CGS698C, Lectures 8-10: Parameter estimation

Himanshu Yadav

2024-03-05

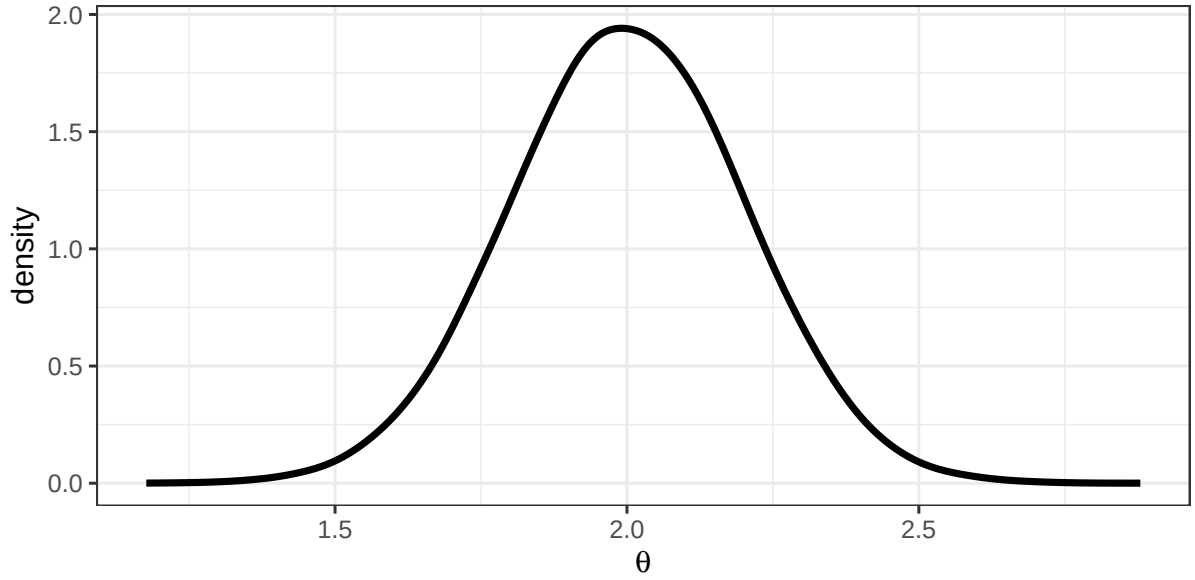
Contents

1 The goal

Given the observed data y, the likelihood function $\mathcal{L}(\theta|y)$, and the prior distribution $p(\theta)$, we want to estimate the posterior distribution of the parameter θ. Not just that, **we need samples from the posterior distribution of** θ.

What do we mean by *samples from the posterior* distribution of a parameter?

```
## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```



The above graph depicts the posterior density function of the parameter θ. The samples from the posterior would be — a set of values of θ that have relatively high posterior density. For example, in the above graph, the values of θ between 1.5 and 2.5 would have higher posterior densities compared to values above 2.5 or below 1.5. Thus, we need a 'sampler' that somehow gives us a lot of values from the high posterior density regions and fewer values from the low posterior density regions.

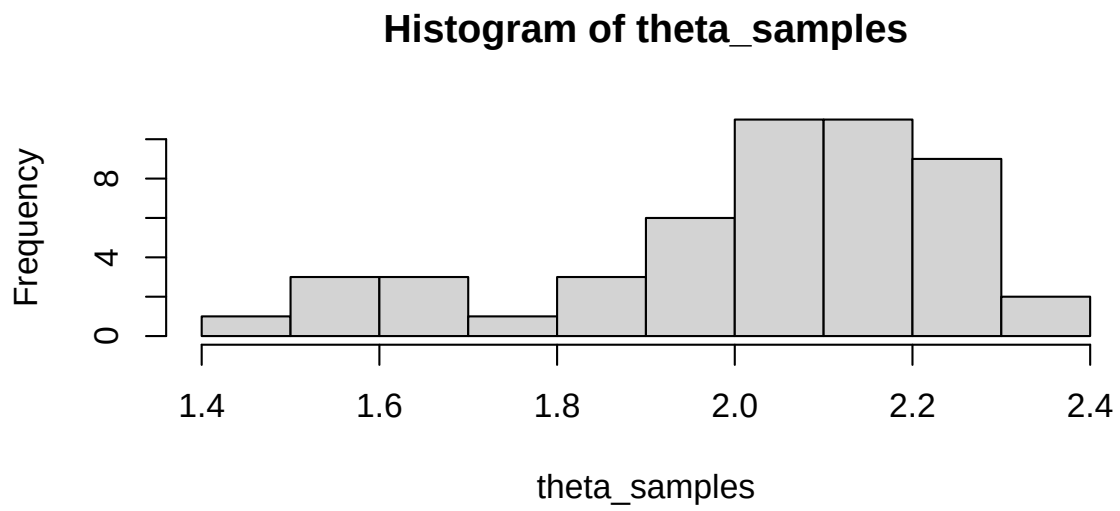
Consider the following vector 'theta_samples'. It contains 50 samples from the posterior distrobution of θ. This is how it looks like.

