Annual Review of Statistics and Its Application

Simulation-Based Bayesian Analysis

Martyn Plummer

Department of Statistics, University of Warwick, Coventry CV4 7AL, United Kingdom; email: martyn.plummer@warwick.ac.uk

Annu. Rev. Stat. Appl. 2023. 10:401–25

First published as a Review in Advance on November 18, 2022

The *Annual Review of Statistics and Its Application* is online at statistics.annualreviews.org

https://doi.org/10.1146/annurev-statistics-122121-040905

Keywords

Bayesian computation, Bayesian inference, Gibbs sampling, graphical model, statistical software, MCMC, INLA, BUGS, Stan, JAGS, PDMPs

Abstract

I consider the development of Markov chain Monte Carlo (MCMC) methods, from late-1980s Gibbs sampling to present-day gradient-based methods and piecewise-deterministic Markov processes. In parallel, I show how these ideas have been implemented in successive generations of statistical software for Bayesian inference. These software packages have been instrumental in popularizing applied Bayesian modeling across a wide variety of scientific domains. They provide an invaluable service to applied statisticians in hiding the complexities of MCMC from the user while providing a convenient modeling language and tools to summarize the output from a Bayesian model. As research into new MCMC methods remains very active, it is likely that future generations of software will incorporate new methods to improve the user experience.

1. INTRODUCTION

Over the last 30 years, Markov chain Monte Carlo (MCMC) methods have expanded the scope of applied Bayesian statistics across diverse scientific fields. The growth in popularity of MCMC has been accelerated by the wide availability of free software. Essential features of this software include:

1. Providing a flexible and expressive modeling language that allows users to build large complex models from simple components. This capability represented a radically different approach from traditional statistical software, which provided a set of canned routines for fixed classes of models. With a flexible Bayesian modeling language, the user can easily extend existing models to account for issues such as measurement error, missing data, censoring and truncation, or hierarchical structure (Gilks et al. 1993).
2. Hiding the implementation details of MCMC from the user. Several competing sampling methods to analyze a given model may be available. The software takes care of the choice of sampling method. In addition, most sampling methods have parameters that need to be tuned for optimal performance. The software also takes care of adaptively tuning the sampling method. Ultimately, the user does not need to care about how the samples are drawn as long as the sampling is efficient. From the point of view of statistical methodology, the software packages are "black boxes," or, perhaps more accurately, "gray boxes," since they may allow partial control over the sampling methods.
3. Communicating with other software. Bayesian software is typically designed to do one thing only—efficient sampling—allowing the complexities of data formatting and analyzing output to be carried out in a more suitable environment such as R or Python.

Examples of Bayesian software with these features include BUGS (Lunn et al. 2012), JAGS (Plummer 2017), NIMBLE (de Valpine et al. 2022), and Stan (Stan Dev. Team 2022). The same description may be used for other packages that do not use MCMC, such as R-INLA (Iterated Nested Laplace Approximation) (Rue et al. 2009).

In this article I review the historical development of software for Bayesian inference. The article focuses on MCMC, which remains the dominant methodology in this area, but also touches on alternatives such as INLA. The review tracks the way that advances in software have been driven by advances in statistical methodology. The main thesis of the review is that the success of general-purpose Bayesian statistical software relies on two important pillars: (*a*) the existence of statistical methods that can be easily tuned for optimal performance and so are widely applicable and (*b*) the availability of reusable software components.

2. MARKOV CHAIN MONTE CARLO

The 1980s saw the convergence of three powerful ideas: graphical models, Gibbs sampling, and Bayesian inference. At the same time, exponential growth in computing power reached a critical point that enabled these ideas to be translated from theory to practice. In this section I review some essential information on MCMC methods, covering only the material necessary to understand the rest of the article. A more comprehensive overview is presented by Brooks et al. (2011).

Bayesian computations involve high-dimensional integration to derive the posterior distribution. Analytic solutions to these integrals are rare and are generally limited to conjugate prior distributions, that is, where the prior and the posterior come from the same family. Conjugate priors are useful only in the simplest models. Numerical approximations such as Gaussian quadrature do not scale well to high dimensions. This situation changed with the rediscovery of MCMC methods in the 1980s (Gelfand & Smith 1990). MCMC was first developed in the 1950s to solve problems in statistical physics (Metropolis et al. 1953). The wide applicability of MCMC to statistical models was not fully appreciated until the rapid development of desktop computing made the methods computationally feasible and accessible. With MCMC, we relax the search for an exact solution to the posterior distribution, instead relying on the empirical distribution of a sequence of samples. Furthermore, we do not require these samples to be independent. Instead, we create a Markov chain that has the target distribution as its stationary distribution. Under suitable regularity conditions, the Markov chain will converge to the target distribution, allowing inference from posterior samples after a suitable "burn-in" period.

To make matters more concrete, suppose we wish to estimate the distribution $p(\boldsymbol{\theta})$ for $\boldsymbol{\theta} \in \mathbb{R}^d$. In Bayesian applications, the distribution that we wish to estimate is the posterior distribution $p(\theta \,|\, D)$ for some data D, but for notational convenience we suppress the dependence on D. The aim of Monte Carlo inference is to conduct inference on $p(\boldsymbol{\theta})$ by drawing a sequence of samples $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \dots \boldsymbol{\theta}^{(T)}$. Then, for a function $g(\boldsymbol{\theta})$ with finite expectation under $p(\boldsymbol{\theta})$, the expectation is estimated by the sample mean

$$\overline{g}^{(T)} = \left(\frac{1}{T} \sum_{t=1}^{T} g(\boldsymbol{\theta}^{(t)}) \right). \qquad 1.$$

In MCMC, the sequence $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \dots \boldsymbol{\theta}^{(T)}$ is a sequence of dependent samples from a Markov chain, with $p(\boldsymbol{\theta})$ as its stationary distribution. Under mild regularity conditions, the Markov chain is ergodic so that $\overline{g}^{(T)}$ is a consistent estimator of $\mathrm{E}(g)$:

$$\lim_{T \to \infty} \overline{g}^{(T)} = \mathrm{E}\left[g(\boldsymbol{\theta}) \right]. \qquad 2.$$

In practice, convergence may be extremely slow, so that the sample mean in Equation 1 may be a poor estimator even with a large number of iterations T. Ideally, the Markov chain should be geometrically ergodic. Then, the Wasserstein distance between the distribution of $\boldsymbol{\theta}^{(t)}$ and the target $p(\boldsymbol{\theta})$ diminishes as $t \to \infty$ at a geometric rate. When the chain is geometrically ergodic, a central limit theorem applies to the sample mean in Equation 1, which has an asymptotic normal distribution as $T \to \infty$.

The particular form of MCMC developed in the 1980s is named Gibbs sampling (Geman & Geman 1984), in honor of Josiah Willard Gibbs (1839–1903). Suppose we have d random variables $\boldsymbol{\theta} = (\theta_1 \dots \theta_d)$ with joint distribution $p(\boldsymbol{\theta})$. At each iteration of the Gibbs sampler, we visit each variable in turn and replace its current value θ_i with a new value θ_i' for i in $1 \dots d$. The new value is a sample from the full conditional distribution $p(\theta_i \mid \boldsymbol{\theta}_{-i})$, where $\boldsymbol{\theta}_{-i} = (\theta_1' \dots \theta_{i-1}', \theta_{i+1} \dots \theta_d)$ is the state of the random variables, except for θ_i, before we draw the new sample.

The worst-case computational complexity of the Gibbs sampler is $\mathcal{O}(d^2)$. If the full conditional density $p(\theta_i \mid \boldsymbol{\theta}_{-i})$ depends on all $d-1$ elements of $\boldsymbol{\theta}_{-1}$, then the computational cost of sampling the new value θ_i' is $\mathcal{O}(d)$ as d increases. At each iteration we draw d samples, once each for $i = 1, \dots d$, which is also $\mathcal{O}(d)$, giving an overall complexity of $\mathcal{O}(d^2)$. An algorithm of this computational complexity would be impractical for very large models. However, in practice many statistical models have a sparse representation in terms of a conditional independence graph (CIG), in which variables are represented by nodes. The absence of an edge between two nodes in a CIG indicates that the two corresponding variables are independent of each other, conditional on the values of all other nodes in the graph.

In a CIG, the full conditional distribution $p(\theta_i \mid \boldsymbol{\theta}_{-i})$ of variable θ_i depends only on its neighbors in the graph. An archetypal CIG is a lattice, illustrated in **Figure 1**. In this example, the

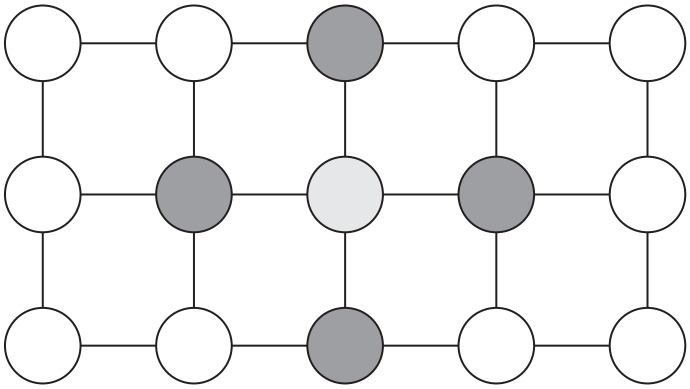

Figure 1

Lattice illustrating conditional independence in a graph. The light gray node is conditionally independent of the white nodes given its dark gray neighbors. The remaining nodes, in white, do not contribute to its full conditional distribution.

highlighted node depends only on its four neighbors. The remaining nodes do not contribute to its full conditional distribution. The same argument applies to all nodes. As a result, sampling a new value for a given node takes constant time, regardless of the size of the lattice. Therefore, the computational complexity of the Gibbs sampler scales linearly with the size of the graph.

Gibbs sampling on a graph fits perfectly with Bayesian inference. If a node on a graph represents an observed quantity, then the Gibbs sampler skips that node, leaving it at its observed value. The stationary distribution of the resulting Markov chain is the conditional distribution of the unobserved nodes given the observed nodes, in other words, the posterior distribution that is the target of Bayesian inference.

