respect to workers. This is because its parallel nature doesn't just produce more samples, it also accelerates mixing when compared to parallel independent chains. For an intuitive explanation, observe that if we have m workers, p full conditional distributions, and workers broadcast their updates at every iteration, then it will take each worker approximately $\frac{p}{m}$ iterations to either sample or receive each full conditional

on average once. This allows each worker to explore the posterior at a much quicker rate, even though on a one-sample single-worker basis, the algorithm may perform worse than standard Gibbs. This can be seen in the sample autocorrelation functions (ACFs) for Asynchronous Gibbs, which can exhibit long-memory tails. Note that this problem is significantly mitigated by what is otherwise an undesirable feature in MCMC: strong positive sample autocorrelation. Conditioning on the most recent known value is quite similar to conditioning on the absolutely most recent sample, because they are positively correlated. The net result of all of these considerations is that, because parallelism can both accelerate and slow down mixing at the same time in different ways, determining how the algorithm scales is difficult. Nonetheless, in many situations, we have observed that if we double the number of workers, Asynchronous Gibbs will be more than twice as fast.

- Asynchronous Gibbs can be performed in a variety of parallel environments. The simplest is the multicore setting, in which each worker corresponds to a CPU core, and shared memory is located on the machine. Our examples focus instead primarily on the cluster setting, in which each worker corresponds to a computer, and shared memory is approximated via network communication. The cluster setting is particularly interesting for problems that are too big to store on one machine, in terms of size of the data set and the parameter space.

Significant care must be taken to properly tune the algorithm and monitor convergence. The standard "effective sample size" calculation used for MCMC diagnostics does not immediately extend to multiple workers, because their chains are not independent. We have found it helpful in practice to examine the sample ACF at large time lags, on the order of thousands or tens of thousands of observations: the point is to check whether the ACF goes to 0 at a number of lags that is lower than the total number of Monte Carlo iterations performed on its corresponding worker.

Our first two examples in Sections 6.1 and 6.2 contain simulated data with a known correct answer. In our second example we are able to perform Gaussian Process regression on $n = 71,500$ observations, with 143 workers on single-core machines communicating over a network and 10,000 Monte Carlo samples per variable, in around 20 minutes (all references to time in this paper are based on clock time, not CPU time). If we only use 70 workers, the algorithm takes approximately twice as long to run (so in this example we do not obtain superlinearity). Our third example in Section 6.3 involves a real-world data set, and we compare our results to the output produced by a sequential-scan Gibbs sampler with identical numerical computation routines that takes about 12 hours to run (whereas Asynchronous Gibbs produces comparable results in about 1 hour). We also compare our results to a highly optimized *C++* [25] implementation to verify accuracy. In all of the problems we have examined to date, we have not yet hit any dimensionality limit. Thus the true high-end capabilities of Asynchronous Gibbs remain unknown.

Our algorithm is implemented in *Scala* [38], a compiled language similar to and interoperable with *Java* [19] that is well equipped for parallel processing use cases. Network communication and cluster management is handled by *Akka* [52], a decentralized actor-based message-passing framework written in *Scala* for large-scale distributed applications. Numerical computation is done via *Breeze* [21], a *Scala* library written for fast and accurate computation, designed for natural language processing and scientific computing. Random number generation is handled via Matsumoto and Nishimura's *Mersenne Twister* [31] random number generation algorithm. With slight modifications, our implementation could run on large-scale distributed data sets in the *Hadoop* [46] environment. However, since Asynchronous Gibbs sampling is not a deterministic algorithm, it does not translate directly into the *MapReduce* [10] paradigm.

5 Convergence and Properties of the Algorithm

5.1 Convergence

In this section we prove that, provided we start with a well-defined Markov Chain, Asynchronous Gibbs sampling will converge to the correct target distribution. Our strategy is two-fold. Firstly, we define a synchronous parallel MCMC algorithm that formalizes the way in which workers draw samples and communicate with one another, under the assumption that communication is instantaneous and synchronous. This part of the proof was inspired by ideas from the coupling of chains in *parallel tempering* [48]. Then we note that MCMC methods belong to the class of *fixed-point* algorithms, and hence we can use a result from the asynchronous convergence of these algorithms, due to Bertsekas [6] and Baudet [5], to prove that the asynchronous version of our parallel algorithm with non-instantaneous communication converges as well. We use total variation $||\cdot||_{\mathrm{TV}}$ as our distance metric.

Our notation is as follows:

- w_i: an arbitrary worker.
- k: the current iteration of the chain.
- p: the total number of full conditional distributions.
- m: the total number of workers.
- $(\Omega, \mathscr{F}, \mathbb{P})$: the probability triple upon which the target distribution is defined.
- $\mathscr{M}_i$: all probability measures on Ω. Note that (trivially) $\mathscr{M}_i = \mathscr{M}_{i'}$ for all i, i'.
- $\mathscr{M} \triangleq \bigtimes_{i=1}^{m} \mathscr{M}_i$: all product probability measures on Ω^m.
- π: the measure corresponding to the target density, i.e., the posterior distribution $f(\boldsymbol{\theta} \mid \boldsymbol{x})$.
- $\Pi \triangleq \bigtimes_{i=1}^{m} \pi$: the product measure of m independent copies of π.
- θ_{ci}: the full conditional random variable corresponding to coordinate c on worker i.

We begin by defining the MCMC algorithm that we wish to parallelize.

Definition 1 (Underlying Chain).
Let the underlying chain be a well-defined first-order Markov operator P satisfying the following:

- Stationarity: $P(\pi) = \pi$.
- Convergence: $||P^k(\cdot) - \pi||_{\mathrm{TV}} \to 0$ as $k \to \infty$.
- Gibbs Kernel: P is constructed via full conditional distributions $f(\theta_c \mid \theta_{-c})$.

Conditions for stationarity and convergence typically follow from ϕ-irreducibility, aperiodicity, and related assumptions required by standard MCMC schemes.

We now define a Markov chain on an extended space that captures the behavior of our algorithm *if communication is instantaneous*, i.e., if there are no asynchronous delays.

Definition 2 (Synchronous Parallel Chain).
Let the synchronous parallel chain be a first-order Markov operator $H : \mathscr{M} \to \mathscr{M}$ with a Metropolis-Hastings transition kernel defined as follows:

(i) Randomly select a worker w_s.

(ii) Randomly select a full conditional coordinate c from the set of coordinates that w_s works on.

(iii) Propose θ'_{cs} from $f(\theta_{cs} \mid \theta_{-cs})$.

(iv) For each worker w_i, accept the proposal with probability

$$\alpha_i = \min\left\{1, \frac{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta_{-cs})}{f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})}\right\}, \tag{4}$$

staying at the previous value otherwise.

Thus the synchronous parallel chain selects a worker at random and proposes from that worker's full conditional *at every worker*. Note that $\alpha_s = 1$, because on worker w_s – whose full conditional was selected – the proposal is exactly a Gibbs step and is hence always accepted. It's clear that if the combined proposal for all workers is evaluated jointly with acceptance probability $\alpha = \prod_{i=1}^{m} \alpha_i$, then this is just a standard Metropolis-Hastings algorithm and thus converges. We now show that if each worker performs the MH test independently given the proposal, the resulting chain also converges.

Theorem 3 (Synchronous Parallel Convergence).
H admits Π as its stationary distribution and unique limiting distribution.

Proof. It's sufficient to show that H satisfies the detailed balance equation

$$f(\theta)\, q(\theta' \mid \theta) = f(\theta')\, q(\theta \mid \theta')\,, \tag{5}$$

in which f is the target density, q is our transition density, θ is the old state, and θ' is the new state. Let $I \subseteq \{1, .., m\}$ be the set of workers that accept the newly proposed θ'_{cs}. We write $I = \{i_1, i_2, .., i_{|I|}\}$. Our chain begins with initial density

$$f(\theta) = \prod_{i=1}^{m} f(\boldsymbol{\theta}_i)\,, \tag{6}$$

and for $i \in I$ it replaces each θ_{ci} with θ'_{cs}. This yields the updated density

$$f(\theta') = \prod_{i \in I} f(\theta'_{cs}, \theta_{-ci}) \prod_{i \notin I} f(\boldsymbol{\theta}_i)\,, \tag{7}$$

corresponding to a move of the form:

$$\begin{array}{rccc} w_{i_1}: & \theta_{ci_1} & \to & \theta'_{cs} \\ w_{i_2}: & \theta_{ci_2} & \to & \theta'_{cs} \\ \vdots & \vdots & & \vdots \\ w_{i_{|I|}}: & \theta_{ci_{|I|}} & \to & \theta'_{cs} \end{array}\,. \tag{8}$$

Our proposal density on each worker is the full conditional distribution for coordinate i on worker w_s. With the MH acceptance probability α_i, the transition density for each worker is

$$q_i(\theta' \mid \theta) = f(\theta'_{cs} \mid \theta_{-cs})\, \alpha_i\,. \tag{9}$$

For all workers taken together, this yields

$$q(\theta' \mid \theta) = \prod_{i \in I} f(\theta'_{cs} \mid \theta_{-cs})\, \alpha_i \,, \tag{10}$$

and the reverse transition density is

$$q(\theta \mid \theta') = \prod_{i \in I} f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \,. \tag{11}$$

Note that the asymmetry in (10) and (11) arises because I singles out workers that accept the proposed move. The reverse move takes the form

$$\begin{array}{rccc} w_{i_1}: & \theta'_{cs} & \to & \theta_{ci_1} \\ w_{i_2}: & \theta'_{cs} & \to & \theta_{ci_2} \\ \vdots & \vdots & & \vdots \\ w_{i_{|I|}}: & \theta'_{cs} & \to & \theta_{ci_{|I|}} \end{array} \,, \tag{12}$$

with reverse acceptance probability:

$$\alpha'_i = \min \left\{ 1, \frac{f(\theta_{ci}, \theta_{-ci}) f(\theta'_{cs} \mid \theta_{-cs})}{f(\theta'_{cs}, \theta_{-ci}) f(\theta_{ci} \mid \theta'_{-cs})} \right\} . \tag{13}$$

Note that by construction $\alpha_i = 1 \iff \alpha'_i < 1$ and $\alpha'_i = 1 \iff \alpha_i < 1$. Let $A = \{i \in I : \alpha_i < 1\}$ and $A' = I \setminus A = \{i \in I : \alpha'_i < 1\}$. Note further that

$$\alpha'_i = 1 \text{ for } i \in A, \text{ and } \alpha'_i = \frac{f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})}{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})} \text{ for } i \in A' \,, \tag{14}$$

and similarly

$$\alpha_i = 1 \text{ for } i \in A' \text{ and } \alpha_i = \frac{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})}{f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})} \text{ for } i \in A \,. \tag{15}$$

With these definitions, we can demonstrate detailed balance as follows:

$$\begin{aligned}
f(\theta)\, q(\theta' \mid \theta) &= \prod_{i=1}^{m} f(\boldsymbol{\theta}_i) \prod_{i \in I} f(\theta'_{cs} \mid \theta_{-cs})\, \alpha_i \\
&= \prod_{i \in I} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})\, \alpha_i \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in A} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})\, \alpha_i \prod_{i \in A'} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs}) \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in A} f(\theta_{ci}, \theta_{-ci}) f(\theta'_{cs} \mid \theta_{-cs}) \frac{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})}{f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})} \prod_{i \in A'} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs}) \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in A} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs}) \prod_{i \in A'} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs}) \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in A} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \prod_{i \in A'} f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs}) \frac{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})}{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})} \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in A} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \prod_{i \in A'} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs}) \frac{f(\theta_{ci}, \theta_{-ci})\, f(\theta'_{cs} \mid \theta_{-cs})}{f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})} \prod_{i \notin I} f(\boldsymbol{\theta}_i)
\end{aligned}$$

$$
\begin{aligned}
&= \prod_{i \in A} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \prod_{i \in A'} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in I} f(\theta'_{cs}, \theta_{-ci})\, f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \prod_{i \notin I} f(\boldsymbol{\theta}_i) \\
&= \prod_{i \in I} f(\theta'_{cs}, \theta_{-ci}) \prod_{i \notin I} f(\boldsymbol{\theta}_i) \prod_{i \in I} f(\theta_{ci} \mid \theta'_{-cs})\, \alpha'_i \\
&= f(\theta')\, q(\theta \mid \theta')\,. \qquad (16)
\end{aligned}
$$

This holds for all $I \subseteq \{1, .., m\}$, and hence for all 2^m possible distinct moves that can occur in one iteration of the chain. Thus H satisfies detailed balance with respect to Π, and admits Π as its stationary distribution and unique limiting distribution. ■

We now move to the second stage of the proof. From here, we would like to show that H converges asynchronously, i.e., convergence is still valid in the setting in which each worker doesn't know the precise current state of all other workers, and instead works with the latest state that it knows about. We begin by stating Frommer and Szyld's model of distributed computation. Some of their notation, such as H and k, overlaps with ours.

