Validation of Software for Bayesian Models using Posterior Quantiles

Samantha R. Cook
Andrew Gelman
Donald B. Rubin

DRAFT

Abstract

We present a simulation-based method designed to establish that software developed to fit a specific Bayesian model works properly, capitalizing on properties of Bayesian posterior distributions. We illustrate the validation technique with two examples. The validation method is shown to find errors in software when they exist and, moreover, the validation output can be informative about the nature and location of such errors.

Keywords: Markov chain Monte Carlo, Gibbs sampler, Posterior distribution, Hierarchical models.

1 Introduction

As modern computing environments become more advanced, statisticians are able to fit more complex models, which have the ability to address adequately complications such as missing data, complex sampling schemes, and treatment noncompliance. With increasing model complexity, however, comes increasing opportunity for errors in the software used to fit such models. This situation is particularly true with the rapidly expanding use of Bayesian statistical models fitted using iterative techniques, such as Markov chain Monte Carlo (MCMC) methods. Not only are MCMC methods computationally intensive, but there is relatively limited software available to aid the fitting of such models. Implementing these methods therefore often requires developing the necessary software "from scratch," and such software is rarely tested to the same extent as most publicly available software (e.g., R, S-plus, SAS); of course, publicly available software is not necessarily error-free either.

Although there is a rich literature on algorithms for fitting Bayesian models (e.g., Gelfand and Smith, 1990; Gelman, 1992; Smith and Roberts, 1993), there is only limited work (Geweke, 2004) related to investigating whether the software developed to implement these algorithms works properly. Standard software testing and debugging methods from the engineering and computer science literature (e.g., Agans, 2002) can sometimes be helpful but these tend to focus on fixing obvious errors and crashes, rather than on determining whether software that runs and appears to work correctly actually does what it claims to do. Here we outline a simulation-based method for testing the correctness of software for fitting Bayesian models using posterior simulation. We begin by describing the design of the validation simulation and analysis of the simulation output in Section 2. Section 3 provides examples using two different pieces of software, and Section 4 presents further discussion and conclusions.

2 Methods for Automatic Software Validation using Simulated Data

People often test software by applying it to data sets for which the "right answer" is known or approximately known, and compare the expected results to those obtained from the software. Such a strategy becomes more complicated with software for Bayesian analyses, whose results are inherently stochastic, but can be extended to develop statistical tests of the correctness of Bayesian model-fitting software. The basic idea is that if we draw parameters from their prior distribution and data from their sampling distribution, and then perform Bayesian inference correctly, then posterior

inferences will be, on average, correct. For example 50% and 95% intervals will contain the true parameter values with probability 0.5 and 0.95, respectively. In this paper, we develop a slightly more elaborate version of this idea.

2.1 Theoretical Framework

Consider the general Bayesian model $p(y|\boldsymbol{\theta})p(\boldsymbol{\theta})$, where $p(y|\boldsymbol{\theta})$ represents the sampling distribution of the data, y, $p(\boldsymbol{\theta})$ represents the proper prior distribution of the parameter vector, $\boldsymbol{\theta}$, and inferences are based on the posterior distribution, $p(\boldsymbol{\theta}|y)$. If a "true" parameter value $\boldsymbol{\theta}^{(0)}$ is generated according to $p(\boldsymbol{\theta})$, and data y^0 are then generated according to $p(y|\boldsymbol{\theta}^{(0)})$, then $(y^0, \boldsymbol{\theta}^{(0)})$ represents a draw from the joint distribution $p(y, \boldsymbol{\theta})$, which implies that $\boldsymbol{\theta}^{(0)}$ represents a draw from $p(\boldsymbol{\theta}|y^0)$, the posterior distribution with y^0 observed. Now consider the posterior sample of size L, $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$, generated from the to-be-tested software with $y = y^0$. If the software is written correctly, then $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$ are a sample from $p(\boldsymbol{\theta}|y^0)$, meaning that $\boldsymbol{\theta}^{(0)}$ and $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$ are drawn from the same distribution. This is also true for any subset of the components of $\boldsymbol{\theta}$.

Our proposed validation methods are based on the posterior quantiles of the true values of scalar parameters. Let ζ represent a scalar component or function of $\boldsymbol{\theta}$; the true value $\zeta^{(0)}$ and posterior sample $(\zeta^{(1)}, \ldots, \zeta^{(L)})$ are obtained from $\boldsymbol{\theta}^{(0)}$ and $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$, respectively. Let $q(\zeta^{(0)}) = \Pr(\zeta^{(0)} > \zeta^{(\ell)})$, the posterior quantile of $\zeta^{(0)}$, where $\zeta^{(\ell)}$ represents a random draw from the distribution of ζ generated by the to-be-tested software. The estimated quantile, $\hat{q}(\zeta^{(0)}) = \widehat{\Pr}(\zeta^{(0)} > \zeta^{(\ell)})$, is equal to $\frac{1}{L}\sum_{\ell=1}^{L} I_{\zeta^{(0)} > \zeta^{(\ell)}}$.

Theorem 1. *If the software is written correctly, for continuous ζ the distribution of $\hat{q}(\zeta^{(0)})$ approaches Uniform(0, 1) as $L \to \infty$.*

Proof. Let $Q_x(f)$ represent the xth quantile of a distribution f. To prove that the distribution of $\hat{q}(\zeta^{(0)})$ approaches uniformity, we need to show that $\lim_{L\to\infty} \Pr(\hat{q}(\zeta^{(0)}) < x) = x$ for any $x \in [0, 1]$:

$$\lim_{L\to\infty} \Pr(\hat{q}(\zeta^{(0)} < x) = \lim_{L\to\infty} \Pr\left(\zeta^{(0)} < Q_x(\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(L)})\right) \tag{1}$$

$$= \Pr\left(\zeta^{(0)} < \lim_{L\to\infty} Q_x(\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(L)})\right) \tag{2}$$

$$= \Pr\left(\zeta^{(0)} < Q_x(p(\zeta|y^0))\right) \tag{3}$$

$$= x. \tag{4}$$

For independently drawn $(\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(L)})$, Equation 3 follows from the convergence of the empirical distribution function to the true distribution function. If the sample $(\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(L)})$ comes from a process such as MCMC that converges to the target distribution, Equation 3 follows from the assumed ergodicity of the simulation process (see, e.g., Tierney, 1998, for MCMC simulation). Equation 4 follows from the fact that $\zeta^{(0)}$ is drawn from $p(\zeta|y^0)$. Thus, as the model-fitting algorithm converges, the distribution of $q(\zeta^{(0)})$ approaches the uniform. □

