Validation of Software for Bayesian Models using Posterior Quantiles

Samantha R. Cook
Andrew Gelman
Columbia University, Department of Statistics
New York, NY 10027

Donald B. Rubin
Harvard University, Department of Statistics
Cambridge, MA 02138

Authors' Footnote

Samantha R. Cook is Post-doctoral Fellow, Department of Statistics, Columbia University, New York, NY 10027 (email: cook@stat.columbia.edu); Andrew Gelman is Professor, Department of Statistics, Columbia University, New York, NY 10027 (email: gelman@stat.columbia.edu); and Donald B. Rubin is Professor, Department of Statistics, Harvard University, Cambridge, MA 02138 (email: rubin@stat.harvard.edu). The authors would like to thank the Associate Editor and three anonymous referees for their insightful comments and suggestions.

Abstract

We present a simulation-based method designed to establish the computational correctness of software developed to fit a specific Bayesian model, 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 complicated models, which have the ability to address adequately complications such as missing data, complex sampling schemes, and treatment noncompliance. Increasing model complexity, however, implies increasing opportunity for errors in the software used to fit such models, particularly 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; Gelman *et al.*, 2003), there is only limited work related to investigating whether the software developed to implement these algorithms works properly. We are aware of only one published paper in this area (Geweke, 2004). Standard approaches to software testing and debugging 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 in Section 2 by describing the design of the validation simulation and the analysis of the simulation output. 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

assessments of the correctness of Bayesian model-fitting software. The basic idea is that if we draw parameters from their prior distribution and draw data from their sampling distribution given the drawn parameters, and then perform Bayesian inference correctly, the resulting posterior inferences will be, on average, correct. For example, 50% and 95% posterior intervals will contain the true parameter values with probability 0.5 and 0.95, respectively. In this paper, we develop a more powerful and systematic version of this basic idea.

2.1 Theoretical Framework

Consider the general Bayesian model $p(y|\Theta)p(\Theta)$, where $p(y|\Theta)$ represents the sampling distribution of the data, y, $p(\Theta)$ represents the proper prior distribution of the parameter vector, Θ, and inferences are based on the posterior distribution, $p(\Theta|y)$. If a "true" parameter value $\Theta^{(0)}$ is generated according to $p(\Theta)$, and data y are then generated according to $p(y|\Theta^{(0)})$, then $(y, \Theta^{(0)})$ represents a draw from the joint distribution $p(y, \Theta)$, which implies that $\Theta^{(0)}$ represents a draw from $p(\Theta|y)$, the posterior distribution of Θ with y observed. The to-be-tested software is supposed to generate samples from $p(\Theta|y)$.

Now consider the posterior sample of size L, $(\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(L)})$, generated from the to-be-tested software. By sample we mean that each of the $\Theta^{(1)}, \ldots, \Theta^{(L)}$ has the same marginal distribution, not necessarily that they are independent or even jointly exchangeable. If the software is written correctly, then $(\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(L)})$ are a sample from $p(\Theta|y)$, meaning that the marginal distribution of $\Theta^{(0)}$ and each of $\Theta^{(1)}, \ldots, \Theta^{(L)}$ is the same. This is also true for any subvector of the components of Θ. Figure 1 illustrates this process of generating Θ and y.

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

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

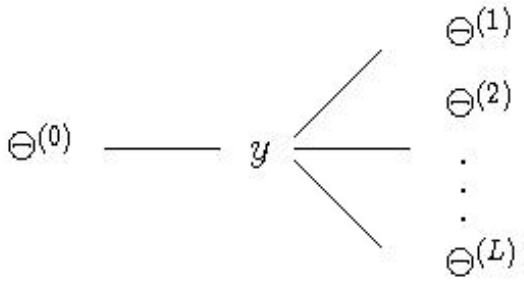

Figure 1: Illustration of data and parameter generation for validation simulation. First draw $\Theta^{(0)}$ from the prior distribution; then draw $y|\Theta^{(0)}$ from the data model; then draw $\Theta^{(1)}, \ldots, \Theta^{(L)}$ from the posterior distribution given y using the to-be-tested software. If the software is written correctly, then $(y, \Theta^{(\ell)})$ should look like a draw from $p(y, \Theta)$, for $\ell = 0, 1, \ldots, L$.