Definition 4 (Asynchronous Computation).
Start with the following fixed-point computation problem:

(P1) Let $E \triangleq \bigtimes_{i=1}^{m} E_i$ be a product space. Index i refers to the component of E belonging to worker w_i.

(P2) Let $H : E \to E$ be a function (here, a Markov operator), and denote its individual components by H_i.

(P3) Let x^* be a fixed point of H, i.e., $x^* = H(x^*)$.

Now, define the following cluster computation model:

- Let $k \in \mathbb{N}_0$ be the total number of iterations performed by all workers.
- Let $s_i(k) \in \mathbb{N}_0$ be the total number of iterations on component i by all workers.
- Let I_k be an index set containing the components updated at iteration k.

Next, assume the following basic regularity conditions on the cluster:

(R1) $s_i(k) \leq k - 1$, i.e., a worker's current state cannot be based on future values.

(R2) $\lim_{k \to \infty} s_i(k) = \infty$, i.e., workers don't stop permanently mid-run.

(R3) $|\{k \in \mathbb{N} : i \in I^k\}| = \infty$, i.e., no component stops being updated or communicated.

Finally, define x^k component-wise via the following:

$$
x_i^k \triangleq \begin{cases} H_i\left(x_1^{s_1(k)}, .., x_m^{s_m(k)}\right) \text{ for } i \in I^k\,, \\ x_i^{k-1} \text{ otherwise.} \end{cases} \qquad (17)
$$

Then x^k is termed an *asynchronous iteration*, and $\{x^k : k \in \mathbb{N}_0\}$ is termed an *asynchronous computation*.

The above definition is broad enough to encompass almost every asynchronous distributed computation that possesses any hope of convergence, and its requirements should be trivially true in any such computation.

In particular, it does not make sense to even think about convergence in situations where work on some portion of the problem stops prematurely and permanently, violating (R2). It also does not make sense to think about asynchronous distributed computations for which there does not exist a way to split up the problem among the workers.

With this computational model in mind, the following general theorem gives a sufficient set of conditions under which the asynchronous iterates x^k converge to the correct answer.

Result 5 (Asynchronous Convergence).
Given a well-defined asynchronous computation as in Definition 4, assume the following conditions hold for all $k \in \mathbb{N}_0$:

(C1) There are sets $E^k \subseteq E$ satisfying $E^k = \bigtimes_{i=1}^m E_i^k$ *(box condition).*

(C2) For E^k in (C1), $H(E^k) \subseteq E^{k+1} \subseteq E^k$ *(nested sets condition).*

(C3) There exists x^* such that $y^k \in E^k \implies y^k \to x^*$ in some metric *(synchronous convergence condition).*

Then $x^k \to x^*$ in the same metric.

Proof. See Frommer and Szyld [14], Bertsekas [6], and Baudet [5]. ■

This result can be used to prove that a wide variety of asynchronous iterative algorithms converge. In particular, we believe that it can be used to construct an alternative proof that the *Hogwild* scheme (asynchronous stochastic gradient descent) [37] converges, as well as for asynchronously parallelizing other iterative stochastic algorithms such as EM [11].

To write a proof using the above result, the main thing that needs to be shown is that the state space for the iterative algorithm in question can be partitioned into a sequence of boxes that shrink toward the fixed point in question in a sequentially continuous way as the iterative operation is applied. Intuitively, once such a sequence is defined, the *Banach Contraction Mapping Theorem* [3] guarantees that it will eventually land within a neighborhood of the fixed point. Within the context of Asynchronous Gibbs sampling, this amounts to proving that a well-defined standard Gibbs sampler can be partitioned according to (P1) and passes conditions (C1–C3) described above.

We need the following basic regularity assumption.

Assumption 6 (No Worker Dies).
All of the following hold:

(i) At least one worker is assigned to each full conditional $\theta_c \mid \theta_{-c}$ defined in the underlying chain.

(ii) For each worker w_i and each coordinate c, the sequence $\theta_{ci} \mid \theta_{-ci}$ is infinite.

(iii) For each worker w_i, coordinate c, and iteration $s_i(k)$, θ_{ci}^k is updated conditional on $\theta_{ci}^{k'}$ where $k' \leq k-1$.

Justification. Without these assumptions, asynchronous convergence is hopeless. Condition (i) ensures that, taken together, the workers actually work on the full problem. Condition (ii) is needed because without it there will be some dimension that stops either being updated or communicated after finite time. Note that in practice, if a worker does crash, it suffices to restart the worker sufficiently quickly. Condition (iii) ensures that workers cannot receive updates from the future.

We need the following result from Markov Chain theory.

Result 7 (Monotonic Convergence).
Suppose P is a Markov operator with unique stationary distribution π, initial distribution $\mu \in \mathscr{M}$. Then

$$||P^{k+1}(\mu) - \pi||_{\text{TV}} \leq ||P^k(\mu) - \pi||_{\text{TV}}\,. \tag{18}$$

Proof. Meyn and Tweedie [34], Proposition 13.3.2. ■

We now proceed with the proof.

Lemma 8 (Box Condition).
Take $E = \mathscr{M}$ and $E_i = \mathscr{M}_i$. Fix initial distribution $\mu \in \mathscr{M}$. Define the following:

$$E^k \triangleq \{\nu \in \mathscr{M} : ||\nu - \pi||_{\text{TV}} \leq ||H^k(\mu) - \pi||_{\text{TV}}\}\,. \tag{19}$$

Then there exist sets E_i^k such that $E^k = \bigtimes_{i=1}^m E_i^k$.

Proof. From the definitions we know that $E = \bigtimes_{i=1}^m E_i$. By construction, $E^k \subseteq E$. Let $E_i^k = E^k \cap E_i$. Because all of these sets are by definition nonempty, and the sets E_i form a partition of E (since $\mathscr{M} = \bigtimes_{i=1}^m \mathscr{M}_i$), we get $E^k = \bigtimes_{i=1}^m E_i^k$. ■

Lemma 9 (Nested Sets Condition).
Let E^k be defined as in the previous lemma. Then $H(E^k) \subseteq E^{k+1} \subseteq E^k$.

Proof. Fix ν and consider E^k. By monotonic convergence, $||H^k(\mu) - \pi||_{\text{TV}}$ is non-increasing in k, and we get $E^{k+1} \subseteq E^k$. We also have $E^{k+1} = H(E^k)$ by construction. ■

Theorem 10 (Asynchronous Gibbs Converges).
Assume all assumptions and theorems above. Then Asynchronous Gibbs sampling converges to π on each worker in total variation.

Proof. Below, we verify that all of the conditions required in Result 5 (Asynchronous Convergence) hold.

(P1) Take $E = \mathscr{M}$.

(P2) Take H as defined in the Synchronous Parallel Chain Definition.

(P3) Take $x^* = \Pi$.

(R*) All satisfied by the No Worker Dies Assumption.

(C1) Satisfied by the Box Condition Lemma.

(C2) Satisfied by the Nested Sets Condition Lemma.

(C3) Satisfied by the Synchronous Parallel Convergence Theorem.

Now invoke the Asynchronous Convergence Result to conclude that the distribution from the combined Asynchronous Gibbs sampling chain across all workers converges to Π in total variation, and thus each worker's chain converges to π marginally. ■

We note in conclusion that this proof does not require any structure on the underlying chain, beyond the basic assumptions required for it to converge. In the example in Section 6.2, for instance, each worker updates all of its assigned variables θ_i in a single block.

5.2 Exchangeable Latent Variable Models and Exact Asynchronous Gibbs

If we are interested in sampling from an exchangeable latent variable hierarchical Bayesian model, the posterior ratio used in the MH acceptance test in exact Asynchronous Gibbs simplifies to an expression involving only one data point – this means that this ratio can be evaluated locally to each worker in a parallel environment. To illustrate, consider the following model:

$$x_i \mid \nu_i \propto A(\nu_i) \qquad \nu_i \mid \theta \overset{\text{iid}}{\sim} B(\theta) \qquad \theta \propto \pi(\theta)\,, \tag{20}$$

in which A and B are arbitrary distributions. We can define a Gibbs sampler of the form

$$\nu_i \mid \theta, x_i \sim C(\theta, x_i) \qquad \theta \mid \nu_1, .., \nu_N \sim D(\nu_1, .., \nu_N)\,, \tag{21}$$

where C and D are arbitrary distributions. Assume that we can sample from C directly. This model's posterior distribution has the form

$$\prod_{i=1}^{N} f(x_i \mid \nu_i)\, f(\nu_i \mid \theta)\, \pi(\theta) \propto \prod_{i=1}^{N} f(\nu_i \mid x_i)\, f(\theta \mid \nu_i) \propto \prod_{i=1}^{N} f(\theta, \nu_i \mid x_i)\,. \tag{22}$$

Now define an Asynchronous Gibbs sampler in which all workers transmit the values of their corresponding ν_i but never transmit θ. Consider a transmitted update from ν_j to ν_j'. Let q be the full conditional proposal distribution on the worker that sent ν_j', and assume that this worker transmits the parameters of that distribution along with ν_j'. Notice that since q is a full conditional distribution, it does not depend on ν_j or the previous value of ν_j' on the transmitting worker. Then the MH acceptance probability takes the form:

$$\min\left\{1, \frac{f(\theta, \nu_j' \mid x_j)\ \prod_{i \neq j} f(\theta, \nu_i \mid x_i)\, q(\nu_j)}{f(\theta, \nu_j \mid x_j)\ \prod_{i \neq j} f(\theta, \nu_i \mid x_i)\, q(\nu_j')}\right\} = \min\left\{1, \frac{f(\theta, \nu_j' \mid x_j)\, q(\nu_j)}{f(\theta, \nu_j \mid x_j)\, q(\nu_j')}\right\}. \tag{23}$$

Thus we can carry out the evaluation using only one data point. The details for doing so are problem-specific and depend on how the data is stored. For example, if x_j is not available on other workers, we can transmit it over network along with ν_j'. If ν_j is also not available on other workers, but the latent variables ν_j form a non-overlapping partition among the workers, then we can transmit (ν_j', ν_j, x_j, q), because ν_j can only be updated on others workers through communication. This situation occurs in some problems where parameters – such as θ in Equation (20) – that are located at the top of a hierarchical model may depend on ν_j only through sufficient statistics, and where storing ν_j for all j on every worker is thus unnecessary. These details illustrate the flexibility Asynchronous Gibbs sampling gives the user in handling large distributed data sets.

Note that due to the large number of MH evaluations, it will typically be too expensive to perform all of them. We instead recommend computing and storing the MH ratios at random with small probability, and using them as a convergence diagnostic – see Section 6.4.