```
theta_samples
```

```
##  [1] 2.008893 2.230791 2.084479 2.191269 1.986300 1.885667 1.983064 2.113245
##  [9] 2.282375 2.085077 2.269505 1.693534 1.712836 1.494848 2.227980 2.098325
## [17] 2.228998 2.209589 1.928681 1.947209 2.042633 2.168113 1.659643 1.821059
## [25] 2.135490 2.208139 2.132353 2.019475 2.122390 1.545021 2.097381 2.154220
## [33] 2.022108 2.202916 2.055284 1.845160 2.309225 1.563315 2.122045 2.116911
## [41] 2.052144 2.016966 2.348653 2.146541 2.128167 1.914622 1.940144 1.624330
## [49] 1.599847 2.282458
```

A histogram of these values should look similar to the posterior density graph.

```
hist(theta_samples,breaks = 8)
```



1.1 *The challenge*

Given the likelihood and the prior, we can calculate the posterior density of θ using Bayes' rule:

$$p(\theta|y) = \frac{\mathcal{L}(\theta|y)p(\theta)}{\int \mathcal{L}(\theta|y)p(\theta)\,d\theta}$$

In order to sample from the posterior distribution of θ, a straightforward way is to analytically derive the posterior density function $p(\theta|y)$ and directly draw independent samples from this function.

However, we can analytically derive the posterior in a limited number of cases. For practical applications, we often do not have the analytical solution of the posterior distribution. That means, in most cases, we cannot directly sample from the posterior density function.

If we cannot directly sample from the posterior density function, we can still use **the relative posterior density** to collect a lot of samples from the high poserior density regions.

We need algorithms that can produce more samples from the high posterior density regions and less samples from the low posterior density regions. You can call them *posterior simulation algorithms*.

In sum, there are two approaches that you can use to draw samples from the posterior distribution of a parameter: *(i) Analytical approach:* Sampling from the analytically-derived posterior distribution, {(ii) Computational approach:} Sampling using a posterior simulation algorithm. I will talk about these two approaches in this lecture.

2 *Parameter estimation using analytically-derived posterior*

If the posterior density function $p(\theta|y)$ belongs to the same probability distribution family as the prior distribution $p(\theta)$, the prior is then called a conjugate prior for the likelihood function $\mathcal{L}(\theta|y)$.

In these cases, where you (can) specify conjugate priors for all the free parameters of the model, you can derive a posterior distribution analytically and you can easily sample from this distribution.

Some examples of the conjugate distributions are:

1. Binomial likelihood — Beta prior (on the probability of success)
2. Poisson likelihood — Gamma prior (on the rate parameter)
3. Normal likelihood — Normal prior on the mean; Inverse gamma prior on the variance
4. Multivariate normal likelihood — Multivariate normal prior on the mean; Inverse wishart prior on the variance-covariance matrix

Consider the case of a Beta-binomial model.

Suppose x_i is number of successes in N trials in an experiment i; x_i is assumed to come from a Binomial likelihood with probability of sucess θ, where θ is assumes to have a Beta prior.

$$x \sim Binomial(N, \theta)$$

$$\theta \sim Beta(\alpha, \beta)$$

Suppose the observed data from n successive experiments is $x_1, x_2, x_3, ..., x_n$.

The anlytical derived posterior of θ given the data $x_{1:n}$will be a Beta distribution with parameters α^* and β^*, where

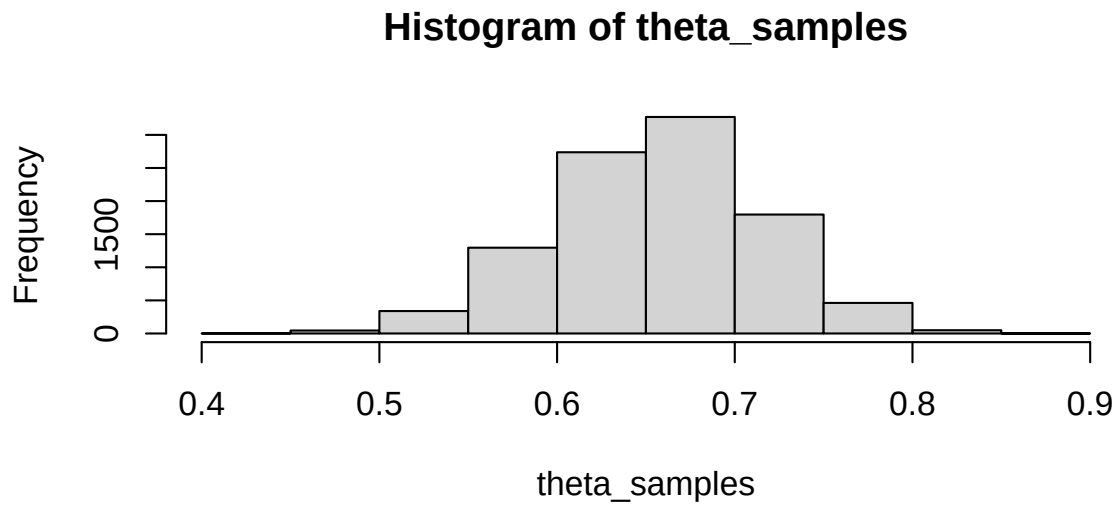
$\alpha^* = \alpha + \sum_{i=i}^{n} x_i$ and $\beta^* = \beta + \sum_{i=1}^{n} N - x_i$

We can write the sampling statement

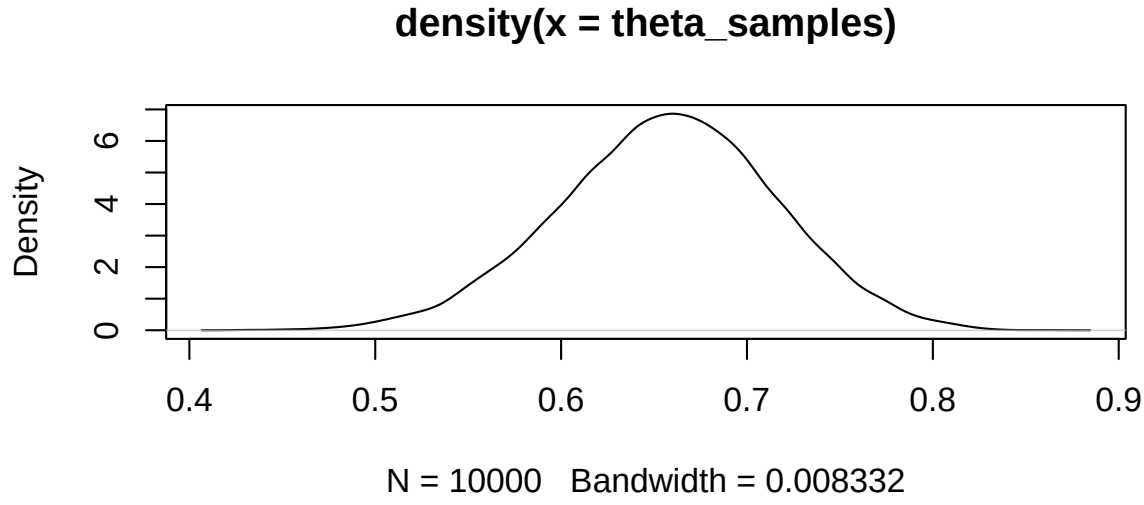
$$\theta | x_{1:n} \sim Beta\Big(\alpha + \sum_{i=i}^{n} x_i, \beta + \sum_{i=1}^{n} (N - x_i)\Big)$$

Let us use the above statement to draw samples from the posterior distribution of θ.