Proof. To prove this result, we need to show that $\lim_{L\to\infty} \Pr(\hat{q}(\theta^{(0)}) < x) = x$ for any $x \in [0, 1]$. Let $Q_x(f)$ represent the xth quantile of a distribution f, and $Q_x(z_1, z_2, \ldots, z_n)$ the xth empirical quantile of $(z_1, z_2, \ldots, z_n)$. Then

$$\begin{aligned}
\lim_{L\to\infty} \Pr(\hat{q}(\theta^{(0)}) < x) &= \lim_{L\to\infty} \Pr\left(\theta^{(0)} < Q_x(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)})\right) \\
&= \Pr\left(\theta^{(0)} < \lim_{L\to\infty} Q_x(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)})\right) \\
&= \Pr\left(\theta^{(0)} < Q_x(p(\theta|y))\right) \\
&= x,
\end{aligned}$$

because $\theta^{(0)}$ is drawn from $p(\theta|y)$. For independently drawn $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)})$, the third line of the above expression follows from the convergence of the empirical distribution function to the true distribution function. If the sample $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)})$ comes from a process such as MCMC that converges to the target distribution, the equation follows from the assumed ergodicity of the simulation process (see, e.g., Tierney, 1998, for MCMC simulation). Thus, as the model-fitting algorithm converges, the distribution of $\hat{q}(\theta^{(0)})$ approaches the uniform, assuming that the target distribution $p(\theta|y)$ has zero probability at any point. □

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 Figures 1 and 2, simply generate and analyze data according to the same

model, and calculate posterior quantiles. Specifically, generate the parameter vector $\Theta^{(0)}$ from $p(\Theta)$. Then conditional on $\Theta^{(0)}$, generate the data y from $p(y|\Theta = \Theta^{(0)})$. Next, analyze y 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, $(\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(L)})$, from the posterior distribution of Θ, $p(\Theta|y = y)$. Finally, for each component of $\Theta^{(0)}$, generically labeled $\theta^{(0)}$, compute its posterior quantile, $\hat{q}(\theta^{(0)})$, with respect to the posterior sample, $(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)})$. Imputed values of missing data and functions of Θ, 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*. Figures 1 and 2 outline the steps of one replication. The simulation *conditions* are specified by unmodeled data and unmodeled 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 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 $\Theta^{(0)}$ from $p(\Theta)$ and y from $p(y|\Theta^{(0)})$ as the data-generating software and the posterior simulation software that samples $(\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(L)})$ 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, which both sample from $p(\Theta|y)$, as explained in Section 2.1. Direct simulation, i.e, the data-generating software, is easier to program and is presumed to be programmed correctly; Geweke (2004) also makes this point and this assumption. It is possible, however, that an apparent error in the model-fitting software could actually be the result of an error in the data-generating software. If desired, the correctness of the data-generating software could be examined by comparing its output with analytically-derived quantities, for example, comparing sample moments of the components of Θ or of the data y with their analytically-derived expectations.

To perform the validation experiment, perform many replications, each drawing $\Theta^{(0)}$ from $p(\Theta)$ and y from $p(y|\Theta^{(0)})$, and then using the to-be-tested model-fitting software to obtain a sample from $p(\Theta|y)$. 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 detect errors in the software.