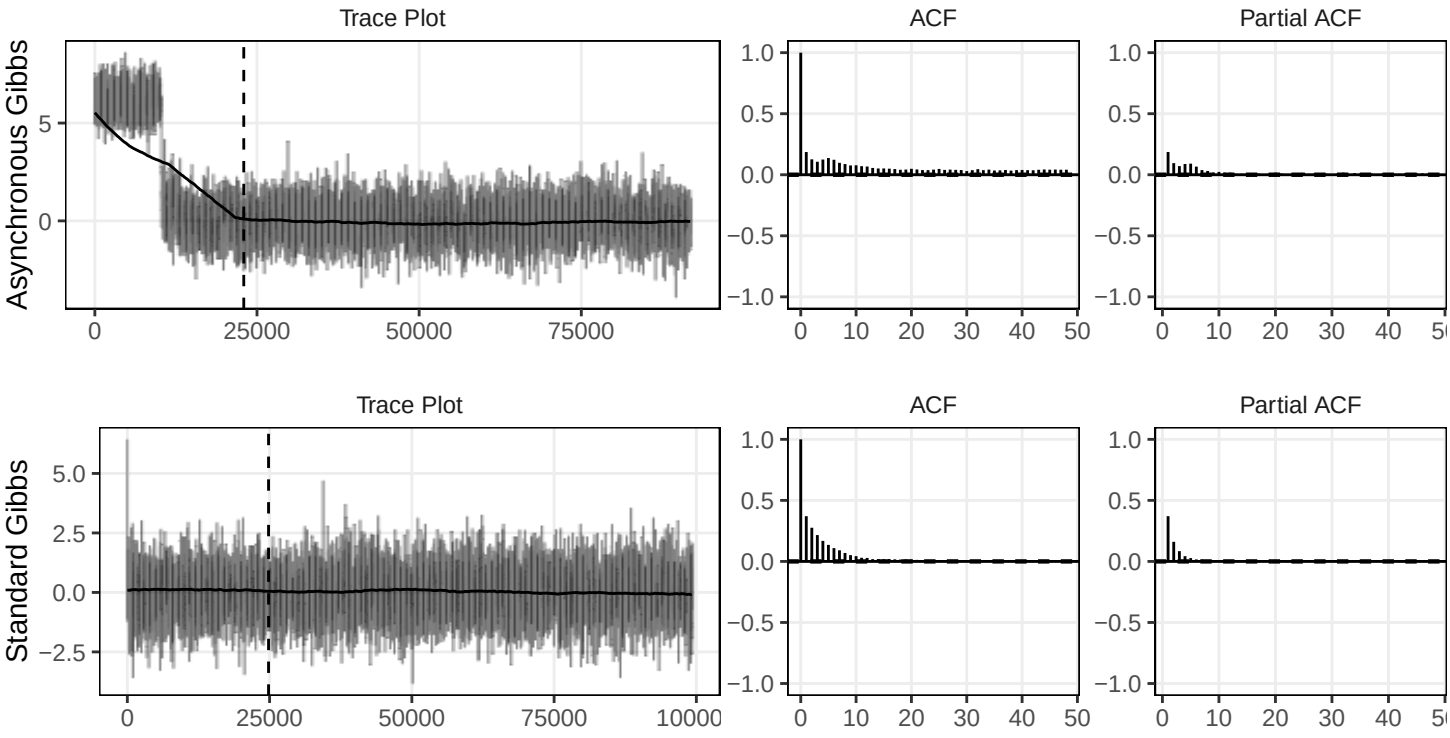


Figure 1: *Diagnostics for Asynchronous Gibbs (top row) and standard Gibbs sampling (bottom row), in the correlated standard multivariate normal example of Section 6.1.*

6 Examples

6.1 Example: Toy Problem, an 8-dimensional correlated standard normal

In this example, we generated samples with approximate Asynchronous Gibbs sampling from a correlated 8-dimensional Gaussian with mean zero, variance 1, and exponential covariance function, i.e., $\mathbf{\Sigma}_{ij} = \exp\{-\phi|i-j|\}$ with ϕ set to 0.5. This example was the first instance of Asynchronous Gibbs sampling that we implemented, with the intention of numerically testing whether the algorithm that we conceptualized would converge to the correct stationary distribution. Note that this is not necessarily a trivial problem from a Gibbs sampling standpoint: a correlated standard normal has a "narrow" density (running along a ridge that is not parallel to the axes), which makes it relatively difficult for the sampler, which can only move at 90° angles, to explore the distribution – the exact difficulty depends on the covariance matrix. Our sampler was started from absurd initial conditions: $\mathbf{10}_8$, a vector with all values set to 10 (i.e., 10 standard deviations away from the correct answer). We used 8 total workers, each of which was assigned one full-conditional dimension of the Gaussian to sample. The workers communicated with each other over the network. Trace plots of the output for the first of the 8 multivariate normal components are presented in Figure 1 (the plots for all other components were essentially identical to those in this figure), along with diagnostic plots of the output of a standard Gibbs sampler. The algorithm quickly finds the correct location parameter, and the QQ plots (Figure 2) show no evidence of non-normality in the samples.

It is difficult to assess whether Asynchronous Gibbs or the standard Gibbs sampler performed better. The

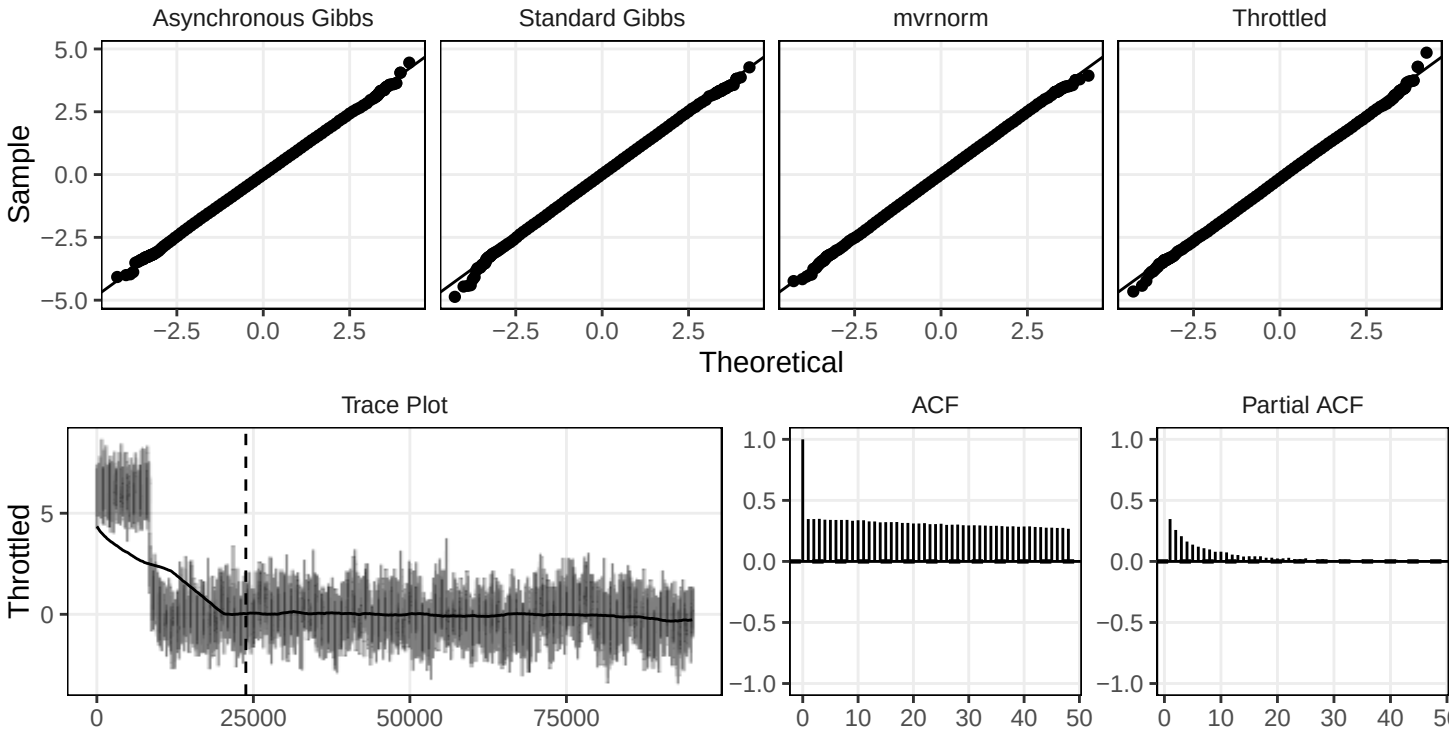


Figure 2: *Top row: QQ plots of Asynchronous Gibbs, standard Gibbs sampling, and* `mvrnorm` [53] *in the example of Section 6.1, as well as the throttled variant described in the text. Bottom row: diagnostics for Asynchronous Gibbs, throttled variation, in the same example. All plots are based on the first of the 8 multivariate normal components.*

trace plots for the standard sampler look better: it was able to find the correct location almost instantly, whereas Asynchronous Gibbs took around 10,000 iterations to do so. The ACF plot for Asynchronous Gibbs has smaller values at low time lags, but it decays to zero at a rate slower than that of standard Gibbs. However, Asynchronous Gibbs runs entirely in parallel, and is able to produce a vastly larger number of iterations in the same amount of time. The partial autocorrelation function (PACF: for a review of properties of the ACF and PACF see, e.g., [8]) goes to zero after about 12 lags, indicating that Asynchronous Gibbs is behaving in a way consistent with a 12th order autoregressive process – this provides a rough measure of how much delay is associated with communicating over the network.

Given that Asynchronous Gibbs worked so well without interference, we ran a subsequent test to study how decreased network traffic would affect the sampling, with the goal of seeing whether the algorithm would converge at all – we call this the "throttled" variation. We instructed each worker to send out messages with probability only 0.01, instead of deterministically sending out messages at each iteration (and allowing the computer's operating system and network infrastructure to handle traffic management). Trace plots for this reduced-communication variant can be seen in Figure 2. The algorithm again finds the correct location, and the QQ plots show no evidence for non-normality. However, the ACF and PACF plots look substantially worse than in the previous case, indicating that the decreased frequency of communication resulted in a slower-mixing chain.

6.2 Example: Gaussian Process regression, a highly simplified spatial model with $n = 71,500$

This problem originally appeared in an example due to Neal [35] – we examine a variation due to Kottas [29] (personal communication). In this example, we used our algorithm to sample from the posterior distribution arising from a simple Gaussian Process regression problem. Our method is close to exact – inexactness is introduced only through a block matrix inverse approximation scheme, and through approximate Asynchronous Gibbs.

This example is far too simple for use in a real spatial statistics problem – rather, we present it as a way to illustrate how approximate Asynchronous Gibbs sampling can be used for computation at scale. We expect that future work will be able to extend our approach to more realistic settings. Our goal was to reconstruct the following function:

$$\tilde{f}(x) = 0.3 + 0.4\,x + 0.4\,\sin(2.7\,x) + \frac{1.1}{1+x^2} \qquad\qquad x \in [-3, 3]\,. \tag{24}$$

This is then reflected and copied around the lines $x = 3, 9, ..,$ and $x = -3, -9, ..,$ in such a way that $\tilde{f}(x)$ becomes periodic with period 6 and is continuous everywhere. To simplify our example, we assumed that our data lives on a grid with spacing equal to 0.06 (i.e., $x_1 = 0, x_2 = 0.06, x_3 = -0.06, ..$). To generate the data, we added Gaussian white noise with standard deviation 0.2. Our model for reconstructing this function is then defined in the following way:

$$y_i = f(x_i) + \varepsilon_i \qquad\qquad f(x_i) \sim \mathrm{GP} \qquad\qquad \varepsilon_i \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma^2)\,. \tag{25}$$

Here $i = 1, .., n = 71,500$ with x on $[-2,145,\ 2,145)$. For simplicity, we selected a Gaussian Process with constant mean function μ and exponential covariance function $-\tau^2 \exp\{-\phi|x - x'|\}$, together with the following hyperpriors:

$$\mu \sim \mathrm{N}(a_\mu, b_\mu) \qquad \sigma^2 \sim \mathrm{IG}(a_\sigma, b_\sigma) \qquad \tau^2 \sim \mathrm{IG}(a_\tau, b_\tau) \qquad \phi \sim \mathrm{U}(0, b_\phi)\,. \tag{26}$$

By introducing latent variables θ_i corresponding to each data point, the model can then be expressed in the following way:

$$y_i \mid \theta_i, \sigma^2 \sim \mathrm{N}(\theta_i, \sigma^2) \qquad \boldsymbol{\theta} \sim \mathrm{N}_n\left(\mu\,\mathbf{1}_n, \tau^2\,\mathbf{H}(\phi)\right) \qquad H_{ij}(\phi) = \exp\left\{-\phi|x_i - x_j|\right\}\,. \tag{27}$$

By conjugate updating, this yields Inverse–Gamma posteriors for σ^2 and τ^2, a normal posterior for μ, and an n-dimensional multivariate normal posterior for $\boldsymbol{\theta}$. Since ϕ is non-conjugate, to simplify things for our example we fixed it at 0.5, which is an interpretable value that is close to its MLE. If n is large, block sampling from this posterior is intractable because it requires the frequent inversion of two $(n \times n)$ matrices. It's possible to integrate $\boldsymbol{\theta}$ out of the model, but this does not avoid large matrix inversion. We propose the following scheme to sample from the posterior of $\mu, \sigma^2, \tau^2, \boldsymbol{\theta}$.

