Bayesian Inference and MCMC for Parameter Estimation

Ruiming Li

Spring 2020 STAT 51 Final Project

Professor Amanda Luby

INTRODUCTION

From what we learned so far, we calculated probability in distributions of fixed parameters and assumed that these parameters are the true parameters for the distribution. To capture uncertainties in our model distribution, we can instead see the parameters themselves as random variables that follow a certain distribution. This paper introduces the theoretical knowledge for estimating a parameter distribution, and applies it to estimate the reproductive number R_0 for COVID-19 in a simple SIR epidemiological model.

OUTLINE

Bayesian Inference

- Bayes's Theorem
- Beta-Binomial conjugacy

Markov Chain Monte-Carlo (MCMC)

- Markov Chains
- The Metropolis-Hastings Algorithm

Estimation of R_0 in SIR model

- Set-up
- Result Analysis

BAYESIAN INFERENCE

Bayesian Theorem

Let S be a sample space and $B_1 \cdots B_n$ be a partition of S such that $\bigcup_{1 \leq i \leq n} B_i = S$ and $B_i \bigcap B_j = \emptyset$ for all $i \neq j$. Let A be any event. For any $1 \leq k \leq n$:

$$P(B_k|A) = \frac{P(A|B_k)P(B_k)}{P(A)} \text{ [1]}$$

$$\text{(discrete)} = \frac{P(A|B_k)P(B_k)}{\sum_{i=1}^{n} P(A|B_i)P(B_i)}$$

$$\text{(continous)} = \frac{P(A|B_k)P(B_k)}{\int_{i=1}^{n} P(A|B_i)P(B_i)}$$

We will then look into detail what each term of the formula represents in Bayesian modeling.

Posterior

The posterior is our objective in Bayesian modeling - finding the distribution of parameters given a dataset. Let θ be the R.V for the parameter and X be the R.V. for the observed dataset. We are interested in the conditional distribution of θ for dataset $X = x$:

$$\boldsymbol{\pi(\theta|X = x)}$$

In real life, the posterior distribution is usually not any familiar distribution we have seen or can be expressed precisely - thus the need for simulation and estimation.

Prior
Prior is what we start with for estimating the parameter θ. The random variable θ itself has a marginal distribution $\boldsymbol{\pi(\theta)}$. This marginal probability captures our prior uncertainty regarding θ. It is like a proposal distribution to start with for estimating the parameter.

Since we usually do not know what the true prior distribution is, we usually choose the normal distribution or the uniform distribution, which are non-informative priors to limit influence on the posterior.

Likelihood
Likelihood is the key connecting prior and posterior. Suppose we try a parameter value θ for the distribution of X, we are interested in the probability that we observe the current dataset $X = \boldsymbol{x}$. This is expressed as $\boldsymbol{P(X = x|\theta)}$. The likelihood is based on a given θ value as the parameter for the distribution of X, evaluated at the point where $X = \boldsymbol{x}$.

Putting Things Together
By Bayes's Theorem, we find a way to calculate posterior probability with a given dataset $X\boldsymbol{x}$ and parameter value θ:

$$\boldsymbol{\pi(\theta|x)} = \frac{\boldsymbol{P(x|\theta)\pi(\theta)}}{\int_{\boldsymbol{\theta}} \boldsymbol{P(x|\theta)\pi(\theta)}\ \boldsymbol{d\theta}}$$

$$Posterior = \frac{Likelihood * Prior}{Normalizing\ Factor}$$

The denominator in the formula is just a normalizing constant equal to the sum of $Likelihood * Prior$ for all values of θ, so that the posterior probability is between 0 to 1. It is computationally expensive to calculate this constant, and if we are just comparing the posterior at different values of θ, we can simply calculate the unnormalized $Likelihood * Prior$ and compare the results relative to one another.

A Beta Prior and Binomial Likelihood
We can illustrate the Bayes's Theorem using a simple example with beta prior and binomial likelihood. Suppose our prior $\pi(\theta)$ follows a beta distribution $Beta(\alpha, \beta)$:

$$\pi(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}$$

and our likelihood $P(X|\theta)$ follows a binomial distribution $Bin(n, \theta)$:

$$P(X|\theta) = \binom{n}{x}\theta^x(1-\theta)^{n-x}$$

Using Bayes's Theorem,

$$P(\theta|X) \propto P(X|\theta)\pi(\theta)$$

$$= \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}\binom{n}{x}\theta^x(1-\theta)^{n-x}$$

$$\propto \theta^{\alpha+x-1}(1-\theta)^{n-x+\beta-1}$$

Surprisingly, the posterior stays in the beta family as $Beta(\alpha + x, n - x + \beta)$! [2]

MARKOV CHAIN MONTE-CARLO

So far we obtained a formula to calculate the posterior in terms of a given dataset $X = \boldsymbol{x}$ and parameter θ. How do we proceed to actually find the best θ for a given X? We have to first study properties of Markov Chains. Since Markov Chain is a huge topic to cover on its own, we are only introducing basic concepts in order to get to the estimation.