2.2 Outline of Validation Simulation

The uniformity of the posterior quantiles motivates powerful diagnostics for assessing whether Bayesian model-fitting software is written correctly. As outlined in Figure 1, simply generate and analyze data according to the same model, and calculate posterior quantiles. Specifically, generate the parameter vector $\boldsymbol{\theta}^{(0)}$ from $p(\boldsymbol{\theta})$. Then conditional on $\boldsymbol{\theta}^{(0)}$ generate the data y^0 from $p(y|\boldsymbol{\theta} = \boldsymbol{\theta}^{(0)})$. Next, analyze y^0 using the model-fitting software based on the same model as was used to simulate the data, thereby creating the random sample of size L, $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$, from $p(\boldsymbol{\theta}|y = y^0)$. Finally, for each component of $\boldsymbol{\theta}^{(0)}$, compute the posterior quantile, $\hat{q}(\zeta^{(0)})$, with respect to the posterior sample, $\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(L)}$. Imputed values of missing data and functions of $\boldsymbol{\theta}$, for example, ratios or cross products, may be considered as well.

We refer to the generation and analysis of a single data set under a fixed condition as one *replication*. Figure 1 outlines the steps of one replication. The simulation *conditions* are unmodeled data and parameters, such as: the sample size, including number of groups and sample sizes within groups (for a hierarchical data structure); values of nonstochastic predictors (for a regression model); and fixed constants in the prior distribution (i.e., parameters in the prior distribution that are set to a priori fixed values rather than estimated from the data). We refer to the number of replications performed under the same condition as the *size* of the simulation for that condition. Finally, we refer to the software that generates $\boldsymbol{\theta}^{(0)}$ and y^0 as the data-generating software and the software that samples $(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$ as the model-fitting software. A key idea of the simulation method is the comparison of results from two computer programs: The data-generating software and the model-fitting software both sample from $p(\boldsymbol{\theta}|y)$. Direct simulation, i.e, the data-generating software, is (presumably) easier to program and easier to program correctly; Geweke (2004) also makes this point.

To perform the validation experiment, perform many replications, each drawing $\boldsymbol{\theta}^{(0)}$ from $p(\boldsymbol{\theta})$ and y^0 from $p(y|\boldsymbol{\theta}^{(0)})$

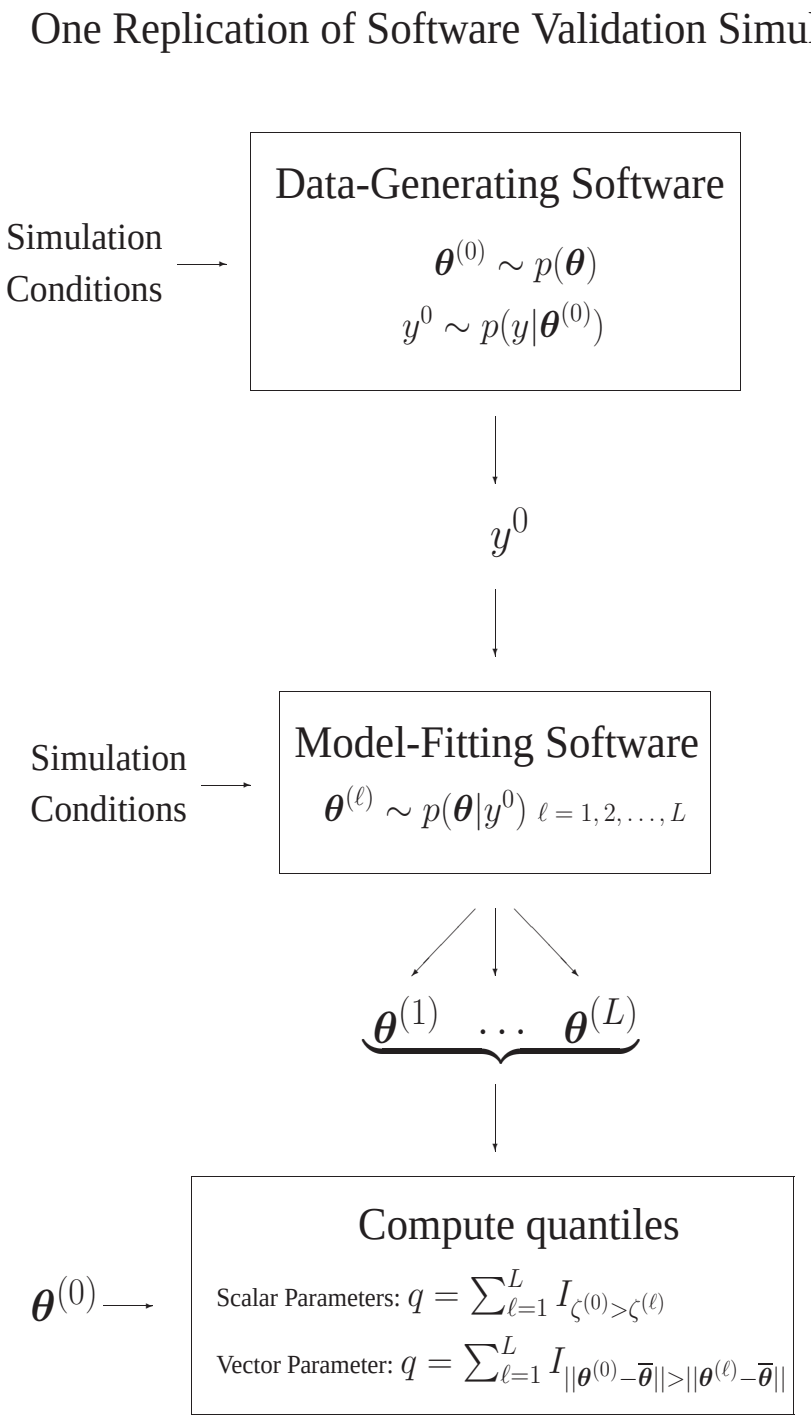

Figure 1: Steps of one replication of validation simulation. Given the simulation conditions, first generate parameters $\boldsymbol{\theta}^{(0)}$ and data y^0 using the data-generating software. Then, given the data y^0, generate a sample from $p(\boldsymbol{\theta}|y^0)$ using the model-fitting software. Finally, from the posterior sample and $\boldsymbol{\theta}^{(0)}$, calculate posterior quantiles.

and then using the to-be-tested software to obtain a sample from $p(\boldsymbol{\theta}|y^0)$. The simulation output is a collection of estimated posterior quantiles; from the quantiles we can calculate test statistics to quantify their deviation from uniformity, and hence test the correctness of the software.