In our approach, we update individual slices of $\boldsymbol{\theta}$, consisting of 500 elements, via Gibbs steps. To do this, we sample from full conditional distributions of the form $\boldsymbol{\theta}_{1:500} \mid \boldsymbol{\theta}_{501:n}, \mu, \sigma^2, \tau^2$ for arbitrary indices (recall that ϕ is fixed). Thus we need to sample from conditional Gaussian distributions of portions of $\boldsymbol{\theta}$, given the rest of $\boldsymbol{\theta}$. To do this without ever constructing the large covariance matrix, which may be too big to store in memory, we need to be able to invert $\mathbf{H}(\phi)$, multiply by τ^{-2}, add $\sigma^{-2}\mathbf{I}_n$, and invert back. The following scheme allows us to do this elementwise, with only one approximate inversion along the way, which can with further work likely be refined into an exact inversion.

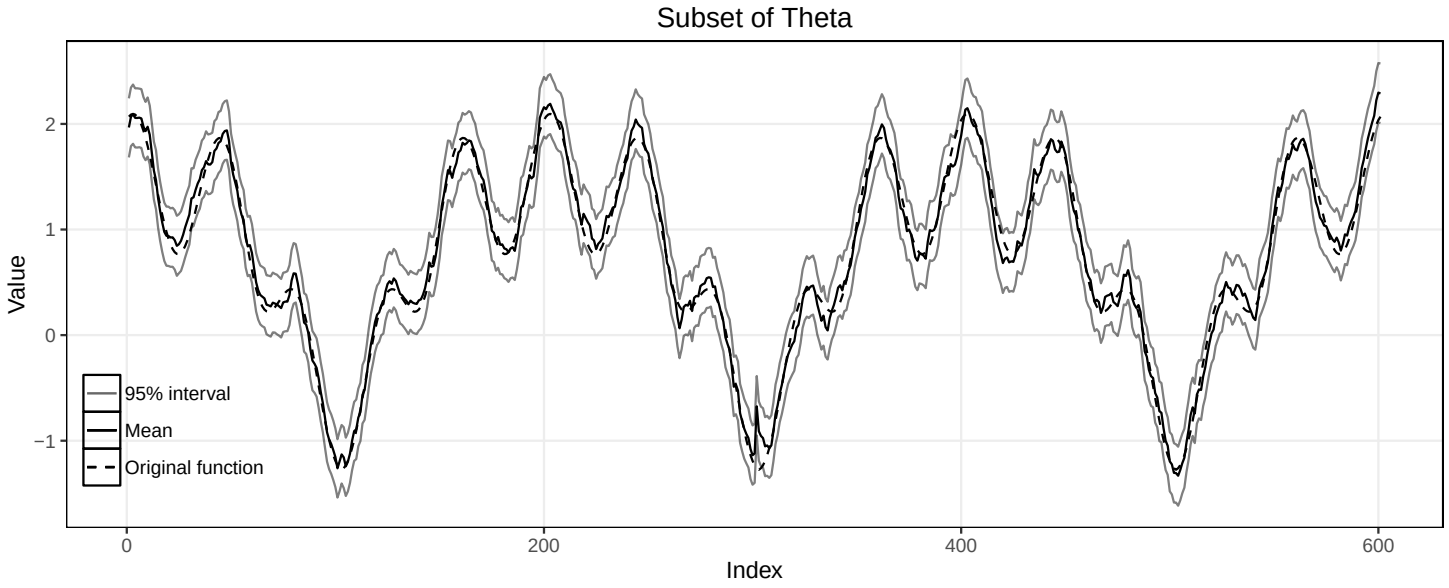


Figure 3: *Partial subset of $\boldsymbol{\theta}$ for two workers (split at center), in the Gaussian Process regression example of Section 6.2.*

Since we have made the simplifying assumption that our grid is evenly spaced, the covariance matrix $\mathbf{H}(\phi)$ is Toeplitz. Additionally, since our covariance function is exponential, the resulting covariance matrix is hyperbolic, and can be inverted element-wise analytically via a technique due to Dow [13], with inverse that simplifies to

$$\mathbf{H}^{-1}(\phi) = \begin{bmatrix} d_0 & a & 0 & \dots & \dots & \dots & 0 \\ a & b & a & 0 & \ddots & \ddots & \vdots \\ 0 & a & b & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & b & a & 0 \\ \vdots & \ddots & \ddots & 0 & a & b & a \\ 0 & \dots & \dots & \dots & 0 & a & d_0 \end{bmatrix} \quad \begin{cases} b = -\coth(-\phi\rho) \\ a = \frac{\operatorname{csch}(-\phi\rho)}{2} \\ d_0 = \frac{e^{-\phi\rho(2N-3)}\operatorname{csch}(-\phi\rho)+1-\coth(-\phi\rho)}{2-2e^{-\phi\rho(2N-3)}} \\ \rho = \text{grid spacing size} = 0.06 \\ N = \text{dimension of } \mathbf{H}(\phi) \end{cases} . \tag{28}$$

Note that this $\mathbf{H}^{-1}(\phi)$ is tridiagonal with modified corner elements. Does this inversion scheme hopelessly limit the generality of our technique? We conjecture that the answer is no. Given their structure, similar closed-form inversion techniques should be possible for other covariance functions. With further work, such techniques could possibly apply to non-Toeplitz covariance matrices as well.

Next, we multiply by τ^{-2}, and add σ^{-2} to the diagonal. The resulting covariance matrix is still tridiagonal with modified corner elements. We do not know how to invert this matrix analytically, but we do know how to invert the general tridiagonal Toeplitz matrix without modified corner elements, via a technique due to Hu and O'Connell [23]. We approximate the tridiagonal form by assuming that $d_0 \doteq b$ in (28) – this works well except at the points where the partition slices of $\boldsymbol{\theta}$ join, where some error is introduced. It turns out that this inversion scheme includes the numerically unstable terms $\operatorname{sech}(x)$ and $\cosh(x)$, which are unbounded and asymptotically increasing on $\mathbb{R}^+$. Fortunately, for large values of x, these terms are approximately equal to $\frac{1}{2}e^x$, allowing us to cancel out numerically unstable terms, and this approximation

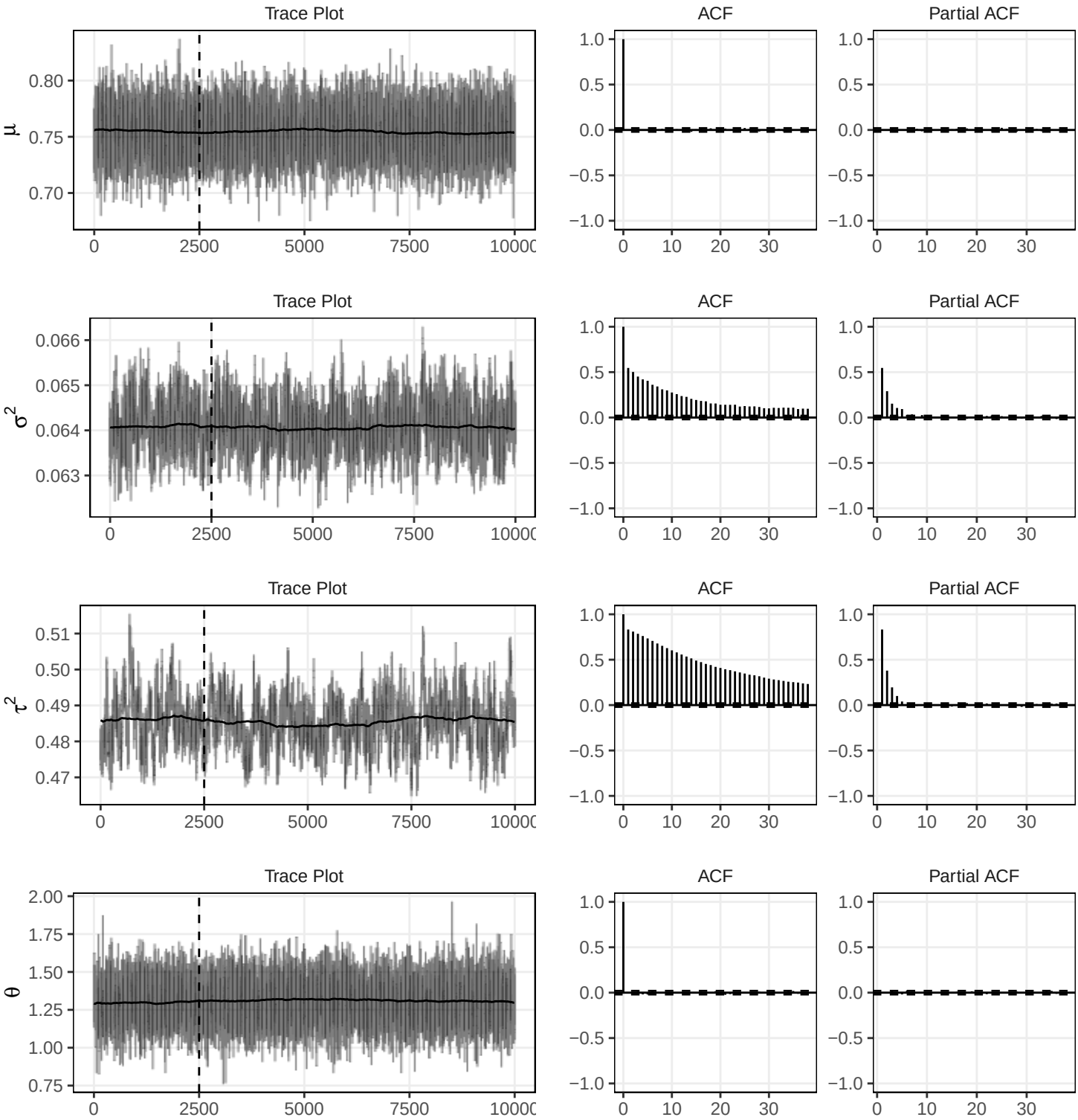


Figure 4: *Diagnostics for μ, σ^2, τ^2, and one θ_i, in the Gaussian Process regression example of Section 6.2.*

is asymptotically exact.

Finally, to find the mean vector, we need to multiply the covariance matrix defined by (28) by a term that

includes the full data. Since the data set can be too large to store on a single machine, this is intractable. However, not all is lost: since the resulting covariance function decreases rapidly to 0 as the distance away from the full conditional of interest increases, this multiplication can be carried out to arbitrary precision by simply taking a slice in the center of the matrix in a neighborhood around the full conditional of interest. In this way we're able to calculate the mean function to within numerical precision without ever performing a computation involving the full data set. This idea also underlies *covariance tapering* [15] and *composite likelihood methods* for spatial problems [47].

After all of these steps, we can sample any slice of $\boldsymbol{\theta}$ full conditionally via the standard Schur complement formula (for a review of this result, see [39]), since the full conditional of a Gaussian is Gaussian.

With standard Gibbs, this technique is still intractable: there are too many full conditionals to sample for the chain to produce its output in a reasonable amount of time. Asynchronous Gibbs lets us parallelize this computation, producing results in minutes (given sufficient hardware) that would have taken days or weeks with standard MCMC. In this example we used 143 workers with 1 CPU each. Each worker was responsible for 500 values of $\boldsymbol{\theta}$ (different from those handled by the other workers), and for μ, σ^2, τ^2 . We started our algorithm from garbage initial values: $\mu = 10, \sigma^2 = 10, \tau^2 = 10, \boldsymbol{\theta} = \mathbf{0}$. Our algorithm converged rapidly to within a small neighborhood of the correct solution. It finished, producing approximately 10,000 samples per worker, in around 20 minutes.