```
#Suppose, you are given
N <- 10
x <- c(7,6,6,5,9,7)
alpha <- 2
beta <- 2

theta_samples <- rbeta(10000,shape1=alpha + sum(x),shape2=beta+sum(10-x))

hist(theta_samples)
```




```
plot(density(theta_samples))
```



3 *Parameter estimation using posterior simulation algorithms*

When we do not have an analytically-derived posterior for sampling, we need to find other solutions for parameter estimation.

Four classes of solutions:

1. Approximate the posterior density function using discretization

   (a) Grid approximation

2. Estimate the denominator using independent samples from a continuous probability distribution
   (a) Monte Carlo integration; importance sampling
3. Draw dependent samples from the posterior based on their *relative posterior densities* (the method uses only the numerator term of Bayes' rule)
   (a) Markov chain Monte Carlo (MCMC) – Metropolis-Hastings
   (b) Hamiltonian Monte Carlo
   (c) Sequential Monte Carlo
4. Draw samples from a *conditional* posterior density function
   (a) Gibbs sampling

The methods (1) and (2) aim to approximate the precise posterior density for a given value of the parameter θ. But you cannot use them draw samples from the posterior. These methods mostly used for estimating the marginal likelihood; the marginal likelihood estimate is needed for quantifying evidence for a model.

The methods (3) and (4) are commonly used for drawing samples from the posterior.

I will introduce methods (1) and (2) first, because they provide some foundational ideas for sampling algorithms.

4 Grid approximation

Goal: we want to approximate the exact posterior densities for a set of parameter values.

4.1 Idea

- Discretise a continuos parameter
- Because it is easier to compute $\sum p(y|\theta)p(\theta)$

4.2 Method

- Suppose we want to estimate the marginal likelihood $\int p(y|\theta)p(\theta)$.
- Divide the parameter space θ into n equally spaced points, $v_1, v_2, .., v_n$ called *grid points*
- For each grid point, v_i, calculate the likelihood $p(y|v_i)$ and the prior density $p(v_i)$
- Add the product of the likelihood and the prior from all grid points to approximate the marginal likelihood, $p(y) \approx \sum_{i=1}^{n} p(y|v_i)p(v_i)$.
- Estimate posterior density at each grid point, v_i, using $\frac{p(y|v_i)p(v_i)}{\sum_{i=1}^{n} p(y|v_i)p(v_i)}$
  - This gives us a discretised version of the posterior distribution

Note: If θ represents a set of m parameters such that $\theta = \{\theta_1, \theta_2, ..., \theta_m\}$, then divide each parameter space, θ_j into n equally spaced points, $v_{j,1}, .., v_{j,n}$ and use all possible combinations to create n^m grid points.

4.3 *Implementation*

Example. A normal model with unknown mean and known variance

Suppose you are given 10 independent and identically distributed data points that are assumed to come from a Normal distribution with mean μ and variance 4. Let y_i be the i^{th} data point,

$y_i \sim Normal(\mu, \sigma^2 = 4)$

$\mu \sim Normal(\mu_0 = 0, \sigma_0^2 = 9)$

We can derive the posterior distribution analytically,

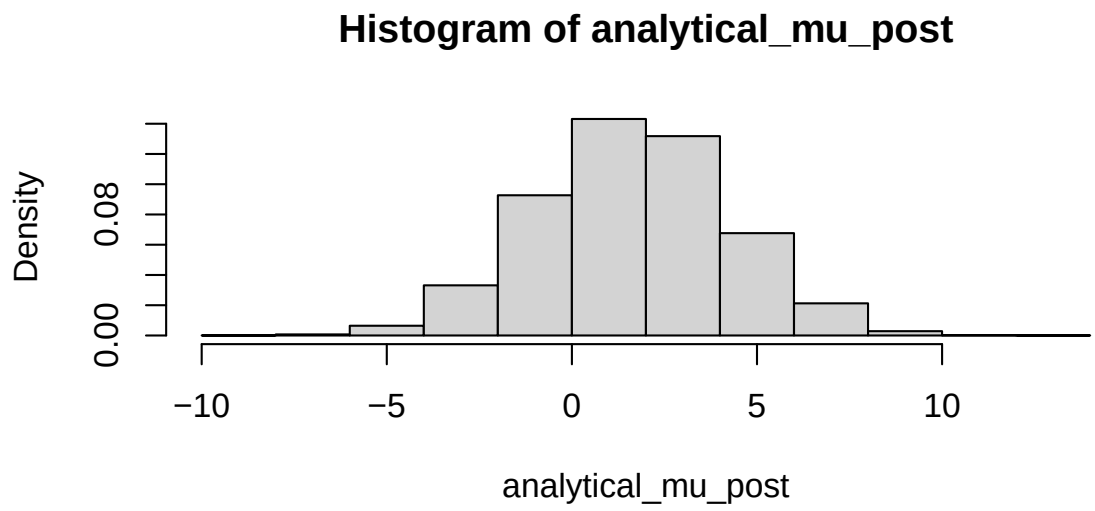
$\theta|y \sim Normal(\frac{\sigma^2\mu_0 + \sigma_0^2 \sum_{i=1}^{n} y_i}{\sigma^2 + n\sigma_0^2}, \frac{1}{\sigma_0^2} + \frac{n}{\sigma^2})$

where n is the number of data points.