Markov Chain

Definition 1 A sequence of random variables $\{X_1, X_2, \cdots, X_t\}$ on a discrete state space Ω is called a first order *Markov Chain* if:

$$P(X_t = x_t | X_{t-1} = x_{t-1}, \cdots, X_1 = x_1) = P(X_t = x_t | X_{t-1} = x_{t-1}) \text{ [3]}$$

This simply means that the current state X_t of the random variable is only dependent on the early one state X_{t-1}. We do not care about how and what happens before it gets into state X_{t-1}.

Definition 2 Let $X_0, X_1, \cdots, X_M$ be a Markov Chain with state space $\{0, 1, \cdots, M\}$, and let $q_{ij} = P(X_t = j | X_{t-1} = i)$ be the transition probability from outcome i in state X_{t-1} to outcome j in state X_t. The $M * M$ matrix $Q = q_{ij}$ is called the *transition matrix* of the chain. [4] Here is an example with outcomes A, B, C for the R.V. X:

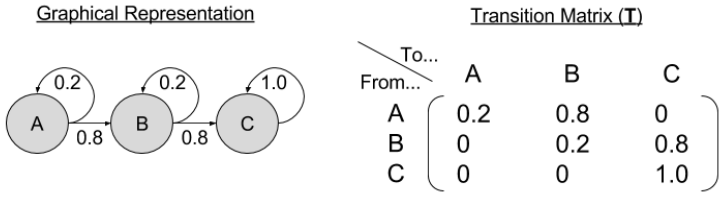


Figure 1. Graphical and matrix representation of transitioning from state X_{t-1} to X_t.
Note that A, B, C are the possible outcomes in one state.

This helps us to express the transition from all possible outcomes at one state to another in a matrix form. For example, if we know the probability distribution over outcomes A,B,C at time $t-1$, we can use the transition matrix to find out the probability distribution at t:

$$X_t = X_{t-1}Q = \begin{bmatrix} A_{t-1} \\ B_{t-1} \\ C_{t-1} \end{bmatrix} \begin{bmatrix} P(A|A) & P(A|B) & P(A|C) \\ P(B|A) & P(B|B) & P(B|C) \\ P(C|A) & P(C|B) & P(C|C) \end{bmatrix}$$

Definition 3 A row vector $s = (s_1, \cdots, s_m)$ such that $s_i \geq 0$ and $\sum_i s_i = 1$ is a *stationary distribution* of a Markov chain with transition matrix Q if:

$$\boldsymbol{s}Q = \boldsymbol{s} \text{ [5]}$$

This means that if we start from a marginal distribution of $\boldsymbol{s}$, no matter how many times the chain proceeds, the distribution stays the same across the outcomes. The probability of going out from an outcome at time $t-1$ equals the probability of going into an outcome t.

The reason for studying Markov Chain is that for complicated distributions (such as the posterior) to model, we want to construct a Markov Chain such that the stationary distribution is our distribution of interest. Here we explore algorithms to achieve precisely that.

The Metropolis-Hastings Algorithm

The following algorithm is cited from *Introduction to Probability* by Joseph K. Blitzstein and Jessica Hwang [6]. I would like not to reinvent the wheel given its precise and accurate explanation:

Let $s = (s_1, ..., s_M)$ be a desired stationary distribution on state space $1, ..., M$. Assume that $s_i > 0$ for all i. Suppose

that $P = (p_{ij})$ is the transition matrix for a Markov chain on state space $1, ..., M$. Intuitively, P is a Markov chain that we know how to run but that does not have the desired stationary distribution.

The goal is to modify P such that the Markov Chain has a stationary distribution $\boldsymbol{s}$. Start at any state X_0, and suppose that the new chain is currently at X_n. To make one move of the new chain, do the following.

1. If $X_n = i$, propose a new state j using the transition probabilities in the i^{th} row of the original transition matrix P.

2. Compute the acceptance probability $a_{ij} = min(\frac{s_j p_{ji}}{s_i p_{ij}}, 1)$

3. Flip a coin that lands Heads with probability a_{ij}.

4. If the coin lands Heads, accept the proposal (i.e., go to j), setting $X_{n+1} = j$. Otherwise, reject the proposal (i.e., stay at i), setting $X_{n+1} = i$.

Here is my intuitive understanding: we first initialize the transition probability p_{ij}, then we calculate the acceptance probability a_{ij} by taking the ratio of the posterior. We flip a coin with the probability landing head a_{ij}. We accept if the coin lands Head and reject otherwise. In the event that the proposed posterior probability is greater:

$$\frac{s_j p_{ji}}{s_i p_{ij}} > 1, \; min(\frac{s_j p_{ji}}{s_i p_{ij}}, 1) = 1$$

meaning that we will definitely move to the new state j. This process will converge to the the desired stationary distribution of θ as the number of samples we draw increase.

Estimating R_0 for COVID-19

A very useful and relevant application of MCMC is to estimate R_0 for COVID-19.

R0 is a number that indicates how contagious an infectious disease is. It is also known as the reproduction number. As an infection is transmitted to new people, it reproduces itself. R0 indicates how contagious a diseases is. If people are not vaccinated against a disease, every one person infected can spread to R0 number of new people infected. [7]

I will be using R to estimate the posterior distribution of R_0 in a simple SIR model for COVID-19 in Hubei, China. The simulation process is shown in Appendix I from the R notebook, with tutorial and dataset from [8][9].