There are three levels of potential subscripting here: replications, draws of Θ within each replication, and components

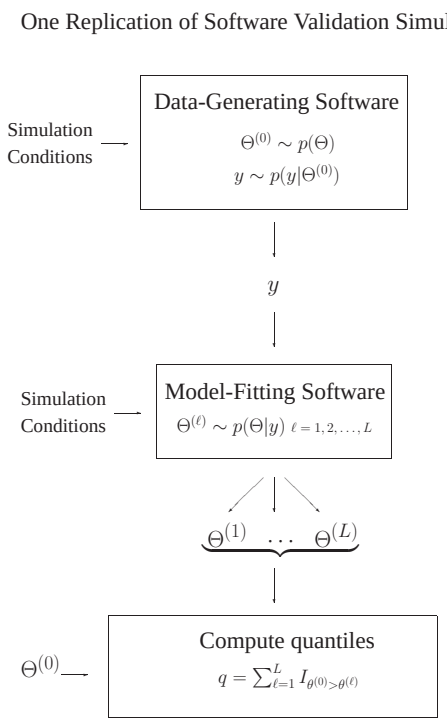

Figure 2: Steps of one replication of validation simulation. Given the simulation conditions, first generate parameters $\Theta^{(0)}$ and data y using the data-generating software. Then, given the data y, generate a sample from $p(\Theta|y)$ using the model-fitting software. Finally, from the posterior sample and $\Theta^{(0)}$, calculate posterior quantiles.

of Θ. We never indicate components of Θ with subscripts; instead, we use θ to denote a generic scalar component of Θ. We denote draws of Θ (or θ) within a replication with superscripts: $\Theta^{(\ell)}$ represents the ℓth draw, with $\ell = 1, \ldots, L$. We denote replications with subscripts: $i = 1, \ldots, N_{rep}$.

2.3 Analyzing Simulation Output

Let $q_i = \frac{1}{L} \sum_{\ell=1}^{L} I_{\theta_i^{(0)} > \theta_i^{(\ell)}}$, the empirical quantile for the ith replication. For any generic function h, we can determine the distribution of $h(q)$ for correctly working software. In particular, if the software works properly and therefore the posterior quantiles are uniformly distributed, then $h(q) = \Phi^{-1}(q)$ should have a standard normal distribution, where Φ represents the standard normal CDF. One way to test the software is to test that the mean of $\Phi^{-1}(q)$ equals 0. However, if an error caused the quantiles follow a non-uniform distribution centered near 0.5 but with mass piled up near 0 and 1, $\Phi^{-1}(q)$ could still appear to have mean zero. Our investigations revealed that posterior quantiles do

sometimes follow such a U-shaped distribution when software has errors; simply testing that $\Phi^{-1}(q)$ has mean equal to zero may therefore not find errors when they exist. Figure 3 shows a histogram of such a distribution obtained from software with an error. Instead, for each scalar θ, we calculate the following statistic:

$$X_\theta^2 = \sum_{i=1}^{N_{rep}} \left(\Phi^{-1}(q_i)\right)^2,$$

which should follow a χ^2 distribution with N_{rep} degrees of freedom when the software works correctly. We can then quantify the posterior quantiles' deviation from uniformity by calculating the associated p-value, p_θ, for each X_θ^2.

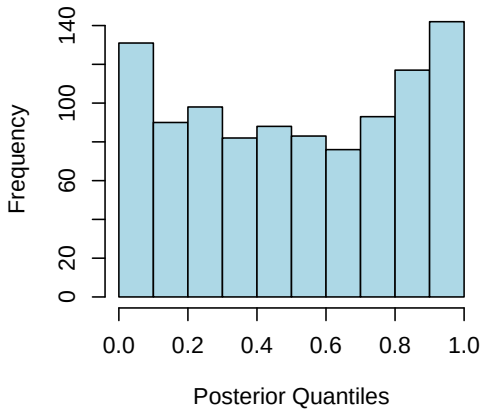

Figure 3: An example of posterior quantiles q from software with error. An effective summary for detecting the error should emphasize quantiles near 0 or 1, such as $h(q) = \left(\Phi^{-1}(q)\right)^2$.

Any extreme p_θ value indicates an error in the software. If the number of parameters is large, however, we expect some of the p_θ values to be small even when the software does work properly. We therefore recommend two complementary approaches for analyzing the simulation output.

As an exploratory tool, we transform the p_θ values into z_θ statistics ($z_\theta = \Phi^{-1}(p_\theta)$) and plot the absolute values of the z_θ statistics in "batches" that are related (examples appear in Figures 4 -9). For example, a batch might consist of the individual-level means in a hierarchical model. Plotting z_θ rather than p_θ makes extreme values more evident.