```
# Assuming, true parameter value, mu=1
# Observed data
y <- rnorm(10,1,2)
y
```

```
##  [1]  2.5737558  3.0636557  1.8655166  3.5973279 -0.2497399  1.4768745
##  [7] -0.1542030  1.6486695  0.6963396  2.4443189
```

```
#Analytical posterior
sigma = 2  # Known standard devation of normal distribution
mu_prior = 0 # Mean of prior distribution on mu
sigma_prior = 3 # Standard deviation of prior distribution on mu
n = 10 # no. of observations
analytical_mu_post <- rnorm(10000,
                            mean=(((sigma^2)*(mu_prior))+
                                    ((sigma_prior^2)*sum(y)))/
                              (sigma^2 + (n*(sigma_prior^2))),
                            sd=(1/(sigma_prior^2))+(n/(sigma^2)))
hist(analytical_mu_post,freq = FALSE)
```



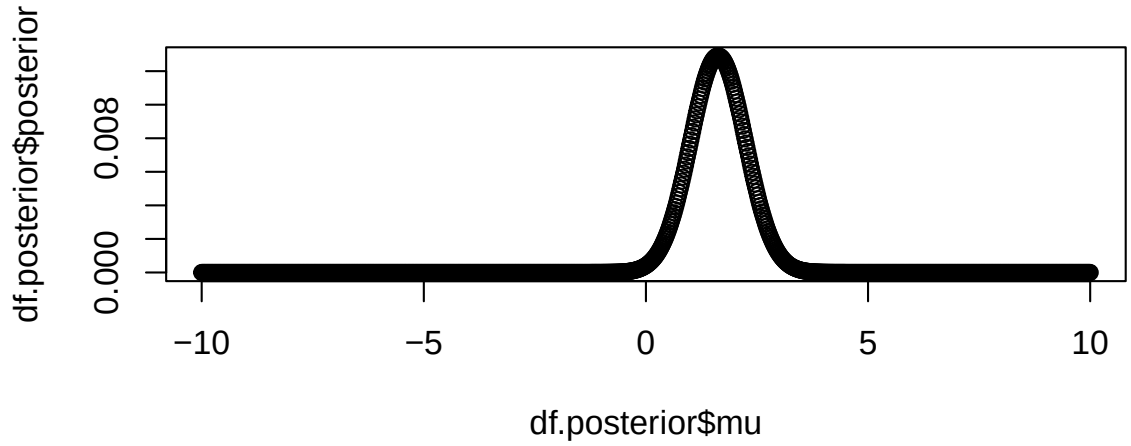

```r
# Grid approximation

# Create grid points
mu_grid <- seq(-10,10,length=1000)
head(mu_grid)
```

```
## [1] -10.00000  -9.97998  -9.95996  -9.93994  -9.91992  -9.89990
```

```r
length(mu_grid)
```

```
## [1] 1000
```

```r
#Calculate likelihood and posterior at each grid point
df.posterior <- data.frame(matrix(ncol=3,nrow=length(mu_grid)))
colnames(df.posterior) <- c("mu","likelihood","prior")
for(i in 1:length(mu_grid)){
  likelihood <- prod(dnorm(y,mu_grid[i],2))
  prior <- dnorm(mu_grid[i],0,3)
  df.posterior[i,] <- c(mu_grid[i],likelihood,prior)
}

#Approximate marginal likelihood
df.posterior$ML <- rep(sum(df.posterior$likelihood*df.posterior$prior),1000)