Bibliography

[1][3] Haugh, Martin. 2017 Columbia University *MCMC and Bayesian Modeling*. Page 1-11.

[2][4][5][6] K. Blitzstein, Joseph and Hwang, Jessica. 2015 CRC Press *Introduction to Probability*. Chapter 8, 11, 12.

[7] Ramirez, Vanessa Bates. April 2020 *What Is R0? Gauging Contagious Infections*. https://www.healthline.com/health/r-nought-reproduction-number

[8] London School of Hygiene & Tropical Medicine. May 2019 *Sampling from an univariate distribution using MCMC*. http://sbfnk.github.io/mfiidd/mcmc.html

[9] Johns Hopkins University. 2020 https://data.humdata.org/dataset/novel-coronavirus-2019-ncov-cases

Figure 1. Binz, Kevin. October 2016 *An Introduction To Markov Chains*. https://kevinbinz.com/2016/10/17/markov-chains/

Appendix I

FinalProjectStats50

Ruiming Li

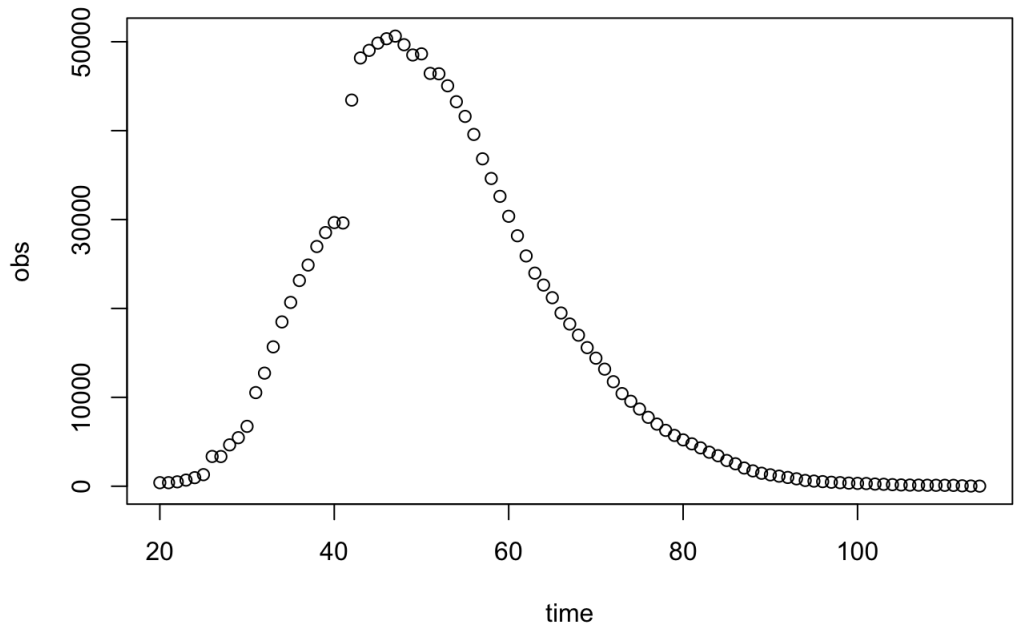
04/26/2020

The Dataset:

I chose dataset from Hubei Province, China since the epidemic has fallen off there and I can easily observe a complete cycle of cases. The original data is a time series data with 3 columns of Suspect, Infected, and Removed number of cases. This is in the form of a simple SIR epidemiological model, where S means people susceptible to the disease, I means the number of infected with the disease, and R means people removed from the disease. The removed population means either recovered or dead, and the model assumes that recovered patients will not contract the disease again.

After preprocessing, we can see the number of active cases by days.

```
hubei <- read.csv("simpleI.csv")
plot(hubei)
```



Likelihood

The likelihood is the probability of observing our Hubei dataset $X = x$ given $R_0 = \theta$ in a SIR model. We can say this is analogous to a "SIR" distribution with parameter R_0, and we want to find $P_{SIR}(X = x | R_0 = \theta)$. Since it is a time series data, the probability of observing the entire dataset across n days is $\sum_{t=0}^{n} P_{SIR}(X_t = x_t | R_0 = \theta)$. This process is complicated as the SIR model involves differential equations. Luckily, the probability in SIR model is given by the function in the package.