As a confirmatory analysis, we perform a Bonferroni correction on a representative subset of the p_θ values. This subset of p_θ values is defined as follows. For each "batch" of related parameters, define a new scalar parameter equal to the mean of all parameters in the batch. The p_θ values for these new parameters can be calculated from the simulated values

of Θ (with no calculation required for batches containing only one parameter) and so no alteration of the model-fitting software is required. The Bonferroni correction simply multiplies each p_θ value in the subset by the total number of batches. From these Bonferroni-adjusted p_θ-values, testing can be carried out at any desired significance level. Using the means of the batched parameters can result in a much smaller number of p_θ values used in the multiple comparisons procedure, and therefore a less conservative procedure. Such a strategy is especially appropriate for groups of parameters sampled using the same code segment (e.g., in a loop), because any error in that segment would generally affect all parameters in the batch as well as their mean. This multiple comparisons procedure borrows from ideas in meta-analysis (e.g., Rosenthal and Rubin, 1986): Each batch of parameters can represent multiple outcomes from a single study; we are first combining within-batch results and then combining batch-level summaries into a single conclusion.

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 the special case of our method where $h(q) = I_{\{0.025<q<.975\}}$. Using $h(q) = q$ or $h(q) = \left(\Phi^{-1}(q)\right)^2$ effectively assesses the coverage of all possible intervals at once.

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 draws from the joint distribution $p(\Theta, y)$. One of the programs (the marginal-conditional simulator) is equivalent to our data-generating software, creating independent samples from $p(\Theta, y)$ by first sampling Θ from $p(\Theta)$ and then sampling y from $p(y|\Theta)$; this piece of software should be straightforward to write and is presumed to be error-free. The second program (the successive-conditional simulator) is created by appending to the algorithm that samples from $p(\Theta|y)$ an additional step that samples y from $p(y|\Theta)$; the limiting distribution of this new algorithm is also $p(\Theta, y)$. The data y are sampled from $p(y|\Theta)$ in both algorithms, but the two algorithms are different because they sample Θ differently. Thus two samples from $p(\Theta, y)$ are generated; z statistics are then calculated to test that the means of (scalar) components and functions of (Θ, 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 $p(\Theta, y)$ rather than repeatedly generating from $p(\Theta|y)$ 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. WinBUGS, for example, cannot be altered to resample the data y at each iteration, and so the successive-conditional simulator cannot be created. 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 (Θ, 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 $a \leq 2$. Because our method uses sample quantiles rather than moments, it does not require that the prior distribution have a finite variance and so can more easily be used to check software for models with vague prior distributions. Of course, generating Θ from nearly improper prior distributions could lead to computational problems with either method.

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:

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

Because the prior distribution is conjugate, it is straightforward to derive the full conditional distribution for each parameter, i.e., its distribution given the data and all other parameters, 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 of 20 replications to validate the MCMC software developed to fit the simple model presented in Section 3.1.1. The sample sizes for the generated data are $J = 6$, $n = (33, 21, 22, 22, 24, 11)$. 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 $\alpha_j/\sigma, j = 1, \ldots, J$. In addition, we monitor $\overline{\alpha} = \sum_{j=1}^{J} \alpha_j$ and $\overline{\alpha}/\sigma$. We perform the Bonferroni correction on the p_θ values for $(\overline{\alpha}, \overline{\alpha}/\sigma, \mu, \tau^2, \sigma^2, \mu/\tau)$ by multiplying each of these p_θ values by 6. In other words, we group the model parameters into 6 batches: one containing $\alpha_1, \ldots, \alpha_J$; one containing $\alpha_1/\sigma, \ldots, \alpha_J/\sigma$; and the remaining four containing one of μ, τ^2, σ^2, μ/τ.