#Estimate posterior density at each grid point
df.posterior <- df.posterior %>%
  mutate(posterior=likelihood*prior/ML)
plot(df.posterior$mu,df.posterior$posterior)
```



4.4 *Drawbacks*

- The curse of dimensionality
  - The number of computations increases exponentially with an increase in the number of parameters
  - Given m parameters, if we divide each parameter space into n equally spaced points, we will create n^m grid points
  - Say we have to estimate 4 parameters and we discretize each parameter space into 1000 points, the number of grid points will be 1000^4
    - If we decrease the number of discrete points, the approximation becomes poorer

5 *Monte Carlo integration*

5.1 *Idea*

Expectation of a function $f(\theta)$ can be approximated using independent samples from probability density function of θ, i.e., $p(\theta)$,

$$\int f(\theta)p(\theta)d\theta \approx \frac{1}{n}\sum_{i=1}^{n} f(\tilde{\theta}_i) \quad \text{where } \tilde{\theta}_i \sim p(\theta)$$

The expression $\frac{1}{n}\sum_{i=1}^{n} f(\tilde{\theta}_i), \tilde{\theta}_i \sim p(\theta)$ means that we draw n independent samples from the distribution $p(\theta)$ and calculate average of function f() applied to all n samples.

- We can use this idea to approximate the marginal likelihood
  - Marginal likelihood, ML, is 'expectation of likelihood function'

  - $ML = E(p(y|\theta)) = \int p(y|\theta)p(\theta)d\theta$
  - $ML \approx \frac{1}{n}\sum_{i=1}^{n} p(y|\tilde{\theta}_i), \tilde{\theta}_i \sim p(\theta)$
    * Means you are drawing n independent samples from the prior distribution and calculating the average of the likelihood of all n samples

- We may choose to not to sample from the prior distribution
  - Use an importance density, $g(\theta)$
  - $\int p(y|\theta)p(\theta)d\theta \approx \frac{1}{n}\sum_{i=1}^{n} \frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{g(\tilde{\theta}_i)}, \tilde{\theta}_i \sim g(\theta)$
  - Importance density $g(\theta)$ should resemble the posterior distribution and have fatter tails than the posterior distribution
  - When it is useful?
    * When the posterior distribution is peaked (too narrow) relative to the prior
    * In this situation, most of the samples from the prior will have likelihood zero

5.2 *Method*

- Draw n independent samples $\tilde{\theta}_1, \tilde{\theta}_2, .., \tilde{\theta}_n$ from the prior distribution $p(\theta)$
- Compute likelihood for each sample, $p(y|\tilde{\theta}_i)$
- Calculate average of likelihoods, $\sum_{i=1}^{n} p(y|\tilde{\theta}_i)$ to approximate the marginal likelihood

5.3 *Implementation*

Example. A beta-binomial model

Suppose n is the sample size, k is the number of successes,
$k|n, \theta \sim Binomial(n, \theta)$
$\theta \sim Beta(a, b)$
We can derive the marginal likelihood, ML, analytically,
$ML = \binom{n}{k} \frac{(k+a-1)!(n-k+b-1)!}{(n+a+b-1)!}$
Suppose
k=2, n=10, a=1, b=1
$ML = 1/(10+1) = 0.0909$
and posterior distribution of θ will be
$\theta|k = 2, n = 10 \sim Beta(3, 9)$
Let us estimate the marginal likelihood and the posterior using a Monte Carlo estimator.

```
df.estimate <- data.frame(matrix(ncol=2,nrow=10000))
colnames(df.estimate) <- c("theta_sample","likelihood")
for(i in 1:10000){
  theta_i <- rbeta(1,1,1) # independent sample from the prior
  likelihood <- dbinom(2,10,theta_i)
  df.estimate[i,] <- c(theta_i,likelihood)
}
```

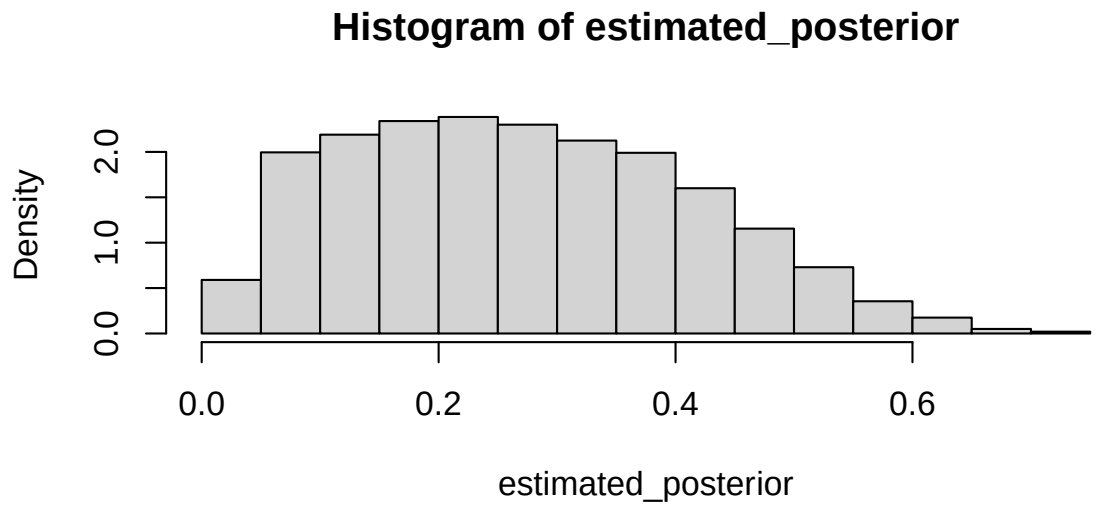
```
# Marginal likelihood
ML <- mean(df.estimate$likelihood)
ML
```