Here is an example of finding the log likelihood of observing the Hubei dataset given $\theta = 1.5$. Notice we also have to set the D_inf, the number of days a person stays infected and the initial SIR numbers respectively. The result is an unormalized log likelihood.

```
# install.packages("devtools")
# library(devtools)
# install_github("sbfnk/fitR")
library(fitR)
```

```
## Loading required package: deSolve
```

```
## Loading required package: adaptivetau
```

```
theta <- c(R0 = 1.5, D_inf = 2)
init.state <- c(S = 600000, I = 1, R = 0)
log.likelihood = dTrajObs(SIR, theta, init.state, hubei, log = TRUE)
print(log.likelihood)
```

```
## [1] -526709.9
```

Prior

The prior function is by default a uniform distribution in the SIR model.

```
SIR$dprior
```

```
## function(theta, log = FALSE) {
##
##     ## uniform prior on R0: U[1,100]
##     log.prior.R0 <- dunif(theta[["R0"]], min = 1, max = 100, log = TRUE)
##     ## uniform prior on infectious period: U[0,30]
##     log.prior.D <- dunif(theta[["D_inf"]], min = 0, max = 30, log = TRUE)
##
##     log.sum <- log.prior.R0 + log.prior.D
##
##     return(ifelse(log, log.sum, exp(log.sum)))
## }
```

Posterior:

With a prior and likelihood function, we essentially obtain a where $\textit{unnormalized posterior} = \textit{prior} * \textit{likelihood function}$.

The function below is the posterior function and we tested it with the same setup as likelihood above.

```
# This is a function that takes 4 arguments:
# - fitmodel, a fitmodel object that defines the model dynamics, prior and likelihoods.
# - theta, a named vector of parameters
# - init.state, a named vector of initial states
# - data, the data set we are fitting the model to
# It should return the posterior for the given model, parameters, initial state and data.
my_dLogPosterior <- function(fitmodel, theta, init.state, data) {

    # calculate the `fitmodel` log-prior for parameter `theta`
    log.prior <- fitmodel$dprior(theta,log=TRUE)
    # calculate the `fitmodel` log-likelihood for parameter `theta` and
    # initial state `init.state`, with respect to the data set `data`.
    log.likelihood <- dTrajObs(fitmodel, theta, init.state, data, log = TRUE)
    # return the logged posterior probability
    log.posterior <- log.prior + log.likelihood
    return (log.posterior)
}
target.current <- my_dLogPosterior(SIR, theta, init.state, hubei)
print(target.current)
```

```
## [1] -526717.9
```

MCMC Algorithm Definition

This is enough for our Metropolis-Hastings Algorithm to simulate and sample different θ values and do acceptance and rejections on those samples!

```
# This is a function that takes four arguments:
# - target: the target distribution, a function that takes one argument
#           (a number) and returns the (logged) value of the
#           distribution of interest
# - init.theta: the initial value of theta, the argument for `target`
# - proposal.sd: the standard deviation of the (Gaussian) proposal distribution
# - n.iterations: the number of iterations
# The function should return a vector of samples of theta from the target
# distribution
my_mcmcMH <- function(target, init.theta, proposal.sd, n.iterations) {

    # evaluate the function "target" at parameter value "init.theta"
    target.theta.current <- target(init.theta)

    # initialise variables to store the current value of theta, the
    # vector of samples, and the number of accepted proposals
    theta.current <- init.theta
    samples <- data.frame(theta = theta.current, target = target.theta.current)
    accepted <- 0

    # repeat n.iterations times:
    for (i in seq_len(n.iterations)){
    # - draw a new theta from the (Gaussian) proposal distribution
    #   with standard deviation sd.
        theta.proposed <- rnorm(n = length(theta.current),
                                mean = theta.current,
                                sd = proposal.sd)
    # - evaluate the function "target" at the proposed theta
        target.theta.proposed <- target(theta.proposed)
    # - calculate the Metropolis-Hastings ratio
        log.acceptance <- target.theta.proposed-target.theta.current
    # - draw a random number between 0 and 1
        a <- runif(1)
    # - accept or reject by comparing the random number to the
    #   Metropolis-Hastings ratio (acceptance probability); if accept,
    #   change the current value of theta to the proposed theta,
    #   update the current value of the target and keep track of the
    #   number of accepted proposals
        if(a < exp(log.acceptance)){
            theta.current <- theta.proposed
            target.theta.current <- target.theta.proposed
            accepted <- accepted + 1
        }

        samples <- rbind(samples, data.frame(theta = theta.current, target = target.theta.current), deparse.le

    }

    return (samples)
}
```

Running MCMC Algo

Before we run the MCMC algorithm, we have to change our posterior function (the target function in MCMC algorithm) a bit to make θ the only input. We make a wrapper function here.

```
my_dLogPosterior_R0 <- function(r0) {

  return(my_dLogPosterior(fitmodel = SIR,
                          theta = c(R0 = r0, D_inf = 2.5),
                          init.state = c(S = 600000, I = 1, R = 0),
                          data = hubei))
}
```