3.1.3 Validation Results: Correctly Written Software

The absolute z_θ statistics from this simulation are plotted in Figure 4. Each row in the plot represents a different batch of parameters. All z_θ statistics are less than 2, thus providing no indication of incorrectly written software. Moreover, the smallest of the unadjusted p_θ values for $(\overline{\alpha}, \overline{\alpha}/\sigma, \mu, \tau^2, \sigma^2, \mu/\tau)$ is 0.2, making its Bonferroni-adjusted p value larger than 1.

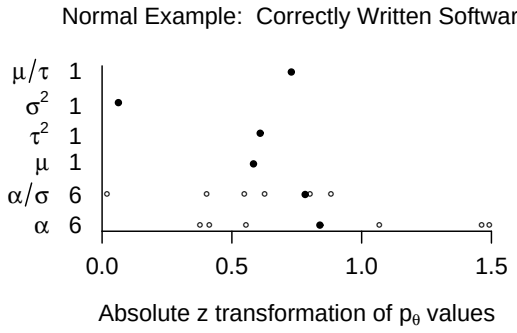

Figure 4: Scalar validation z_θ statistics: Simple model, correctly written software. Each row represents a scalar parameter or batch of parameters; the circles in each row represent the z_θ statistics associated with that parameter or batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

3.1.4 Validation Results: Incorrectly sampling α

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 full 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 20 replications.

Figure 5 plots the absolute z_θ statistics from this simulation; note the scale of the x axis. It is clear that there is an error somewhere in the software: All of the z_θ statistics are extreme. The parameters with the smallest z_θ statistics are τ^2 and the coefficient of variation μ/τ, whereas the parameters $\boldsymbol{\alpha}$, σ^2, μ, and $\boldsymbol{\alpha}/\sigma$ all have even more extreme z_θ statistics. In addition, the smallest Bonferroni-adjusted p_θ value is essentially zero, clearly indicating a problem. In such a situation we recommend first 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_θ statistics.

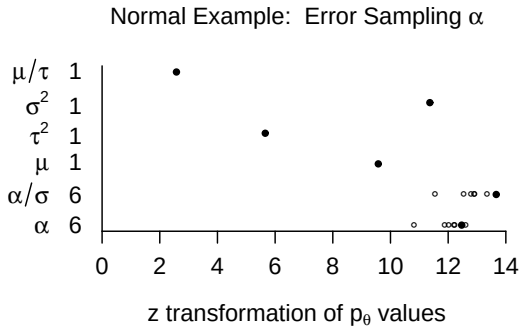

Figure 5: Scalar validation z_θ statistics: Simple model, incorrectly sampling $\boldsymbol{\alpha}$. Each row represents a scalar parameter or batch of parameters; the circles in each row represent the z_θ statistics associated with that parameter or batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

3.1.5 Validation Results: Incorrectly Sampling μ

For another example, we consider an error in the specification of the full conditional distribution of μ in the Gibbs sampler, which is normal with mean $\frac{\sum_{j=1}^{J}\alpha_j}{\tau^2}+\frac{5}{5^2}$ 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 $\frac{\sum_{j=1}^{J}\alpha_j}{\tau^2}+\frac{5}{5}$ and variance $\frac{1}{\frac{J}{\tau^2}+\frac{1}{5}}$, and again perform a simulation of 20 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.

As can be seen from Figure 6, the absolute z_θ statistics for μ and μ/τ are extreme, suggesting an error in the part of the software that samples μ. The smallest Bonferroni-adjusted p_θ value (that associated with μ) equals 0.002.

3.2 A Hierarchical Repeated-Measures Regression Model

We now present an example of software validation for a much more complicated model.