```
## [1] 0.08987182
```

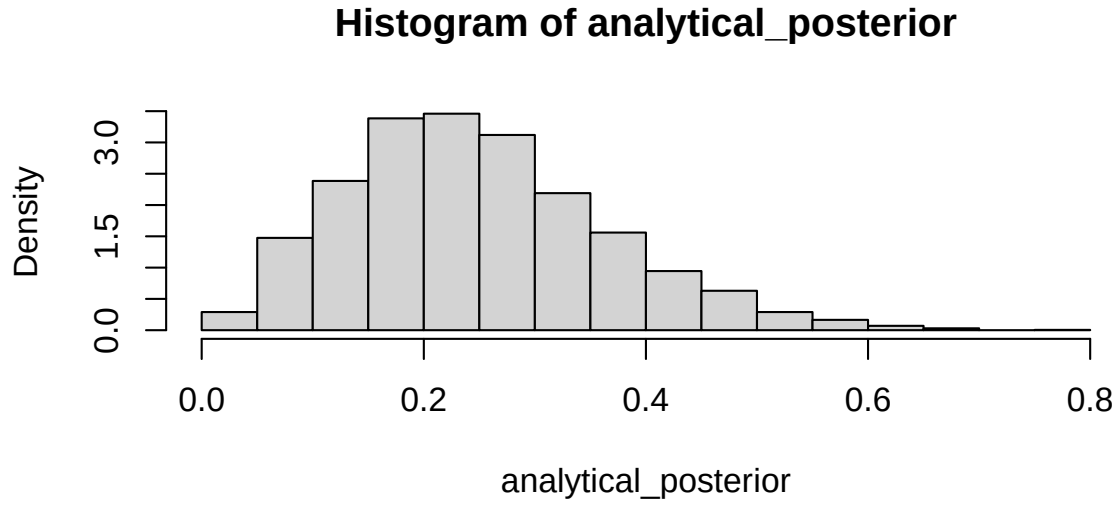
5.4 Complications

- But what about the posterior distribution?
- You have estimated the marginal likelihood ML, you know the likelihood function, you know the prior distribution; can you draw independent samples from the posterior distribution?
  - Not directly! But you can compute posterior density for a parameter value
  - Apply *rejection sampling* to draw independent samples from the posterior
    - Draw a sample, θ^* from a uniform distribution Uniform(a,b), where [a,b] is 'possible' range of parameter values having non-zero posterior density
    - Compute posterior density for the sample θ^*, call it pd^*
    - Sample t from a uninform distribution, $t^* \sim Uniform(0,1)$
    - If $pd^* > t$, accept, θ^* as a sample from the posterior distribution
  - **Drawbacks:**
    - only a small proportion of the initial samples are accepted
    - Inefficiency increases exponentially with the number of parameters
  - Another way to draw independent samples from the posterior
    - Ignore the denominator term
    - Sample from the 'un-normalized' posterior distribution
    - Histogram of independent samples from the un-normalized posterior will be a proxy for the posterior density function

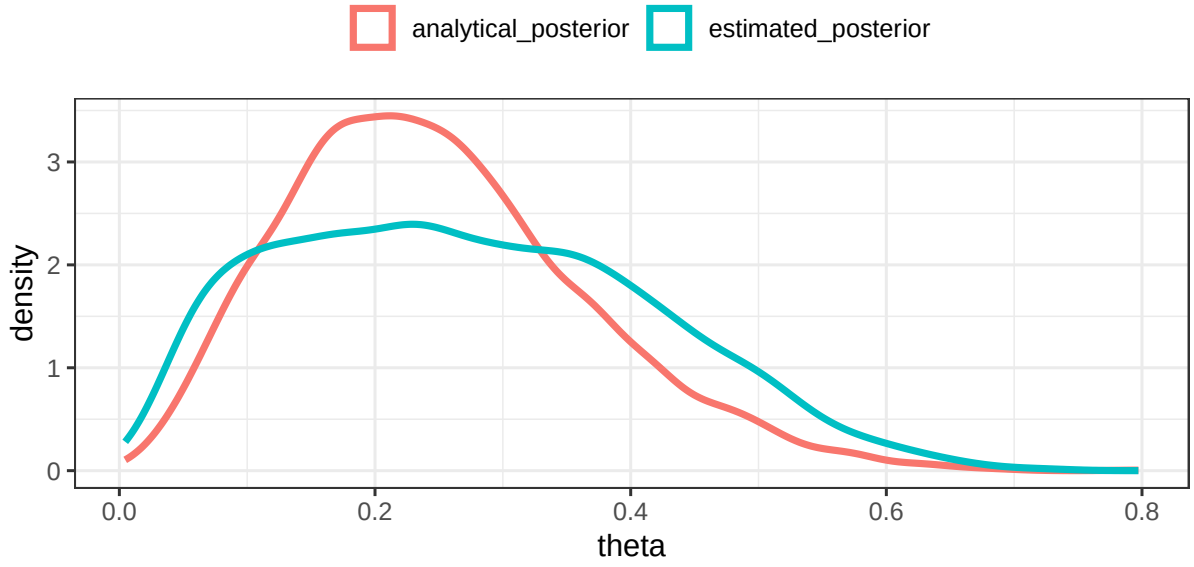
```
# Samples from the un-normalized posterior distribution
estimated_posterior <- sample(df.estimate$theta_sample,
                              size = 4000,
                              prob = df.estimate$likelihood)

hist(estimated_posterior,freq = FALSE)
```




```
analytical_posterior <- rbeta(4000,3,9)
hist(analytical_posterior,freq = FALSE)
```




