Fitting a Bayesian Linear Model Using R and C++: bayesMCMC R Package

By Liam Abbott and David Reynolds

Introduction:

In a Bayesian framework, parameters are random variables with prior and posterior distributions. Prior distributions represent preconceived beliefs about the values the parameters may take on, while the posterior distributions take into account both those prior beliefs and the evidence provided by collected data.

In this report, the parameters we are interested in are those involved in the standard multiple linear regression model. We have n observations of a response variable y and a set of predictor variables $X=\left(X_0,X_1,X_2,...,X_p\right)$, where $X_0=1$ for all observations, representing a constant effect.

We model the relationship between the responses and the predictors using the equation:

$$y=X\beta+\varepsilon\ ,$$

where $\beta=\left(\beta_0,\beta_1,...,\beta_p\right)^T$ are the coefficient parameters, $\varepsilon=\left(\varepsilon_1,\varepsilon_2,...,\varepsilon_n\right)^T$ are the individual error terms for each observation, and each error term $\varepsilon_i\sim N\left(0,\sigma^2\right)$.

The conditional joint probability density of the vector of responses y given values of σ^2 and β can be written as:

$$f\left(y\mid\beta,\sigma^2\right)=\left(\frac{1}{2\pi\sigma^2}\right)^{n/2}\exp\left(-\frac{1}{2\sigma^2}||y-X\beta||^2\right)\ .$$

The parameters we are interested in estimating are σ^2 and β. Given the Bayesian framework laid out above, we would like to incorporate both our prior beliefs about the parameters and the observed data when estimating values of σ^2 and β.

To begin, we characterize our prior beliefs about σ^2 and β by assigning them the following prior distributions:

$$\sigma^2\sim IG(a,b)\qquad\qquad\beta\mid\sigma^2\sim N\left(0,\frac{\sigma^2}{\kappa}I_{p+1}\right)$$

for fixed values a, b, and κ.

Combining the densities of both of these prior distributions, we get the following joint prior density for σ^2 and β:

$$\pi\left(\beta,\sigma^2\right) \propto \left(\frac{1}{\sigma^2}\right)^{a+\frac{p+1}{2}+1} \exp\left(-\frac{b}{\sigma^2} - \frac{\kappa\left\|\beta\right\|^2}{2\sigma^2}\right)$$

Using Bayes' formula, we can combine the joint prior density and the conditional density of the data to obtain the joint posterior density of σ^2 and β. The posterior density we obtain is as follows:

$$\pi\left(\beta,\sigma^2 \mid y\right) \propto \left(\frac{1}{\sigma^2}\right)^{a+\frac{n}{2}+\frac{p+1}{2}+1} \exp\left(-\frac{b}{\sigma^2} - \frac{\kappa\left\|\beta\right\|^2 + \left\|y - X\beta\right\|^2}{2\sigma^2}\right),$$

where we omit a normalizing term that is constant with respect to β and σ^2, as it does not affect the inference process.

The goal of the techniques discussed in the following section and implemented in the included "bayesMCMC" R package is to obtain sample draws of σ^2 and β from this joint posterior density. Obtaining these sample draws is what we refer to as "fitting" the linear regression model in the Bayesian context.

Note that in this particular case, the joint posterior density takes on the form of a normal density and so we would be able to obtain our posterior sample draws using simpler, more traditional methods. However, the methods discussed in the following section give us a way to sample from the posterior density even in more complicated situations where the posterior density is not as easily recognizable.

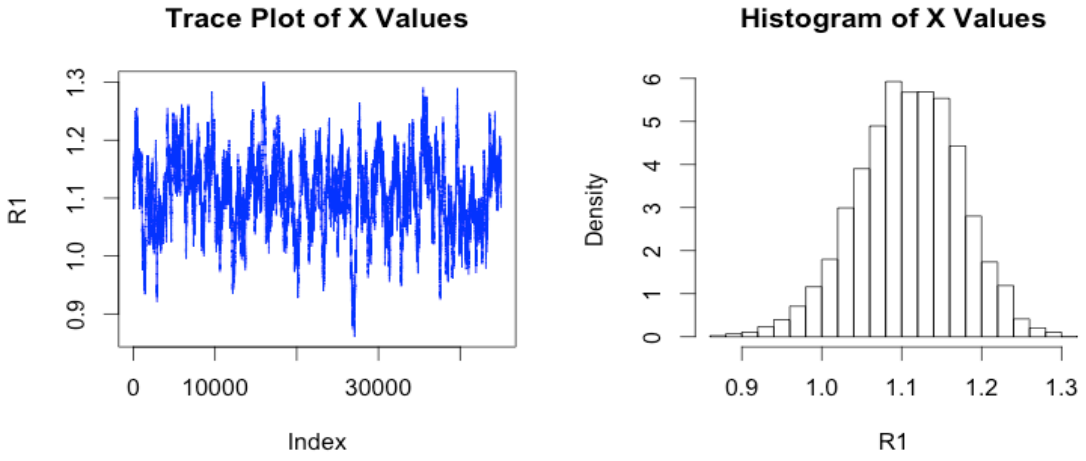
Methods:

