STA 141C Final Project: MCMC with Gibbs Sampling vs Frequentist Methods for Estimating Regression Coefficients on Simulated Data

Jordan Bowman, Samuel Van Gorden, Riley Adams

23 March 2023

Abstract

[[Do we need one of these?]]

1 Introduction

Linear regression is a fundamental problem in statistical analysis and over the years there have been many methods introduced to produce better predictive results in a frequentist setting. Notable examples include the the Ridge introduced by Hoerl and Kennard (1970) [[!!!SOURCE RIDGE]] and the LASSO introduced by Tibshirani (1996) [[!!!SOURCE LASSO]] among many others. Both of these methodologies in particular function by penalizing the norm the standard OLS solution. Although both the Ridge and LASSO compute different results, the general premise is that we introduce bias and nudge the computed coefficients toward zero in order to, with a properly selected penalty term, drastically reduce the variance of the predicted coefficients and thus the overall mean squared error estimate in the regression model.

The linear regression problem can also be constructed under a Bayesian framework by assuming that the regression coefficients follow some prior distribution. By adjusting these prior, we can impart previous knowledge upon the posterior distribution of our regression coefficient estimates, and, through some measure of center, thereby impact the point estimates of our regression coefficients. In fact, Park and Casella (2008) [[!!!SOURCE BAYESIAN LASSO]] showed that the LASSO can be formulated in a Bayesian sense with properly constructed Laplace priors however that is outside the scope of this project.

The purpose of this project is to demonstrate the use of Markov Chain Monte Carlo (MCMC) methods for obtaining parameter estimates. Specifically, we will be utilizing Bayesian MCMC with Gibbs sampling in order to obtain posterior distributions of linear regression coefficients and variance under the assumption of homoskedasticity. We use noninformative conjugate priors to generate posteriors for the iterative sampling of our coefficients and variance. Finally, we compare the estimates via MSE of prediction from our Bayesian implementations to those generated using frequentist techniques such as OLS, Ridge and LASSO regression.

The advantage of using Bayesian MCMC over frequentist methods is that having posterior distributions allows us to determine probability intervals. This allows us to have a better understanding of the spread, or variance, of our parameter estimates instead of just getting a point estimate as is the case in frequentist methods. The accuracy of these estimates and probability intervals depends of course on the priors we specify, which is not something we need to worry about when using frequentist methods. Furthermore, MCMC can also be computationally costly compared to frequentist methods (so we will look at some ways to reduce the computational complexity?).

2 Proposed Methods

2.1 MCMC and Gibbs Sampling

Markov Chain Monte Carlo methods combine stochastic processes (Markov Chains) with random sampling (Monte Carlo) to estimate calculations that would otherwise be untenable with classical estimation methods.

Monte Carlo algorithms work by randomly sampling from a domain and then calculating the proportion of sampled points that meet some criteria. For example, area under a curve can be estimated by calculating the proportion of points in a rectangle of area A that lie below the curve and then multiplying that proportion by A.

Markov Chains are stochastic processes in which each state in a process depends only on the previous state and none of the earlier ones. They are incorporated with Monte Carlo techniques by allowing each sample taken to be dependent on the previous sample that was taken. In our MCMC implementation, we utilize Markov Chains to select parameter estimation samples that are based on the previously selected samples. In theory, doing so will cause the samples to gradually converge to being drawn from the true parameter distributions.

Gibbs sampling is a special case of Metropolis-Hastings sampling. In Metropolis-Hastings, each parameter estimate is drawn from a posterior distribution based on its prior and the likelihood of our data. The posterior distribution is conditioned on the data and the previously drawn values of every other parameter:

$$
\begin{aligned}
&f(\theta_1^{(t)}|\mathbf{X}, \theta_2^{(t-1)}, \theta_3^{(t-1)}, \ldots, \theta_p^{(t-1)}) \\
&f(\theta_2^{(t)}|\mathbf{X}, \theta_1^{(t)}, \theta_3^{(t-1)}, \ldots, \theta_p^{(t-1)}) \\
&f(\theta_3^{(t)}|\mathbf{X}, \theta_1^{(t)}, \theta_2^{(t)}, \ldots, \theta_p^{(t-1)}) \\
&\qquad \vdots \\
&f(\theta_p^{(t)}|\mathbf{X}, \theta_1^{(t)}, \theta_2^{(t)}, \ldots, \theta_{p-1}^{(t)})
\end{aligned}
$$

where $\theta_i^{(j)}$ refers to the j^{th} sample of the i^{th} coefficient and $\mathbf{X}$ is our dataset. Each $\theta_i^{(0)}$ must be initialized to a reasonable starting value.