There are three levels of potential subscripting here: replications, draws of $\boldsymbol{\theta}$ within replications, and components of $\boldsymbol{\theta}$. We never indicate components of $\boldsymbol{\theta}$ with subscripts; instead, we use ζ to denote a generic scalar component of $\boldsymbol{\theta}$. We denote draws of $\boldsymbol{\theta}$ (or ζ) within a replication with superscripts: $\boldsymbol{\theta}^{(\ell)}$ represents the ℓth draw, with $\ell = 1, \ldots, L$. We denote replications with subscripts: $i = 1, \ldots, N_{rep}$.

2.3 Analyzing Simulation Output

Now let $q_i = \frac{1}{L}\sum_{\ell=1}^{L} I_{\zeta_i^{(0)} > \zeta_i^{(\ell)}}$, the empirical quantile for the ith replication. Also, let h be a generic function and $h(U)$ that function applied to a random variable with Uniform(0, 1) distribution. Then for each scalar ζ, we calculate a z score summarizing the departure from uniformity of the posterior quantiles as computed from the N_{rep} replications:

$$z_\zeta = \frac{\frac{1}{N_{rep}}\sum_{i=1}^{N_{rep}} h(q_i) - \mathrm{E}(h(U))}{\sqrt{\mathrm{Var}(h(U))/N_{rep}}}. \tag{5}$$

By the Central Limit Theorem, z_ζ follows a standard normal distribution when the software is correctly written and N_{rep} is large.

For scalar components of $\boldsymbol{\theta}$, we suggest using the following function to test for uniformity: $h(q_i) = (q_i - .5)^2$, which has mean equal to $\frac{1}{12}$ and variance equal to $\frac{1}{180}$ when quantiles are uniformly distributed (i.e., $\mathrm{E}(h(U)) = \frac{1}{12}$ and $\mathrm{Var}(h(U)) = \frac{1}{180}$). This proposed transformation can be more informative than the raw quantiles about deviations from uniformity if the quantiles follow a non-uniform distribution that is centered near 0.5 but has mass piled up near 0 and 1. Our investigations revealed that posterior quantiles do sometimes follow such a U-shaped distribution when software has errors; simply testing that the posterior quantiles have mean equal to 0.5 may therefore not find errors when they exist. Figure 2 shows a histogram of such a distribution obtained from software with an error. Other choices of h are, for example, $h(q) = q$ or $h(q) = I_{\{0.25<q<.75\}}$ (an indicator of 50% interval coverage). Direct tests of uniformity such as the Kolmogorov-Smirnov one-sample test (Kolmogorov, 1933) are also possible, but may have low power when N_{rep} is small.

We can look for extreme values of the z_ζ statistics graphically. To help diagnose the location of errors when they exist,

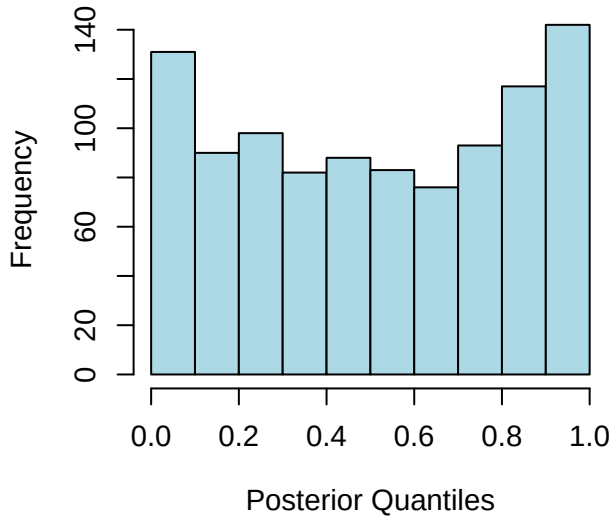

Figure 2: An example of posterior quantiles q from software with error. An effective summary for detecting the error should emphasize quantiles near 0 or 1, such as $h(q) = (q - \frac{1}{2})^2$.

we recommend plotting parameters in "batches" that are related (see Figures 3 - 8). Such plots of the z_ζ statistics are often informative about errors in the software when they exist; however, when the number of scalar components monitored is large, we expect some z_ζ statistics to be extreme (e.g., 5% at the 5% level) just by chance. We could use corrections for multiple comparisons, but prefer not to because most multiple comparison procedures assume the multiple tests are independent, which they almost surely are not here. Instead, for each draw of $\boldsymbol{\theta}$ we calculate its multivariate (Mahalanobis) distance from the joint posterior mean of $\boldsymbol{\theta}$. Since this distance is itself a scalar, we can calculate a test statistic similar to z_ζ to test the correctness of the software.

Specifically, let $\boldsymbol{\theta}^{(\ell)}$ represent a posterior draw of the model parameters, $\overline{\boldsymbol{\theta}}$ the posterior mean vector, and $V_{\boldsymbol{\theta}}$ the posterior covariance matrix. Within each replication, calculate the Mahalanobis distance $||\boldsymbol{\theta}^{(\ell)} - \overline{\boldsymbol{\theta}}|| = (\boldsymbol{\theta}^{(\ell)} - \overline{\boldsymbol{\theta}})' V_{\boldsymbol{\theta}}^{-1} (\boldsymbol{\theta}^{(\ell)} - \overline{\boldsymbol{\theta}})$ for $\ell = 1, \ldots, L$. Also calculate the distance for the "true value" $\boldsymbol{\theta}^{(0)}$: $(\boldsymbol{\theta}^{(0)} - \overline{\boldsymbol{\theta}})' V_{\boldsymbol{\theta}}^{-1} (\boldsymbol{\theta}^{(0)} - \overline{\boldsymbol{\theta}})$. The posterior quantile for the ith replication is then $q_i = \frac{1}{L} \sum_{\ell=1}^{L} I_{||\boldsymbol{\theta}^{(0)} - \overline{\boldsymbol{\theta}}|| > ||\boldsymbol{\theta}^{(\ell)} - \overline{\boldsymbol{\theta}}||}$ and we can calculate a test statistic $z_{\boldsymbol{\theta}}$ as in Equation 5. When the software is correctly written, q_i should follow a uniform distribution; when software has errors, q_i should be closer to 1 because $\boldsymbol{\theta}^{(0)}$ will be far from $\overline{\boldsymbol{\theta}}$ relative to the posterior sample. In this case, we recommend using $h(q_i) = q_i$ to calculate the $z_{\boldsymbol{\theta}}$ statistics. When $h(q_i) = q_i$, $\mathrm{E}(h(U)) = 0.5$ and

