Application of Markov Chain Monte Carlo in Bayesian Logistic Regression

by

Benjamin Lutz

An undergraduate honors project submitted
in partial fulfillment of the requirements for the

Fariborz Maseeh Department of Mathematics and Statistics
Honors Track

Under the supervision of
Prof. Ge Zhao

Portland State University

June 11, 2022

Introduction

In today's day and age, model accuracy is becoming increasingly important. In both science and industry, the stakes for accurate prediction can be very high. Particularly, in the field of medical research there are often human lives depending on the performance of new and improved medical technology. Because of the immense importance of accurate modeling, statistical researchers are encouraged to explore new methods of prediction. One such method, which is increasingly finding its place in many applications of modeling, is the Bayesian regression model.

In this paper, we examine the Bayesian logistic regression model. We also introduce a method of parameter estimation called a Markov Chain, and we use this to estimate distributions for each model parameter. Lastly, we discuss the application of this type of model in the statistical analysis of a medical device.

The Bayesian Model

Let us first introduce the Bayesian model. While this type of regression model is less main-stream than the traditional approach, it is useful in a couple of different scenarios. One such scenario is the situation where there is preexisting data or domain knowledge that can be incorporated into the analysis. Another is when it is desirable to quantify uncertainty in the predictions. Rather than compute point estimates for model parameters as in the frequentist approach, the result of Bayesian analysis will yield distributions for model parameters which allow for uncertainty in the model.

The result of a fitted Bayesian regression model looks very similar to that of a frequentist model. The difference between the two is in the method for estimating the model parameters. While ordinary regression uses the MSE to find point estimates and checks them against a p-value, Bayesian regression uses a stochastic process such as a Markov Chain to find distributional estimates for these parameters. The result is a range of potential model fits with varying corresponding probabilities.

The process for estimating model parameters using the Bayesian approach is to sample from a target posterior distribution using information we have about the current state of the parameter. To estimate this posterior distribution, we use the likelihood of our current data and multiply it to a prior. The posterior distribution for a model parameter is given as follows:

$$\begin{aligned} Posterior &= \frac{Likelihood * Prior}{Normalization} \\ P(\beta \mid y, X) &= \frac{P(y \mid \beta, X) * P(\beta \mid X)}{P(y \mid X)} \end{aligned}$$

In this equation, the likelihood is the probability function for a distribution of the response based on data from the current study. The prior is a probability function for a distribution that may or may not be informed by previous analyses. If there is no previous knowledge about the data, it is common to use an educated guess for the prior made by a professional, but Bayesian models that have informed priors are often far less controversial than ones that don't. Notice that this formula contains the normalization constant which was not mentioned previously. In most statistical analyses, the normalization is computationally intractable and is simply excluded from the calculation.

Once the posterior distribution is estimated, we use a stochastic process such as a Markov Chain to sample from, and update, the probabilities in this distribution. The Markov Chain samples through the state space of the parameter and determines the probability of a proposed state based on the current state. After many iterations, the Markov Chain will stabilize, which means the probabilities for each state of the Markov Chain will converge and resemble the probabilities of the true target distribution. We will elaborate further on the algorithm for this in later discussion.

Bayesian Logistic Regression

The example we use in this paper is that of Bayesian logistic regression. We will analyze the "logit" data set, and use the Markov Chain Monte Carlo, Metropolis-Hastings algorithm to find estimates for the parameter distributions.

In general, logistic regression is a type of regression that is used when we have a data set with a binary response. When a proper logistic regression model has been fit, the output of the fitted model will be the probability of obtaining one of the two possible response values. A multiple logistic regression model with n regressors follows the form

$$P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\ldots+\beta_n X_n)}}$$

This formula is the outline for our regression model. Since we are conducting a Bayesian regression, we will find distributions for β_0, β_1, ..., β_n rather than point estimates.