Normally, in Metropolis-Hastings each sample is only accepted with a probability given by

$$\min\{1, \frac{f(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t-1)})f(\boldsymbol{\theta}^{(t-1)})}{f(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^{(t)})f(\boldsymbol{\theta}^{(t)})}\} \tag{1}$$

to that of the previous parameter draws. However, with Gibbs sampling the second component of the min operator takes value 1, and thus we always accept each draw. We are able to use Gibbs sampling in our project because we have known full conditional posteriors given our choice of priors [**?**].

MCMC involves repeating the above process for a certain number of iterations and discarding the first n iterations as a burn-in phase (when the process has not yet converged). We save the parameter values estimated in each iteration, minus the burn-in, and use these to construct a joint distribution of all parameters. From here we can find point estimates using measures of centrality—such as the mean or median–and probability intervals given these estimates and the parameter distribution.

2.2 Simple Bayesian Linear Regression

First we consider a standard linear regression model as given by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{2}$$

where response vector $\mathbf{y} \in \mathbb{R}^n$, design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, coefficients $\boldsymbol{\beta} \in \mathbb{R}^p$ and error term $\boldsymbol{\epsilon} \in \mathbb{R}^n$. Recall that each $\epsilon_i \overset{\text{iid}}{\sim} N(0, \sigma^2)$. In Bayesian linear regression we assume, as done by Scanlon (2021) [[!!! SOURCE SCANLON]] that $\boldsymbol{\beta}$ is produced by a multivariate normal distribution as follows:

$$\boldsymbol{\beta} \sim N_p(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta).$$

Futhermore, we assume that the variance of the error term σ^2 results from an inverse-gamma distribution as follows:

$$\sigma^2 \sim IG(\alpha_0, \beta_0).$$

These are the prior distributions of $\boldsymbol{\beta}$ and σ^2. Note that these priors are conjugate priors and so the full conditional posterior distributions belong to the same multivariate normal and inverse-gamma distributions respectively. As derived by Scanlon (2021) [[!!! SOURCE SCANLON]], the posterior distribution for $\boldsymbol{\beta}$ is

$$[\boldsymbol{\beta}|\cdot] \sim N_p(\mathbf{A}_\beta^{-1}\mathbf{b}_\beta, \mathbf{A}_\beta^{-1}), \tag{3}$$

where

$$\mathbf{A}_\beta = \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X} + \boldsymbol{\Sigma}_\beta^{-1}, \tag{4}$$

and

$$\mathbf{b}_\beta = \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{y} + \boldsymbol{\Sigma}_\beta^{-1}\boldsymbol{\mu}_\beta, \tag{5}$$

and the posterior distribution for σ^2 is

$$\left[\sigma^2|\cdot\right] \sim IG\left(\alpha_0 + \frac{n}{2}, \beta_0 + \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\right). \tag{6}$$

We can then use these posterior distributions to construct a Gibbs sampler to compute our regression coefficients.

As one final note, notice for a truly noniformative prior on $\boldsymbol{\beta}$ such that $\boldsymbol{\Sigma}_\beta^{-1} \to \mathbb{0}_{p\times p}$, which for a diagonal $\boldsymbol{\Sigma}_\beta = \text{diag}(s_1^2, ..., s_p^2)$ means each $s_j^2 \to \infty$, implies that

$$[\boldsymbol{\beta}|\cdot] \sim N_p\left((\mathbf{X}^\top\mathbf{X})\mathbf{X}\mathbf{y}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}\right). \tag{7}$$

Recall that this is the mean and variance estimate of the OLS solution of the linear regression coefficients $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ for a given σ^2.