```
posteriors <- data.frame(analytical_posterior,estimated_posterior)
ggplot(melt(posteriors),aes(x=value,colour=variable))+
  geom_density(size=1.2)+theme_bw()+
  xlab("theta")+theme(legend.title = element_blank(),
                      legend.position = "top")

## No id variables; using all as measure variables
```



- We have not estimated the absolute posterior density; but we are able to estimate relative posterior density of the samples quite well
  - Relative posterior density is the basis of dependent sampling in Markov chain Monte Carlo algorithms

Unresolved issues:

- The parameter space explored by the estimator depends on the proposal distribution, i.e., prior distribution or importance distributon
  - Ideally, we want most of the samples from non-zero posterior density regions

6 Markov chain Monte Carlo (MCMC)

6.1 Idea

- Explore the parameter space in such a way that the histogram of their samples produces the target distribution
  - Do not care about absolute posterior density
  - Draw zero samples from zero posterior density regions
  - Draw relatively more samples from higher posterior density regions
- Collection of samples from one iteration to the other is a Markov process.
  - Evolution of the chain only depends on the current position; past samples cannot be used to determine new positions in parameter space
- A proposal sample is evaluated based on its relative posterior density

6.2 Method

Suppose we want to estimate the posterior distribution, $p(\theta|y)$,
$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$
We are going to generate a chain of n samples from the posterior distribution of θ.

- Initialize the chain for parameter θ with some value θ_0
- for $i = 1$ to $i = n$:
  - Generate a proposal: $\theta^* \sim q(\theta^*|\theta_i)$
    - θ_i is the current state of the chain
    - $q()$ is the proposal distribution; I will explain it in a while
  - Compute likelihood and prior for θ^* and θ_i
  - Compute Hastings ratio: $H = \frac{p(y|\theta^*)p(\theta^*)q(\theta_i|\theta^*)}{p(y|\theta_i)p(\theta_i)q(\theta^*|\theta_i)}$
  - Sample from uniform distribution, $p \sim Uniform(0, 1)$
  - if $p < min(1, H)$,
    - Update the chain: $\theta_{i+1} = \theta^*$
- The proposal distribution q proposes a new parameter value conditional on the current state of the chain.
  - A commonly used proposal distribution is a normal distribution with mean θ_i and some variance σ^2, such that $\theta^* \sim Normal(\theta_i, \sigma^2)$
  - σ is called the step-size parameter and it is critical for the algorithm to work
- The chain will converge to the target distribution if the proposal distribution, q has following three properties:
  - We must be able to explore the parameter space in a finite number of steps
    - This will be impossible if step-size is too small
  - We must be able to re-visit previously explored parameter space in a finite number of steps
  - The chain should not get stuck in cycles
- For computational feasibility, the step-size should not be too large
  - Rejection rate increases with increase in the value of the step-size parameter
- MCMC algorithms may differ in the choice of proposal distribution or in methods to evaluate the relative posterior density of the samples

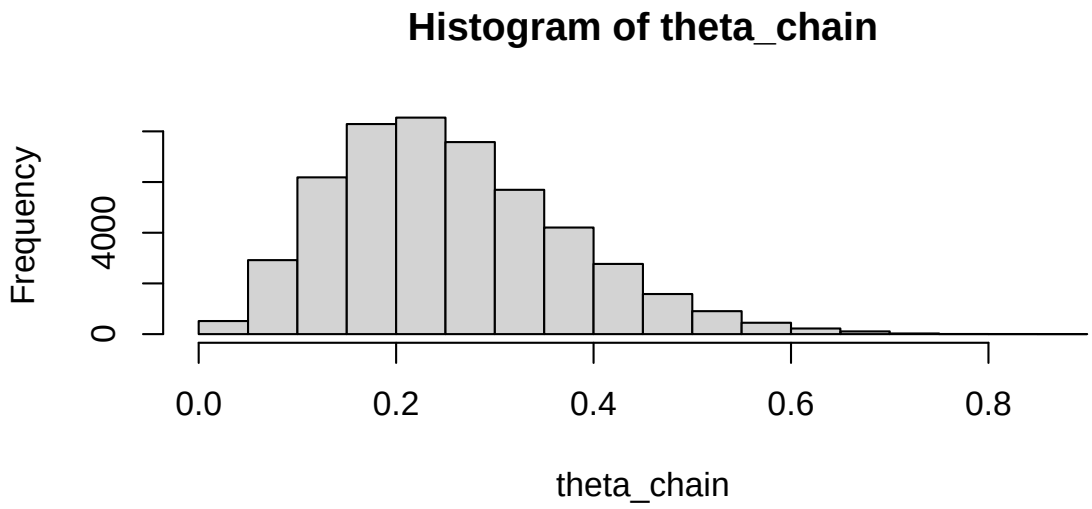
6.3 Implementation

Example. A beta-binomial model

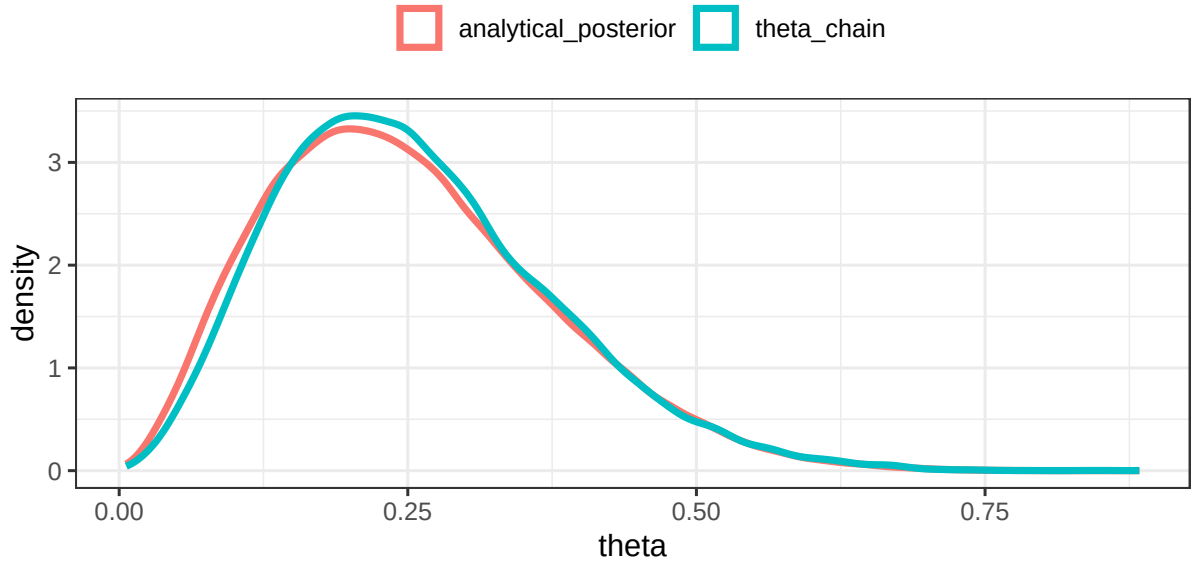
Suppose n is the sample size, k is the number of successes,
$k|n,\theta \sim Binomial(n,\theta)$
$\theta \sim Beta(a,b)$
We can derive the marginal likelihood, ML, analytically,
$ML = \binom{n}{k}\frac{(k+a-1)!(n-k+b-1)!}{(n+a+b-1)!}$
Suppose
k=2, n=10, a=1, b=1
$ML = 1/(10+1) = 0.0909$
and posterior distribution of θ will be
$\theta|k=2,n=10 \sim Beta(3,9)$
Let us estimate the posterior distribution using a simple Metropolis-Hastings sampler.