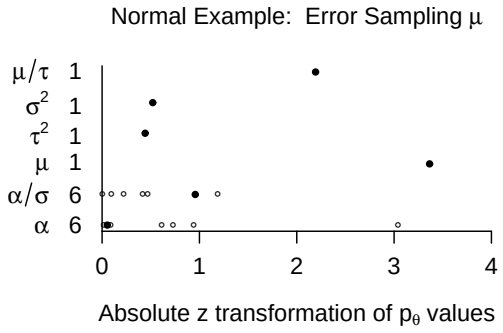

Figure 6: Scalar validation z_θ statistics: Simple model, incorrectly sampling μ. Each row represents a scalar parameter or batch of parameters; the circles in each row represent the z_θ statistics associated with that parameter or batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

3.2.1 Model

This model was developed to impute missing data in a clinical trial. The data are repeated blood measurements, and a Bayesian hierarchical regression model is used 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:

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

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 represents 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 $\boldsymbol{\xi} = (\boldsymbol{\eta}, \boldsymbol{\delta}_\beta, \boldsymbol{\delta}_\mu)$. 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

We will present results for three validation simulations; each simulation consists of 20 replications and requires approximately 2.5 hours of computing time. Each replication generates a sample of $L = 5{,}000$ draws from the posterior distribution. 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 $\boldsymbol{\eta}$. We perform the Bonferroni correction on the p_θ values for the means of the following nine vector parameters: $(\boldsymbol{\mu}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\sigma}^2, \boldsymbol{\eta}, \boldsymbol{\eta} \times \boldsymbol{\eta}, \boldsymbol{\delta}_\beta, \boldsymbol{\delta}_\mu, \boldsymbol{y}_{mis})$, where $\boldsymbol{y}_{mis}$ refers to the imputed values of missing data and $\boldsymbol{\eta} \times \boldsymbol{\eta}$ refers to the cross products of $\boldsymbol{\eta}$.

3.2.3 Validation Results: Correctly written software

We first present results from testing the correctly coded WinBUGS software. The absolute z_θ statistics from this simulation are plotted in Figure 7. None of the z_θ statistics is extreme, thus providing no indication of incorrectly written software. The smallest of the Bonferroni-adjusted p-values is larger than 1.

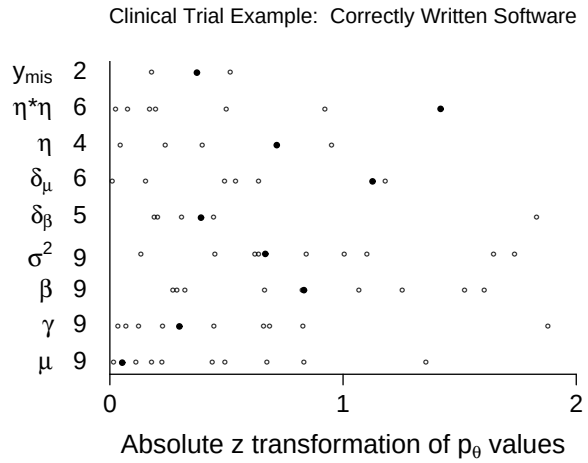

Figure 7: Scalar validation z_θ statistics: Complex model, correctly written software. Each row represents a batch of parameters; the circles in each row represent the z_θ statistics associated with that batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

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

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

thereby using t_{ij}^3 instead of t_{ij}^2 as the covariate associated with γ_j. We again perform 20 replications.

Figure 8 plots the absolute z_θ statistics, showing that many are extreme. The parameters with the largest deviations from uniformity are those related to $\boldsymbol{\gamma}$. The z_θ statistics for the nine values of γ_j are all larger than six, and several components of $\boldsymbol{\eta}$ and $\boldsymbol{\eta} \times \boldsymbol{\eta}$ (the parameters governing the prior distribution of $\boldsymbol{\gamma}$) are also extreme. The smallest of the nine Bonferroni-adjusted p_θ values is that associated with $\boldsymbol{\gamma}$ and is equal to 7.1×10^{-29}, clearly indicating an error related to sampling $\boldsymbol{\gamma}$.