2.3 Gibbs Sampler Implementation of Bayesian Linear Regression

For a large number of iterations K, we will create a Gibbs sampler in Python that iteratively samples from each of the posterior distributions described in the previous section for each iteration $k = 1, ..., K$. This will lead to the following being computed at each iteration k:

$$\mathbf{A}_\beta = \frac{1}{\sigma^{2(k-1)}}\mathbf{X}^\top\mathbf{X} + \boldsymbol{\Sigma}_\beta^{-1}, \tag{8}$$

$$\mathbf{b}_\beta = \frac{1}{\sigma^{2(k-1)}}\mathbf{X}^\top\mathbf{y} + \boldsymbol{\Sigma}_\beta^{-1}\boldsymbol{\mu}_\beta, \tag{9}$$

which will then be used to used to sample the current iteration of

$$\left[\boldsymbol{\beta}^{(k)}|\cdot\right] \sim N_p(\mathbf{A}_\beta^{-1}\mathbf{b}_\beta, \mathbf{A}_\beta^{-1}), \tag{10}$$

and this allows for the sampling of

$$\left[\sigma^{2(k)}|\cdot\right] \sim IG\left(\alpha_0 + \frac{n}{2}, \beta_0 + \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)})\right). \tag{11}$$

We must also set initial values of α_0, β_0, $\boldsymbol{\mu}_\beta$ and $\boldsymbol{\Sigma}_\beta$. Scanlon (2021) uses $\alpha_0 = \beta_0 = 0.01$, $\boldsymbol{\mu}_\beta = \mathbb{0}_p$ and $\boldsymbol{\Sigma}_\beta = 100\mathbf{I}_p$ for a sample model in which $p = 3$ [[!!! SOURCE SCANLON]]. This assumes identically and independently distributed β_i which conveniently allows for a trivial computation of $\boldsymbol{\Sigma}_\beta^{-1} = \frac{1}{100}\mathbf{I}_p$. We have used the same setup in the construction of our Gibbs sampler found in A.2.

To improve computational efficiency, we have also identified some matrix computations that are repeated at every iteration and thus only must be computed once. These are $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}^\top\mathbf{y}$. Unfortunately, $\mathbf{A}_\beta^{-1}$ must be computed at every iteration k which is computationally expensive on the order of $O(p^3)$ time. [[!!!However we have implemented (INSERT METHOD) to make this process more efficient.]]

2.4 Frequentist Methods OLS/Ridge/Lasso

For a linear model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ as defined previously, the oridnary least squares (OLS) solution to linear regression is the solution to the following minimization problem:

$$\hat{\boldsymbol{\beta}}_{OLS} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \tag{12}$$

This problem has a closed form solution that can be computed directly as $\hat{\boldsymbol{\beta}}_{OLS} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$.

In the case of Ridge regression we add a scaled squared ℓ_2-norm $\lambda\|\boldsymbol{\beta}\|_2^2$ penalty term to this minimization. Thus the solution to the Ridge regression optimization problem is defined as:

$$\hat{\beta}_{Ridge} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2. \tag{13}$$

Again, we have a closed form solution to this problem that is $\hat{\boldsymbol{\beta}}_{Ridge} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{y}$.

For LASSO, this penalty term is instead the the scaled ℓ_1-norm $\lambda\|\boldsymbol{\beta}\|_1$, which means that

$$\hat{\beta}_{Lasso} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1. \tag{14}$$

However, unlike OLS and Ridge, Lasso does not have a closed form solution and so the solution must be computed through some form of iterative algorithm.

For Ridge and LASSO, an optimal λ of the can be approximated through k-folds cross-validation. The `sklearn.linear_model.LassoCV` and `sklearn.linear_model.LassoCV` functions automatically perform this cross-validation step with a 5-fold setup when fitting a model. Although likely not the most efficient setup, we will choose to use these functions for this project for the sake of brevity.

As a final remark, we will also be using the `sklearn.linear_model.LinearRegression` function. This function is far more computationally costly than computing the OLS solution through Cholesky decomposition for instance. However, as only one computation of the linear model was needed for each model, it seemed sensible to use this function.

3 Data Simulation

We have created two simulated samples to test our Gibbs sampler against. Both of these models are uncorrelated sparse models, that is they all have zero true covariance values and there are nonsignificant predictors. Each of these models are used to generate equally sized test and training sets.