Sampling from the full conditional distribution $p(\theta_i \mid \boldsymbol{\theta}_{-i})$ is not always feasible. However, it is not necessary to draw an independent sample from the full conditional. It is sufficient to make a transition $\theta_i \rightarrow \theta_i'$ that is reversible. Let $p_t\left(\theta_i \rightarrow \theta_i'\right)$ be the probability of a transition from the current value θ_i to the new value θ_i'. Reversibility is defined by the detailed balance condition:

$$p\left(\theta_i \mid \boldsymbol{\theta}_{-i}\right) p_t\left(\theta_i \rightarrow \theta_i'\right) = p\left(\theta_i' \mid \boldsymbol{\theta}_{-i}\right) p_t\left(\theta_i' \rightarrow \theta_i\right). \qquad 3.$$

There are two qualitatively different probability terms in Equation 3. The conditional probability $p\left(\theta_i \mid \boldsymbol{\theta}_{-i}\right)$ is derived from the joint distribution $p(\boldsymbol{\theta})$, which is defined by the model. The transition probability $p_t\left(\theta_i \rightarrow \theta_i'\right)$ is determined by the transition kernel of the Markov chain and is completely independent of $p(\boldsymbol{\theta})$. Many choices of $p_t\left(\theta_i \rightarrow \theta_i'\right)$ exist for a given model. We emphasize the distinction between the two probabilities with the subscript t for the transition probabilities. The goal of statistical software for MCMC is to determine appropriate transition probabilities $p_t\left(\theta_i \rightarrow \theta_i'\right)$ without any user input and, in fact, to hide these probabilities from the user.

Under broad conditions, any transition kernel can be made reversible by the addition of a Metropolis–Hastings acceptance step (Metropolis et al. 1953, Hastings 1970). We can rewrite the detailed balance condition in Equation 3 as $R = 1$, where

$$R = \frac{p\left(\theta_i' \mid \boldsymbol{\theta}_{-i}\right)}{p\left(\theta_i \mid \boldsymbol{\theta}_{-i}'\right)} \times \frac{p_t\left(\theta_i \rightarrow \theta_i'\right)}{p_t\left(\theta_i' \rightarrow \theta_i\right)} \qquad 4.$$

is the Metropolis–Hastings acceptance ratio. The move $\theta_i \rightarrow \theta_i'$ generated by the transition kernel is considered a proposal, which is accepted with probability $\min(1, R)$. If the proposal is rejected, we remain at the current value θ_i. The addition of the acceptance step makes the transition reversible. The modified algorithm is known as Metropolis within Gibbs.

2.1. Markov Chain Monte Carlo in Practice

While Markov chain theory guarantees that the chain will converge to the target distribution, it is relatively silent on how long convergence will take. Theoretical research on convergence bounds did not lead to practically usable criteria. Therefore, the determination of MCMC convergence remains largely empirical.

In practice, it is customary to discard the initial portion of the Markov chain. This process is referred to as burn-in. The burn-in period may coincide with the samplers adapting their behavior for optimal efficiency (Andrieu & Thoms 2008). After burn-in, samples of the parameters of interest can be monitored. Visual inspection of the monitored values normally uses trace plots, which show how each scalar parameter changes with time. When trace plots show high autocorrelation, the chain is said to display poor mixing. In this case it is customary to thin the chain by monitoring only every m iterations for $m \gg 1$. The discarded samples do not contribute any additional information about the target distribution when the autocorrelation is high.

It is also customary to run several parallel chains, where the initial values and the transitions are completely independent of one another. The use of the term "parallel chain" predates the current multicore era of computing and simply refers to the realizations of the Markov chain being statistically independent. However, in practice most software will run parallel chains on different cores.

It remains to make a determination of whether the chain has converged sufficiently for the monitored values to be considered an approximate sample from the posterior distribution. After some vigorous activity on convergence diagnostics in the early 1990s (reviewed in Cowles & Carlin 1996), consensus has largely settled on the Gelman–Rubin diagnostic (Gelman & Rubin 1992), which is based on starting parallel Markov chains from widely dispersed starting points. The degree of convergence can then be assessed by comparing within-chain and between-chain variation, with a summary statistic $\widehat{R}$ that tends to one as the Markov chains converge. The Gelman–Rubin diagnostic has evolved over the years, notably with the introduction of the split $\widehat{R}$ that separates the beginning and the end of each chain, and it continues to be refined (Gelman et al. 2014, Vats & Knudson 2021, Vehtari et al. 2021).

3. BUGS

BUGS (Bayesian inference Using Gibbs Sampling) is a long-running project to develop Bayesian modeling software. The BUGS project began in 1989 at the Medical Research Council (MRC) Biostatistics Unit in Cambridge, United Kingdom.

The BUGS developers maintain that the BUGS project was independent of Gelfland & Smith's (1990) paper, which appeared around the time of the launch of the BUGS project. It was in fact inspired by earlier developments in artificial intelligence. In the 1980s it was understood that expert systems could represent qualitative knowledge in the form of a directed acyclic graph (DAG), or influence diagram, and that uncertainty could be represented by a probability distribution on the graph (Lauritzen et al. 1990). Furthermore, efficient algorithms for updating the probability in the presence of new data were developed (Lauritzen & Spiegelhalter 1988). These ideas, together with an understanding of the value of object-oriented programming, formed the conceptual foundation of the BUGS project.

The BUGS software has evolved through four different versions, which, confusingly, have different names. These versions are described below, but first I describe the key conceptual contribution of the BUGS project to Bayesian modeling, which is the BUGS language.

3.1. The BUGS Language and Directed Acyclic Graphs

Central to the success of BUGS has been the BUGS language (Thomas 2006) for specifying graphical models. The BUGS language has been adopted by other projects, such as JAGS and NIMBLE, so it is important to distinguish the language from the software.

The BUGS language is syntactically similar to S, or R, but is declarative, not procedural. Statements in BUGS define the static relationships between variables in the model, not instructions on how to carry out a calculation.

I illustrate the BUGS language with the Eight Schools example, which was originally developed by Rubin (1981) and has since been popularized by Gelman et al. (2014, section 5.5) as a teaching example. The data come from an experiment to see how short-term coaching could improve scores on a standardized school test. The coaching was applied across $J = 8$ schools with varying results. The estimates of the coaching effect $y_1 \ldots y_J$ from the eight schools have known standard errors $\sigma_1 \ldots \sigma_J$, which vary across schools because varying numbers of students take the training in each school. The aim is to estimate the average effect, accounting for the heterogeneous information coming from the participating schools.

The model used to analyze the Eight Schools data is an example of a normal-normal hierarchical model,

$$\begin{aligned} y_j \mid \theta_j &\sim N(\theta_j, \sigma_j^2), \\ \theta_j \mid \mu, \tau &\sim N(\mu, \tau^2), \end{aligned} \tag{5.}$$

where θ_j is a random effect representing the effect of training in school j. The same model has an alternative noncentered parameterization, which is used as the basis of the BUGS model:

$$\begin{aligned} y_j \mid \theta_j &\sim N(\theta_j, \sigma_j^2), \\ \theta_j &= \mu + \tau\eta_j, \\ \eta_j &\sim N(0, 1). \end{aligned} \tag{6.}$$

Listing 1 shows the BUGS code for the Eight Schools example. The code is an almost exact translation of the mathematical description of the noncentered parameterization in Equation 6, except that we must also supply proper prior distributions for `mu` and `tau`.

Listing 1 BUGS code for the Eight Schools example

```
model {
  for (j in 1:J) {
    y[j]~dnorm(theta[j], precision.y[j])
    theta[j] <- mu + tau * eta[j]
    eta[j] ~ dnorm(0, 1)
    precision.y[j] <- pow(sigma[j], -2)
  }
  mu ~ dnorm(0.0, 1.0E-6)
  tau ~ dunif(0, 1000)
}
```

The BUGS language uses an S-like syntax to define the model in terms of a series of relations. Relations may be of two types. A stochastic relation, denoted by a tilde (~), defines a random variable on the left-hand side that is parameterized in terms of the quantities on the right. A logical relation, denoted by a left arrow (`<-`), defines a quantity on the left-hand side that is a deterministic function of the quantities on the right.

Listing 1 shows that relations indexed by j appear inside a for loop. Despite appearances, the for loop is not a control flow statement but may be considered a macro that expands the contents inside the curly braces J times, replacing index j with every integer value from 1 to J.

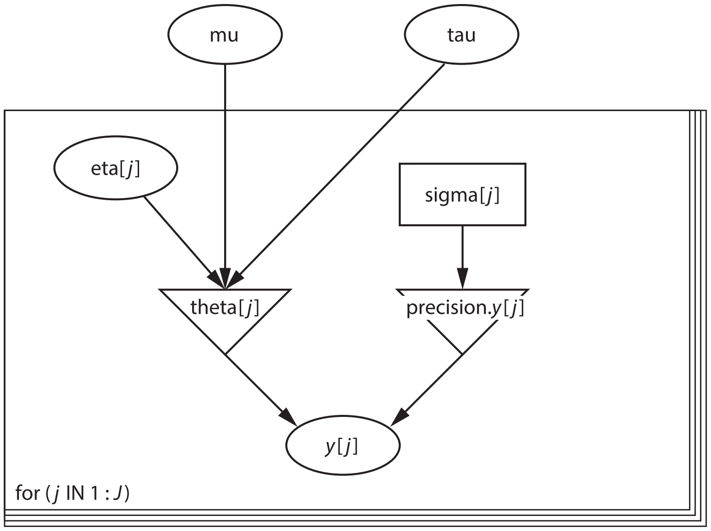

Figure 2

Directed acyclic graph for the Eight Schools model. The graph was created with the WinBUGS Doodle tool, which allows users to interactively create a graphical model.

The normal distribution in the BUGS language (`dnorm`) is parameterized in terms of the mean and the precision, which is the reciprocal of the variance. This was an early language design decision that simplified the use of conjugate prior distributions (a gamma prior on the precision of a normal random variable is conjugate). If the user wishes to define a weakly informative prior for a normal random variable, then it should have a low precision, as observed in the small value of 10^{-6} assigned to the precision parameter of `mu`. For the outcome variable `y[j]`, the standard deviation parameter `sigma[j]` is transformed into a precision parameter `precision.y[j]`, which is then passed as the second argument of the `dnorm` distribution.

The BUGS language is largely agnostic about what is data and what is a parameter. There are some restrictions, however. Any quantity that does not appear on the left-hand side of a relation must be supplied as data. In the Eight Schools example (Listing 1), these quantities are `J`, used as the limit of the for loop, and `sigma[1]`...`sigma[J]`, used to define the precision of `y[1]`...`y[J]`. Apart from this restriction, the BUGS language makes no distinction between data and parameters. This distinction is made when the data are supplied to the model. For the Eight Schools example, we may choose not to supply data values for `y[1:J]`, and in this case the parameters are sampled from their prior distribution. If we supply data values for `y[1:J]`, then BUGS will choose appropriate samplers for the unobserved stochastic variables and sample them from their posterior distribution given `y[1:J]`.