Before loading the data, let us first introduce the Metropolis-Hastings (MH) algorithm. Metropolis-Hastings is a very straightforward algorithm for a Markov Chain that can be easily implemented using basic R code. The MH algorithm is defined as follows.

Metropolis Hastings Algorithm

1. For a given parameter β, select an initial value β_0 (not to be confused with the intercept, β_0)
2. If m is the number of iterations of the algorithm, repeat the following for $i = 1, 2, \ldots, m$.
   (a) Draw a candidate $\beta^* \sim q(\beta^* \mid \beta_{i-1})$
   (b) Next, we calculate the Hastings ratio α given by
   $$\alpha = \frac{g(\beta^*)q(\beta^* \mid \beta_{i-1})}{g(\beta_{i-1})q(\beta_{i-1} \mid \beta^*)}$$
   (c) Now if $\alpha \geq 1$ we accept β^* and set
   $$\beta_i = \beta^*$$
   If $0 < \alpha < 1$ we accept β^* and set
   $$\beta_i = \beta^*$$
   with probability α.
   OR, we reject β^* and set
   $$\beta_i = \beta_{i-1}$$
   with probability $1-\alpha$.

In this algorithm, the Hastings ratio is the probability of accepting the proposed value for β. This Hastings ratio is the same as the probability of the proposed value divided by the probability of the current value, where both probabilities are drawn from the evolving posterior distribution. For each iteration of the algorithm, more of the state space for β is explored and our posterior distribution becomes more accurate.

The Code

Now to analyze the "logit" data set. An ordinary logistic regression model using point estimates would be done as follows.

```
library(mcmc)
data(logit)
out <- glm(y ~ x1 + x2 + x3 + x4, data = logit,
  family = binomial(), x = TRUE)
summary(out)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x3 + x4, family = binomial(), data = logit,
##     x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7461  -0.6907   0.1540   0.7041   2.1943
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.6328     0.3007   2.104  0.03536 *
## x1            0.7390     0.3616   2.043  0.04100 *
## x2            1.1137     0.3627   3.071  0.00213 **
## x3            0.4781     0.3538   1.351  0.17663
## x4            0.6944     0.3989   1.741  0.08172 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.628  on 99  degrees of freedom
## Residual deviance:  87.668  on 95  degrees of freedom
## AIC: 97.668
##
## Number of Fisher Scoring iterations: 6
```

Based on the p-values displayed in the table, we see that $x3$ would require further inspection. However, we will not go into this as the focus of this paper is Bayesian regression.

Posterior Distribution

To estimate the parameters using the Bayesian method, we first define a function to estimate a posterior distribution for each. Since we do not have any previous information pertaining to this data, we choose a non informative prior. Note that this would be somewhat controversial if applied in the real world. However, for this toy example, we will use a normal distribution with mean 0 and standard deviation 2. The code follows.

```
data(logit)

posterior <- function(param){
    # Store the current state for each parameter in variables
    beta1 = param[1]
    beta2 = param[2]
    beta3 = param[3]
    beta4 = param[4]
    beta5 = param[5]
    # This is used at various points in the logistic regression pdf, so we
    # assign it to a variable.
    pred = beta1*logit$x1 + beta2*logit$x2 + beta3*logit$x3 + beta4*logit$x4 + beta5
    eta = as.numeric(pred)

    # Log likelihood for y=1
    logp = ifelse(eta < 0, eta - log1p(exp(eta)),- log1p(exp(- eta)))
    # Log likelihood for y=0
    logq = ifelse(eta < 0, - log1p(exp(eta)), - eta - log1p(exp(- eta)))
    # Total log likelihood
    logLik = sum(logp[logit$y == 1]) + sum(logq[logit$y == 0])

    # For the prior, we choose a normal distribution with mean 0 and standard
    # deviation 2.
    # Total log prior
    logPrior = sum(dnorm(param, mean = 0, sd = 2, log = T))


    return(logLik + logPrior)
}
```

This function computes the log of the posterior distribution by adding the natural logarithm of the likelihood to the natural logarithm of the prior. Note that this is the same as the natural log of the product of the likelihood and the prior due to laws of logarithms.