3.1 Small Model

Our first model is a low p large n model. That is to say, it has a small number of predictors p and a comparatively large number of samples n. In this case the model $p = 6$ and $n = 300$. The response vector $\mathbf{y} \in \mathbb{R}^n$ is sampled as

$$\mathbf{y} = 7\mathbf{x}_1 + 1\mathbf{x}_2 + 0\mathbf{x}_3 + 4\mathbf{x}_4 + 3\mathbf{x}_5 + 0\mathbf{x}_6 + \boldsymbol{\epsilon} \tag{15}$$

where each $\mathbf{X}_j \in \mathbb{R}^n$ is a vector of n observations for $j = 1, ..., j$. Further, every $\epsilon_i \overset{\text{iid}}{\sim} N(0, 9)$ and each $x_{i,j}$ is i.i.d. under some uniform distribution for each value of j. These uniform distributions were chosen arbitrarily and are as follows

$$\begin{aligned}
x_{i,1} &\sim U(0, 30) \\
x_{i,2} &\sim U(20, 50) \\
x_{i,3} &\sim U(15, 18) \\
x_{i,4} &\sim U(0, 50) \\
x_{i,5} &\sim U(0, 10) \\
x_{i,6} &\sim U(25, 27)
\end{aligned}$$

for each $i = 1, ..., n$. In this model the regression coefficients were also chosen arbitrarily. The code for this small model generation is found in A.3.

3.2 Large Model

Our second model has a large p and a relatively small n, however $n > p$ still holds. Accordingly, we have selected $p = 1000$ and $n = 1050$ for this model. These p and n values are not on the scale of some of the datasets that may be encountered in the modern day, however they were approaching the limits of our computational abilities due to the time required to run our Gibbs sampler defined above. The response variable $\mathbf{y} \in \mathbb{R}^n$ was constructed as follows

$$\mathbf{y} = \mathbf{XG}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{16}$$

with design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, coefficients $\boldsymbol{\beta} \in \mathbb{R}^n$, error vector $\boldsymbol{\epsilon} \sim N_p(\mathbb{0}_p, \sigma^2 \mathbf{I}_p)$ and "trigger" matrix $\mathbf{G} = \text{diag}(g_1, ..., g_p)$ where each $g_j \in \{0, 1\}$. Every $g_j \sim \text{Bernoulli}(p = 0.5)$ which will then randomly set an expected 50% of the β_j values to zero. We use this construction as an easy way of creating sparsity considering the number of parameters we are generating. Further, for this model we are sampling every $x_{ij} \sim U(0, 100)$ and $\boldsymbol{\beta} \in N_{1000}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta)$ with $\mu_j \sim U(-10, 10)$ and $\boldsymbol{\Sigma} = \text{diag}(\sigma^2_{\beta_1}, ..., \sigma^2_{\beta_p})$ such that every $\sigma^2_{\beta_j} \sim U(0, 20)$. These parameters and distributions were largely chosen arbitrarily in order to produce a random $\mathbf{X}$ and $\boldsymbol{\beta}$. The code for the generation of this larger model is found in A.4.

4 Results & Comparisons

4.1 Small Model

	True Value	OLS	Ridge	Lasso	Gibbs
$\hat{\beta}_1$	7.0	6.945047	6.942184	6.938708	6.943393
$\hat{\beta}_2$	1.0	0.970877	0.970560	0.961613	0.968138
$\hat{\beta}_3$	0.0	0.833671	0.796489	0.000000	0.581158
$\hat{\beta}_4$	4.0	3.980728	3.980158	3.977486	3.981427
$\hat{\beta}_5$	3.0	2.920467	2.910189	2.814856	2.911239
$\hat{\beta}_6$	0.0	0.491147	0.443575	0.000000	-0.253574
MSE_{pred}		84.387188	84.344407	83.401259	83.736562

Table 1: Predicted $\hat{\beta}_j$ for each method and MSE of prediction on the test data set.

The point estimates we get for the coefficients are very close to the true values for all methods. Lasso appears to be the best method for detecting parameters with no influence which is what we would expect since one property of LASSO is that it will 'nudge' variables close to zero to be zero. The optimal values of λ for Lasso and Ridge regression were found by cross-validation to be 0.92396 and 10.0, respectively.