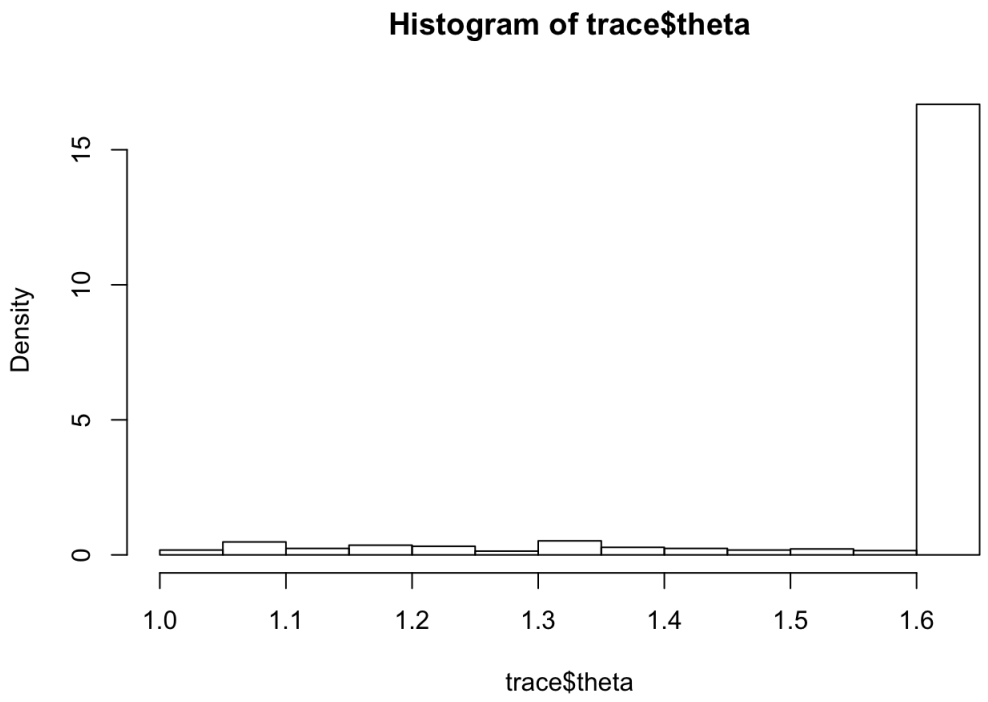
Then we call the MCMC algorithm above using the target function after setting the parameters.

```
trace <- my_mcmcMH(target = my_dLogPosterior_R0, init.theta=1, proposal.sd=0.01, n.iterations = 1000)
```

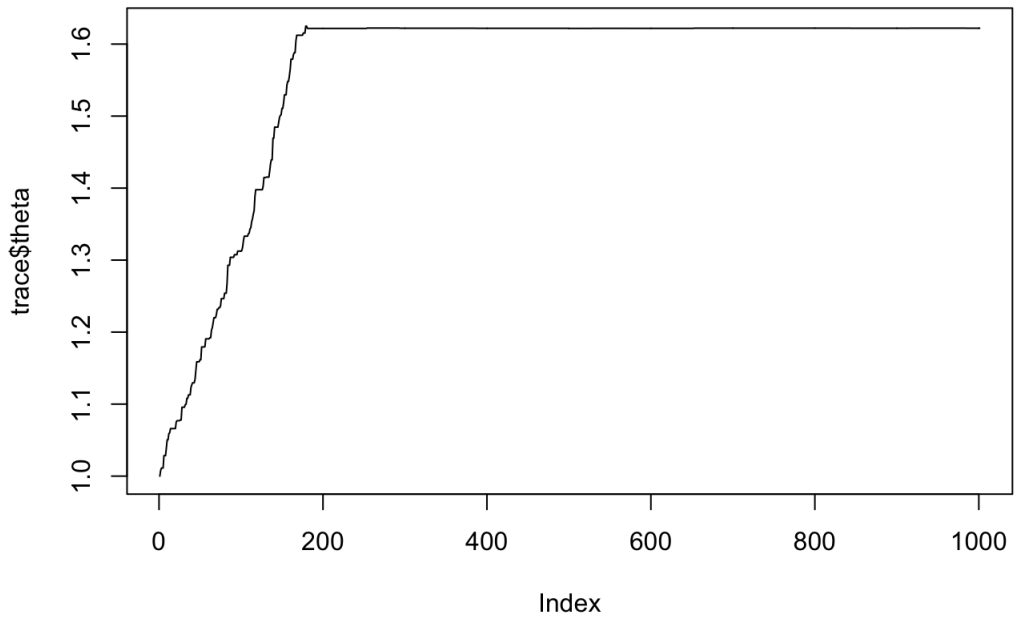
Plotting and Evaluating Results

We first plot a historgram of all the θ values sampled as well as a trace of θ over the 1000 iterations. We also pick the best θ value with the highest occurence (mode) to fit our model.