Notice also that we use “ifelse” statements in the calculation of the likelihood function. This is because we must only consider the case where “eta” is negative in order to avoid overflow. Also notice that we calculate the likelihoods for the cases where the response 1 and 0 separately. This allows us to avoid catastrophic cancellation which would otherwise significantly throw off the results by accepting too few proposed parameter values.

Proposal Function

Next we create a proposal function to select a candidate value for the next state based on the current one. It will select from a distribution with mean equal to the current state and a random standard deviation. Since the standard errors will evolve over thousands of iterations, we select random start values for each parameter’s standard deviation. This step is equivalent to part (a) of the MH algorithm.

```
proposalfunction <- function(param){
    return(rnorm(5, mean = param, sd = c(0.001, 0.002, 0.0005, 0.0015, 0.0001)))
}
```

The Markov Chain

Now we use the Metropolis-Hastings algorithm to create a function that will run a Markov Chain.

```
run_metropolis_MCMC <- function(startvalue, iterations){
    # Array to hold current and past states of the Markov Chain
    chain = array(dim = c(iterations+1, 5))
    # Set the initial state to a random start value
    chain[1,] = startvalue
    # For each iteration of the Markov Chain, do the following
    for (i in 1:iterations){
        # Get the proposed value for the current iteration
        proposal = proposalfunction(chain[i,])

        # Calculate the Hastings ratio and either accept or reject the proposal
        probab = exp(posterior(proposal) - posterior(chain[i,]))
        if (probab > 1){
            chain[i+1,] = proposal
        }else{
            chain[i+1,] = chain[i,]
        }
    }
    # Return an array of the accumulated accepted values of the parameters
    return(chain)
}

# Create a vector of randomely chosen start values for each parameter
startvalue = c(0.13, 0.13, 0.03, 0.08, 0.56)
# Define the number of iterations of the Markov Chain
iter = 10000
# Run the Markov Chain and store the results in the data frame "chain"
chain = run_metropolis_MCMC(startvalue, iter)
```

Model Analysis

Now that we have successfully run the Markov Chain, it is good practice to check the acceptance rate. The acceptance rate tells us how many samples are being accepted by the Metropolis-Hastings update, and should generally be between 20% and 30%. This way, the Markov Chain is able to explore the full state space of the parameter while still accepting enough samples within a reasonable amount of time. There is logic behind the $20\% - 30\%$ rule that is not explained here, though it is important to note that this rule does not apply in all scenarios. The acceptance rate is displayed below.

```
acceptance = 1-mean(duplicated(chain))
acceptance
```

```
## [1] 0.2278772
```

It is also good practice to create a traceplot of the estimates, which will help us guess whether the Markov Chain has stabilized. We can then decide whether it is safe to use the results as distributional estimates, or if it is necessary to run the Markov chain for more iterations. The traceplot for this run of the Markov Chain is displayed below.