4.2 Large Model

	OLS	Ridge	Lasso	Gibbs
Runtime (s)	0.4880088	0.603109	5.262211	3162.128275
MSE_{pred}	2.791440e+07	2.791382e+07	2.769916e+07	2.791400e+07

Table 2: Runtime for each method and MSE of prediction on the test data set.

The results presented in Table 2 provide a comparison of the four methods we evaluated for estimating regression coefficients: Ordinary Least Squares (OLS), Ridge, Lasso, and Gibbs sampling using the Markov Chain Monte Carlo (MCMC) algorithm. The performance of each method is evaluated based on two metrics: runtime and mean squared error (MSE) of prediction on the test dataset.

From the table, it is evident that the MSE of prediction is of the same order of magnitude for all four methods. Lasso has the lowest MSE, suggesting that it provides the best model fit among the methods compared. In terms of runtime, OLS and Ridge are the fastest methods, with OLS being slightly faster than Ridge. Lasso takes over 10 times longer to execute than OLS, however, we must keep in mind that we ran the LASSO regression with built in 5-fold cross-validation. So the computation speed might actually be much faster than OLS and Ridge.

Examining the Gibbs MCMC algorithm, it is apparent that it has the longest runtime by far, taking approximately 600 times longer than Lasso. This suggests that the algorithm might not be the most efficient choice for this particular problem, especially when runtime is a concern. Gibbs sampling may only be more suitable for high-dimensional data when obtaining probability intervals is crucial, and when the time available for computation is not a limiting factor.

Overall, the Lasso method seems to provide the best trade-off between model accuracy and computational efficiency. It achieves the lowest MSE of prediction and takes a reasonable amount of time compared to the other methods. The choice of the most appropriate method ultimately depends on the specific requirements of the analysis, such as the desired level of accuracy and the available computational resources. For this reason, it seems that LASSO is the most effective method to try when p is close to n if point estimates are all that are needed.

In conclusion, the results of this comparison highlight the limitations of the Gibbs MCMC algorithm in terms of computational efficiency for this particular problem. Although it provides a comparable model fit in terms of MSE, its substantially longer runtime makes it less attractive when time is a constraint. However, the Gibbs MCMC algorithm can still be a valuable tool in situations where high-dimensional data is involved and obtaining probability intervals is crucial, provided that the available computational resources and time are sufficient. As always, the choice of the most appropriate method will depend on the specific requirements of the analysis and the characteristics of the dataset and problem at hand.

5 Conclusion

Our results showed that Gibbs MCMC and OLS, Ridge, and Lasso all produce similar results in terms of estimates and prediction error for both low- and high-dimensional datasets. However, the difference in runtime between the Bayes and frequentist methods became enormous upon increasing from a 300x6 dataset to a 1050x1000 dataset. For this reason, we conclude that MCMC Gibbs is probably not the best method to use for estimation of a model with a very large number of parameters (close to the number of observations).

Going forward, it would be useful to determine if there are modifications that can be made to our MCMC algorithm that could perform better on high-dimensional datasets. Gibbs sampling is already an optimization over the traditional Metropolis-Hastings algorithm when the full conditionals are known. We could also attempt non-MCMC Bayesian techniques that do not rely on computing a large number of iterations but still provide posterior distributions if we wish to obtain probability intervals for our estimates. However, we would need to take additional considerations if we were dealing with difficult to compute posterior distributions.

!!!REMOVED!!!

[[Source these claims maybe? or rewrite them]] The data being used in this project is simulated in order to test specific scenarios that in theory should be better suited to Bayesian MCMC. These include using a high parameter dimension compared to the number of observations and drawing our true parameter values from known distributions to allow us to specify accurate priors. Our hypothesis is that Bayesian MCMC estimation techniques should provide better estimates (lower MSE) than frequentist methods at the expense of significantly longer execution times. Bayesian techniques are also advantageous over frequentist techniques in that they provide distributions for our estimates, rather than just point estimates, which allow for specifying probability intervals. [[How is it advantageous to have probability intervals]]