$\mathrm{Var}(h(U)) = \frac{1}{12}$. When calculated for scalar parameters, $z_{\boldsymbol{\theta}}$ gives essentially the same results as z_ζ; both measure distances between $\zeta^{(0)}$ and the "center" of $(\zeta^{(1)}, \ldots, \zeta^{(L)})$. In a simulation where we calculated $z_{\boldsymbol{\theta}}$ as well as z_ζ for each scalar parameter, their correlation was 0.99. In order to perform this multivariate test, L must be larger than the dimension of $\boldsymbol{\theta}$. If the dimension of $\boldsymbol{\theta}$ is large compared to L, we can perform a similar multivariate test with $V_{\boldsymbol{\theta}}$ replaced by the diagonal matrix with entries equal to the variances of components of $\boldsymbol{\theta}$.

As we illustrate in Section 3, examining the z_ζ statistics graphically can be helpful for finding the parts of the software that are written incorrectly if software does have errors. When these initial diagnostics show no indication of errors, $z_{\boldsymbol{\theta}}$ provides an omnibus test of the null hypothesis of correctly written software.

2.4 Other Approaches

An intuitive approach to testing software involves repeatedly generating data from the assumed model, using the software to calculate, say, 95% intervals, and comparing the observed interval coverage with the nominal coverage. For Bayesian models, such an approach is a special case of our method, where $h(q) = I_{\{0.025<q<.975\}}$. For classical confidence intervals, however, such an approach is not generally useful for software validation because most frequentist confidence intervals have nominal coverage only asymptotically.

Geweke (2004) presents an alternative simulation strategy for testing Bayesian model-fitting software. This approach also involves comparing the results from two separate programs, in this case two programs that generate from the joint distribution $p(\boldsymbol{\theta}, y)$. One of the programs is equivalent to our data-generating software, creating independent samples from $p(\boldsymbol{\theta}, y)$ by first sampling $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta})$ and then sampling y from $p(y|\boldsymbol{\theta})$; this piece of software should be straightforward to write and is presumed to be error-free. The second program is created by adding to the algorithm that samples from $p(\boldsymbol{\theta}|y)$ an additional step that samples y from $p(y|\boldsymbol{\theta})$; the limiting distribution of this new algorithm is also $p(\boldsymbol{\theta}, y)$. Thus two samples from $p(\boldsymbol{\theta}, y)$ are generated; z statistics are then calculated to test that the means of (scalar) components and functions of $(\boldsymbol{\theta}, y)$ are the same from both programs.

Geweke's approach has the advantage that only one replication needs to be performed, because the two programs generate from $p(\boldsymbol{\theta}, y)$ rather than repeatedly generating from $p(\boldsymbol{\theta}|y)$ for different values of y. A disadvantage is that it requires altering the software to be tested: If the software in question is truly a "black box," altering it to re-sample

the data at each iteration may not be possible. Even when this alteration is possible, it may still be desirable to test the version of the software that will actually be used, rather than an altered version. In addition, Geweke's method requires that the functions of $(\boldsymbol{\theta}, y)$ compared from the two programs have finite variance, a condition that is not met for certain proper but vague prior distributions, such as the Cauchy or Inverse-gamma(a, a) if $a \leq 2$.

3 Examples

We illustrate the posterior quantile-based software validation techniques with one simple and one complex example. For each example, we first present simulation results from correctly written software and then illustrate how various errors propagate to diagnostic measures.

3.1 A Simple One-Way Hierarchical Model

3.1.1 Model

We illustrate first with a simple hierarchical normal model. The model applies to grouped or clustered data, with means varying across groups and constant variance:

$$\begin{aligned} y_{ij}|\alpha_j, \sigma^2 &\sim \mathrm{N}(\alpha_j, \sigma^2) \qquad i = 1, \ldots, n_j, \qquad j = 1, \ldots, J \\ \alpha_j|\mu, \tau^2 &\sim \mathrm{N}(\mu, \tau^2) \qquad j = 1, \ldots, J. \end{aligned}$$

We assume a simple conjugate prior distribution:

$$\begin{aligned} \sigma^2 &\sim \mathrm{Inv}{-}\chi^2(5, 20) \\ \mu &\sim \mathrm{N}(5, 5^2) \\ \tau^2 &\sim \mathrm{Inv}{-}\chi^2(2, 10). \end{aligned}$$

Because the prior distribution is conjugate, it is straightforward to derive the complete conditional distribution for each parameter and simulate the posterior distribution using Gibbs sampling (Geman and Geman, 1984; Gelfand and Smith, 1990).

3.1.2 Validation Simulation

We perform a simulation to validate the MCMC software developed to fit the simple model presented in Section 3.1.1. We generate data with sample size $J = 6$, $n = (33, 21, 22, 22, 24, 11)$. The simulation consists of 1,000 replications. Within each replication, we generate a sample of L = 5,000 draws from the posterior distribution of the model parameters. We monitor all parameters listed in Section 3.1.1, as well as (arbitrarily) the coefficients of variation: μ/τ and $\alpha_j/\sigma, j = 1, \ldots, J$.

3.1.3 Validation Results: Correctly Written Software

Scalar Validation. The z_ζ statistics from this simulation are plotted in Figure 3. Here we differentiate among $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}/\sigma$, μ, τ^2, σ^2, and μ/τ. Each row in the plot represents a different batch of parameters. The vertical lines are at $\pm$1.96; 95% of the z_ζ statistics should be between these lines when software is written correctly. In this simulation the expected number of significant (at the 5% level) z_ζ statistics is 0.8; none of the observed z_ζ statistics is significant, thus providing no indication of errors in the software. As illustrated later in this section, when errors are present, examining these plots is often sufficient to diagnose the aspect of a program that is written incorrectly.