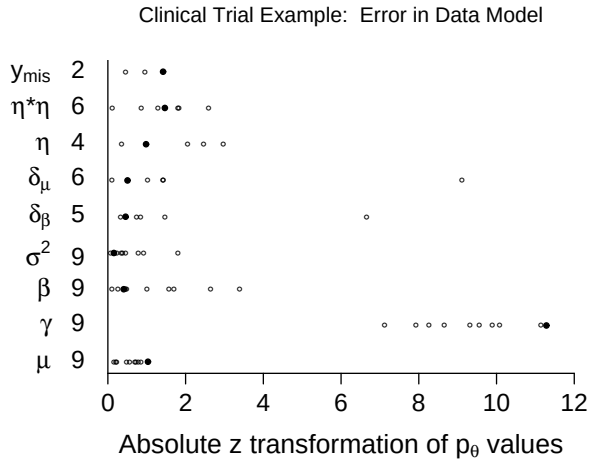

Figure 8: Scalar validation z_θ statistics: Complex model, error in likelihood specification. Each row represents a batch of parameters; the circles in each row represent the z_θ statistics associated with that batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

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 a prior distribution that is independent across components of $\boldsymbol{\eta}$. The correlation matrix used to sample $\boldsymbol{\eta}$ in the data-generating

software is

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

Figure 9 plots the absolute z_θ statistics for this simulation. Again, the simulation results 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_θ statistics, suggesting that there is an error in the software related to $\boldsymbol{\eta}$. The z_θ 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 $\boldsymbol{\eta}$. The smallest of the Bonferroni-adjusted p_θ values is that associated with $\boldsymbol{\eta} \times \boldsymbol{\eta}$ and is equal to 8.2×10^{-15}.

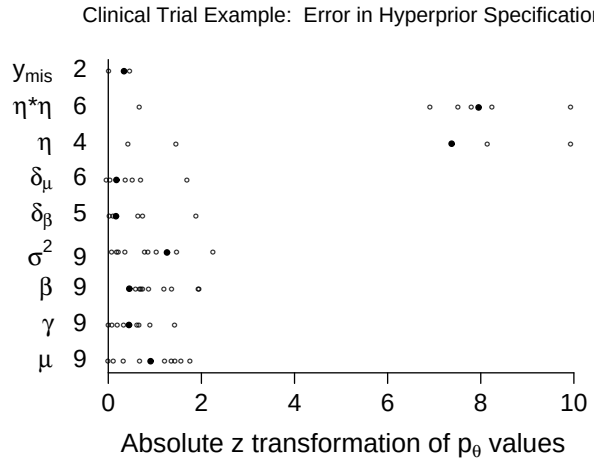

Figure 9: Scalar validation z_θ statistics: Complex model, error in hyperprior specification. Each row represents a batch of parameters; the circles in each row represent the z_θ statistics associated with that batch of parameters. Solid circles represent the z_θ statistics associated with the mean of that batch of parameters. The numbers on the y axis indicate the number of parameters in the batch.

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 for this model when the model is correctly specified.

3.3 Comparison with the Method of Geweke (2004)

As discussed in Section 2.4, Geweke's method compares output from two pieces of software that sample from the distribution $p(\Theta, y)$, whereas our method repeatedly compares output from two programs that sample from $p(\Theta|y)$ for many values of y. One might expect Geweke's method to be substantially faster than ours because it requires only a single replication; however, it also requires sampling from a larger model which may take longer to converge.

For the examples in Section 3.1, implementing Geweke's method in fact requires *more* computing time than our method. The simulations in Sections 3.1.3 - 3.1.5 each required about one minute of computing time, using in each replication the convergence criterion that the Gelman and Rubin (1992) statistic $\sqrt{\hat{R}} < 1.05$ for all parameters monitored. Performing Geweke's single-replication method, even using the less stringent convergence criterion of $\sqrt{\hat{R}} < 1.2$ for all parameters, required about four minutes of computing time for the successive-conditional simulator to converge. The results from the two methods were similar: Both methods found existing errors and did not indicate errors in the correctly written software.

For the examples in Section 3.2, Geweke's method cannot be used because the successive-conditional simulation cannot be implemented in WinBUGS. This is a clear advantage of our method in such cases, because there are models for which a correctly coded WinBUGS program does not sample from the correct target distribution. For complex models especially, it can be important to test that a specific WinBUGS program works properly.