A Appendix

A.1 Discussing Code Optimization

A.2 Gsblr

A.3 Small Model Generation

A.4 Large Model Generation

A.5 Model Computations

A.6 Graphics Generation

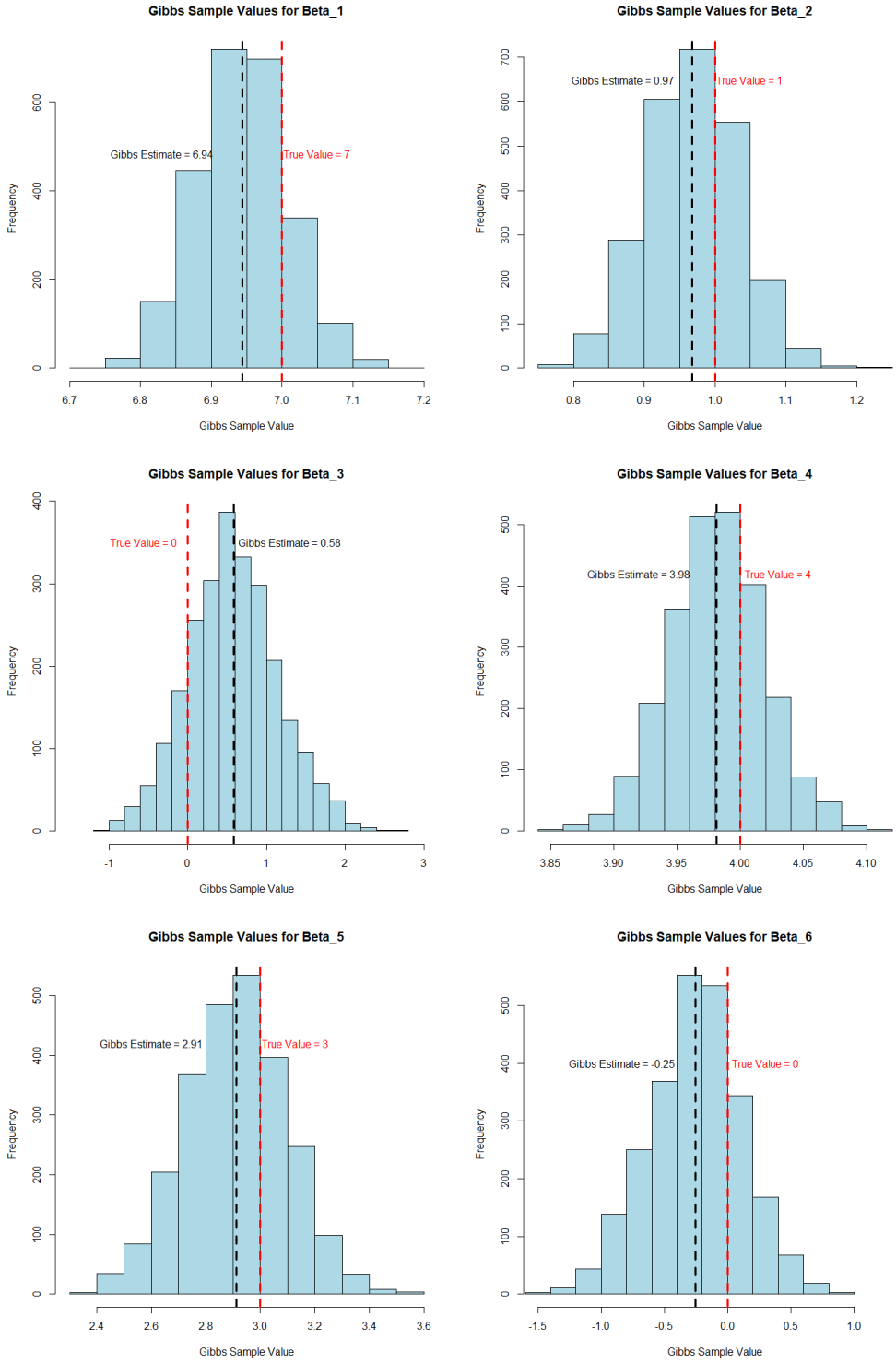

Figure 1: Each plot shows the MCMC samples that were generated for one coefficient. The distributions appear Normal which is expected due to the use of Normal conjugate prior.