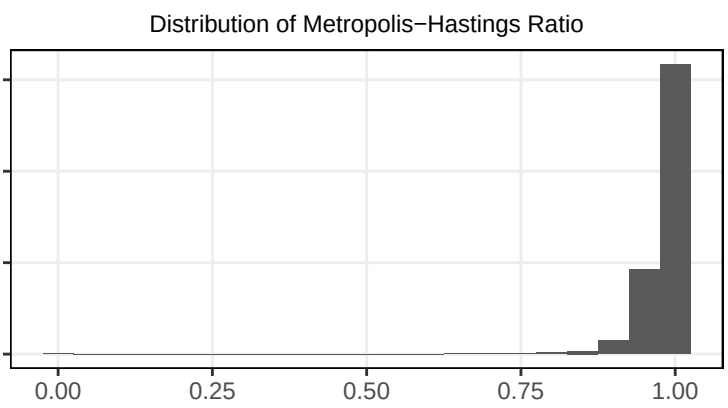
In Figure 3 we plot a slice of the data, together with the correct solution. As noted above, our matrix inversion approximation scheme is inaccurate around the edges of each slice of $\boldsymbol{\theta}$ (this can be seen in the middle of Figure 3, at index 300), and hence these values are not as accurate as those elsewhere. The algorithm converged in an analogous fashion for all other slices of the data. Figure 4 presents diagnostic plots for μ, σ^2, τ^2 and one θ_i.

All of of the components of (μ, σ, τ) are relatively easy to learn. We have observed similar results for a wide variety of statistical models in the large-data setting: parameters that depend on the full data tend to be easy to learn, with low posterior variance, and can often safely be fixed at their MLEs if doing so helps computation. Note that if we had not fixed ϕ, we would have needed to compute a large matrix expression involving $\mathbf{H}^{-1}(\phi)$ in its entirety for every sample of τ^2 and μ. Because $\mathbf{H}^{-1}(\phi)$ is available analytically, this is tractable, but for simplicity, we chose to fix ϕ instead. This choice is reasonable given that ϕ and τ^2 are (jointly) weakly identifiable.

We conclude that Asynchronous Gibbs sampling allows us to fit Gaussian Process regression models at scale without variational methods and the other classes of approximations described in Section 3.

6.3 Example: Mixed-effects regression, a complex hierarchical model with $n = 1,000,000$

The following model, due to von Brzeski et al. [54], was used in a large-scale analysis of product updates at eBay Inc. Because users choose when to update to the latest version of the product, analysis of product updates is done not by experiment but by observational study, and causal inference is difficult. In particular, it's necessary to control for the *early-adopter effect*, in which the behavior of the response is correlated with how quickly a user adopts the treatment after release. To adjust for this effect, a Bayesian hierarchical mixed-effects regression model was selected. Since we are primarily interested in the computational aspects of this problem, we omit further discussion of how and why the particular model was selected and interpretation of the results – such discussion can be found in the original publication. A



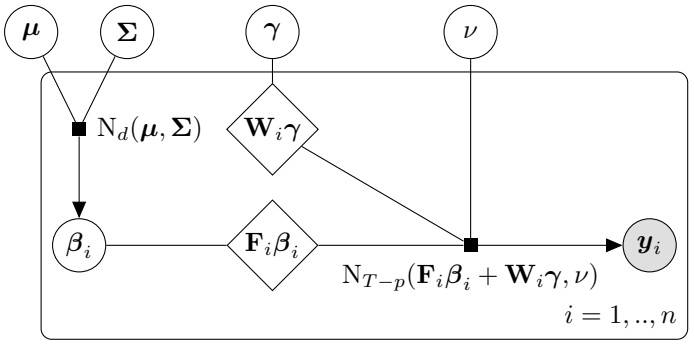


Figure 5: *(Left panel) Distribution of MH acceptance probability, and (right panel) directed acyclic graph for the model, in the hierarchical mixed-effects regression example of Section 6.3.*

variety of different data sets have been used with this model – the data set that we employed, selected for convenience, consists of $n = 1,000,000$ users. The model can be written in the following way:

$$\boldsymbol{y}_i = \mathbf{F}_i\boldsymbol{\beta}_i + \mathbf{W}_i\boldsymbol{\gamma} + \boldsymbol{\varepsilon}_i \qquad \boldsymbol{\beta}_i \mid \boldsymbol{\mu}, \boldsymbol{\Sigma} \overset{\text{iid}}{\sim} \mathrm{N}_d(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \qquad \boldsymbol{\varepsilon}_i \mid \nu \overset{\text{iid}}{\sim} \mathrm{N}_{T-p}(\mathbf{0}, \nu\,\mathbf{I}_{T-p})\,, \tag{29}$$

with the following data:

$$\boldsymbol{y}_i : (T-p) \times 1 \qquad \mathbf{F}_i : (T-p) \times d \qquad \mathbf{W}_i : (T-p) \times (T-p)\,, \tag{30}$$

the following parameters:

$$\boldsymbol{\beta}_i : (d \times 1) \qquad \boldsymbol{\gamma} : (T-p) \times 1 \qquad \boldsymbol{\mu} : (d \times 1) \qquad \boldsymbol{\Sigma} : (d \times d) \qquad \nu : \text{scalar}\,, \tag{31}$$

and the following priors:

$$\boldsymbol{\mu} \sim \mathrm{N}_d(\mathbf{0}, \kappa_\mu \mathbf{I}_d) \qquad \boldsymbol{\Sigma} \sim \mathrm{IW}_d(d+1, \mathbf{I}_d) \qquad \boldsymbol{\gamma} \sim \mathrm{N}_{T-p}(\mathbf{0}, \kappa_\gamma \mathbf{I}_{T-p}) \qquad \nu \sim \mathrm{IG}(\epsilon/2, \epsilon/2)\,. \tag{32}$$

Here $i = 1, .., n$ indexes individual data points (eBay users), $\boldsymbol{y}_i$ is a vector of values representing customer satisfaction for user i over time (aggregated to the weekly level), $\mathbf{F}_i$ and $\mathbf{W}_i$ are user-specific matrices of known constants (fixed effects), d is the length of the random-effects vector, $T = 52$ is the number of weeks of data for each user, p is the number of lags of autoregression in the model (typically no more than 5), and $\kappa_\mu, \kappa_\gamma, \epsilon$ are fixed hyperparameters. The posterior distribution arising from this model has the following full conditionals:

$$\begin{aligned}
\boldsymbol{\beta}_i \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\gamma}, \nu, \text{data} &\sim \mathrm{N}_d(\boldsymbol{m}_i, \mathbf{C}_i) & \mathbf{C}_i &= (\boldsymbol{\Sigma}^{-1} + \nu^{-1}\mathbf{F}_i^T\mathbf{F})^{-1} \\
& & \boldsymbol{m}_i &= \mathbf{C}_i(\nu^{-1}\mathbf{F}_i^T(\boldsymbol{y}_i - \mathbf{W}_i\boldsymbol{\gamma}) + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) \\
\boldsymbol{\mu} \mid \bar{\boldsymbol{\beta}}, \boldsymbol{\Sigma} &\sim \mathrm{N}_d(\boldsymbol{a}, \mathbf{B}) & \mathbf{B} &= (\kappa_\mu^{-1}\mathbf{I}_d + n\boldsymbol{\Sigma}^{-1})^{-1} \\
& & \boldsymbol{a} &= \mathbf{B}(n\boldsymbol{\Sigma}^{-1}\bar{\boldsymbol{\beta}}),\ \text{with } \bar{\boldsymbol{\beta}} = \tfrac{1}{n}\textstyle\sum_{i=1}^n \boldsymbol{\beta}_i \\
\boldsymbol{\gamma} \mid \boldsymbol{\beta}_{i=1,..,n}, \nu, \text{data} &\sim \mathrm{N}_{T-p}(\boldsymbol{c}, \mathbf{D}) & \mathbf{D} &= (\nu^{-1}\textstyle\sum_{i=1}^n \mathbf{W}_i^T\mathbf{W}_i + \kappa_\gamma^{-1}\mathbf{I}_{T-p})^{-1} \\
& & \boldsymbol{c} &= \nu^{-1}\mathbf{D}\boldsymbol{g},\ \text{with } \boldsymbol{g} = \textstyle\sum_{i=1}^n \mathbf{W}_i(\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i) \\
\boldsymbol{\Sigma} \mid \boldsymbol{\beta}_{i=1,..,n}, \boldsymbol{\mu} &\sim \mathrm{IW}_d(n+d+1, \mathbf{S} + \mathbf{I}_d) & \mathbf{S} &= \textstyle\sum_{i=1}^n (\boldsymbol{\beta}_i - \boldsymbol{\mu})(\boldsymbol{\beta}_i - \boldsymbol{\mu})^T \\
\nu \mid \boldsymbol{\beta}_{i=1,..,n}, \text{data} &\sim \mathrm{IG}\left(\frac{\epsilon + n(T-p)}{2}, \frac{\epsilon + l}{2}\right) & l &= \sum_{i=1}^n (\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i - \mathbf{W}_i\boldsymbol{\gamma})^T(\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i - \mathbf{W}_i\boldsymbol{\gamma})\,.
\end{aligned} \tag{33}$$

These full conditionals can be sampled using a standard sequential-scan Gibbs sampler. To increase efficiency, the variables $\boldsymbol{\beta}_{i=1,..,n}$ can be treated as a block and updated in parallel. Since they're all conditionally independent given $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\gamma}, \nu$, this multicore algorithm is still a standard sequential-scan Gibbs sampler.

Unfortunately, this parallelization scheme extends poorly from the multicore setting to the cluster setting, for a variety of reasons:

(i) *Synchronization:* The cluster must wait for all of the nodes to finish updating the β_i before proceeding with updating other variables.

(ii) *Full-data Reduce Operations:* Large sums, such as those involved in calculating l and $\mathbf{S}$, need to be performed in a distributed setting. Due to network overhead, these sums take significant time to compute.

(iii) *Resource Allocation:* Some parameters, such as ν, are both approximately constant and expensive to sample (see the discussion at the end of Section 6.2), slowing down the entire algorithm. As noted previously, this feature is common in large-data problems.

(iv) *Fault Tolerance:* If a single node performing any task goes offline, the entire algorithm stops. Given sufficient nodes, this will occur with probability close to 1.

Approximate Asynchronous Gibbs can enable this computation to be performed fully in parallel by an arbitrarily-large cluster, while eliminating many of these difficulties. It addresses each of them in the following way:

(i) By design, Asynchronous Gibbs is completely lockfree and fully asynchronous.

(ii) To avoid *Reduce* operations over the full data, we maintain a *cache* of $\bar{\boldsymbol{\beta}}$, $\mathbf{S}$, $\boldsymbol{g}$, l. To illustrate this, consider a new update of a single $\boldsymbol{\beta}_i$. When it's generated or received, the cache is updated by subtracting the portion of the sum corresponding to the old $\boldsymbol{\beta}_i$ and adding the portion corresponding to the new value. This significantly speeds up computation, but results in higher memory use. Note that for $\mathbf{S}$, since storing large matrices in memory is expensive, we instead store the value of $(\boldsymbol{\beta}_i - \boldsymbol{\mu})$ for each i – this is equivalent to storing $(\boldsymbol{\beta}_i - \boldsymbol{\mu})(\boldsymbol{\beta}_i - \boldsymbol{\mu})^T$, as it can be quickly computed when the need arises.

(iii) Each worker updates $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $\boldsymbol{\gamma}$, ν with the same probability as each individual element $\boldsymbol{\beta}_i$. With 12 workers and 1,000 iterations for each $\boldsymbol{\beta}_i$, the algorithm generates 12(1,000) = 12,000 total samples for each variable. This helps with mixing, leading to higher Monte Carlo accuracy.