The BUGS language description defines a DAG, as shown in **Figure 2**, with directed edges leading from quantities on the right-hand side of a relation to the quantity on the left. The BUGS program takes the description of the model and translates it into a virtual graphical model (VGM) in computer memory. The joint prior distribution of the variables in the model factorizes as

$$\prod_{i=1}^{d} p\,(v_i \mid \text{Parents}(v_i)), \qquad 7.$$

where Parents(v_i) denotes the set of parents of node v_i.

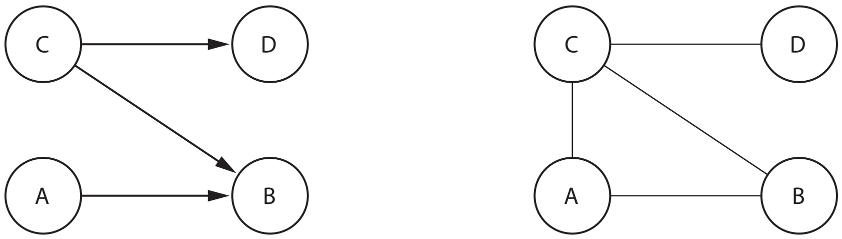

Figure 3

Deriving a conditional independence graph (*right*) from a directed acyclic graph (*left*). Nodes A and C are "married" in the conditional independence graph because they have a common child node, B.

In order to determine the conditional independence relationships for Gibbs sampling, the DAG must be translated into a CIG. This process involves three steps. First, we marginalize out any deterministic nodes so that only stochastic nodes remain. Second, we draw an undirected edge between two nodes with a common stochastic child. Third, we replace all directed edges in the graph with undirected edges. This step is known as moralizing the graph because it marries parents. This process is illustrated for a simple graph in **Figure 3**.

3.2. Classic BUGS

The first version of the BUGS program, retrospectively named classic BUGS, was written in Modula-2 and ran on MS-DOS and Unix (Gilks et al. 1994, Spiegelhalter et al. 1996a). It included a scripting language for running a model as well as facilities for reading and writing data files. Input data could be prepared in S-PLUS, the commercially available distribution of S, and then written to file using the `dump()` function. Conversely, output files from BUGS could be read into S-PLUS and analyzed with a suite of functions called CODA (Convergence Diagnostics and Output Analysis), which was later developed into an R package (Plummer et al. 2006).

The workhorse algorithm for classic BUGS was adaptive rejection sampling (ARS) (Gilks 1992, Gilks & Wild 1992). ARS draws a sample from a univariate distribution with a log concave density. It works by first constructing a piecewise-linear hull around the log density that is easy to sample from. Then, rejection sampling is used to determine whether to accept the sampled value on the basis of how closely the hull approximates the log density. As the name suggests, the algorithm is adaptive: Rejected sampling points are used to improve the accuracy of the hull approximation so that the probability of a successful draw increases with every rejection.

An expert system built into classic BUGS recognized situations where the full conditional distribution of a node was log concave and applied the ARS algorithm. This ability to identify and sample log-concave distributions enabled Gibbs sampling on a graph without the user being restricted to conjugate priors, the strong condition that had previously restricted the application of Bayesian models. The algorithm was later generalized to adaptive rejection Metropolis sampling (ARMS), which added a Metropolis–Hastings acceptance step when the full conditional is not log concave, although ARMS is efficient only in situations when the log density is close to being log concave (Gilks et al. 1995).

In addition to providing the software, the BUGS authors provided a rich set of examples from the contemporaneous literature illustrating the use of BUGS (Spiegelhalter et al. 1996b,c). Many of these examples were taken from Breslow & Clayton (1993), who assembled a set of examples to illustrate approximate inference for generalized linear mixed models (GLMMs). These worked examples provided templates for new users to get started with modeling in BUGS by adapting an existing example to their own problems.

3.3. WinBUGS

In 1996, the BUGS project moved to Imperial College London, where the second generation of BUGS was developed. WinBUGS (Lunn et al. 2000) was written in Component Pascal and built with the Black Box Component Builder, originally a commercial product from Oberon Microsystems that is now available as open source software (see **https://blackboxframework.org**). As its name suggests, WinBUGS was designed to run on Windows, and it had a rich graphical user interface (GUI). Initially, the user was required to use the GUI to run the model, but that changed in later versions, when a scripting language was introduced. An interface to the R language was developed in the form of the R2WinBUGS package (Sturtz et al. 2005).

A distinguishing feature of the Black Box Component Builder is its focus on compound documents, which may contain a richer set of entities than text alone. This feature was used in the WinBUGS manual (Spiegelhalter et al. 2003), where, for each example, a single compound document contains the model description, the data, initial values, and a human-readable rich-text description. This might be considered an early example of reproducible research. However, Black Box compound documents did not gain widespread use in statistics outside of WinBUGS.

The modular structure of WinBUGS allowed a richer set of samplers than the classic BUGS software and led to the development of several extensions for specialized tasks:

- The WinBUGS Differential Interface for models defined by differential equations.
- The WinBUGS Developer Interface, which enabled users to define their own functions and hard-code them into WinBUGS (Lunn 2003, Wetzels et al. 2010).
- PKBugs (Lunn et al. 1999), an interface for defining Bayesian pharmacokinetic/pharmacodynamic models (Wakefield et al. 1999).
- The WinBUGS Jump Interface (Lunn et al. 2009) for variable selection models using reversible jump MCMC (Green 1995).

Some of these extensions improve on the flexibility of the BUGS language by allowing users to define special classes of models more easily while retaining the flexibility of WinBUGS. Some extensions provide specialized sampling routines to allow for more efficient sampling.

The final version of WinBUGS was 1.4.3. It is still available at the MRC Biostatistics Unit website but is no longer actively developed.

3.4. OpenBUGS

OpenBUGS (see **https://www.openbugs.net**) was the third incarnation of the BUGS software. It began in 2004, when lead developer Andrew Thomas moved to Helsinki. OpenBUGS was the first open source version of BUGS and was released under the GNU General Public License, version 2. OpenBUGS is not limited to the Windows operating system; it also runs on Linux using a command line interface rather than a GUI. An interface to R is provided by the BRugs package (Thomas et al. 2006).

OpenBUGS was developed in parallel with WinBUGS, with the rationale that WinBUGS was the stable release version while OpenBUGS was the experimental development version subject to incompatible changes from one release to another. OpenBUGS was considered stable enough to be a reliable replacement for WinBUGS from version 3.0.7. OpenBUGS 3.2.3 was the final release of OpenBUGS.

The GeoBUGS extension to OpenBUGS (Thomas et al. 2004) includes some features specifically for geospatial models. However, the WinBUGS extensions described in the previous section were not ported over to OpenBUGS.

3.5. MultiBUGS

The current version of BUGS is MultiBUGS (Goudie et al. 2020), which was designed specifically for a multicore computing environment. Previous generations of software could count on Moore's law, which posited a doubling of the number of transistors every 18 months, thereby guaranteeing increased performance on new hardware without fundamental changes to the software. In the mid-2000s, Moore's law was opposed by the limitations of clock speeds and power consumption, so hardware design switched to providing multicore processors. These offer improved performance only if the software is designed for a multicore environment.

Gibbs sampling can be parallelized in three ways. The first way is to run parallel chains on different cores. The second way involves parallelizing the log density calculations that occur at each step of the MCMC algorithm. Many MCMC algorithms require the calculation of the log density of the variables being sampled. This density is typically a sum of separate terms, and when the number of terms is large, there may be a speed benefit in doing these calculations in parallel. The third way to parallelize MCMC calculations is based on parallel updating of different nodes in the same model. This can be done if the CIG can be partitioned into sets of nodes $\{S_1 \ldots S_K\}$ such that no nodes in S_k have an edge between them for $k = 1 \ldots K$.

Partitioning the CIG in this way is an example of a vertex coloring problem. **Figure 4** illustrates a coloring of the lattice from **Figure 1**. Only $K = 2$ colors are required to separate nodes that are conditionally independent of one another. The Gibbs sampling algorithm can alternate between updating nodes of the first color in parallel and nodes of the other color in parallel.

The regular structure of the lattice in **Figure 4** makes the coloring problem easy. In general, vertex coloring is computationally hard. For $K > 2$ it is NP-complete to determine whether a graph admits a K-coloring (Garey et al. 1976). However, a CIG in a BUGS model is derived from a DAG with a rich underlying structure that can be used to simplify the problem. Goudie et al. (2020) note that each node in the DAG has a depth defined by the shortest distance from a root node (one with no parents). Nodes with the same depth can be updated in parallel if they have no common child nodes.

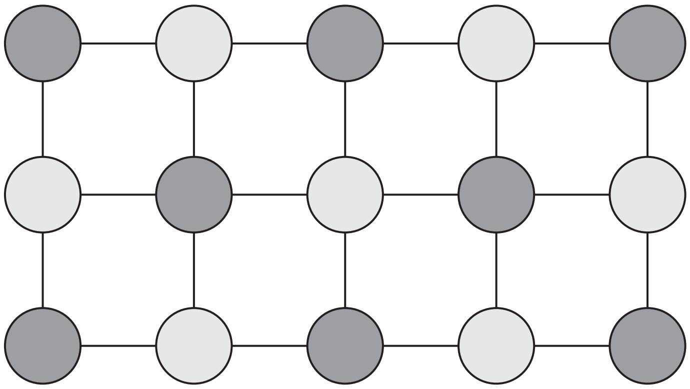

Figure 4

The lattice admits a 2-coloring, such that all dark gray nodes are conditionally independent of one another and can be updated in parallel. Likewise, all the light gray nodes can be updated in parallel.

In a modern high-performance computing environment with many cores, the three forms of parallelization can yield important improvements in run time. Goudie et al. (2020) cite the example of a large GLMM with 425,112 observations and 20,426 random effects. Using 48 cores, MultiBUGS was faster than OpenBUGS by a factor of 64.