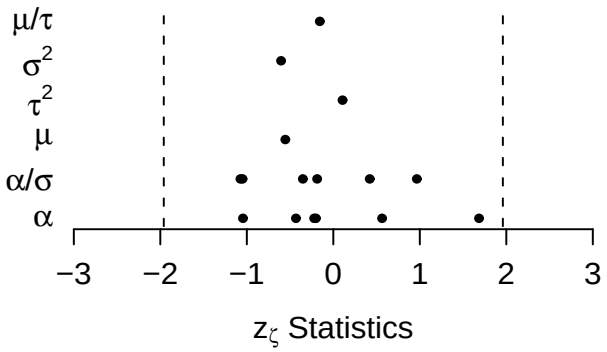

Figure 3: Scalar validation z_ζ Statistics: Simple model, correctly written software. Each row represents a scalar parameter or batch of parameters; the points in each row represent the z_ζ statistics associated with that parameter or batch of parameters. Vertical lines at $\pm$1.96 represent 95% bounds.

Omnibus Validation. In addition to plotting the z_ζ statistics, we perform the $z_{\boldsymbol{\theta}}$ test. For this simulation the test statistic is $z_{\boldsymbol{\theta}} = -1.01$ with corresponding p-value 0.31, also providing no evidence of incorrectly written software.

3.1.4 Validation Results: Incorrectly sampling $\boldsymbol{\alpha}$

We have just seen how the validation results should look when software is written correctly. We now show analogous results for software with errors, and illustrate how the diagnostics advocated here can be used to locate errors when they exist.

The complete conditional distribution of α_j in the Gibbs sampler is normal with mean $\frac{\frac{\mu}{\tau^2}+\frac{\sum_{i=1}^{n_j} y_{ij}}{\sigma^2}}{\frac{1}{\tau^2}+\frac{n_j}{\sigma^2}}$ and variance $\frac{1}{\frac{1}{\tau^2}+\frac{n_j}{\sigma^2}}$. We change the model-fitting software to incorrectly use $N = \sum_{j=1}^{J} n_j$ instead of n_j when sampling α_j, i.e., we sample α_j with mean $\frac{\frac{\mu}{\tau^2}+\frac{\sum_{i=1}^{n_j} y_{ij}}{\sigma^2}}{\frac{1}{\tau^2}+\frac{N}{\sigma^2}}$ and variance $\frac{1}{\frac{1}{\tau^2}+\frac{N}{\sigma^2}}$. We perform another simulation with 1,000 replications.

Scalar Validation. Figure 4 plots the z_ζ statistics from this simulation; note the scale of the x axis. It is clear that there is an error somewhere in the software. All z_ζ statistics are significant at the 5% level; in fact, all are larger than 7. The parameters with the smallest z_ζ statistics are τ^2 and the coefficient of variation μ/τ, whereas the parameters $\boldsymbol{\alpha}$,

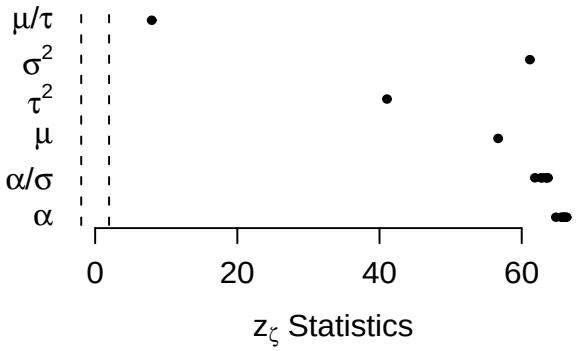

Figure 4: Scalar validation z_ζ Statistics: Simple model, incorrectly sampling $\boldsymbol{\alpha}$. Each row represents a scalar parameter or batch of parameters. Vertical lines at ± 1.96 represent 95% bounds.

σ^2, μ, and $\boldsymbol{\alpha}/\sigma$ all have much more extreme z_ζ statistics. In this situation we would recommend looking for errors in the sections of the program that sample $\boldsymbol{\alpha}$ and σ^2, because these parameters and functions of them have such extreme z_ζ statistics.

Omnibus Validation. The omnibus distance-based z statistic is $z_{\boldsymbol{\theta}} = 54$ (p-value ≈ 0), clearly indicating an error in the software.

3.1.5 Validation Results: Incorrectly Sampling μ

For another example, we consider an error in the specification of the complete conditional distribution of μ in the Gibbs sampler, which is normal with mean $\frac{\frac{\sum_{j=1}^{J}\alpha_j}{\tau^2}+\frac{5}{5^2}}{\frac{J}{\tau^2}+\frac{1}{5^2}}$ and variance $\frac{1}{\frac{J}{\tau^2}+\frac{1}{5^2}}$. We change the model-fitting program so that 5 rather than 5^2 is used in these conditional distributions, i.e., we sample μ with mean $\frac{\frac{\sum_{j=1}^{J}\alpha_j}{\tau^2}+\frac{5}{5}}{\frac{J}{\tau^2}+\frac{1}{5}}$ and variance $\frac{1}{\frac{J}{\tau^2}+\frac{1}{5}}$, and again perform a simulation of 1,000 replications. The results from this simulation are not as extreme as those in the previous section; however, they still clearly indicate an error in the software and, moreover, are informative about the source of the error.

Scalar Validation. As can be seen from Figure 5, the z_ζ statistics for μ and μ/τ are highly significant, whereas the z_ζ statistics for the other parameters are within the expected range for standard normal random variables. These results clearly suggest an error in the software and indicate that the error is in the part of the software that samples μ.

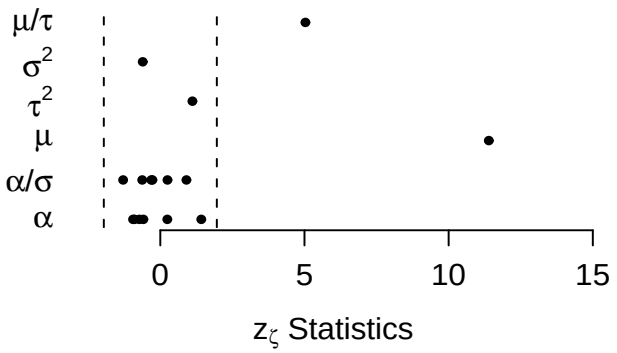

Figure 5: Scalar validation z_ζ Statistics: Simple model, incorrectly sampling μ. Each row represents a scalar parameter or batch of parameters. Vertical lines at ± 1.96 represent 95% bounds.