(iv) A single worker going offline effectively results in the temporary loss of a small number of data points, which – given the large data set size – is highly unlikely to affect the posterior distributions of the parameters.

For a fair performance comparison between approximate Asynchronous Gibbs and standard Gibbs (with multithreaded sampling of $\boldsymbol{\beta}_{i=1,..,n}$), we implemented a simple sequential-scan Gibbs sampler in *Scala*, using the exact same matrix computation routines (*Breeze*) as in our cluster sampler. For data size $n = 1,000,000$ and 1,000 Monte Carlo iterations, running in parallel with 8 threads, the sequential-scan Gibbs sampler takes about 12 hours to finish. (Due to expected length of time, we did not attempt to run a single-threaded version.) Asynchronous Gibbs was much faster: with 20 workers, each with 8 threads (160 threads in total), the algorithm finished in about 1 hour.

Figure 5 gives the distribution of the MH acceptance probabilities, together with a directed acyclic graph

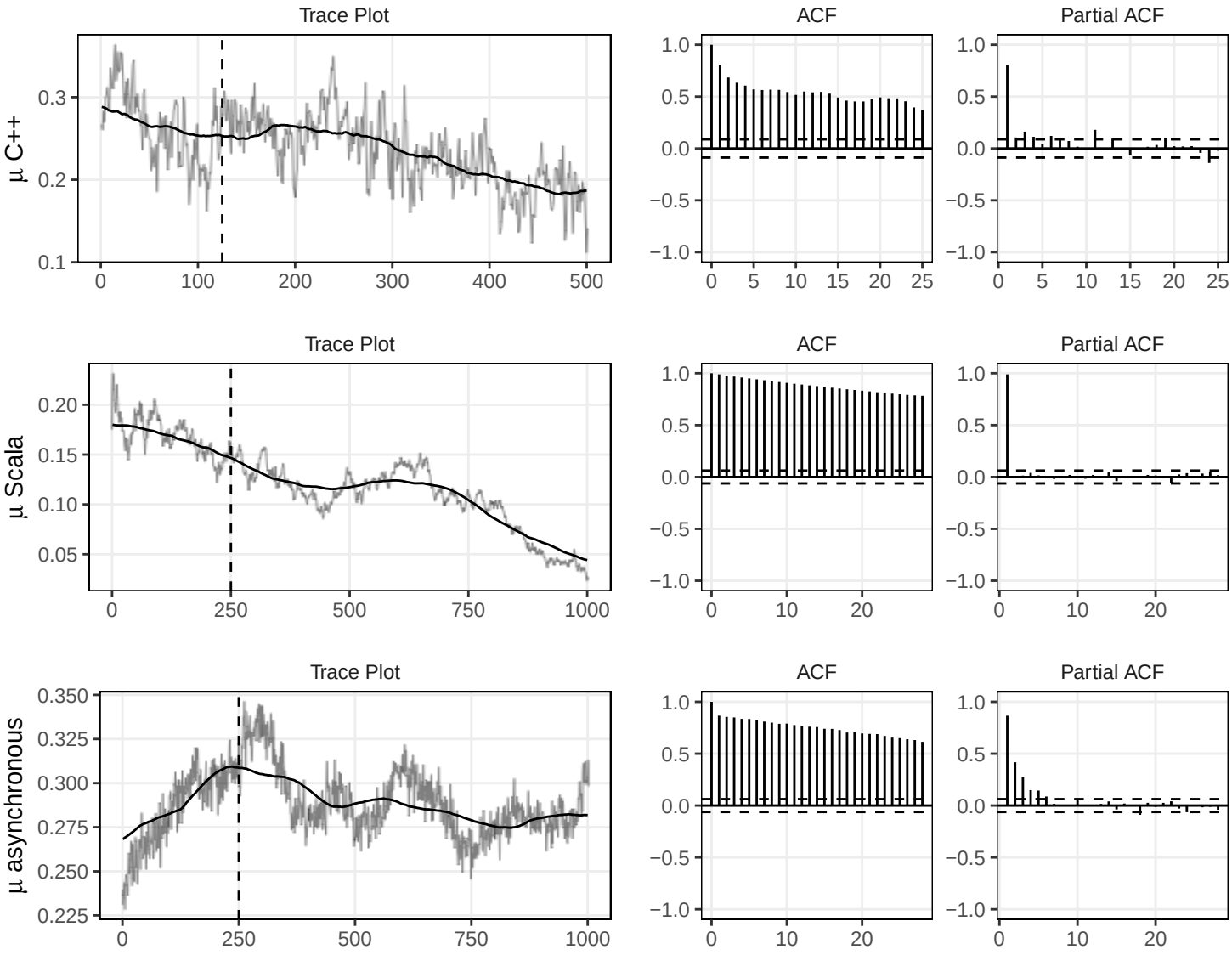


Figure 6: *Diagnostics for element 1 of* $\boldsymbol{\mu}$*, in the mixed-effects regression example of Section 6.3. Top row: sequential-scan, original C++ implementation. Middle row: sequential scan, Scala implementation. Bottom row: Asynchronous Gibbs, worker 1 only.*

representation of model (29–32). The probability of rejecting a random update is about 0.02, indicating that the behavior of the approximate algorithm is close to what the exact algorithm would have done (up to Monte Carlo noise) – see Section 6.4 for further discussion of this diagnostic. Figures 6 and 7 give diagnostics for row 1 of the $\boldsymbol{\mu}$ vector, one of the β_i and ν. From a Monte Carlo accuracy standpoint, it's again hard to tell whether the sequential-scan Gibbs sampler or Asynchronous Gibbs is better. Both chains have poor diagnostic plots, indicating issues with mixing. Note that the increased amount of total Monte Carlo iterations per variable (12,000 total, 1,000 from each worker) increases the Monte Carlo accuracy of Asynchronous Gibbs sampling, but this is not reflected in these figures because it would take too much space to plot all of the chains together.

Note from Figure 7 that it takes substantially longer for ν to reach equilibrium with the Asynchronous

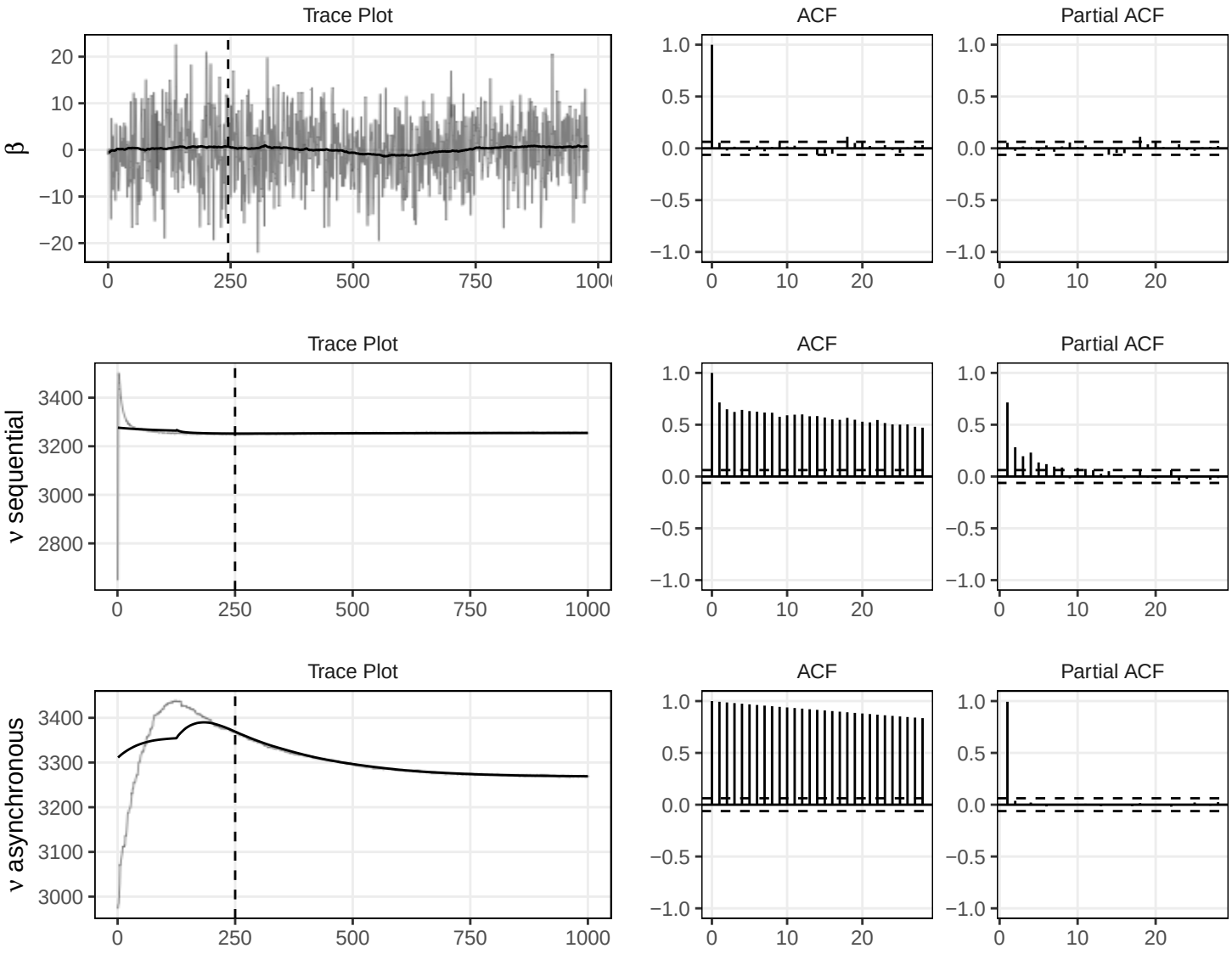


Figure 7: *Diagnostics for an unspecified $\boldsymbol{\beta}_i$ (top row) and ν (middle and bottom rows) in the mixed-effects regression example of Section 6.3. Middle row: sequental scan Scala implementation. Bottom row: Asynchronous Gibbs, worker 1 only, with the caching option turned on.*

Gibbs sampler: this is a result of caching. Before we implemented caching, the Asynchronous Gibbs trace plot looked similar to the sequential-scan trace plot, but the algorithm ran substantially more slowly due to time spent computing the sum. Note also that caching helps to ensure that all variables take a similar amount of time to sample, which is needed to ensure that the parallel chain is approximately time-homogeneous. An example due to Murray Pollock (personal communication) has demonstrated that violating time-homogeneity in a systematic way can sometimes introduce additional bias into the algorithm. Given that ν is approximately constant and not a primary parameter of interest in the real-world problem, it's quite computationally wasteful to sample it at all as part of the MCMC scheme. Instead, it would be more efficient to run Asynchronous Gibbs with ν fixed at its MLE. This can be done by using a maximization algorithm such as *Stochastic Gradient Descent*, which (as noted earlier) can be computed in parallel asynchronously via the *Hogwild* scheme [37].

Given the massively faster run time, Asynchronous Gibbs appears to outperform multicore sequential-scan Gibbs sampling. We were unable to acquire a cluster with more hardware on which to test our algorithm at larger scale – thus, the top end of its actual capability remains unknown. To summarize this example, Asynchronous Gibbs sampling produces output that appears to have sufficient Monte Carlo accuracy for the real-world problem at hand. The resulting chain mixes poorly, but the real-world problem only demands minimal precision: the main factors of interest in the original analysis are the signs of the individual components of $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$. The output of Asynchronous Gibbs sampling appears sufficient for these purposes, and the benefits of parallelism and speed appear to outweigh the costs associated with slower mixing.

6.4 Jacobi Sampling and Approximate Asynchronous Gibbs

We now illustrate a way in which Asynchronous Gibbs sampling without a Metropolis-Hastings correction can fail. The example here is due to Johnson [26], personal communication. Suppose that we have a Gibbs sampler on $(\theta_1, .., \theta_m)$ with target distribution $\pi \sim \mathrm{N}_m(\mathbf{0}, \boldsymbol{\Sigma})$.