4. JAGS

JAGS (Just Another Gibbs Sampler) is a clone of BUGS that has a completely independent code base but aims for similar functionality (Plummer 2017). JAGS is written in C++ and runs on Windows, MacOS, and Linux. It is published under the GNU General Public License, version 2. JAGS would not have been possible without the R math library, which provides high-quality algorithms for random number generation and calculation of quantities associated with probability distributions. At least four interfaces between JAGS and R exist (Su & Yajima 2015, Denwood 2016, Kellner 2021, Plummer 2021).

JAGS has a modular structure. A module may be loaded at run time that adds new functions and distributions to the BUGS language. A module may also define new sampling methods and new monitors, which are ways of summarizing the streaming data generated by the Markov chain. Modules contributed by third parties include the `wiener` module (Wabersich & Vandekerckhove 2014) for inference on Wiener processes and the `pexm` module (Mayrink et al. 2021) for models using piecewise-exponential distributions.

The workhorse sampling method for JAGS is slice sampling (Neal 2003), which can be applied to both continuous- and discrete-valued nodes. The `glm` module, which comes with the JAGS distribution, incorporates efficient samplers for GLMMs. These samplers are based on the principle of data augmentation, a commonly used technique to simplify the structure of a graphical model by adding new nodes (Hobert 2011). In this case, data augmentation reduces GLMMs with binary outcomes (Albert & Chib 1993, Holmes & Held 2006, Polson et al. 2013) or binary and Poisson outcomes (Frühwirth-Schnatter et al. 2009) to a linear model with normal outcomes.

This reduction to a normal linear model allows block updating of all the parameters in the linear predictor, which is much more efficient than Gibbs sampling. The underlying engine for the linear model uses sparse matrix algebra (Davis 2006), which handles fixed and random effects simultaneously.

Block updating solves one of the disadvantages of Gibbs sampling. It is strongly dependent on the parameterization of the model. If two variables have a high posterior correlation but are updated independently using Gibbs sampling, then the Markov chain will exhibit high autocorrelation for both variables. Practitioners can learn to recognize situations where high posterior correlation is likely to occur and reparameterize the model to avoid the problem. A better solution is to do block updating of correlated parameters so that they move together.

5. NIMBLE

NIMBLE (Numerical Inference for statistical Models for Bayesian and Likelihood Estimation) is another engine for Bayesian modeling (de Valpine et al. 2017, 2022). NIMBLE uses a dialect of BUGS language with an extended syntax. One of these extensions is the use of named arguments for distributions. This feature allows distributions to have multiple parameterizations, freeing the user from the historical restriction to parameterizations that were convenient for conjugacy but not convenient for understanding (de Valpine et al. 2022).

The extended BUGS syntax can be illustrated with the Eight Schools example (Listing 2), which can be compared with the BUGS version (Listing 1). In the NIMBLE version, the user can specify directly that σ_i is the prior standard deviation of y_i, removing the need for a transformation.

Likewise, the prior distribution of μ is defined in terms of its large standard deviation rather than its small precision.

Listing 2 NIMBLE BUGS code for the Eight Schools example

```
schoolsCode <- nimbleCode({
  for (j in 1:J) {
    y[j] ~ dnorm(theta[j], sd=sigma[j])
    theta[j] <- mu + tau * eta[j]
    eta[j] ~ dnorm(0, 1)
  }
  mu ~ dnorm(0.0, sd=1000)
  tau ~ dunif(0, 1000)
})
```

NIMBLE is built on top of R, and it brings the flexibility and transparency of the R language to Bayesian modeling. Model objects can be queried and modified, and the choice of samplers can be customized. NIMBLE users can also extend the scope of the BUGS language by writing their own functions and distributions. New samplers can also be defined in a simplified version of the R language.

Running MCMC models in R is too slow to be practically useful. In order to overcome this limitation, NIMBLE has a back end that converts R code to C++ and then compiles it into object code. The details of this compilation process are hidden from the user, who continues to use the R interface and does not need any understanding of C++. The NIMBLE compiler is not tied to the use of Bayesian models but can be used to speed up any numerical code written in R as long as it is compatible with the simplified version of R supported by the compiler. The NIMBLE compiler allows familiar control flow statements from R and supports linear algebra through the Eigen library (Guennebaud et al. 2010).

NIMBLE's modular design creates a cleaner separation between the front end and the back end. Models defined in the BUGS language do not have to be analyzed using MCMC for inference but can in principle be analyzed with any algorithm. Sequential Monte Carlo algorithms (bootstrap particle filter, auxiliary particle filter, ensemble Kalman filter, iterated filter2, and particle MCMC) are provided in the nimbleSMC package (Michaud et al. 2021, NIMBLE Dev. Team 2021).

6. GRADIENT-BASED METHODS

Gradient-based methods for MCMC may be motivated by analogy with optimization problems. Suppose that we are interested only in finding the posterior mode, not the full posterior distribution. In this case, we could use an optimization method to find the maximum value of the log density. This could be done with direct search optimization methods, which rely only on function evaluations. However, it is more efficient to use optimization methods that use the gradient of the log density, such as gradient ascent.

Returning to the problem of finding the full posterior distribution, MCMC methods that rely only on evaluating the log density without using gradient information are inefficient. This inefficiency manifests as random walk behavior, where the Markov chain tends to go back over a previously explored area of the sample space instead of moving to a new one, thus providing no new information about the posterior distribution while using computation time.

Random walk behavior can cause long excursions into the tail of the distribution, where the density is low and the chain is supposed to spend only a small proportion of time. When tail excursions occur, practical remedies such as extending the run time and increasing the thinning interval will not, in general, be effective, as the run time will be too long to be practically useful.

Gradient-based MCMC methods rely on evaluating the gradient of the log density at each step. The implementation of gradient-based methods relies heavily on automatic differentiation (autodiff), a methodology from computer algebra for efficiently calculating the derivatives of functions expressed as computer programs (Baydin et al. 2018). Briefly, autodiff takes a function that evaluates the log density, analyzes the syntax of the function body, and creates a new function that returns the gradient. This process needs to be done only once, and then the gradient function can be called at each iteration. Autodiff works with functions of multiple arguments and can therefore calculate the partial derivatives of the log density with respect to all continuous parameters, allowing block updating of all of these parameters.

6.1. Langevin Diffusions

Langevin diffusions are continuous-time Markov processes based on the Langevin stochastic differential equation,

$$\mathrm{d}\boldsymbol{\theta}_t = \frac{1}{2}\nabla \log p(\boldsymbol{\theta}_t)\mathrm{d}t + \mathrm{d}W_t, \qquad 8.$$

where W_t is a Wiener process or Brownian motion. A Langevin diffusion has $p(\boldsymbol{\theta})$ as its stationary distribution but has a tendency to drift toward areas of higher density. The drift term $\nabla \log p(\boldsymbol{\theta})\mathrm{d}t$ suppresses some, but not all, of the random walk behavior contributed by the Brownian motion.

For MCMC, a discrete-time approximation to the Langevin dynamics is used with a time step of size τ:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \frac{\tau}{2}\nabla \log p\left(\boldsymbol{\theta}^{(t)}\right) + \sqrt{\tau}\boldsymbol{\epsilon}^{(t)},$$
$$\boldsymbol{\epsilon}^{(t)} \sim N(0, I_d).$$

This approximation requires the use of a Metropolis–Hastings acceptance step after each move in order to correct for any errors in the discrete-time approximation. Therefore, the corresponding algorithm is called the Metropolis-adjusted Langevin algorithm (MALA).

MALA has a single parameter, the step size τ, which needs to be tuned for optimal performance. Note that the aim is not to accurately reproduce the Langevin dynamics of Equation 8 but rather to efficiently explore the parameter space while preserving $p(\boldsymbol{\theta})$ as the stationary distribution. Therefore, there is a compromise between large values of τ, which permit larger moves at each time step, and smaller values of τ, which have higher acceptance probabilities. In high dimensions, the optimal acceptance rate is 0.574, and τ should be adaptively tuned to attain this target (Roberts & Rosenthal 1998). MALA has been somewhat superseded by Hamiltonian Monte Carlo, which contains MALA as a special case.

6.2. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC), or hybrid Monte Carlo (Duane et al. 1987, Neal 2011, Betancourt 2017), is based on Hamiltonian dynamics for physical systems. In the statistical implementation of Hamiltonian dynamics, the parameter space $\boldsymbol{\theta} \in \mathbb{R}^d$ is supplemented with a set of momentum parameters $\boldsymbol{\phi} \in \mathbb{R}^d$. The joint coordinates $(\boldsymbol{\theta}, \boldsymbol{\phi})$ define the state of the dynamical system in phase space.

The Hamiltonian

$$H = -\log p(\boldsymbol{\theta}) + \frac{\boldsymbol{\phi}^T M \boldsymbol{\phi}}{2},$$
$$= K(\boldsymbol{\theta}) + V(\boldsymbol{\phi})$$

represents the total energy of the system as the sum of potential energy $K(\boldsymbol{\theta})$ and kinetic energy $V(\boldsymbol{\phi})$. The mass matrix M is usually taken to be the identity matrix I_d.

The evolution of the dynamical system in time is determined by the Hamiltonian equations

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = M\boldsymbol{\phi},$$

$$\frac{\mathrm{d}\boldsymbol{\phi}}{\mathrm{d}t} = -\nabla \log p(\boldsymbol{\theta}),$$

which determine the transfer of energy between potential and kinetic energy while keeping total energy (H) constant. Hamiltonian dynamics are time reversible and preserve volumes in phase space.

Hamiltonian dynamics can be used for MCMC. At the start of each iteration, new momentum variables are drawn at random $\boldsymbol{\phi} \sim N_d(\mathbf{0}, M^{-1})$. Then the process evolves according to Hamiltonian dynamics for a given time T. The transition in the position parameters $\boldsymbol{\theta}$ is a reversible MCMC transition in detailed balance with $p(\boldsymbol{\theta})$. Refreshing the momentum variables at the start of each iteration ensures that the chain is ergodic. Without refreshment, Hamiltonian dynamics is periodic and will follow a closed loop back to its starting point.