Omnibus Validation. The $z_{\boldsymbol{\theta}}$ statistic for this simulation equals 2.59 (p-value = 0.01).

3.2 A Hierarchical Repeated-Measures Regression Model

Many models that require specialized software are much more complex than the simple model in the previous example. As models become more complex it becomes more important to test the model-fitting software. At the same time, however, computation time may prevent simulations as large as those presented for the simple example in Section 3.1

for complex models. We now present an example of software validation for a much more complicated model, and show that the validation techniques work well even with far fewer replications.

3.2.1 Model

This model was developed to impute missing data in a clinical trial. The data are repeated blood measurements, and we use a Bayesian hierarchical regression model to describe them; see Cook (2004) for more details. The model forces a decreasing trend in the outcome measurements over time for each subject; the complete-data model is:

$$y_{ij} \sim N(\mu_j - \exp(\beta_j)t_{ij} - \exp(\gamma_j)t_{ij}^2, \sigma_j^2) \qquad i = 1, \ldots, n_j, \qquad j = 1, \ldots, J, \tag{6}$$

where t_{ij} is the time of the ith measurement for the jth patient. The prior distribution of $(\mu_j, \beta_j, \gamma_j)$ is trivariate normal with means that depend on covariates; we parameterize this distribution in factored form:

$$\begin{aligned}
\gamma_j|\sigma^2, \boldsymbol{\xi}, \boldsymbol{X}_j &\sim N(\eta_0 + \eta_1 X_j + \eta_2 X_j^2, \omega^2) \\
\beta_j|\gamma_j, \sigma^2, \boldsymbol{\xi}, \boldsymbol{X}_j &\sim N(\delta_{\beta_0} + \delta_{\beta_1} X_j + \delta_{\beta_2} X_j^2 + \delta_{\beta_3}\gamma_j, \omega_\beta^2) \\
\mu_j|\gamma_j, \beta_j, \sigma^2, \boldsymbol{\xi}, \boldsymbol{X}_j &\sim N(\delta_{\mu_0} + \delta_{\mu_1} X_j + \delta_{\mu_2} X_j^2 + \delta_{\mu_3}\gamma_j + \delta_{\mu_4}\beta_j, \omega_\mu^2),
\end{aligned}$$

where X_j is related to the baseline measurement for the jth patient; $\boldsymbol{\eta} = (\eta_0, \eta_1, \eta_2, \log(\omega))'$; $\boldsymbol{\delta}_\beta = (\delta_{\beta_0}, \delta_{\beta_1}, \delta_{\beta_2}, \delta_{\beta_3}, \log(\omega_\beta))'$; $\boldsymbol{\delta}_\mu = (\delta_{\mu_0}, \delta_{\mu_1}, \delta_{\mu_2}, \delta_{\mu_3}, \delta_{\mu_4}, \log(\omega_\mu))'$; and $\boldsymbol{\xi} = (\boldsymbol{\eta}, \omega, \boldsymbol{\delta}_\beta, \omega_\beta, \boldsymbol{\delta}_\mu, \omega_\mu)$. The vector $\boldsymbol{\eta}$ has an informative multivariate normal prior distribution with fixed parameters. The correlation matrix of this distribution, $\boldsymbol{r}$, is not diagonal. The remaining scalar components of $\boldsymbol{\xi}$ have independent normal prior distributions with fixed input parameters. In addition to the model parameters, we also monitor imputed values of missing data: For each generated data set n_{mis} data points were masked out and then imputed.

3.2.2 Validation Simulation

Each replication consists of generating a sample of $L = 5{,}000$ draws from the posterior distribution. Because of the time required to fit this more complex model, we perform only 100 replications. Using WinBUGS (Spiegelhalter *et al.*, 1994, 2003) to fit the models, a simulation of 100 replications requires approximately twelve hours of computing time. For each patient, t_{ij} takes the integer values from 0 to $n_j - 1$. The sample sizes are $J = 9$,

$n = (12, 13, 9, 17, 11, 11, 13, 8, 15)$, and $n_{mis} = 2$. We monitor all model parameters listed in Section 3.2.1, as well as imputations of the two missing data points and cross products of the vector $\boldsymbol{\eta}$.

3.2.3 Validation Results: Correctly written software

We first present results from testing the correctly coded WinBUGS software.

Scalar Validation. The z_ζ statistics from this simulation are plotted in Figure 6; y_{mis} refers to the imputed values of missing data and $\boldsymbol{\eta} \times \boldsymbol{\eta}$ refers to the cross products of $\boldsymbol{\eta}$. One of the 59 z_ζ statistics is significant at the 5% level and three are nearly significant, which is not surprising considering that 59 $\times$ .05 = 2.95.

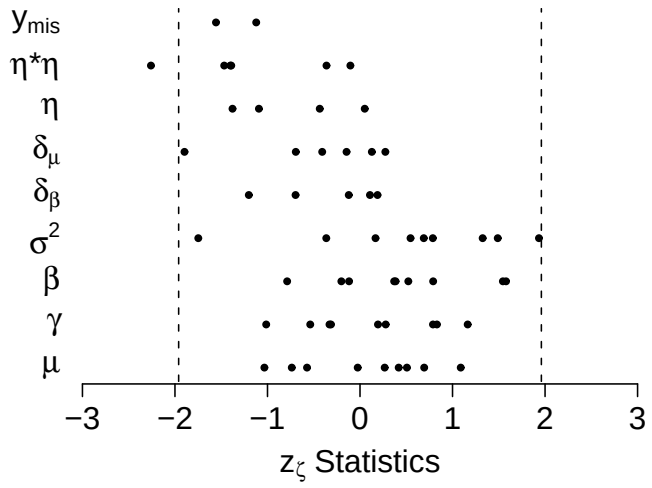

Figure 6: Scalar validation z_ζ Statistics: Complex model, correctly written software. Each row represents a batch of parameters. Vertical lines at ± 1.96 represent a 95% bounds.

Omnibus Validation. The $z_{\boldsymbol{\theta}}$ statistic from this simulation equals -0.52 (p-value = 0.6). These results show no indication of incorrectly written software.

3.2.4 Validation Results: Error in Data Model Specification

The first error we create in the WinBUGS model-fitting software is incorrectly coding the likelihood as

$$y_{ij} \sim N(\mu_j - \exp(\beta_j) t_{ij} - \exp(\gamma_j) t_{ij}^3, \sigma_j^2), \tag{7}$$

thereby using t_{ij}^3 instead of t_{ij}^2 as the covariate associated with γ_j. We again perform 100 replications.