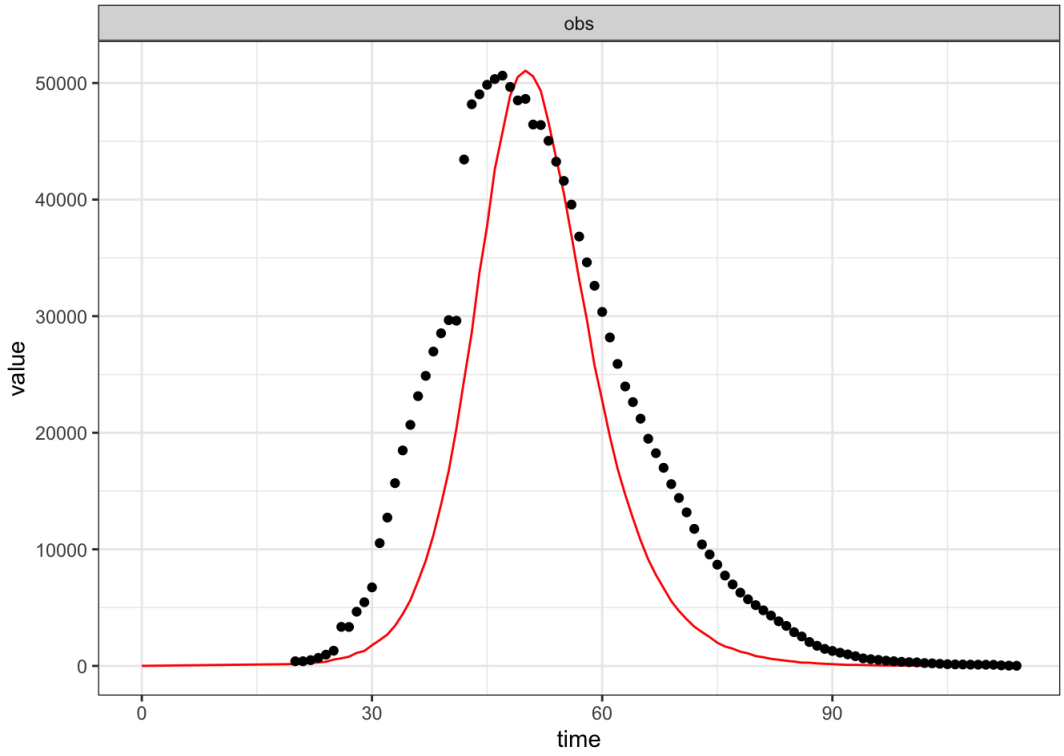
```
h <- hist(trace$theta, freq=FALSE)
```




```
plot(trace$theta,type = "l")
```



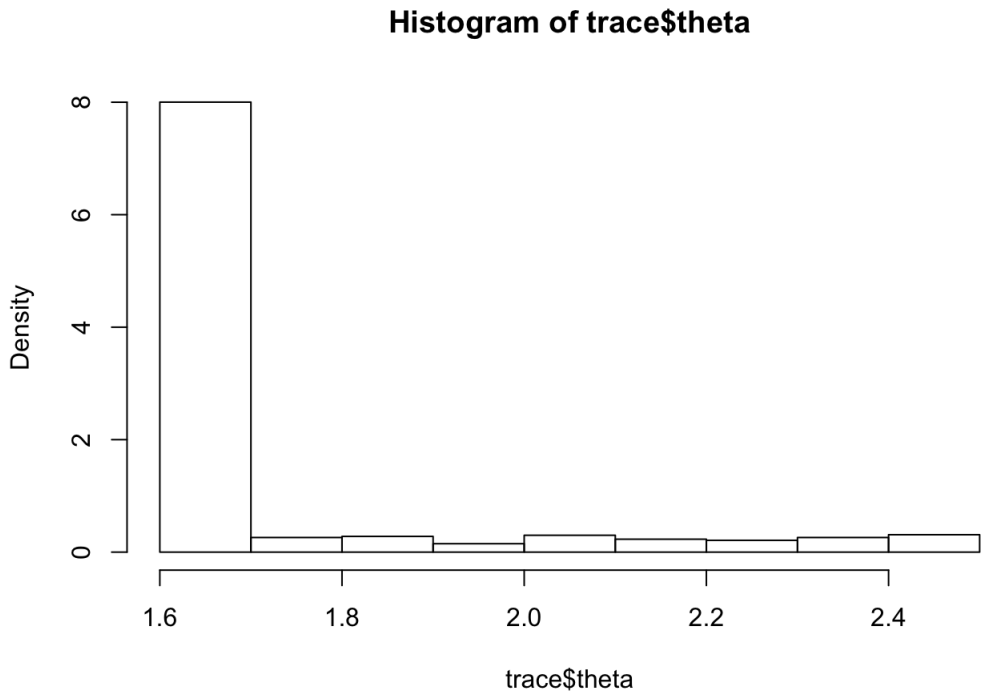
```
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
mode <- getmode(trace$theta)
theta <- c(R0 = mode, D_inf = 2.5)
init.state <- c(S = 600000, I = 1, R = 0)
plotFit(SIR, theta, init.state, hubei)
```