As with the Langevin diffusion, Hamiltonian dynamics in continuous time cannot be exactly simulated on a computer. For this reason, a discretized version is used for MCMC, which alternates updates to the position parameters (Equation 9) and the momentum parameters (Equation 10) in discrete time steps of size h:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + h\boldsymbol{\phi}, \quad 9.$$

$$\boldsymbol{\phi}^{(t+1)} = \boldsymbol{\phi}^{(t)} - h\nabla \log p(\boldsymbol{\theta}). \quad 10.$$

Each HMC move runs for L steps, simulating Hamiltonian dynamics for time $T = Lh$. The discretization process introduces errors, of which the most important is that the discretized version of Hamiltonian dynamics does not preserve volumes in phase space. As a consequence, the discretized Hamiltonian dynamics will either collapse to a single point or tend to infinity (Neal 2011). This serious problem can be solved using the Störmer–Verlet or leapfrog integrator, in which each HMC update begins and ends with a half-update of the momentum parameters (Equation 10) with step size $h/2$ instead of h (Neal 2011). The shortest leapfrog path, which contains a single update to the position parameters (Equation 9), is equivalent to MALA when $M = I_d$.

HMC has two parameters: the step size h and the number of steps L. The step size h needs to be sufficiently small to ensure a high acceptance probability, but not so small that the complex density and gradient calculations are wasted without advancing the dynamics. The current standard is to choose h during an adaptation phase to reach a target acceptance probability of 0.65, derived by Beskos et al. (2013) for the high-dimensional limit of HMC.

The choice of integration time T is considerably harder especially when the variance of the target distribution is heterogeneous between different parameters $\theta_1 \ldots \theta_d$. In this case, use of a constant T can lead to arbitrarily poor mixing (Neal 2011). This problem can be mitigated by drawing T from an exponential distribution (Bou-Rabee & Sanz-Serna 2017).

The breakthrough in adaptively choosing the integration time T came from the No U-Turn Sampler (NUTS) (Hoffman & Gelman 2014). NUTS exploits the periodicity of Hamiltonian dynamics to find a maximum integration time T. To maximize the efficiency with which HMC explores the parameter space, the path should go as far as possible without turning back on itself, that is, without executing a U-turn. NUTS can be thought of as a stopping rule for HMC.

The subtlety of the NUTS algorithm lies in the way it preserves reversibility. From the current point, NUTS explores discretized Hamiltonian dynamics (Equations 9 and 10) in both forward and backward directions, randomly choosing the direction and doubling the number of leapfrog steps at each iteration. This process continues until the ends of the path start to turn back toward each other. The next NUTS sample is drawn at random from the points on the path.

The NUTS path is constructed in such a way that the probability of constructing the given path is the same starting from any one of the points it contains. This uniform distribution ensures reversibility of the chain.

7. STAN

Stan (Stan Dev. Team 2022), named in honor of the physicist Stanislaw Ulam (1909–1984), who coinvented the Monte Carlo method (Metropolis & Ulam 1949), is the latest general-purpose software for Bayesian modeling. Stan is designed for scalability and speed and was written from the start to exploit a multicore computing environment. Stan is written in C++ and published under the 3-clause BSD license.

Stan takes a different approach to model building from BUGS, NIMBLE, and JAGS by having a different language to define the model. Rather than creating a VGM, the Stan language defines the terms contributing to the log density of the joint distribution of the parameters $\log p(\boldsymbol{\theta})$. Users can use built-in density functions for common distributions, but they can also define their own functions in the Stan language to calculate contributions to the log density.

Listing 3 shows the Eight Schools example coded in the Stan language. Unlike the BUGS language, the Stan language is very explicit about the distinction between data and parameters, which must be declared in separate blocks. Deterministic functions of other variables are declared in a third block. Each variable declaration declares the data type, possibly including bounds, and the dimensions for vector or matrix quantities.

Although the declarations make the Stan code much longer than the BUGS code, the definition of the model itself is very succinct, due to the vectorization of the Stan language. The model is defined by three lines. Two lines in the model block show the contributions of the variables $\boldsymbol{\eta}$ and $\boldsymbol{y}$ to the log density (target), and the declaration of $\boldsymbol{\theta}$ also defines it as a linear function of μ, τ, and $\boldsymbol{\eta}$.

Listing 3 Stan code for the Eight Schools example

```
data {
  int<lower=0> J;
  real y[J];
  real<lower=0> sigma[J];
} parameters {
  real mu;
  real<lower=0> tau;
  vector[J] eta;
} transformed parameters {
  vector[J] theta = mu + tau * eta;
} model {
  target += normal_lpdf(eta | 0, 1);          // prior log-density
  target += normal_lpdf(y | theta, sigma); // log-likelihood
}
```

Another difference between the BUGS and Stan code is that Stan does not require all parameters to have a proper prior distribution. In the Eight Schools example, there is no prior distribution for the parameters `theta` and `tau`. Implicitly, these parameters have an improper flat prior.

Stan uses a C++ compiler to compile the model description into object code, which can then be executed with any data set as input. This property distinguishes Stan from BUGS and JAGS, which need to reinterpret the model description and create a new VGM if there are any changes to the data. This difference makes Stan ideal for MCMC analysis of simulated data, as the expensive compilation step needs to be run only once.

The main inference engine for Stan is HMC with NUTS. The HMC/NUTS algorithm is used to update all parameters of the model simultaneously. In contrast, BUGS, NIMBLE, and JAGS may use different sampling methods for different parameters. As a consequence of this approach, Stan does not admit discrete-valued parameters. Models that use discrete-valued parameters in the BUGS language may be rewritten in Stan to remove the discrete parameters by marginalization.

Like NIMBLE, Stan allows different forms of inference. In addition to NUTS for MCMC, Stan allows penalized maximum-likelihood estimation as well as approximate Bayesian inference with variational inference (Blei et al. 2016).

Stan has interfaces to R (Stan Dev. Team 2021) and Python (Riddell et al. 2021), among others. In addition, the rstanarm (Goodrich et al. 2020) and brms (Bürkner 2017) packages allow users to define regression models using the compact Wilkinson–Rogers-style notation used by the `(g)lm` and `(g)lmer` functions and still have access to the Stan HMC sampler as a back end.

8. INLA

INLA (Rue et al. 2009) is an alternative methodology for Bayesian inference applicable to a large class of models. Unlike MCMC, INLA is a deterministic approximation to the posterior.

INLA can be used for latent Gaussian models. These are two-level hierarchical models where the observable data $y_1 \ldots y_n$ are conditionally independent given a set of latent random variables $\boldsymbol{\eta}$ and possibly a set of hyperparameters $\boldsymbol{\theta}_1$:

$$p(\boldsymbol{y} \mid \boldsymbol{\eta}) = \prod_{i=1}^{n} p(y_i \mid \eta_i, \boldsymbol{\theta}_1). \qquad 11.$$

The latent variables $\boldsymbol{\eta}$ are expressed in terms of other latent quantities as a linear predictor,

$$\eta_i = \alpha + \sum_{j=1}^{n_b} \beta_j z_{ji} + \sum_{k=1}^{n_f} f^{(k)}(u_{ki}) + \epsilon_i, \qquad 12.$$

where $\boldsymbol{\beta}$ is a vector of coefficients for covariates $\boldsymbol{z}_j \in \mathbb{R}^{n_b}$, $f^{(1)} \ldots f^{(n_f)}$ are unknown functions of the covariates $\boldsymbol{u}_i \in \mathbb{R}^{n_f}$, and $\epsilon_1 \ldots \epsilon_n$ represent unstructured variation.

Let $\boldsymbol{x}$ be the collection of all latent quantities in the model:

$$\boldsymbol{x} = (\alpha, \beta_1 \ldots \beta_{n_b}, f^{(1)} \ldots f^{(n_f)}, \epsilon_1, \ldots \epsilon_n).$$

Then, $\boldsymbol{x}$ has a prior Gaussian distribution with mean zero and precision matrix $Q(\boldsymbol{\theta}_2)$ depending on hyperparameters $\boldsymbol{\theta}_2$.

Practical restrictions are that the full set of hyperparameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ is of low dimension and the precision matrix Q is sparse. If Equation 12 seems abstract and difficult to understand, that is because it is written in a way that admits the full generality of models that can be analyzed by INLA. This class includes time series models, generalized additive models, generalized additive mixed models, geoadditive models, and univariate volatility models.

INLA provides fast and accurate deterministic approximations to the marginal posteriors $p(x_i \mid \boldsymbol{y})$ and $p(\boldsymbol{\theta} \mid \boldsymbol{y})$. It does not provide the joint posterior distribution of $\boldsymbol{x}$, but the correlations are often not of interest. INLA is distinguished by being exceptionally fast compared with MCMC. Rue et al. (2009, p. 322) suggested that INLA "outperforms MCMC algorithms to such an extent that, for latent Gaussian models, resorting to MCMC sampling rarely makes sense in practice."

The R-INLA software (see **https://www.r-inla.org**) provides an implementation of INLA as an R package. The INLA method has been extremely successful in a wide range of applied problems. Its success illustrates that there is an alternative to the approach taken by most Bayesian software. Instead of trying to provide a universal solution to an unlimited class of Bayesian models, it is possible to conduct inference accurately and efficiently on a prescribed class of models.

9. PROBABILISTIC PROGRAMMING LANGUAGES

In the last 10 years, a convergence of interests in machine learning and computer science has given rise to the concept of probabilistic programming. This statistical modeling tool takes concepts from programming language design and applies them to the design and analysis of statistical models.

A full survey of probabilistic programming languages is beyond the scope of this review. Notable probabilistic programming languages include Church, based on Scheme (Goodman et al. 2008); Figaro, based on Scala (Pfeffer 2016); and PyMC3, based on Python (Salvatier et al. 2016). Notably, PyMC3 provides an implementation of the NUTS algorithm for HMC. The Stan developers describe Stan as a probabilistic programming language (Gelman et al. 2015). Arguably, NIMBLE is a probabilistic programming language that extends R.

10. FUTURE PERSPECTIVES

10.1. Robustness Versus Efficiency

The optimal behavior of MCMC methods in high dimensions has been well studied for the case when the joint probability distribution of the parameters $\boldsymbol{\theta}$ factorizes into independent and identically distributed (i.i.d.) components (Gelman et al. 1997, Roberts & Rosenthal 1998, Beskos et al. 2013),

$$p(\boldsymbol{\theta}) = \prod_{i=1}^{d} f(\theta_i), \qquad 13.$$

for some density function $f()$ common to all dimensions. This scenario is somewhat simplistic, especially when compared with the complex distributions to which MCMC is applied in practice. Nevertheless, it permits analysis of the optimal step size for each algorithm and the way it scales with dimension d. **Table 1** summarizes the results. For all methods, the optimal step size decreases with d, so the Markov chain becomes less efficient with increasing dimension of the parameter space. However, the rate at which the optimal step size decreases is slower for MALA compared with random walk Metropolis–Hastings, and slower for HMC than for MALA.