Scalar Validation. Figure 7 plots the z_ζ statistics, with some larger than 20. The parameters with the largest deviations from uniformity are those related to $\boldsymbol{\gamma}$. The z_ζ statistics for the nine values of γ_j are all larger than 15. Most of the components of $\boldsymbol{\eta}$ and $\boldsymbol{\eta} \times \boldsymbol{\eta}$ (the parameters governing the prior distribution of $\boldsymbol{\gamma}$) are highly significant. Finally, the parameters δ_{β_3} and δ_{μ_3}, the coefficients on γ_j in the prior distribution of β_j and μ_j, respectively, are also highly significant. These results strongly suggest that the error in the software is related to $\boldsymbol{\gamma}$.

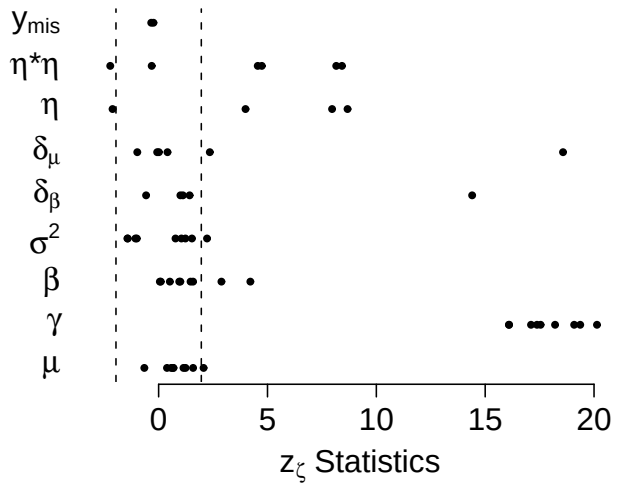

Figure 7: Scalar validation z_ζ Statistics: Complex model, error in likelihood specification. Each row represents a batch of parameters. Vertical lines at ± 1.96 represent a 95% bounds.

Omnibus Validation. The omnibus z statistic is $z_{\boldsymbol{\theta}} = 17$ (p-value ≈ 0).

3.2.5 Validation Results: Error in Hyperprior Specification

The second error we create treats $\boldsymbol{r}$ as a diagonal matrix in the WinBUGS model-fitting software, i.e., using an independent prior distribution on $\boldsymbol{\eta}$. The correlation matrix used to sample $\boldsymbol{\eta}$ in the data-generating software is

$$\boldsymbol{r} = \begin{bmatrix} 1 & 0.57 & 0.18 & 0.56 \\ 0.57 & 1 & 0.72 & 0.16 \\ 0.18 & 0.72 & 1 & 0.14 \\ 0.56 & 0.16 & 0.14 & 1 \end{bmatrix}.$$

Scalar Validation. Again, the z_ζ statistics suggest there is an error in the software and indicate the source of the error. The components of $\boldsymbol{\eta}$ and their cross-product terms have the largest z_ζ statistics; the plot in Figure 8 suggests that there is an error in the software related to $\boldsymbol{\eta}$. The z_ζ statistics are somewhat less extreme for the components of $\boldsymbol{\eta}$ than for their cross products, indicating that the error may be related to the correlations between these parameters. Because these parameters are dependent in their prior distribution, the simulation results suggest first looking for errors in the prior specification of $\boldsymbol{\eta}$.

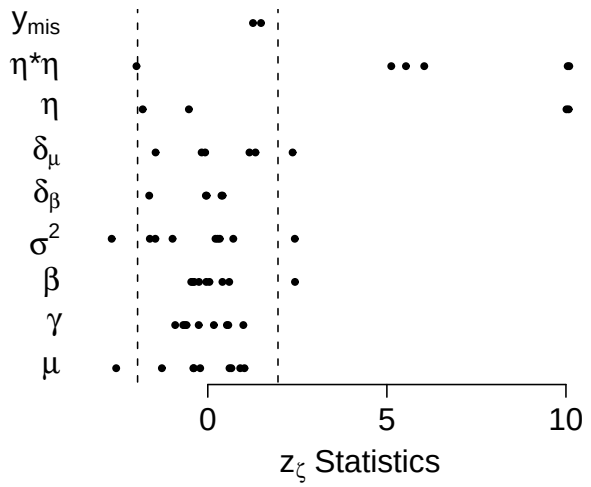

Figure 8: Scalar validation z_ζ Statistics: Complex model, error in hyperprior specification. Each row represents a batch of parameters. Vertical lines at ± 1.96 represent 95% bounds.

Omnibus Validation. Finally, the $z_{\boldsymbol{\theta}}$ statistic for this simulation equals 7 (p-value $< 10^{-10}$).

These errors are in the specification of the model sent to WinBUGS and are not a problem with WinBUGS itself; the results from Section 3.2.3 show that WinBUGS works properly when the model is correctly specified.

4 Conclusions

Substantial time and energy have been spent developing new statistical model-fitting techniques. Indeed, entire books (e.g., Gilks *et al.*, 1996; Liu, 2001) have been devoted to the application of MCMC methods (e.g., hybrid sampling, reversible jump, slice sampling), and each new method is generally presented with theoretical results proving its

validity. For these methods to work properly in practice, the algorithms must also be programmed correctly. Here we have presented a simulation-based method for testing the correctness of Bayesian model-fitting software. Although potentially computationally expensive, the methodology is straightforward and generally easy to implement once the model-fitting software itself has been developed. Of course, results from the type of simulation presented here do not "prove" that a piece of software is written correctly. Rather, as with hypothesis testing, they may simply provide no evidence against a null hypothesis that the software works properly. With an infinite number of replications, our proposed method will find errors if they exist; with finite N_{rep}, the power of the method depends on the nature of the errors. When software is written incorrectly, the simulation results often provide specific clues to where in the program the errors are located. The examples presented here indicate that when higher-level parameters are being sampled incorrectly, their z_ζ statistics can be extreme, whereas the z_ζ statistics for other parameters may appear approximately standard normally distributed. When z_ζ statistics are extreme for all parameters, it is more likely that the error is in sampling a lower-level parameter, e.g., a parameter at the likelihood level. Based on the example of Section 3.2.5, we recommend monitoring all cross-products of parameters that are not independent in their prior distribution. Our method implicitly assumes that in each replication data are generated correctly; an error in the data-generating software could lead to incorrect conclusions about the correctness of the model-fitting software. This same point is made by Geweke (2004).