<u>Metropolis-Hastings:</u>

In the Metropolis Hastings algorithm, we take a random walk through the parameter space and favor parameter values that have a relatively high posterior probability. At each step in the Markov Chain, there is a proposed step to a new 'area' in the parameter space. The direction and magnitude of the step itself is sampled randomly from a proposal distribution. The proposed step is accepted or rejected according to the relative density of the posterior at the proposed position and the current position. If the posterior density is higher at the proposed position than at the current position, then the step to the proposed value is definitely

accepted. If it is lower at the proposed position, the step is accepted with a probability equal to the ratio of the posterior densities.

One area of difficulty in practice is deciding if your MCMC algorithm has undergone enough iterations and whether it has done a decent job of converging on parameter values. The dominant approach we take in assessing results is viewing trace plots of parameter draws (where we seek stationarity) and also in visualizing histograms of parameter values (where we seek normality). Two such charts are seen below.



After implementing our Metropolis Hastings algorithm in R and C++, we find a significant performance gain by way of the C++ implementation.

Gibbs Sampler:

An alternative method for sampling from the joint posterior density is the Gibbs sampler. To use the Gibbs sampler algorithm, the only requirement is that we are able to sample directly from the conditional posterior densities of each of the parameters given the observed data and values of the other parameters.

From the joint posterior density discussed in the introduction, we can derive the following conditional posterior densities for σ^2 and β:

$$\sigma^2 \mid y,\beta \sim IG\left(a+\frac{n+p+1}{2}, b+\frac{\kappa||\beta||^2+||y-X\beta||^2}{2}\right)$$

$$\beta \mid y,\sigma^2 \sim N\left(\left(\kappa I_{p+1}+X^TX\right)^{-1}X^Ty, \sigma^2\left(\kappa I_{p+1}+X^TX\right)^{-1}\right)$$

With these densities written down, the algorithm is relatively straightforward:

1. Start with initialize β estimates (can be $\beta_0 = \beta_1 = \ldots = \beta_p = 0$)
2. Sample from the conditional posterior density of $\sigma^2 \mid y, \beta$
3. Use the new sample σ^2 draw to sample from the conditional posterior density of $\beta \mid y, \sigma^2$
4. Using new sample β draws, repeat from step 2

We implemented this Gibbs sampler algorithm first in R and then in C++ using Rcpp and RcppEigen to see if using C++ source code would improve performance of the algorithm. Results using a simulated data example are discussed below.

Example:

To illustrate the use of Gibbs sampler algorithm, we provide an example using simulated data. The dataset is simulated using the following R code:

```
# simulate data
set.seed(2)
n = 10000
beta_star = c(1, -12, 50, 22)
p = length(beta_star)
sigmasq_true = 0.1
X = cbind(rep(1, n), matrix(rnorm((p-1)*n), ncol=(p-1)))
y = X %*% beta_star + sqrt(sigmasq_true)*rnorm(n, 1)
```

The simulated dataset contains 10,000 observations of a response variable y and four predictor variables $X = (X_0, X_1, X_2, X_3)$, where the predictor $X_0 = 1$ for each observation.

For each observation i, y_i is simulated as a normal random variable centered around a mean of $\sum_{j=0}^{3} \beta_j x_j$ with a variance of σ^2. These "true" values of β and σ^2 are chosen to be $\beta = (\beta_0, \beta_1, \beta_2, \beta_3) = (1, -12, 50, 22)$ and $\sigma^2 = 0.1$.