More generally, we expect Geweke's method to be more useful in some settings and ours to work better in others. It is also possible for both methods to fail; for example, if software intended to sample from the posterior distribution actually samples from the prior distribution, neither our method nor Geweke's will find the problem.

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. Software for implementing these validation methods is available at *http://www.stat.columbia.edu/~cook*.

4.1 Detecting Errors

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. The power of the method depends on the nature of the errors.

In our examples, the simulation results provide specific clues to where in the program the errors are located. 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. Monitoring cross-products can also be helpful to detect indexing errors, for example, drawing a subvector of $\Theta^{(t+1)}$ conditional on $\Theta^{(t-1)}$ rather than $\Theta^{(t)}$ in an MCMC algorithm. An algorithm with this type of error could still yield correct marginal distributions; monitoring cross-products can reveal errors in the joint distribution.

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 scale parameter s. If software were incorrectly written to use s^2 instead of s, the software could still appear to work correctly if tested with the fixed value of s set to 1 (or very close to 1), but would not work correctly for other values of s.

4.2 Computing Time

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 we used posterior samples of size 5000. When computing time is an issue, our recommendation, based on limited experience, is to perform fewer replications rather than to generate smaller posterior samples. Our examples suggest that the proposed validation methods can find errors (at least serious ones) when they exist even with a limited number of replications. Additionally, the p_θ values are exact in the sense that they do not assume or require a large number of replications.

When hierarchical models have many individual-level parameters, the internal replication of parameters can allow for software testing with a single replication. Such analyses could be helpful for screening purposes to detect and correct obvious errors before performing a large expensive simulation. For example, after only a single replication of the simulation in Section 3.1.4 (incorrectly sampling $\boldsymbol{\alpha}$), 15 of the 16 p_θ values calculated were less than 0.05, clearly suggesting an error after only a few seconds of computing time. Sequential testing strategies may also be helpful when computing time is a concern, stopping the simulation early if an extreme result appears. Formal methods for sequential testing are described, for example, in Armitage *et al.* (1969).

Parallel (or distributed) computing environments can decrease computation time: Because each replication is performed independently of the others, multiple replications can be carried out simultaneously. When varying the simulation conditions, choosing small data sample sizes can speed computation; even complex models can often be fit fairly quickly when sample sizes are small.

The number of replications performed also necessarily depends on the purpose for which the software is being developed. For commercial or public-use software (e.g., a SAS procedure), it may be desirable to perform hundreds of replications.

4.3 Proper Prior Distributions

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 (i.e., integrable) 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, so that the resulting software can be tested in a systematic 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. In addition, using a proper prior distribution guarantees a proper posterior distribution.

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 relatively 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, which can also speed the rate of convergence of the model-fitting algorithm, and thereby speed the validation simulation.

4.4 Related Methods

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 *et al.* (1996), Bayarri and Berger (1999), and Sinharay and Stern (2003). The model checking methods of Box (1980) and Dey *et al.* (1998) are similar to our software checking method and involve comparison of either the observed data or observed posterior distribution with the distribution of the data or parameters implied by the model.

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.

Armitage, P., McPherson, C., and Rowe, B. (1969). Repeated significance tests on accumulating data. *Journal of the Royal Statistics Society, Series A* **132**, 235–244.

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.

Box, G. (1980). Sampling and Bayes' inference in statistical modelling and robustness (with discussion). *Journal of the Royal Statistical Society, Series B* **143**, 383–430.

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.

Dey, D., Gelfand, A., Swartz, T., and Vlachos, P. (1998). Simulation based model checking for hierarchical models. *Test* **7**, 325–346.

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., Carlin, J., Stern, H., and Rubin, D. (2003). *Bayesian Data Analysis, 2nd Edition*. Chapman & Hall, London.

Gelman, A., Meng, X.-L., and Stern, H. (1996). Posterior predictive assessment of model fitness via realized discrepancies (with discussion). *Statistica Sinica* **6**, 733–807.

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.

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.

Rosenthal, R. and Rubin, D. B. (1986). Meta-analytic procedures for combining studies with multiple effect sizes. *Psychological Bulletin* **99**, 400–406.

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.

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