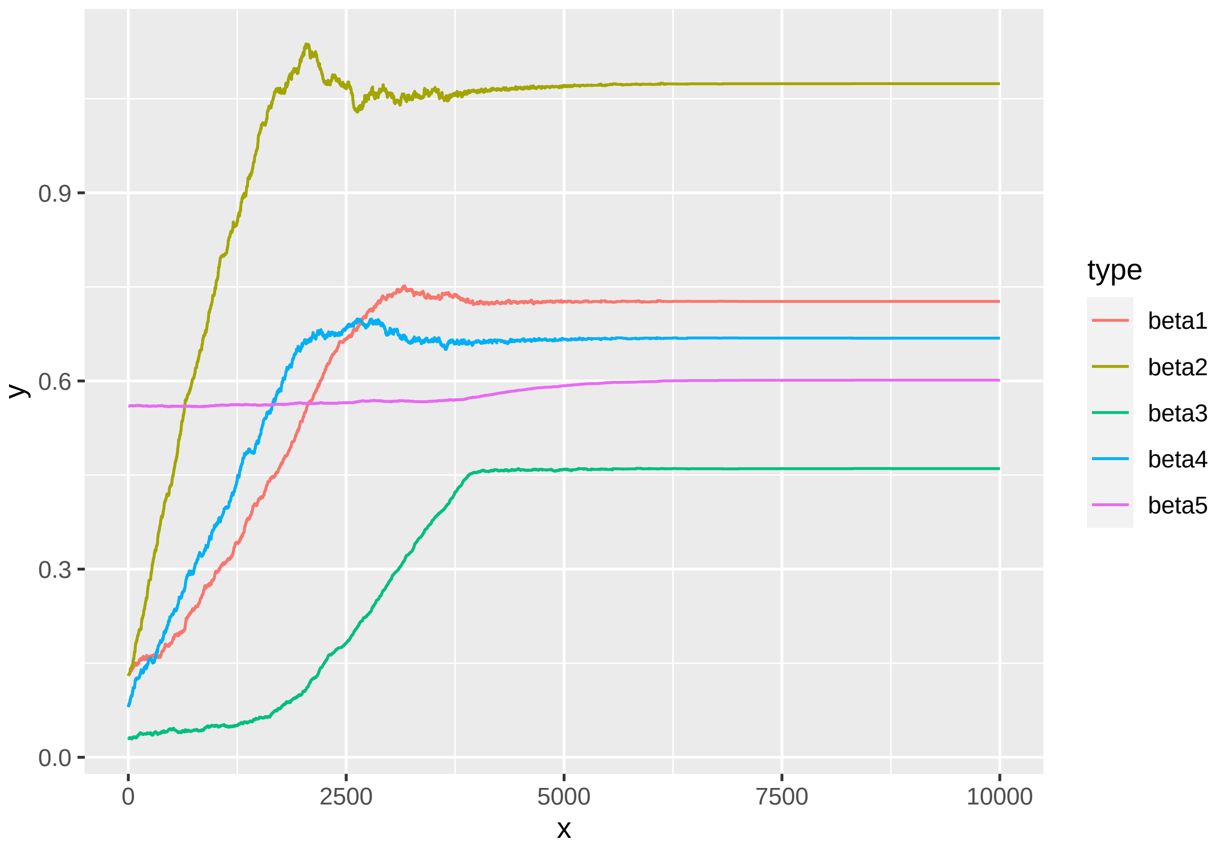
```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.5     v purrr   0.3.4
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
xval = c(1:(iter+1))

mydata = data.frame(xval, chain[, 1], chain[, 2], chain[, 3], chain[, 4], chain[, 5])

df <- data.frame(x=xval, y=chain[, 1], type='beta1')
df <- rbind(df, data.frame(x=xval, y=chain[, 2], type='beta2'))
df <- rbind(df, data.frame(x=xval, y=chain[, 3], type='beta3'))
df <- rbind(df, data.frame(x=xval, y=chain[, 4], type='beta4'))
df <- rbind(df, data.frame(x=xval, y=chain[, 5], type='beta5'))
ggplot(df, aes(x, y, group=type, col=type)) + geom_line()
```



Based on the traceplot, it appears that the Markov Chain stabilizes after roughly 5000 iterations. Because a Markov Chain can take many iterations to stabilize, it is often a good idea to establish a “burn in”, where we discard the estimates at the begging that are inaccurate. In the following code, we create a burn in variable that we will use later.

```r
# Create a burn in variable
burnIn = 5000
```