Since the i.i.d. limit in Equation 13 is somewhat artificial, the behavior of all these methods may be very poor in practice. The robustness of MCMC methods has attracted increasing interest. The question of robustness is extremely important for general-purpose software for Bayesian inference, which aims to efficiently solve a large class of applied problems.

Table 1 Optimal scaling of MCMC methods as a function of *d*, the dimension of the parameter space

Method	Optimal step size	Reference
Random walk M-H	$\mathcal{O}(d^{-1})$	Gelman et al. (1997)
MALA	$\mathcal{O}(d^{-1/3})$	Roberts & Rosenthal (1998)
HMC	$\mathcal{O}(d^{-1/4})$	Beskos et al. (2013)

Abbreviations: HMC, Hamiltonian Monte Carlo; MALA, Metropolis-adjusted Langevin algorithm; MCMC, Markov chain Monte Carlo; M-H, Metropolis–Hastings.

The word "robust" is an overloaded word in statistics, and its application to MCMC methods is no different. There are different ways to interpret the notion of robustness in this context. One is to consider which kinds of distributions an MCMC method can efficiently address. For heavy-tailed distributions, MALA and HMC are not geometrically ergodic (Roberts & Tweedie 1996, Livingstone et al. 2019). If the gradient of the log density goes to zero in the tails, then gradient-based methods no longer have useful additional information for efficiently exploring the parameter space and the random walk behavior that gradient-based methods are meant to suppress reappears. Conversely, for light-tailed distributions, the calculation of the gradients required by MALA and HMC may break down (Roberts & Tweedie 1996, Livingstone et al. 2019), rendering these methods unusable when simpler methods remain applicable.

Another way of thinking about robustness is to consider how well MCMC methods can be adaptively tuned for optimal performance. HMC exhibits poor behavior when the posterior distribution is heterogeneous, with much higher variance in some dimensions than others (Neal 2011). The mixing behavior of the worst component can be arbitrarily poor (Riou-Durand & Vogrinc 2022). In certain applications, this problem can be alleviated by some form of preconditioning. This might be straightforward manipulation of the data, such as standardizing predictor variables, as recommended by Gelman et al. (2014, section 16.3) for logistic regression.

Livingstone & Zanella (2022) have proposed an alternative to MALA, called the Barker proposal, which maintains the $d^{-1/3}$ scaling with dimension of MALA but has improved robustness to tuning. The Barker proposal uses a proposal distribution that is skewed in the direction of the gradient of the log density. In contrast, MALA has separate deterministic and random noise terms in each step.

10.2. Nonreversible Markov Chain Monte Carlo

Recent developments in MCMC have focused on nonreversible Markov chains as a way of suppressing random walk behavior and thus providing more efficient samples (Neal 1999, Diaconis et al. 2000, Bierkens 2016). **Figure 5** shows a mixture of two normals with a bimodal distribution,

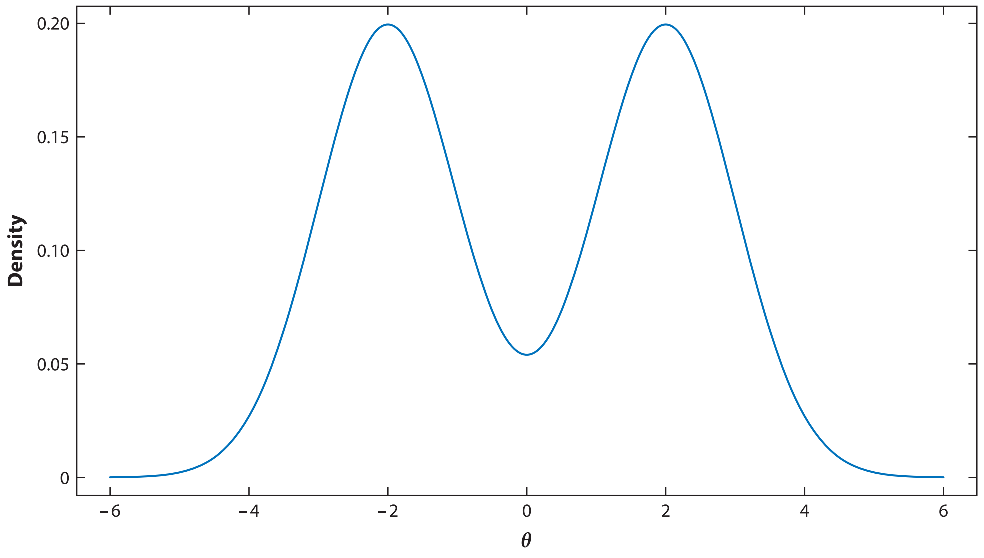

Figure 5

A bimodal distribution constructed from a mixture of two Gaussians.

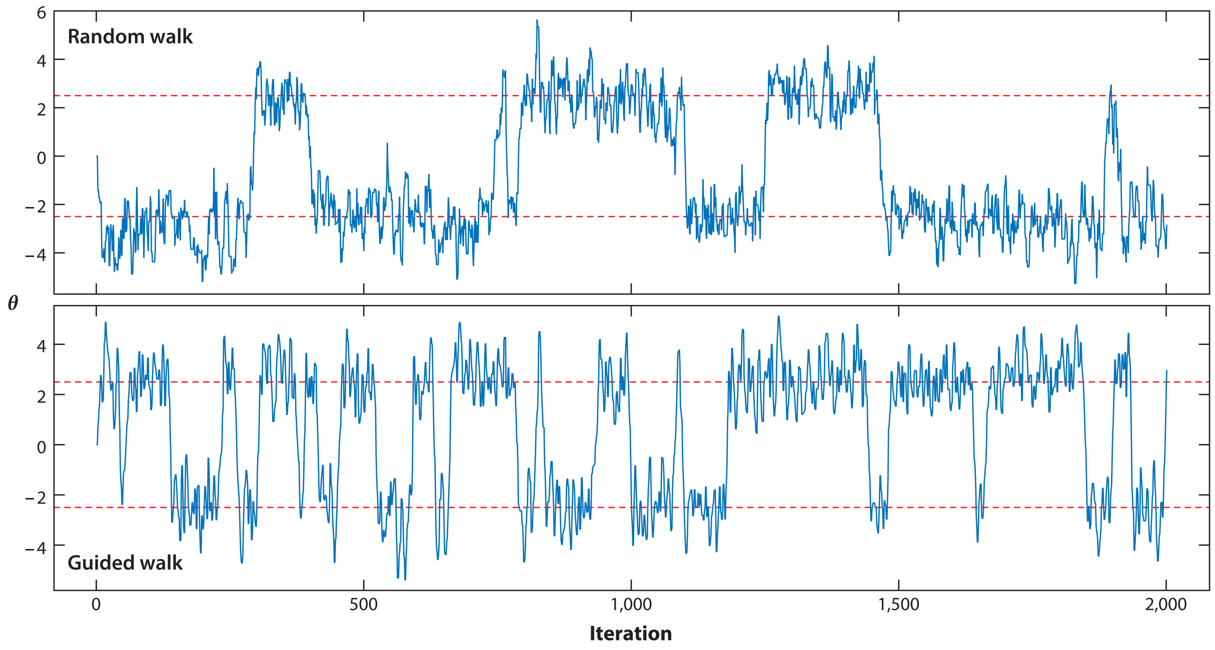

Figure 6

Trace plots for a random walk Metropolis algorithm and a guided walk on the bimodal target density shown in **Figure 5**. Red horizontal lines indicate the two modes of the distribution.

and **Figure 6** shows two Markov chains exploring this distribution. The first is a random walk Metropolis–Hastings sampler, which is reversible. The second is a guided walk (Gustafson 1998), which is the same as the random walk except that the direction of each step is the same as the previous iteration until a proposal is rejected. The guided walk then changes direction at the next iteration. The modified sampler is nonreversible. As shown by the trace plots, the nonreversible sampler is more efficient at switching between the two modes of the distribution. The momentum of the guided walk allows it to cross the gap between the two modes.

This an example of a so-called lifted Markov chain that has been augmented by a velocity variable taking values in $\{-1, 1\}$ (Chen et al. 1999, Turitsyn et al. 2011). Lifted Markov chains are easier to study in the continuous-time limit when we make the step size smaller and smaller, adjusting the timescale accordingly.

Figure 7 shows the effect of reducing the step size by a factor of 10. As the figure suggests, in the limit, the sampler becomes a piecewise-deterministic process. Change points are determined by a nonhomogeneous Poisson process in which the rate depends on the gradient of the log density in the direction of travel. Notably, the rate of change is zero when the process is moving toward a higher-density area. The chain always crosses one of the modes of the distribution before changing direction.

There are two multivariate generalizations of this process: the zig-zag sampler (Bierkens et al. 2016) and the bouncy particle sampler (Bouchard-Côté et al. 2018). In the zig-zag sampler, each component changes direction independently, whereas in the bouncy particle sampler, the whole path reflects off a tangent to the log density. Reference implementations of these samplers can be found in the R package RZigZag (Bierkens 2019).

The zig-zag and bouncy particle samplers are examples of piecewise-deterministic Markov processes (PDMPs) (Davis 1984). These are continuous-time stochastic processes that evolve

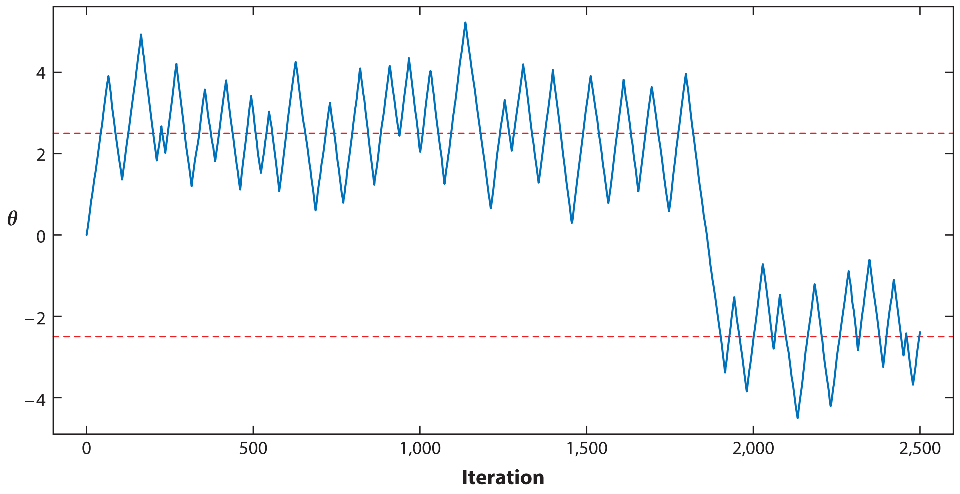

Figure 7

Continuous-time limit of the guided walk shown in **Figure 6** as the step size decreases.

deterministically in between change points that occur randomly in time. The events that occur at change points may also be stochastic. For example, the momentum vector may either change in a deterministic way or be refreshed by random sampling to ensure irreducibility. The first use of PDMPs for MCMC simulation was in the computational physics literature (Peters & de With 2012).

Like HMC, PDMPs augment the parameter space with additional parameters representing momentum. The momentum component allows a PDMP to quickly reach the mode of the distribution when the initial value is far away (Bierkens et al. 2016). Unlike HMC, however, the velocity is always constant. The simpler deterministic dynamics of PDMPs are easy to model on a computer without approximation error. In particular, PDMPs have no rejection step.

Fearnhead et al. (2018) note that PDMPs are a large class of MCMC methods whose potential has not yet been fully explored. A promising feature of PDMPs in the era of big data is that the log gradient $\nabla \log p(\boldsymbol{\theta})$ does not need to be calculated exactly but can be estimated from a subsample of data points. Unlike other MCMC methods, the subsampling of the gradient does not cause any bias, so the stationary distribution of the PDMP remains the same (Bierkens et al. 2016). This attribute makes PDMPs highly suitable for "tall" data with a very large number of observations.

PDMPs have no tuning parameters. As such, they are highly promising methods for implementation in general-purpose MCMC software. However, there is an additional layer of complexity in implementing PDMPs, which is sampling from a nonhomogeneous Poisson process to find the change-point times. Such sampling is feasible only if a suitably tight upper bound can be found for the gradient of the log density, which determines the rate of the Poisson process. The bound can be piecewise constant and thus needs to be calculated only in the range $(t, t + \delta)$ for some fixed value δ. When a bound on the gradient is determined, the next change point can then be simulated from a time-homogeneous Poisson process and thinned by rejection sampling (Lewis & Shedler 2012). For general-purpose Bayesian software, the outstanding problem is how to compute a bound on the gradient for an arbitrary posterior distribution. This may require the use of optimization methods to find the maximum of the log gradient within the time window $(t, t + \delta)$.

The practical challenge of implementing PDMPs into existing software is that existing frameworks are based on discrete-time MCMC. The latest generation of software, including NIMBLE and Stan, has separated the inference algorithm from the model description and hence could, in principle, incorporate continuous-time Markov processes as an alternative back end to discrete-time MCMC. Alternatively, it may be possible to have a discrete-time analog of PDMPs (Sherlock & Thiery 2022, Bertazzi et al. 2021). However, it is not certain whether the advantages of PDMPs can be retained in discrete time.

11. SUMMARY

As noted in Section 1, the high-dimensional integrals required for Bayesian inference are not generally tractable. However, several compromises that give approximate solutions are available. For MCMC, the main compromise is the time required to get a sufficient number of samples to accurately approximate the posterior distribution. INLA offers another compromise for the wide class of latent Gaussian models. INLA gives up on estimating the full posterior but provides fast and accurate approximations to the posterior margins. Users who require only a point estimate of the posterior mode may turn to the fast approximations provided by variational Bayes methods (Blei et al. 2016). In summary, different methods offer a range of compromises to applied Bayesian statisticians. The enduring popularity of MCMC suggests that it currently offers the most favorable compromise in many applications. Whether this will remain true as the demand for analyses of larger data sets increases remains to be determined. Green et al. (2015) conclude that approximate methods will be at the heart of future progress in statistical computing. It is notable that many of the theoretical innovations in MCMC methodology have come from applications in physics (Metropolis et al. 1953, Duane et al. 1987, Peters & de With 2012) before being adapted more widely. This observation suggests that methodologists should keep a weather eye on the computational physics literature.

Over the history of Bayesian software for MCMC, there has been a shift away from Gibbs sampling on a graph toward block-updating methods that act simultaneously on all parameters and are capable of scaling to high-dimensional parameter spaces. MCMC methods continue to be developed along these lines. PDMP methods are interesting but have not yet bridged the gap between theoretical development and practice. There are also promising innovations in robustness of MCMC methods that will improve the user experience when implemented in software.

The existence of freely available statistical software is essential for propagating statistical methods. For example, the R language and its associated CRAN repository (see **https://cran.r-project.org**), as of late 2022, have 19,000 third-party packages. Among these are reference implementations of some of the methods discussed in this review. While Bayesian software has not matched this level of developer engagement, it has allowed scientific problems in many different domains to be addressed by Bayesian methods. This level of engagement with applied statisticians needs to be maintained.

DISCLOSURE STATEMENT

The author is not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

LITERATURE CITED

Albert J, Chib S. 1993. Bayesian analysis of binary and polychotomous response data. *J. Am. Stat. Assoc.* 88:669–79

Andrieu C, Thoms J. 2008. A tutorial on adaptive MCMC. *Stat. Comput.* 18(4):343–73

Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. 2018. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.* 18:1–43

Bertazzi A, Bierkens J, Dobson P. 2021. Approximations of piecewise deterministic Markov processes and their convergence properties. arXiv:2109.11827 [math.PR]

Beskos A, Pillai N, Roberts G, Sanz-Serna J, Stuart A. 2013. Optimal tuning of the hybrid Monte Carlo algorithm. *Bernoulli* 19:1501–34

Betancourt M. 2017. A conceptual introduction to Hamiltonian Monte Carlo. arXiv:1701.02434 [stat.ME]

Bierkens J. 2016. Non-reversible Metropolis–Hastings. *Stat. Comput.* 26:1213–28

Bierkens J. 2019. *RZigZag: Zig-Zag sampler*. R Package Version 0.2.1. **https://cran.r-project.org/package=RZigZag**

Bierkens J, Fearnhead P, Roberts G. 2016. The Zig-Zag process and super-efficient sampling for Bayesian analysis of big data. *Ann. Stat.* 47(3):1288–320

Blei DM, Kucukelbir A, McAuliffe JD. 2016. Variational inference: a review for statisticians. *J. Am. Stat. Assoc.* 112(518):859–77

Bou-Rabee N, Sanz-Serna JM. 2017. Randomized Hamiltonian Monte Carlo. *Ann. Appl. Probab.* 27(4):2159–94

Bouchard-Côté A, Vollmer SJ, Doucet A. 2018. The bouncy particle sampler: a nonreversible rejection-free Markov chain Monte Carlo method. *J. Am. Stat. Assoc.* 113(522):855–67

Breslow NE, Clayton DG. 1993. Approximate inference in generalized linear mixed models. *J. Am. Stat. Assoc.* 88(421):9–25

Brooks S, Gelman A, Jones G, Meng XL, eds. 2011. *Handbook of Markov Chain Monte Carlo*. Boca Raton, FL: CRC

Bürkner PC. 2017. brms: an R package for Bayesian multilevel models using Stan. *J. Stat. Softw.* 80:1–28

Chen F, Lovász L, Pak I. 1999. Lifting Markov chains to speed up mixing. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing* (*STOC '99*), pp. 275–81. New York: ACM

Cowles MK, Carlin BP. 1996. Markov chain Monte Carlo convergence diagnostics: a comparative review. *J. Am. Stat. Assoc.* 91(434):883–904

Davis MHA. 1984. Piecewise-deterministic Markov processes: a general class of non-diffusion stochastic models. *J. R. Stat. Soc. B* 46(3):353–88

Davis TA. 2006. *Direct Methods for Sparse Linear Systems.* Philadelphia: SIAM

de Valpine P, Paciorek C, Turek D, Michaud N, Anderson-Bergman C, et al. 2022. *NIMBLE: MCMC, particle filtering, and programmable hierarchical modeling*. R Package Version 0.12.2. **https://r-nimble.org**

de Valpine P, Turek D, Paciorek C, Anderson-Bergman C, Temple Lang D, Bodik R. 2017. Programming with models: writing statistical algorithms for general model structures with NIMBLE. *J. Comput. Graph. Stat.* 26(2):403–13

Denwood MJ. 2016. runjags: an R package providing interface utilities, model templates, parallel computing methods and additional distributions for MCMC models in JAGS. *J. Stat. Softw.* 71(9):1–25

Diaconis P, Holmes S, Neal RM. 2000. Analysis of a nonreversible Markov chain sampler. *Ann. Appl. Probab.* 10(3):726–52

Duane S, Kennedy A, Pendleton BJ, Roweth D. 1987. Hybrid Monte Carlo. *Phys. Lett. B* 195(2):216–22

Fearnhead P, Bierkens J, Pollock M, Roberts GO. 2018. Piecewise deterministic Markov processes for continuous-time Monte Carlo. *Stat. Sci.* 33(3):386–412

Frühwirth-Schnatter S, Frühwirth R, Held L, Rue H. 2009. Improved auxiliary mixture sampling for hierarchical models of non-Gaussian data. *Stat. Comput.* 19(4):479–92

Garey MR, Johnson DS, Stockmeyer LJ. 1976. Some simplified NP-complete graph problems. *Theor. Comput. Sci.* 1(3):237–67

Gelfand AE, Smith AFM. 1990. Sampling-based approaches to calculating marginal densities. *J. Am. Stat. Assoc.* 85(410):398–409

Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. 2014. *Bayesian Data Analysis.* Boca Raton, FL: CRC. 3rd ed.

Gelman A, Gilks WR, Roberts GO. 1997. Weak convergence and optimal scaling of random walk Metropolis algorithms. *Ann. Appl. Probab.* 7(1):110–20

Gelman A, Lee D, Guo J. 2015. Stan: a probabilistic programming language for Bayesian inference and optimization. *J. Educ. Behav. Stat.* 40(5):530–43

Gelman A, Rubin DB. 1992. Inference from iterative simulation using multiple sequences. *Stat. Sci.* 7(4):457–72

Geman S, Geman D. 1984. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.* 6:721–41

Gilks WR. 1992. Derivative-free adaptive rejection sampling for Gibbs sampling. In *Proceedings of the 4th Valencia International Meeting* (*Bayesian Statistics 4*), ed. J Bernardo, J Berger, AP Dawid, AFM Smith, pp. 641–49. Oxford, UK: Clarendon

Gilks WR, Best NG, Tan KKC. 1995. Adaptive rejection Metropolis sampling within Gibbs sampling. *J. R. Stat. Soc. C* 44(4):455–72

Gilks WR, Clayton DG, Spiegelhalter DJ, Best NG, McNeil AJ, et al. 1993. Modelling complexity: applications of Gibbs sampling in medicine. *J. R. Stat. Soc. B* 55(1):39–52

Gilks WR, Thomas A, Spiegelhalter DJ. 1994. A language and program for complex Bayesian modelling. *J. R. Stat. Soc. D* 43(1):169–77

Gilks WR, Wild P. 1992. Adaptive rejection sampling for Gibbs sampling. *J. R. Stat. Soc. C* 41(2):337–48

Goodman ND, Mansinghka VK, Roy D, Bonawitz K, Tenenbaum JB. 2008. Church: a language for generative models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence* (*UAI'08*), pp. 220–29. Arlington, VA: AUAI

Goodrich B, Gabry J, Ali I, Brilleman S. 2020. *rstanarm: Bayesian applied regression modeling via Stan*. R Package Version 2.21.1. **https://mc-stan.org/rstanarm**

Goudie RJB, Turner RM, De Angelis D, Thomas A. 2020. MultiBUGS: a parallel implementation of the BUGS modeling framework for faster Bayesian inference. *J. Stat. Softw.* 95(7):1–20

Green PJ. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* 82(4):711–32

Green PJ, Latuszyński K, Pereyra M, Robert CP. 2015. Bayesian computation: a summary of the current state, and samples backwards and forwards. *Stat. Comput.* 25:835–62

Guennebaud G, Jacob B, et al. 2010. *Eigen, version 3*. Template Library. **http://eigen.tuxfamily.org**

Gustafson P. 1998. A guided walk Metropolis algorithm. *Stat. Comput.* 8:357–64

Hastings WK. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57(1):97–109

Hobert J. 2011. The data augmentation algorithm: theory and methodology. See Brooks et al. 2011, pp. 253–94

Hoffman MD, Gelman A. 2014. The No-U-Turn Sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* 15:1593–623

Holmes C, Held L. 2006. Bayesian auxiliary variable models for binary and multinomial regression. *Bayesian Anal.* 1(1):145–68

Kellner K. 2021. *jagsUI: a wrapper around 'rjags' to streamline JAGS analyses*. R Package Version 1.5.2. **https://CRAN.R-project.org/package=jagsUI**

Lauritzen SL, Dawid AP, Larsen BN, Leimer HG. 1990. Independence properties of directed Markov fields. *Networks* 20(5):491–505

Lauritzen SL, Spiegelhalter DJ. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *J. R. Stat. Soc. B* 50(2):157–224

Lewis PAW, Shedler GS. 2012. Simulation of non-homogeneous Poisson processes by thinning. *Naval Res. Logist. Q.* 26(3):403–13

Livingstone S, Betancourt M, Byrne S, Girolami M. 2019. On the geometric ergodicity of Hamiltonian Monte Carlo. *Bernoulli* 25(4):A3109–38

Livingstone S, Zanella G. 2022. The Barker proposal: combining robustness and efficiency in gradient-based MCMC. *J. R. Stat. Soc. B* 84(2):496–523

Lunn D. 2003. WinBUGS development interface (WBDev). *ISBA Bull.* 10(3):10–11

Lunn D, Best N, Whittaker J. 2009. Generic reversible jump MCMC using graphical models. *Stat. Comput.* 19:395

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D. 2012. *The BUGS Book: A Practical Introduction to Bayesian Analysis.* London: CRC/Chapman & Hall

Lunn D, Thomas A, Best N, Spiegelhalter D. 2000. WinBUGS, a Bayesian modeling framework: concepts, structure and extensibility. *Stat. Comput.* 10:325–37

Lunn DJ, Wakefield J, Thomas A, Best N, Spiegelhalter D. 1999. *PKBugs: an efficient interface for population PK/PD within WinBUGS*. Stat. Softw., Cambridge Univ., Cambridge, UK. **https://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/winbugs-development/pkbugs-an-efficient-interface-for-population-pk-pd-within-winbugs**

Mayrink VD, Duarte JDN, Demarqui FN. 2021. pexm: a JAGS module for applications involving the piecewise exponential distribution. *J. Stat. Softw.* 100(8):1–28

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. 1953. Equation of state calculations by fast computing machines. *J. Chem. Phys.* 21(6):1087–92

Metropolis N, Ulam S. 1949. The Monte Carlo method. *J. Am. Stat. Assoc.* 44(247):335–41

Michaud N, de Valpine P, Turek D, Paciorek CJ, Nguyen D. 2021. Sequential Monte Carlo methods in the nimble and nimbleSMC R packages. *J. Stat. Softw.* 100(3):1–39

Neal RM. 1999. Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. In *Learning in Graphical Models*, ed. MI Jordan, pp. 205–28. Cambridge, MA: MIT Press

Neal RM. 2003. Slice sampling. *Ann. Stat.* 31(3):705–67

Neal RM. 2011. MCMC using Hamiltonian dynamics. See Brooks et al. 2011, pp. 113–62

NIMBLE Dev. Team. 2021. *nimbleSMC: sequential Monte Carlo methods for 'nimble.'* R Package Version 0.10.1. **https://cran.r-project.org/package=nimbleSMC**

Peters EAJF, de With G. 2012. Rejection-free Monte Carlo sampling for general potentials. *Phys. Rev. E* 85:026703

Pfeffer A. 2016. *Practical Probabilistic Programming*. Shelter Island, NY: Manning

Plummer M. 2017. *JAGS Version 4.3.0 User Manual*. **https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x**

Plummer M. 2021. *rjags: Bayesian graphical models using MCMC*. R Package Version 4-13. **https://CRAN.R-project.org/package=rjags**

Plummer M, Best N, Cowles K, Vines K. 2006. CODA: convergence diagnosis and output analysis for MCMC. *R News* 6(1):7–11

Polson NG, Scott JG, Windle J. 2013. Bayesian inference for logistic models using Pólya–Gamma latent variables. *J. Am. Stat. Assoc.* 108(504):1339–49

Riddell A, Hartikainen A, Carter M. 2021. *PyStan version 3.5.0. PyPI*. Statistical Software. **https://pypi.org/project/pystan**

Riou-Durand L, Vogrinc J. 2022. Metropolis adjusted Langevin trajectories: a robust alternative to Hamiltonian Monte Carlo. arXiv:2202.13230 [stat.CO]

Roberts GO, Rosenthal JS. 1998. Optimal scaling of discrete approximations to Langevin diffusions. *J. R. Stat. Soc. B* 60(1):255–68

Roberts GO, Tweedie RL. 1996. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli* 2(4):341–63

Rubin DB. 1981. Estimation in parallel randomized experiments. *J. Educ. Behav. Stat.* 6(4):377–401

Rue H, Martino S, Chopin N. 2009. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *J. R. Stat. Soc. B* 71(2):319–92

Salvatier J, Wiecki T, Fonnesbeck C. 2016. Probabilistic programming in Python using PyMC3. *PeerJ Comput. Sci.* 2:e55

Sherlock C, Thiery AH. 2022. A discrete bouncy particle sampler. *Biometrika* 109(2):335–49

Spiegelhalter D, Thomas A, Best N, Gilks W. 1996a. *BUGS 0.5—Bayesian Inference Using Gibbs Sampling* (*Version II*). Cambridge, UK: Inst. Public Health

Spiegelhalter D, Thomas A, Best N, Gilks W. 1996b. *BUGS Examples—Version 0.5*. Vol. 1. Cambridge, UK: MRC Biostat. Unit

Spiegelhalter D, Thomas A, Best N, Gilks W. 1996c. *BUGS Examples—Version 0.5*. Vol. 2. Cambridge, UK: MRC Biostat. Unit

Spiegelhalter D, Thomas A, Best N, Lunn D. 2003. *WinBUGS User Manual, Version 1.4*. Cambridge, UK: MRC Biostat. Unit. **https://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs**

Stan Dev. Team. 2021. *RStan: the R interface to Stan*. R Package Version 2.21.3

Stan Dev. Team. 2022. *Stan User's Guide and Reference Manual 2.30*. **https://mc-stan.org/users/documentation**

Sturtz S, Ligges U, Gelman A. 2005. R2WinBUGS: a package for running WinBUGS from R. *J. Stat. Softw.* 12(3):1–16

Su YS, Yajima M. 2015. *R2jags: using R to run JAGS*. R Package Version 0.5-7. **https://CRAN.R-project.org/package=R2jags**

Thomas A. 2006. The BUGS language. *R News* 6(1):17–21

Thomas A, Best N, Lunn D, Arnold R, Spiegelhalter D. 2004. *GeoBUGS User Manual*. Cambridge, UK: MRC Biostat. Unit. **https://www.mrc-bsu.cam.ac.uk/software/bugs/thebugs-project-geobugs**

Thomas A, O'Hara B, Ligges U, Sturtz S. 2006. Making BUGS open. *R News* 6(1):12–17

Turitsyn KS, Chertkov M, Vucelja M. 2011. Irreversible Monte Carlo algorithms for efficient sampling. *Physica D* 240(4):410–14

Vats D, Knudson C. 2021. Revisiting the Gelman–Rubin diagnostic. *Stat. Sci.* 36(4):518–29

Vehtari A, Gelman A, Simpson D, Carpenter B, Bürkner PC. 2021. Rank-normalization, folding, and localization: an improved $\widehat{R}$ for assessing convergence of MCMC (with discussion). *Bayesian Anal.* 16(2):667–718

Wabersich D, Vandekerckhove J. 2014. Extending JAGS: a tutorial on adding custom distributions to JAGS (with a diffusion model example). *Behav. Res. Methods* 46:15–28

Wakefield JC, Aarons L, Racine-Poon A. 1999. The Bayesian approach to population pharmacokinetic/pharmacodynamic modelling. In *Case Studies in Bayesian Statistics*, Vol. 4, ed. BP Carlin, AL Carriquiry, C Gatsonis, A Gelman, RE Kass, et al., pp. 205–65. New York: Springer

Wetzels R, Lee M, Wagenmakers E. 2010. Bayesian inference using WBDev: a tutorial for social scientists. *Behav. Res. Methods* 42:884–97

Annual Review of Statistics and Its Application

Volume 10, 2023

Contents

Errata

An online log of corrections to *Annual Review of Statistics and Its Application* articles may be found at http://www.annualreviews.org/errata/statistics