Using this simulated data, we obtain a sample of size $100{,}000$ of σ^2 and β draws from the joint posterior density using the Gibbs sampler algorithm with $nMC = 100{,}000$ iterations. We run the algorithm using the version coded natively in R, and the version coded using Rcpp and RcppEigen to compare the performance between the versions:

```r
# R version
system.time(gibbsSampler(nMC = 100000, y=y, X=X))
```

```
##    user  system elapsed
##  45.251   0.311  45.749
```

```r
# Rcpp/RcppEigen version
system.time(gibbsSamplerC(nMC = 100000, y=y, X=X))
```

```
##    user  system elapsed
##   3.587   0.018   3.622
```

Stan Implementation

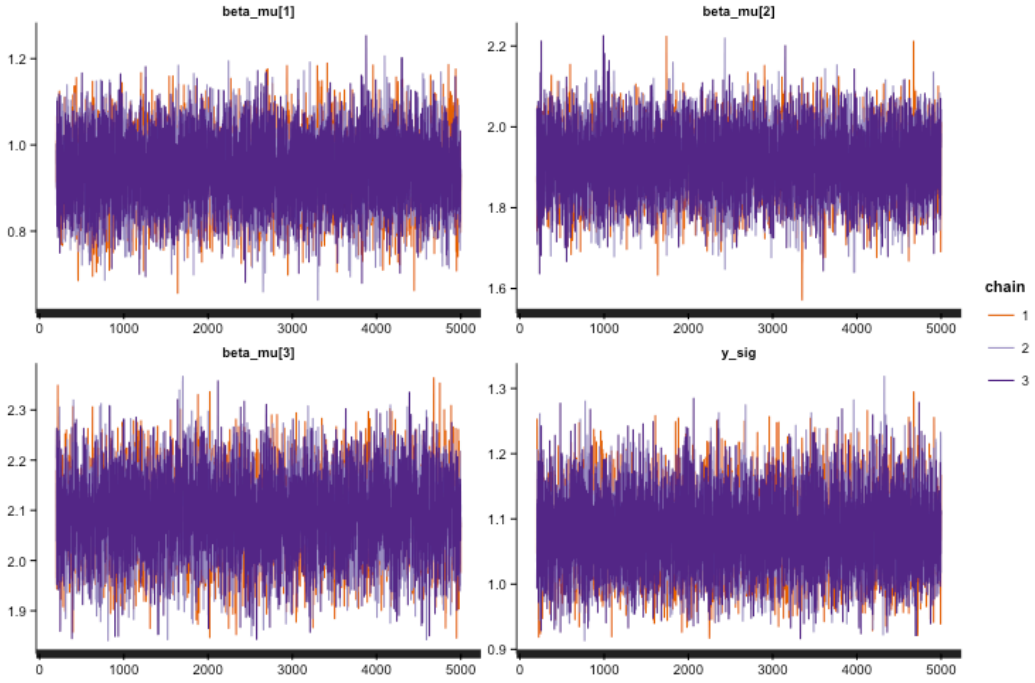
As a final step, we compare our MCMC implementations with Stan. Stan is a relatively modern probabilistic programming language implementing full Bayesian statistical inference. Stan is written in C++ and has packages and libraries for programming languages such as R, Python, and Matlab.

The main difference in Stan's implementation is the use of Hamiltonian Monte Carlo. It is important to note that in the context of the Metropolis Hastings algorithm, the proposal distribution is symmetrically centered on the current position. In complex multidimensional parameter spaces, the proposal distribution remains symmetric regardless of where you are in the parameter space. This can lead to inefficiencies in multiple dimensions. For example, in the tales of the posterior distribution, the proposals will just as often move away from the posterior mode as toward it; therefore, there will be many rejections. Stan corrects for this inefficiency by utilizing an algorithm known as Hamiltonian Monte Carlo (HMC). HMC uses a proposal distribution that changes depending on the current position. The algorithm figures out the direction in which the posterior distribution increases and directs the proposal distribution toward this gradient.

One useful analogy to describe this phenomenon is rolling a ball on the posterior distribution turned upside down. The ball tends to move down to the mode of the posterior density. For this reason, Stan's implementation of HMC often leads to more efficient sampling as compared with Metropolis Hastings and Gibbs Sampling.

However, building a basic linear regression model is not what Stan is optimized for and the Stan implementation actually takes 3x longer to run than our Metropolis Hastings implementation. We believe that as the complexity of the model increases, Stan's relative performance would drastically improve. Despite a somewhat slow run-time, Stan's performance is great, as seen from the trace plots below that represent samples from the parameters in our linear regression.

Stan generated Trace Plots of Parameter Values



Conclusion:

The versions of the Metropolis Hastings and Gibbs sampler algorithms coded using Rcpp and RcppEigen performed much better than the versions coded in R. Using a dataset containing 10,000 observations of 5 different variables, the 100,000 iterations of the Gibbs sampler algorithm in R took about 45 seconds to run, while the Rcpp/RcppEigen version took less 4 seconds. We can expect that this performance gain would only increase with larger datasets and more iterations of the algorithm.