With the burn in established, we can now generate summary statistics for each parameter distribution estimate and compare them to the point estimates in the frequentist model.

```r
summary(chain[-(1:burnIn), 5])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.5923  0.6003  0.6011  0.6001  0.6012  0.6012
```

```r
summary(chain[-(1:burnIn), 1])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7251  0.7268  0.7268  0.7268  0.7268  0.7280
```

```r
summary(chain[-(1:burnIn), 2])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.069   1.073   1.074   1.073   1.074   1.075
```

```r
summary(chain[-(1:burnIn), 3])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4567  0.4601  0.4601  0.4601  0.4603  0.4607
```

```r
summary(chain[-(1:burnIn), 4])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6649  0.6681  0.6683  0.6681  0.6684  0.6693
```

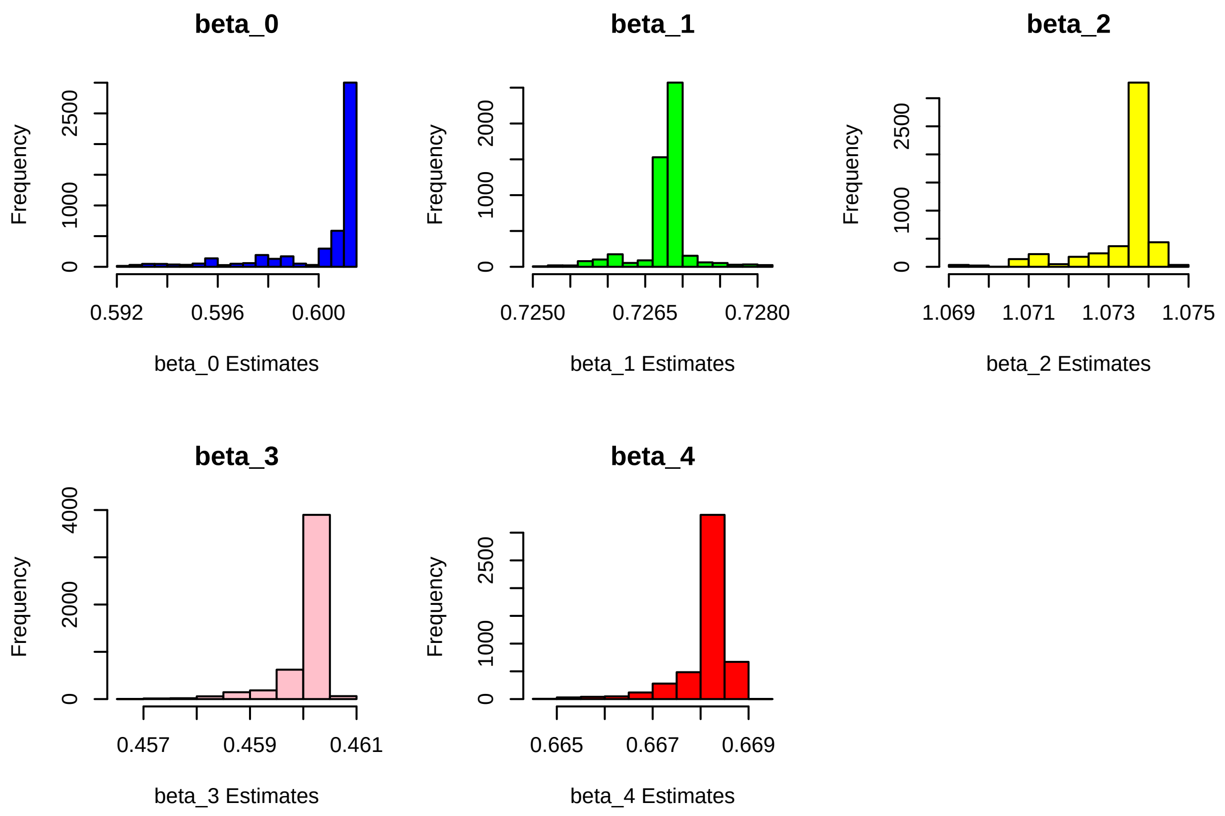
```r
summary(out)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x3 + x4, family = binomial(), data = logit,
##     x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7461  -0.6907   0.1540   0.7041   2.1943
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.6328     0.3007   2.104  0.03536 *
## x1            0.7390     0.3616   2.043  0.04100 *
## x2            1.1137     0.3627   3.071  0.00213 **
## x3            0.4781     0.3538   1.351  0.17663
## x4            0.6944     0.3989   1.741  0.08172 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.628  on 99  degrees of freedom
## Residual deviance:  87.668  on 95  degrees of freedom
## AIC: 97.668
##
## Number of Fisher Scoring iterations: 6
```