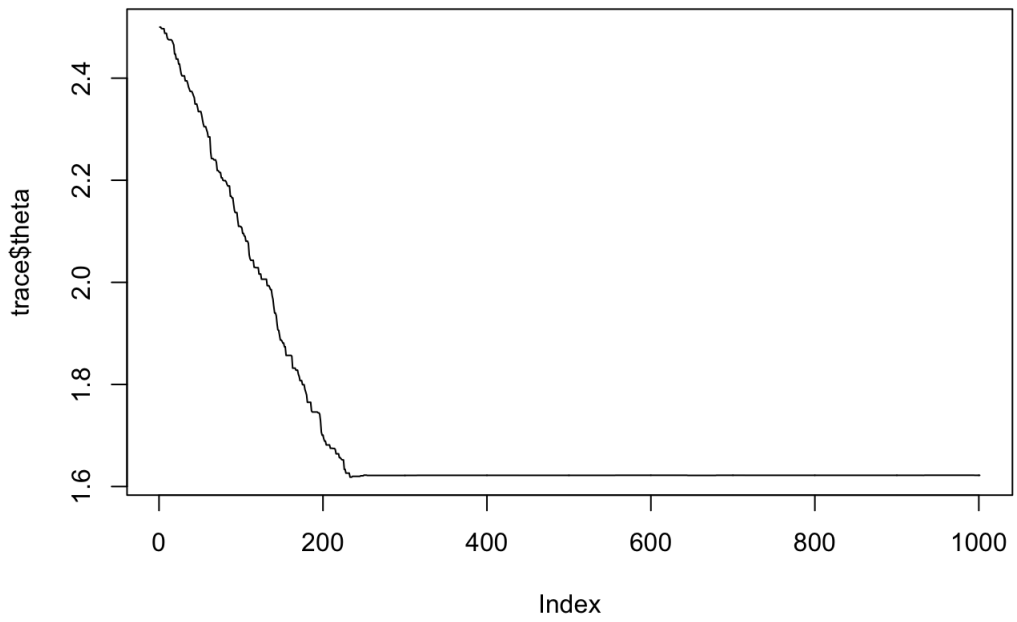
Improving Results

From the above trace of θ over the iterations, it is obvious that we started of the initial value too low and the random walk is mostly climbing up towards 1.6 and stay flat there. Hence we initialize a larger value and tried MCMC again.

```
trace <- my_mcmcMH(target = my_dLogPosterior_R0, init.theta=2.5, proposal.sd=0.01, n.iterations = 1000)
h <- hist(trace$theta, freq=FALSE)
```