Consider the following partially synchronous sampler with workers $(w_1, .., w_m)$, each of which updates one coordinate. Initialize arbitrary $(\theta_0^0, .., \theta_m^0)$ and, in parallel, update the following:

$$w_1 : \text{update } \theta_1^1 \mid \theta_2^0, .., \theta_m^0 \qquad .. \qquad w_m : \text{update } \theta_m^1 \mid \theta_1^0, .., \theta_{m-1}^0 \,. \tag{34}$$

Now synchronize by writing to shared memory and then reading from it, and repeat indefinitely. This sampling scheme does not converge for all $\boldsymbol{\Sigma}$ [7]. In particular, it can diverge if the precision matrix $\boldsymbol{\Sigma}^{-1}$ is not diagonally dominant. Furthermore, even when it does converge, the sample covariance matrix of the output can be incorrect. We call this algorithm *Jacobi sampling*, because the mean vector at each update is an iteration of the Jacobi algorithm for solving linear systems [44] – for the corresponding linear system, diagonal dominance suffices to ensure stability of the iterations.

We analyze the following case:

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} 1.01 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1.01 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1.01 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1.01 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1.01 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1.01 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1.01 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1.01 \end{bmatrix} \quad \boldsymbol{\Sigma} \approx \begin{bmatrix} 87.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 \\ -12.5 & 87.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 \\ -12.5 & -12.5 & 87.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 \\ -12.5 & -12.5 & -12.5 & 87.5 & -12.5 & -12.5 & -12.5 & -12.5 \\ -12.5 & -12.5 & -12.5 & -12.5 & 87.5 & -12.5 & -12.5 & -12.5 \\ -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & 87.5 & -12.5 & -12.5 \\ -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & 87.5 & -12.5 \\ -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & -12.5 & 87.5 \end{bmatrix}. \tag{35}$$

This is clearly a rather difficult target from the parallel sampling perspective due to strong dependence. Note that unlike the exponential covariance matrix in the Toy Example of Section 6.1, there is no spatial decay. We call the $\boldsymbol{\Sigma}$ in (35) the Jacobi covariance matrix to distinguish it from the $\boldsymbol{\Sigma}$ in the Toy Example.

Jacobi Sampling with a target that has the covariance matrix (35) diverges to ∞. What goes wrong? Without the MH accept-reject step, the algorithm is a Noisy Monte Carlo [1] approximation to exact

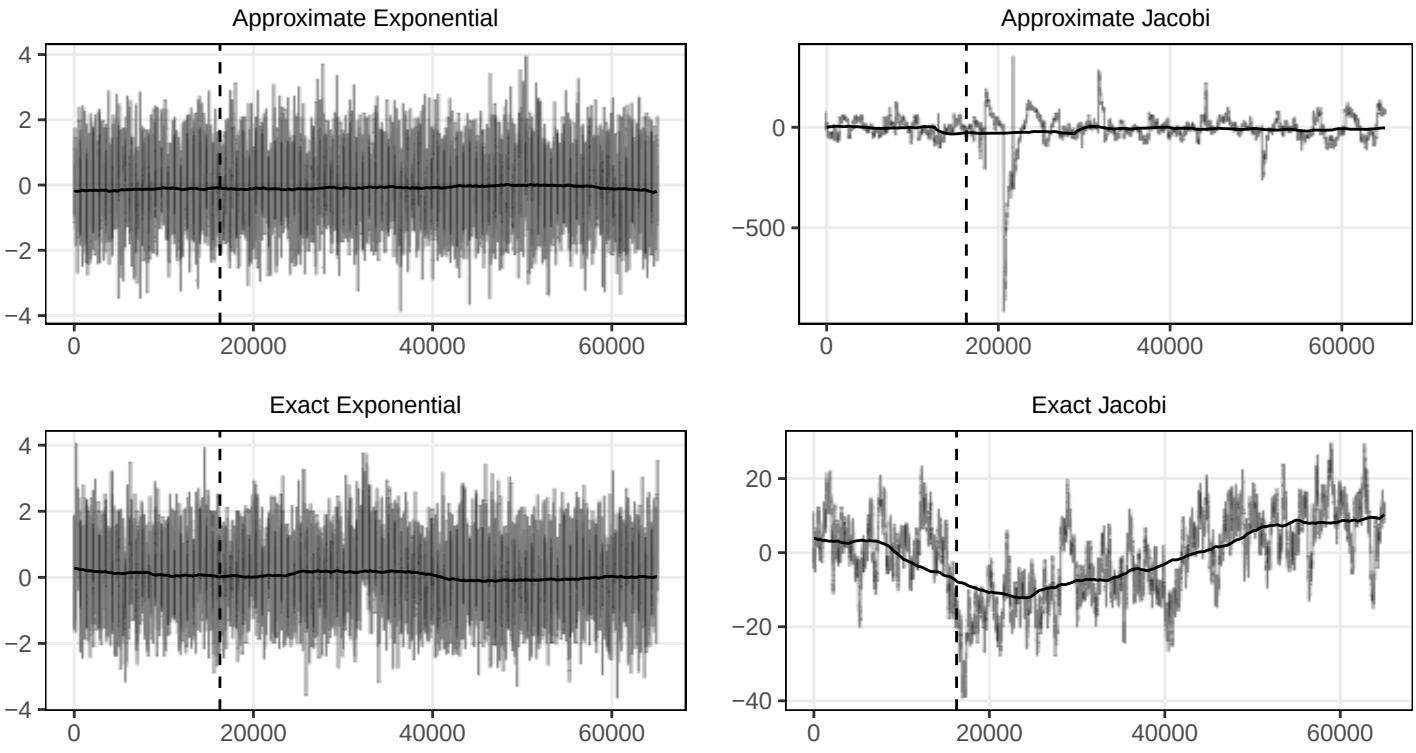


Figure 8: *Diagnostic for the Jacobi sampling algorithm, exact and approximate variations, with exponential and Jacobi covariance matrices: approximate exponential (upper left), approximate Jacobi (upper right), exact exponential (lower left), exact Jacobi (lower right).*

Asynchronous Gibbs: the MH acceptance probability is replaced by a biased estimator, namely 1. If this approximation is bad, the algorithm can fail. By comparing this target to the one in Section 6.1, we seek to provide some characterization of when this approximation is good and therefore when the MH correction in exact Asynchronous Gibbs can safely be ignored.

We now examine a variation of Jacobi sampling that numerically converges to the correct mean and incorrect covariance matrix if the approximate algorithm is used. Suppose that there are 4 workers, each with 2 full conditionals assigned to them from our 8–dimensional Gaussian target. Each worker selects one of its full conditionals at random, performs a Gibbs step, and transmits the resulting draw to each other worker with probability 0.75. The other workers then perform a Metropolis-Hastings calculation and either accept or reject the transmitted value. As long as the MH correction step is performed, this algorithm's convergence is implied by our theorem in Section 5. If the MH step is ignored, the algorithm's convergence will depend on the target covariance matrix.

We implemented both the exact and approximate versions of this variation with the covariance matrix (35) on one machine with simulated parallelism. For comparison, we also ran the variation with the same exponential covariance matrix that we used in the Toy Example. Diagnostic plots are given in Figure 8. Clearly, the algorithm does far better with the exponential covariance. The exact algorithm with Jacobi covariance matrix mixes poorly, but ends up yielding a Monte Carlo mean and covariance matrix that are not too far away from the correct answer. The approximate algorithm with Jacobi covariance matrix yields the correct mean, but vastly incorrect covariance matrix.

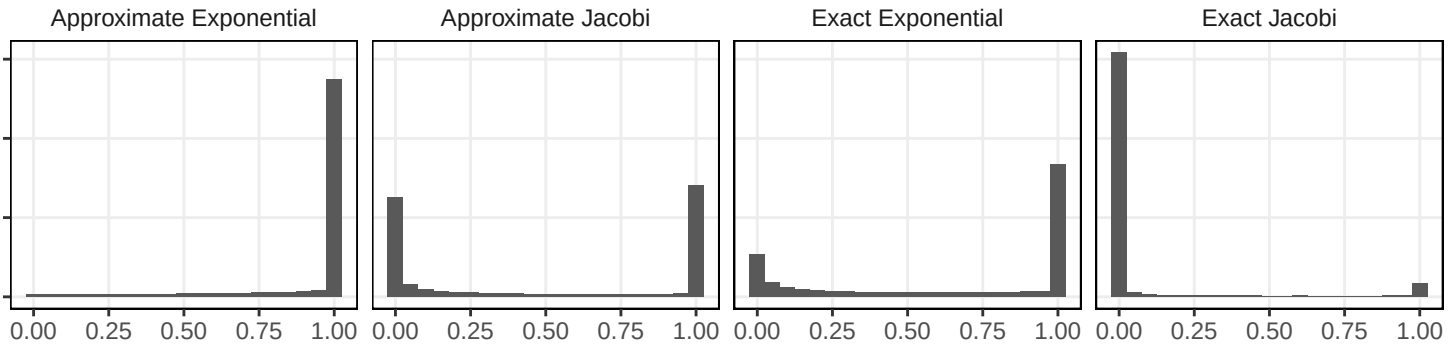


Figure 9: *Distribution of the Metropolis-Hastings acceptance ratio for both approximate and exact Asynchronous Gibbs, with the algorithm described in Section 6.4, and exponential and Jacobi covariance matrices.*

To further study the differences between the exact and approximate algorithm, we examined the distribution of the MH acceptance ratios in all four examples (Figure 9) – in the case of the approximate algorithm this was accomplished by calculating and storing the MH probabilities and then ignoring them by accepting all updates (as noted previously, this calculation would be expensive if performed on every iteration, but it's only necessary to perform it a small percentage of the time at random). This distribution was concentrated around 1 for the approximate exponential case. It was substantially lower for the approximate Jacobi case that yielded the wrong answer. Interestingly, it was also different when comparing both exact algorithms to their approximate counterparts. We are not sure how to interpret this difference, other than to note that the exact and approximate algorithms may behave differently even in situations where the approximation is relatively good.

The intuition suggested by this example leads to the following diagnostic:

Diagnostic A.
Approximate Asynchronous Gibbs is reasonable if the distribution of the Metropolis-Hastings acceptance ratio in the approximate algorithm is concentrated around 1.

If the condition in Diagnostic A is satisfied, the behavior of the approximate algorithm will be similar to that of the exact algorithm in the posterior regions that it explores. This suggests that the bias introduced though the Noisy Monte Carlo [1] approximation should not be too large.

Further work is needed to formalize this intuition. In particular, additional results on Noisy Monte Carlo would grant additional understanding of how such approximations can affect the results of the algorithm. See Section 7 for additional discussion.

To conclude, we provide the following heuristic for describing problems in which the approximate algorithm is appropriate.

Heuristic B.
Asynchronous Gibbs without Metropolis-Hastings correction produces a good approximation if all of the following hold:

(i) The target density π does not possess too much dependence between all coordinates.

(ii) The dimensionality of π is significantly larger than the number of workers.

(iii) All transmitted variables are drawn via Gibbs steps.

We propose this Heuristic for the following reasons: (i) suggests that full conditional distributions in nearby posterior regions are similar, (ii) suggests that there is not too much movement happening at once, and (iii) suggests, given the previous two conditions, that the algorithm will consist of moves that are approximately Gibbs steps and hence should be accepted often.

Both Diagnostic A and Heuristic B are intuitive tools designed to help practitioners use approximate Asynchronous Gibbs in situations where it is likely to work well. We cannot at this time prove any theorems formalizing the intuition that we have provided. Future work is necessary to understand the behavior of the algorithm.

7 Concluding Remarks

Approximate Asynchronous Gibbs sampling appears to offer a way forward for Bayesian inference at scale in a large class of models (as noted previously, the exact algorithm will not scale as well as the approximate algorithm, because the number of Metropolis-Hastings evaluations that need to be performed will increase with the number of workers). In particular, models with a structure similar to the one found in the hierarchical mixed-effects regression example of Section 6.3 – in which each data point has its own latent variable – appear especially promising, because the dependence between almost all dimensions, for instance two vectors $\boldsymbol{\beta}_i, \boldsymbol{\beta}_j$ for $i \neq j$, in the posterior is weak. Models with strong posterior dependence will likely remain difficult, because we expect poor mixing in that context. One way around this would be to tailor blocking of the Gibbs sampler to the specific problem at hand. For example, the performance of the Gaussian Process regression in Section 6.2 could be further improved by considering an overlapping block scheme, such as the *Additive Schwartz* method [43].