Notice the mean for each parameter distribution resembles the corresponding point estimate in the ordinary logistic regression model. Thus it appears we have successfully found accurate distributional estimates for the Bayesian model parameters. Below are histograms representing each of these distributions.

```r
par(mfrow = c(2,3))

hist(chain[-(1:burnIn),5], col='blue', main="beta_0", xlab="beta_0 Estimates")
hist(chain[-(1:burnIn),1], col='green', main="beta_1", xlab="beta_1 Estimates")
hist(chain[-(1:burnIn),2], col='yellow', main="beta_2", xlab="beta_2 Estimates")
hist(chain[-(1:burnIn),3], col='pink', main="beta_3", xlab="beta_3 Estimates")
hist(chain[-(1:burnIn),4], col='red', main="beta_4", xlab="beta_4 Estimates")
```



Applications of Bayesian Regression

Though frequentist regression is more mainstream in industry than the Bayesian approach, there are still scenarios where the flexibility of Bayesian regression is favorable. Say you flip a coin ten times and every flip lands with heads. A frequentist analysis would tell us that a the next flip will be heads with probability

$p = 1$. However, a Bayesian analysis could use prior knowledge we have about this coin, and would allow for variability in the response.

This property of the Bayesian method becomes very useful when examining medical data. One example of a real-world application of Bayesian statistics is the analysis of a medical device's performance. As mentioned at the beginning of this paper, the Bayesian model is useful when we have prior information regarding the data set, or when we want to allow for some variability in the response.

In the clinical trial of a medical device, we often times have substantial previous data on similar versions of the device. Though this data is not from the current device, it can provide well informed prior information to our current study of the device. Of course, the use of prior information in the pivotal study requires that modifications to the device are minor, and that the device effects are local, rather than systemic (Center for Devices and Radiological Health, 2010). However, if these criteria are satisfied, we can incorporate information from past models and from studies of the device conducted overseas. Since there may be economical concerns about testing a device many times, this strong prior may allow us to have a smaller sample size for the pivotal study, thus saving a significant amount of money.

Conclusion

Though it is also common to apply Bayesian regression in scenarios where there is not a well-informed prior, studies like one described above are much less controversial. Bayesian regression is an excellent alternative to the traditional frequentist approach to statistics, and is becoming much more mainstream than it used to be. Though it is computationally intense, it is worth the effort to achieve accurate results when ordinary regression fails. A more widespread acceptance of Bayesian statistics could be the key to making good predictions about the future.

References

Brooks, S., Gelman, A., Jones, G., & Meng, X. L. (Eds.). (2011). Handbook of markov chain monte carlo. CRC press.

Center for Devices and Radiological Health. (2010, February). Guidance for the use of Bayesian statistics in medical device clinical. U.S. Food and Drug Administration. Retrieved June 4, 2022, from <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/guidance-use-bayesian-statistics-medical-device-clinical-trials>

Hartig, F. (2021, March 23). A simple Metropolis-Hastings MCMC in R. theoretical ecology. Retrieved June 4, 2022, from <https://theoreticalecology.wordpress.com/2010/09/17/metropolis-hastings-mcmc-in-r/>

Stephens, Matthew. "The Metropolis Hastings Algorithm." FiveMinuteStats, 23 Apr. 2018, https://stephens999.github.io/fiveMinuteStats/MH_intro.html.

Wikimedia Foundation. (2022, May 22). Markov chain Monte Carlo. Wikipedia. Retrieved June 6, 2022, from <https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo>