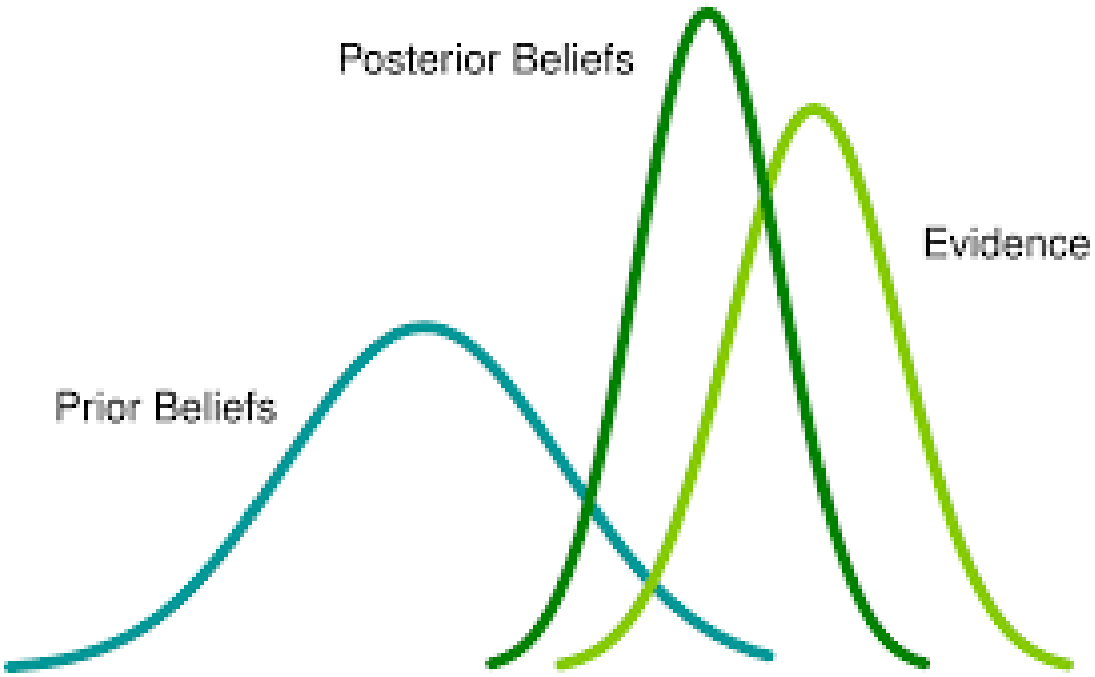
Comparing Metropolis and Gibbs Sampling Methods

Yunzhe Li: madli@ucdavis.edu
Ruriko Imai: raimai@ucdavis.edu
Yizi Zhang: yizzhang@ucdavis.edu

GitHub Link



Introduction

The purpose of this project is to explore sampling methods derived from algorithms in the Monte Carlo Markov Chain (MCMC) family called the Metropolis sampling and Gibbs sampling methods. These MCMC algorithms are used in a bayesian setting where direct sampling from the posterior distribution is difficult. By being able to sample from complicated distributions we are able to estimate mean and other parameters to further analyze the data. In order to check for performance (speed and accuracy)of those sampling methods, we shall use a known posterior distribution so the distribution obtained from Metropolis and Gibbs sampling methods can be compared. Therefore, the setting of the sampling distribution is Multinomial - Dirichlet conjugate with K = 3 categories (three-dimensional parameters) to update.

Difference between Gibbs and Metropolis Algorithm

Metropolis-Hastings sampler is the umbrella algorithm for both Gibbs and Metropolis sampling methods. These sampling methods are used when direct sampling is deemed difficult to perform on a given posterior distribution. Gibbs sampling is a special case of Metropolis hastings algorithm with a probability acceptance rate of 1. Metropolis sampling is also a special case of Metropolis hastings, requiring a symmetric proposal distribution. Although the algorithms are slightly different, the purpose is to sample from the posterior distribution.
Some issues to consider are:

- determine a good burn-in amount to decrease dependence on starting values
- determine a good thinning amount to decrease autocorrelation
- determine a good proposal distribution for Metropolis sampler
- partial correlation between parameters
- convergence diagnostics

We will explore these issues throughout this simulation process.

Set-Up

- Data:
  - True parameters: $\theta_1 = 0.1$, $\theta_2 = 0.7$, $\theta_3 = 0.2$
  - Hyperparameters: $\alpha_1 = 2$, $\alpha_2 = 2$, $\alpha_3 = 2$
- Prior: $X_i|\theta \overset{iid}{\sim} Multinom(n, \theta_1, \theta_2, \theta_3); i = 1, \ldots, n$
- Likelihood function: (singular) $Multinom(\theta_1, \theta_2, \theta_3)$.
- Proposal: $Dirichlet(\alpha_1, \alpha_2, \alpha_3)$
- Posterior: $Dirichlet(\alpha_1 + \sum X_{1i},\ \alpha_2 + \sum X_{2i} + \alpha_3 + \sum X_{3i})$

Algorithms

Metropolis algorithm:

- Choose a starting point for θ's from a starting distribution
- For t = 1, 2, ...
  - Sample a proposed $\theta^\star$ from a symmetric jumping/proposal distribution at time t.
  - Calculate the density ratio:

$$ratio = \frac{p(\theta^\star)|y}{\theta^{t-1}|y} \tag{1}$$

  - Set:

$$\theta^t = \begin{cases} \theta^\star, & \text{with propability min(ratio, 1)} \\ \theta^{t-1}, & \text{otherwise} \end{cases} \tag{2}$$

Gibbs algorithm:

- Set initial values for θ's
- Set $Y = \frac{\theta_1}{1-\theta_2}|\theta_2$
  $Y \sim Beta(\alpha_1 + \sum X_{1i},\ \alpha_2 + \sum X_{2i} + \alpha_3 + \sum X_{3i})$
- Set $Z = \frac{\theta_2}{1-\theta_1}|\theta_1$
  $Z \sim Beta(\alpha_2 + \sum X_{2i},\ \alpha_1 + \sum X_{1i} + \alpha_3 + \sum X_{3i})$
- Obtain updated sample of θ's
- Simulate steps 1-4 with burn-in and thinning

Statistics

Metropolis Simulation Data:

Burn-In	Thinning	Time-Spent	μ_1	μ_2
5000	100	38.78	0.146	0.672
0	100	39.18	0.147	0.671
5000	0	38.78	0.146	0.672

Gibbs Simulation Data:

Burn-In	Thinning	Time-Spent	μ_1	μ_2
5000	100	0.85	0.151	0.662
0	100	0.83	0.151	0.665
5000	0	0.88	0.151	0.664

Analysis

As indicated in the above graphs, comparison of different burn-in and thinning amounts were made for each MCMC algorithm to access whether burn-in and thinning play a significant role in decreasing dependence on starting values and autocorrelation due to Markov Chain.

Figure 1 and Figure 2 represent the sampled values by the Metropolis and Gibbs sampling methods and the distribution of the estimated parameters. The true parameters for the target/posterior distribution are set up to be 0.1, 0.7, and 0.2. We then set up 3 control groups for each sampling method, as indicated in the above table, to compare the effects of burn-in and thinning. In this case, we didn't observe a significant difference in the sampled means of posterior parameters between the 3 control groups, indicating that the burn-in and thinning amount matters less when a large number of iterations is run(N=100000). In addition, both algorithms perform with approximately the same accuracy, however, Gibbs sampler runs faster (less than 1 second) than the Metropolis sampler (approximately 39 seconds). This is because the gibbs function was written in a way that does not call on multiple functions whereas the Metropolis function calls on proposal and posterior functions and so on. The main reason for the difference in computing speed is that Gibbs sampling method breaks the curse of dimensionality by dealing with the conditional distributions of only 2 parameters and updates parameters based on the previous ones without utilizing the rejection sampling scheme(the acceptance ratio is taken as 1). On the contrary, Metropolis algorithm has to compute with all 3 parameters and calculate an acceptance ratio to reject or accept the proposed parameter values.

The partial correlation plots in Figure 3 indicate that there is a negative correlation of around -0.50 between the sampled theta1 and theta2. This is due to the fact that the likelihood function is a multinomial distribution which presumes correlation between parameters.

Conclusion

Although these algorithms have dependence on starting values, the values will eventually converge. Having good starting values is better because the algorithm will converge faster but it is not necessary. We are not able to observe significant differences in the sampled posterior means of the parameters between different burn-in and thinning amount. This indicates that a large sample size will help significantly to decrease the dependence and autocorrelation issues. In a situation where conditional densities are accessible, Gibbs sampling method is preferable over Metropolis sampling method since it is easier to implement and faster to compute. Otherwise, Metropolis performs better when dealing with more complex target distributions.

Difficulties

There were various difficulties we ran into while working on this project. One major problem was the fact that we could not work with our initial proposal due to our misunderstanding and the difficulty of obtaining the jumping distribution to calculate the acceptance ratio for the MCMC-MH algorithm. Due to time constriction, we simplified our project to comparing the Gibbs sampling method and the Metropolis sampling method. This was much viable since we understood and knew how these algorithms worked in order for us to simulate and analyze their performances.

Appendix

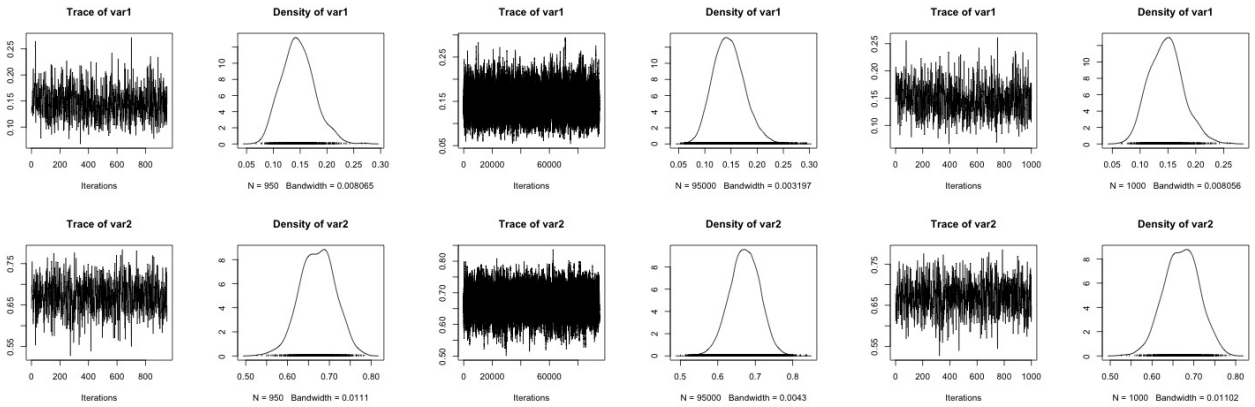


Figure 1: Metropolis Chain Values and Density of Parameters

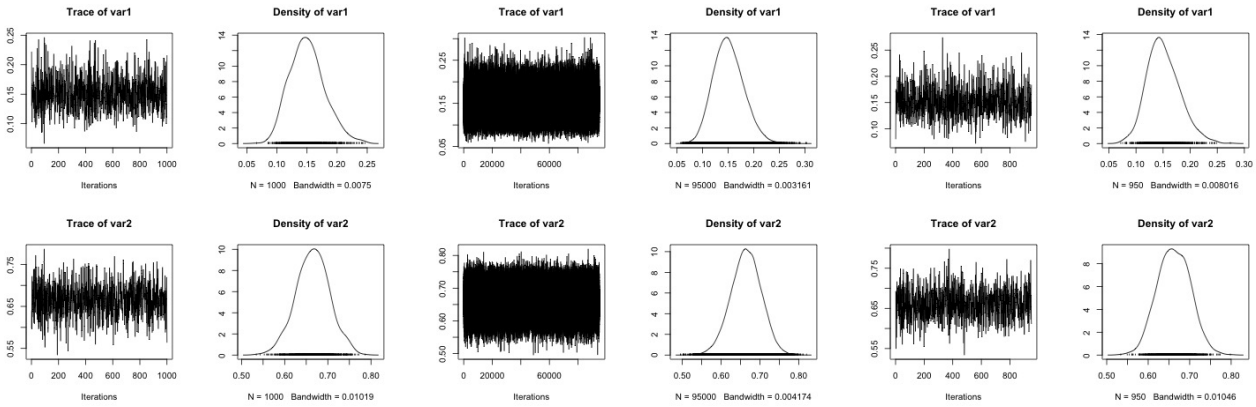


Figure 2: Gibbs Chain Values and Density of Parameters

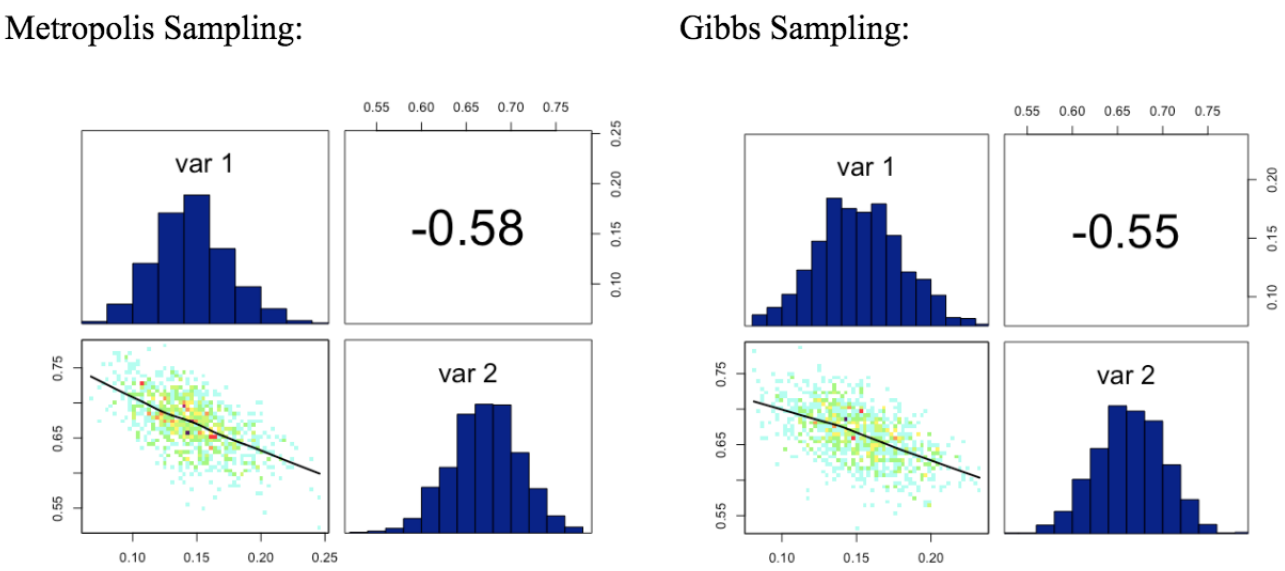


Figure 3: Partial Correlations

```
library(MASS)
library(coda)
library(MCMCpack)
library(BayesianTools)

set.seed(54321)

true_theta<-c(0.1,0.7,0.2)
alpha<-c(2,2,2)
x<-rmultinom(20,size=7,prob = true_theta)    # FIX THIS

gibbs_func<-function(start_value,burnin,thin,iter){
  update<-start_value
  theta_mat<-mat.or.vec(2,iter)
  for(i in 1:iter){
    theta1<-(1-update[2])*(rbeta(1,shape1 = alpha[1]+sum(x[1,]),shape2 = alpha[3]+sum
    theta2<-(1-theta1)*(rbeta(1,shape1 = alpha[2]+sum(x[2,]),shape2 = alpha[3]+sum(x
    update<-c(theta1,theta2)
    theta_mat[,i]<-update
  }
  sample1<-theta_mat[1,][seq(burnin+1,iter,thin)]
  sample2<-theta_mat[2,][seq(burnin+1,iter,thin)]

  chain<-matrix(nrow=length(sample1), ncol=2)
  chain[,1]<-sample1
  chain[,2]<-sample2

  return(chain)
}

# Posterior
posterior_func <- function(param){
  single_likelihood <- apply(x, 2, function(t) dmultinom(t, prob=param))
  single_prior <- ddirichlet(param, alpha)
  nlikelihood <- sum(log(single_likelihood))
  nprior <- sum(log(single_prior))
  return(nlikelihood + nprior)
  #posterior = sum(apply(post_alpha, 2, function(t) ddirichlet(param, t)))
  # posterior = ddirichlet(param, post_alpha)
}

# Proposal
scale_factor = 1000  # Scale variance of proposal function
proposal_func <- function(param, scale_factor){
  param <- rdirichlet(1, param * scale_factor)
  return(param)
}

# Metropolis MCMC algorithm
run_metropolis_MCMC <- function(start_value, burnin, thin, iter, s = scale_factor){
  theta_mat = array(dim = c(iter+1,3))
  theta_mat[1,1:3] = start_value
```

```
  for(i in 1:iter){
    proposal = proposal_func(theta_mat[i,1:3], s)
    prob = exp(posterior_func(proposal) - posterior_func(theta_mat[i,1:3]))
    if (runif(1) < prob){
      theta_mat[i+1,1:3] = proposal
    }else{
      theta_mat[i+1,1:3] = theta_mat[i,1:3]
    }

  }
  sample1<-theta_mat[,1][seq(burnin+1,iter,thin)]
  sample2<-theta_mat[,2][seq(burnin+1,iter,thin)]
  chain<-matrix(nrow=length(sample1), ncol=2)
  chain[,1]<-sample1
  chain[,2]<-sample2

  return(chain)
}

start_value = apply(x, 1, sum) / 140

start = Sys.time()
gibbs_chain<-gibbs_func(start_value[1:2],1000,100,100000)
cat('gibbs time spent: ', Sys.time() - start, '\n')

start = Sys.time()
metro_chain = run_metropolis_MCMC(start_value, 1000,100,100000)
cat("metropolis time spent: ", Sys.time() - start, '\n')
# analysis
summary(gibbs_chain)
par(mar=c(2,2,2,2))
plot(mcmc(gibbs_chain))
plot(mcmc(metro_chain))
## check partial correlatation btw parameters
combinedchains = mcmc.list(mcmc(gibbs_chain), mcmc(metro_chain))
plot(combinedchains)
gelman.diag(combinedchains)
gelman.plot(combinedchains)
```

References

[1] **https://theoreticalecology.wordpress.com/2010/09/17/metropolis-hastings-mcmc-in-r/**

[2] **https://theoreticalecology.wordpress.com/2011/12/09/mcmc-chain-analysis-and-convergence-diagnostics-with-coda-in-r/**

[3] **https://stats.stackexchange.com/questions/185631/what-is-the-difference-between-metropolis-hastings-gibbs-importance-and-rejec/185643#185643**

[4] Sneha's discussion 9