The methods presented here assume that the posterior distributions have been calculated exactly, i.e., that the posterior sample is large enough that there is no uncertainty in the posterior quantiles. In the examples presented here we used posterior samples of size 5,000. When computing time is an issue, our recommendation based on limited experience is to perform fewer replications rather generating smaller posterior samples. The example in Section 3.2 suggests that the proposed validation methods can work well even with a fairly limited number of replications. When hierarchical models have many "person-level" parameters, the internal replication of parameters can allow for software testing with a single replication. Although not all parameters in the model can be formally tested with a single replication, such analyses could be helpful for screening purposes to detect obvious errors before performing a large expensive simulation.

Because models are often changed slightly over the course of a project and used with multiple data sets, we often would like to confirm that software works for a variety of conditions. In this case, we recommend performing simulations

under a variety of conditions. To help ensure that errors, when present, are apparent from the simulation results, we caution against using "nice" numbers for fixed inputs or "balanced" dimensions in these simulations. For example, consider a generic hyperprior variance parameter s^2. If software were incorrectly written to use s instead of s^2, the software could still appear to work correctly if tested with the fixed value of s^2 set to 1 (or very close to 1), but would not work correctly for other values of s^2.

Parallel (or distributed) computing environments can decrease computation time significantly: Because each data set is generated and analyzed independently of the others, multiple replications can be carried out simultaneously. When varying the simulation conditions, choosing small data sample sizes can speed computation. When the model-fitting software is based on MCMC, computing time can also be decreased by using the true parameter values as the starting values of the Markov chains to minimize convergence issues. Even when starting Markov chains at the true parameter value, convergence should be monitored, e.g., using multiple chains and the Gelman and Rubin (1992) $\sqrt{\hat{R}}$ statistic. A lack of convergence may sometimes indicate an error in the software as well.

The software validation simulations presented here should not be confused with model-checking strategies. The simulations described in Section 2 only test that a piece of software is producing the correct posterior distribution implied by the assumed Bayesian model and the observed data. Model checking is another important area of research and is discussed, for example, by Rubin (1984), Gelman and Meng (1996), Bayarri and Berger (1999), and Sinharay and Stern (2003).

The simulation studies presented here can only be applied to Bayesian models. Moreover, these validation simulations are technically only applicable for Bayesian models with proper prior distributions, because the "true" parameter vector $\theta^{(0)}$ must be repeatedly generated from its prior distribution. This provides strong incentive in practice to use prior distributions that are proper (i.e., integrable), so that the resulting software can be tested in a principled way. Most distributions can be made arbitrarily diffuse while still remaining proper, so there is generally little argument in favor of using improper prior distributions rather than diffuse proper distributions. Proper prior distributions are also required when using WinBUGS software to fit Bayesian models. Generating data from very diffuse prior distributions can sometimes lead to overflow problems when fitting the models, or, in the case of Metropolis-type algorithms that require fine-tuning of jumping distributions, can make it difficult to develop a single algorithm that will work efficiently for all data sets generated from the model. As mentioned previously, if software is tested with different values of the

fixed inputs to the hyperprior distribution, it is generally reasonable to conclude it will work for other values of the fixed inputs as well. The fixed inputs used in the simulations may then be chosen so that the prior distribution is less diffuse.

Software checking techniques exist for non-Bayesian methods as well. For example, in a maximum likelihood analysis one may calculate the derivative of the likelihood function at the maximum likelihood estimate to confirm that it is equal to zero; when maximum likelihood estimates are computed using the EM algorithm (Dempster *et al.*, 1977) one can check that the observed-data log-likelihood increases at each iteration. Most frequentist confidence intervals are exact only asymptotically and therefore cannot be reliably used to test software with finite samples; however, monitoring interval coverage could work for models that yield frequentist confidence intervals that are exact in the sense of Neyman (1934). We encourage principled software validation whenever possible.

References

Agans, D. (2002). *Debugging*. Amacom, New York.

Bayarri, M. and Berger, J. (1999). Quantifying surprise in the data and model verification. In J. Bernardo, J. Berger, A. Dawid, and A. Smith, eds., *Bayesian Statistics 6*. Oxford University Press, Oxford, U.K.

Cook, S. R. (2004). Modeling monotone nonlinear disease progression and checking the correctness of the associated software. Ph.D. Thesis, Harvard University.

Dempster, A., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood estimation from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B* **39**, 1–38.

Gelfand, A. and Smith, A. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* **85**, 398–409.

Gelman, A. (1992). Iterative and non-iterative simulation algorithms. *Computing Science and Statistics* **24**, 433–438.

Gelman, A. and Meng, X. (1996). Model checking and model improvement. In W. Gilks, S. Richardson, and D. Spiegelhalter, eds., *Practical Markov Chain Monte Carlo*. Chapman & Hall, New York.

Gelman, A. and Rubin, D. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science* **7**, 457–511. With Discussion.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transaction on Pattern Analysis and Machine Intelligence* **6**, 721–741.

Geweke, J. (2004). Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association* **99**, 799–804.

Gilks, W., Richardson, S., and Spiegelhalter, D., eds. (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall, Boca Raton, FL.

Kolmogorov, A. (1933). Sulla determinazione empirica di una legge di distribuzione. *Giornale dell' Istituto Italiano degli Attuari* **4**, 83–91.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer, New York.

Neyman, J. (1934). On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society* **97**, 558–625.

Rubin, D. B. (1984). Bayesianly justifiable and relevant frequency calculations for the applied statistician. *Annals of Statistics* **12**, 1151–1172.

Sinharay, S. and Stern, H. (2003). Posterior predictive model checking in hierarchical models. *Journal of Statistical Planning and Inference* **111**, 209–221.

Smith, A. and Roberts, G. (1993). Bayesian computation via the Gibbs sampler and related Markov Chain Monte Carlo methods (with disussion). *Journal of the Royal Statistical Society B* **55**, 3–102.

Spiegelhalter, D., Thomas, A., Best, N., Gilks, W., and Lunn, D. (1994, 2003). *BUGS: Bayesian inference using Gibbs sampling*. Medical Research Council, Biostatistics Unit, Cambridge, England. www.mrc-bsu.cam.ac.uk/bugs.

Tierney, L. (1998). A note on the Metropolis Hastings algorithm for general state spaces. *Annals of Applied Probability* **8**, 1–9.