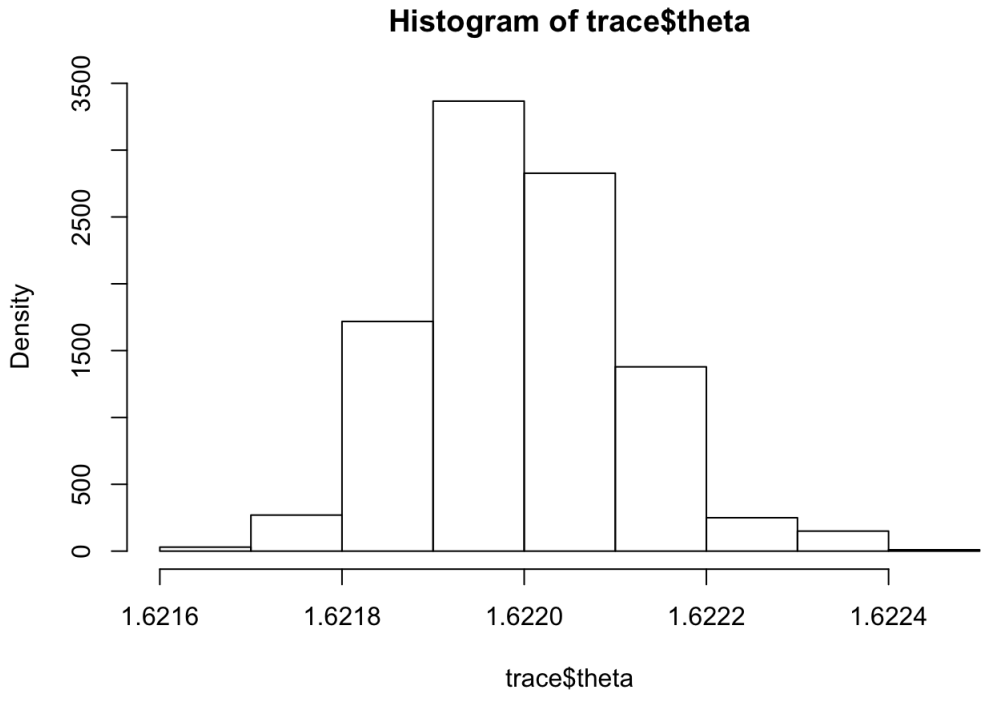


```
plot(trace$theta,type = "l")
```

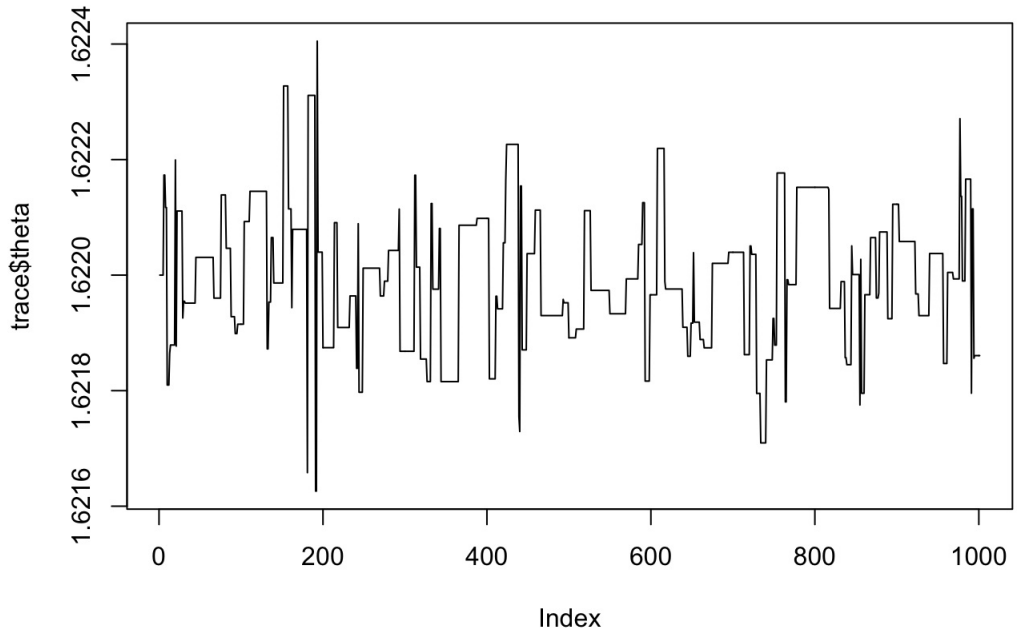


Again we saw a clear slope in the MCMC trace for θ, indicating a lower value needed. From both directions we saw that 1.6 is the converged value. Hence we start from 1.622 and take samller walks using a standard deviation of 0.001. This gives us a better fit overall and a clear distribution of θ!

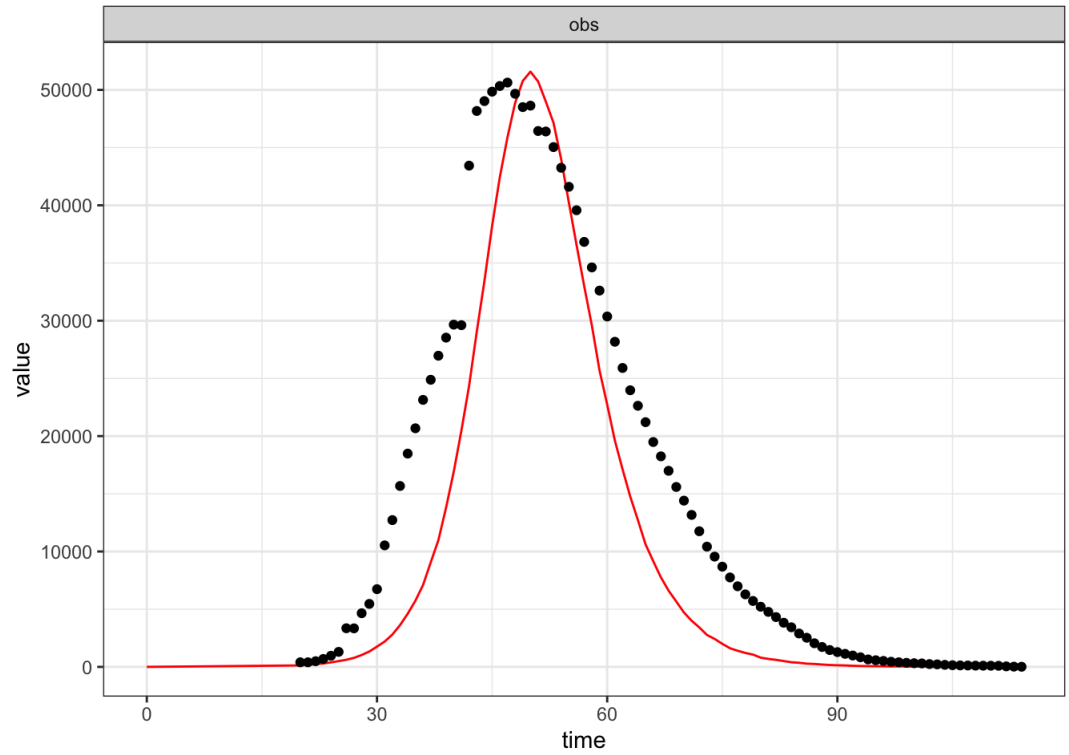
```
trace <- my_mcmcMH(target = my_dLogPosterior_R0, init.theta=1.622, proposal.sd=0.001, n.iterations = 1000)
h <- hist(trace$theta, freq=FALSE)
```




```
plot(trace$theta,type = "l")
```




```
mode <- getmode(trace$theta)
theta <- c(R0 = mode, D_inf = 2.5)
init.state <- c(S = 600000, I = 1, R = 0)
plotFit(SIR, theta, init.state, hubei)
```



Limitation and Future Work

The MCMC algorithm has a few limitations: as we saw above, there are actually two parameters in the SIR model - the R0 and D_inf - number of days a person stays infected. I hard coded the day of infection based on literature and estimation but to improve the reliability of the results we should do treat θ as a vector and estimate R0 and D_inf together.

The initial population S is also quite arbitrary here because I am not sure how many people are in close contact with one another and susceptible to the disease. The current value is simply an approximation based on the population and trial and errors results.

Another limitation is that we are assigning one number of θ to the entire dataset, but in real life the parameter is constantly changing, especially in phases when social distancing measures are put in place.

- The algorithms above are works inspired by http://sbfnk.github.io/mfiidd/mcmc.html, which provides framework for MCMC and I completed the code on my own as the practice exercise.