Other models, such as those involving high-dimensional (for example, of order 10,000 or more) Inverse Wishart priors, remain intractable because they unavoidably require the construction of matrices that are too big to invert quickly, or perhaps even too big to fit into a single computer's memory. Similarly, models that depend strongly on each individual data point will not perform well in this context, as some amount of data-level redundancy is needed to make the scheme robust against temporary worker failures. We conjecture that the solution to these problems will not arise wholly from within the realm of statistical computing, but will also rely on new insight into Bayesian modeling of large data sets.

Finally, models involving excessive *Reduce* operations over large portions of the data also remain intractable, because these computations massively slow down the speed at which individual steps of Asynchronous Gibbs proceed – this difficulty can be mitigated through caching, but this may result in unacceptably slow mixing.

The theory of Asynchronous Gibbs sampling can be further expanded. While we focused on convergence, it would be useful to quantify the degree to which a lack of synchronicity affects the performance of the algorithm. As can be seen in the partial autocorrelation plots in Section 6, Asynchronous Gibbs produces a higher order Markov chain, the direct analysis of which is non-trivial. However, we believe that a finer understanding of the interplay between the convergence behavior of the chain and Markov chain's order will be a useful step toward developing "partially asynchronous" MCMC methods, which may mix better or

possess other useful properties, and could potentially use asynchronous steps to hide latency during the global *Reduce* operations required for synchronization. This would mirror recent advances in massively parallel iterative algorithms for solving linear systems [18].

Further research in Noisy Monte Carlo [1] is needed to understand the approximate algorithm. By replacing the Metropolis-Hastings ratio with a biased estimator (namely 1), we are able to get significantly better scaling, but this introduces some bias into our answer. A variety of results would help in understanding this bias. It's also unclear whether properties such as geometric ergodicity are inherited by the synchronous parallel chain H from the underlying chain P. Further work is needed to understand the approximate algorithm from these perspectives.

Asynchronous Gibbs sampling is nontrivial to implement. It is a fully parallel method that in our implementation mixes thread-based and actor-based parallelism, which is widely considered in the parallel computation literature to be a very bad idea due to the coding complexity involved, including increased potential for *race conditions* and other nondeterministic errors that are difficult to debug. All of the data structures used need to be capable of functioning correctly in parallel. We highly recommend *Scala* as the programming language of choice, because its design makes it much more compatible with parallel programming than most other languages. Implementation is also specific to both the problem being solved and the hardware used – in particular, it's necessary to decide how to divide the workload among all of the workers. We found that different choices produced widely different mixing efficiencies: in the extreme, one worker can bottleneck the entire algorithm if it's sampling, at too slow a rate, a dimension upon which all other workers depend. Similar issues can occur with respect to network traffic control: if one worker is producing output too fast, it can flood the network, preventing other workers from communicating with each other. This is not solely an issue in complex problems – at one point in time (due to an incorrect *Akka* configuration) this difficulty manifested itself in the Toy Example: because the variables can be sampled quickly, a large number of them were produced, resulting in significant network traffic in spite of the problem's small size. Thus care was required to properly tune the algorithm and ensure that it converged in the problems that we studied.

Our current implementation is nowhere near optimal. *Akka* is designed for large-scale distributed applications rather than high-performance computing. This makes for convenient development, but does not give us the kind of low-level hardware control available in a framework like *MPI* [20]. For examine, we cannot easily tell our application's threads to run on CPU 1 while working only on variables stored in a particular memory address range. This may be desirable because CPU 1 may have faster access to one portion of the memory, and slower access to other portions. Our cluster also was selected for convenience rather than performance – indeed, the machines of which it is comprised are physically located in data centers in three different US states. This is an extremely high latency environment from a high-performance computing perspective. While this illustrates our algorithm's robustness, it also means that we do not know how it will perform in the traditional scientific computing environment.

These challenges are common to any parallel computation scheme, where fully generic solutions are difficult. Instead, we have focused on building Asynchronous Gibbs for solving a common class of big-data Bayesian problems, for which the number of latent variables grows with the number of data points. In doing this, we traded off some sequential efficiency for the promise of better parallel behavior, and the initial results are highly promising. Asynchronous computing is almost completely unexplored for Bayesian computation – in our view, Asynchronous Gibbs makes a strong argument that this should no longer be the case.

Acknowledgments

We are grateful to Chris Severs, Joseph Beynon, and Matt Gardner for many useful discussions and much help with the coding. DS (and perhaps to a lesser extent AT and DD) would like to thank Christian Robert for sending him an early version of this paper that allowed him, during an otherwise uneventful trip to buy trousers, to remember a strategy of proof for asynchronous algorithms. We are grateful to Matt Johnson for sending us the *Jacobi Sampling* example and thereby demonstrating that approximate Asynchronous Gibbs is not exact, the subsequent analysis of which made much of this work possible. We thank Christopher De Sa for sending us a different counterexample to exactness of the approximate algorithm on $\{0, 1\}^3$. We are grateful to Cyril Chimsov for pointing out a subtle mathematical error in our original proof. We thank Gareth Roberts, Murray Pollock, Radford Neal, Daniele Venturi, Rajarshi Guhaniyogi, Tatiana Xifara, and Torsten Erhardt for interesting and useful discussions. We are also grateful to eBay, Inc. for providing the computational resources used for running our examples. Membership on this list does not imply agreement with the ideas given here, nor are any of these people responsible for any errors that may be present. The figures in the paper were created in *R* [40], using the *ggplot2* [55] and *magrittr* [2] packages, and in *TikZ* [49] using the *BayesNet* [12] package.

References

[1] P. Alquier, N. Friel, R. Everitt, and A. Boland. Noisy Monte Carlo: Convergence of Markov chains with approximate transition kernels. *Statistics and Computing*, 26(1-2):29–47, 2016 (cit. on pp. 6, 25, 27, 29).

[2] S. M. Bache and H. Wickham. magrittr: A Forward-Pipe Operator for R. 2014. URL: `http://CRAN.R-project.org/package=magrittr` (cit. on p. 30).

[3] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3(1):133–181, 1922 (cit. on p. 12).

[4] A. Banerjee, D. B. Dunson, and S. T. Tokdar. Efficient Gaussian process regression for large datasets. *Biometrika*:75–89, 2012 (cit. on p. 4).

[5] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the Association for Computing Machinery*, 25(2):226–244, 1978 (cit. on pp. 8, 12).

[6] D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983 (cit. on pp. 8, 12).

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, 1989 (cit. on p. 25).

[8] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, New York, 2015 (cit. on p. 16).

[9] R. T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13, 1946 (cit. on p. 1).

[10] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008 (cit. on p. 7).

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*:1–38, 1977 (cit. on p. 12).

[12] L. Dietz. Directed factor graph notation for generative models. 2010 (cit. on p. 30).

[13] M. Dow. Explicit inverses of Toeplitz and associated matrices. *ANZIAM Journal*, 44:185–215, 2008 (cit. on p. 18).

[14] A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1):201–216, 2000 (cit. on p. 12).

[15] R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3), 2006 (cit. on p. 20).

[16] A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990 (cit. on p. 3).

[17] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984 (cit. on p. 3).

[18] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014 (cit. on p. 29).

[19] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley. The Java language specification. 2000. URL: `https://docs.oracle.com/javase/specs/` (cit. on p. 7).

[20] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999 (cit. on p. 29).

[21] D. Hall. Breeze. 2009. URL: `https://github.com/scalanlp/breeze/` (cit. on p. 7).

[22] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970 (cit. on pp. 2, 3).

[23] G. Hu and R. F. O'Connell. Analytical inversion of symmetric tridiagonal matrices. *Journal of Physics A: Mathematical and General*, 29(7):1511–1513, 1996 (cit. on p. 18).

[24] A. Ihler and D. Newman. Understanding errors in approximate distributed latent Dirichlet allocation. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):952–960, 2012 (cit. on p. 4).

[25] International Organization for Standardization. ISO International Standard ISO/IEC 14882:2014(E) – Programming Language C++. 2014. URL: `https://isocpp.org/std/the-standard` (cit. on p. 7).

[26] M. Johnson. Jacobi sampling. Personal Communication. 2015 (cit. on p. 25).

[27] M. Johnson, J. Saunderson, and A. Willsky. Analyzing Hogwild parallel Gaussian Gibbs sampling. In *Advances in Neural Information Processing Systems*, 2013, pp. 2715–2723 (cit. on p. 4).

[28] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999 (cit. on p. 3).

[29] A. Kottas. Bayesian inference for Gaussian process nonparametric regression. Personal Communication. 2014 (cit. on p. 17).

[30] J. S. Liu, W. H. Wong, and A. Kong. Covariance structure and convergence rate of the Gibbs sampler with various scans. *Journal of the Royal Statistical Society, Series B*:157–169, 1995 (cit. on p. 3).

[31] M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998 (cit. on p. 7).

[32] N. Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949 (cit. on p. 3).

[33] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953 (cit. on pp. 2, 3, 5).

[34] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993 (cit. on p. 13).

[35] R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. *arXiv:physics/9701026*, 1997 (cit. on p. 17).

[36] R. M. Neal. Slice sampling. *Annals of Statistics*, 31(3):705–741, 2003 (cit. on p. 5).

[37] F. Niu, B. Recht, C. Re, and S. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2011, pp. 693–701 (cit. on pp. 12, 24).

[38] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger. The Scala language specification. 2004. URL: `http://www.scala-lang.org/` (cit. on p. 7).

[39] D. V. Ouellette. Schur complements and statistics. *Linear Algebra and its Applications*, 36:187–295, 1981 (cit. on p. 20).

[40] R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing, 2015. URL: `https://www.R-project.org/` (cit. on p. 30).

[41] G. O. Roberts and J. S. Rosenthal. Surprising convergence properties of some simple Gibbs samplers under various scans. 2015 (cit. on p. 3).

[42] G. O. Roberts and R. L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*:341–363, 1996 (cit. on p. 5).

[43] Y. Saad. *Iterative methods for sparse linear systems*. PWS Publishing Company, 1996 (cit. on p. 28).

[44] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2nd ed., 2003 (cit. on p. 25).

[45] S. L. Scott, A. W. Blocker, F. V. Bonassi, H. A. Chipman, E. I. George, and R. E. McCulloch. Bayes and Big Data: The Consensus Monte Carlo algorithm. In *Bayes 250*. Vol. 16, 2013 (cit. on p. 4).

[46] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10 (cit. on p. 7).

[47] M. L. Stein, Z. Chi, and L. J. Welty. Approximating likelihoods for large spatial data sets. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 66(2):275–296, 2004 (cit. on p. 20).

[48] R. H. Swendsen and J.-S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607–2609, 1986 (cit. on p. 8).

[49] T. Tantau. The TikZ and PGF Packages. Dec. 2013. URL: `http://sourceforge.net/projects/pgf/` (cit. on p. 30).

[50] S. Tavaré, D. J. Balding, R. C. Griffiths, and P. Donnelly. Inferring coalescence times from DNA sequence data. *Genetics*, 145(2):505–518, 1997 (cit. on p. 4).

[51] A. Terenin and D. Draper. Rigorizing and Extending the Cox-Jaynes Derivation of Probability: Implications for Statistical Practice. *arXiv:1507.06597*, 2015. URL: `http://arxiv.org/abs/1507.06597` (cit. on p. 1).

[52] Typesafe Inc. Akka Scala Documentation. 2015. URL: `http://akka.io/` (cit. on p. 7).

[53] W. N. Venables and B. D. Ripley. *Modern applied statistics with S*. Springer, 2002 (cit. on p. 16).

[54] V. v. Brzeski, M. Taddy, and D. Draper. Causal Inference in Repeated Observational Studies: A Case Study of eBay Product Releases. *arXiv:1509.03940*, 2015 (cit. on p. 20).

[55] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer, New York, 2009. URL: `http://had.co.nz/ggplot2/book` (cit. on p. 30).