```
k <- 2
n <- 10
a <- 1
b <- 1

# Markov chain
nsamp <- 50000
theta_chain <- rep(NA,nsamp)

#Initialization of Markov chain
theta_chain[1] <- rbeta(1,1,1)

#Evolution of Markov chain
i <- 1
step <- 0.08 # step-size for proposal distribution
while(i<nsamp){
  #Sample from proposal distribution
  proposal_theta <- rnorm(1,theta_chain[i],step)
  # This is not a very good proposal distribution,
  # because proposed values can go out of [0,1] range
  if(proposal_theta>0&proposal_theta<1){
    # Compute prior*likelihood
    post_new <- dbinom(k,n,proposal_theta)*dbeta(proposal_theta,a,b)
    post_prev <- dbinom(k,n,theta_chain[i])*dbeta(theta_chain[i],a,b)
    #Compute Hastings ratio
    Hastings_ratio <- (post_new*dnorm(theta_chain[i],proposal_theta,step))/
      (post_prev*dnorm(proposal_theta,theta_chain[i],step))
    p_str <- min(Hastings_ratio,1) # probability of acceptance
    if(p_str>runif(1,0,1)){
      theta_chain[i+1] <- proposal_theta
```

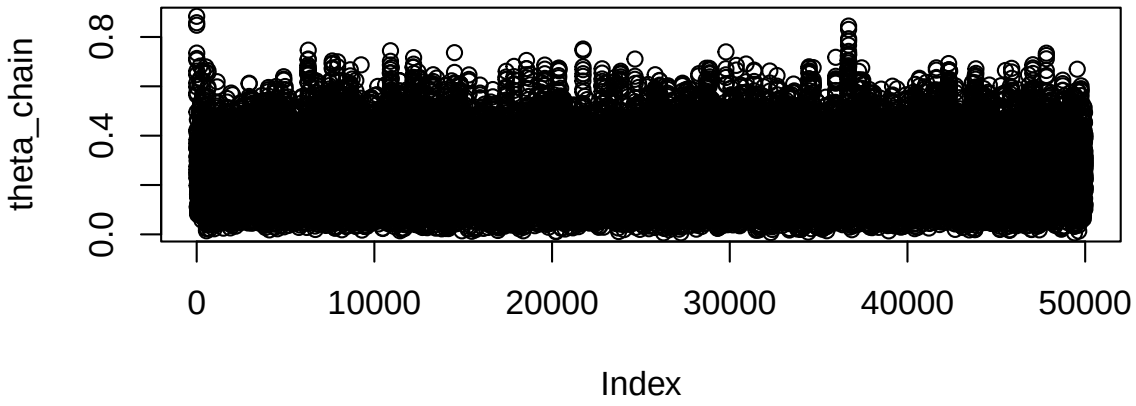
```
      i <- i+1
    }
  }
}
hist(theta_chain)
```




```
analytical_posterior <- rbeta(50000,3,9)
posteriors <- data.frame(analytical_posterior,theta_chain)
ggplot(melt(posteriors),aes(x=value,colour=variable))+
  geom_density(size=1.2)+theme_bw()+
  xlab("theta")+theme(legend.title = element_blank(),
                      legend.position = "top")

## No id variables; using all as measure variables
```



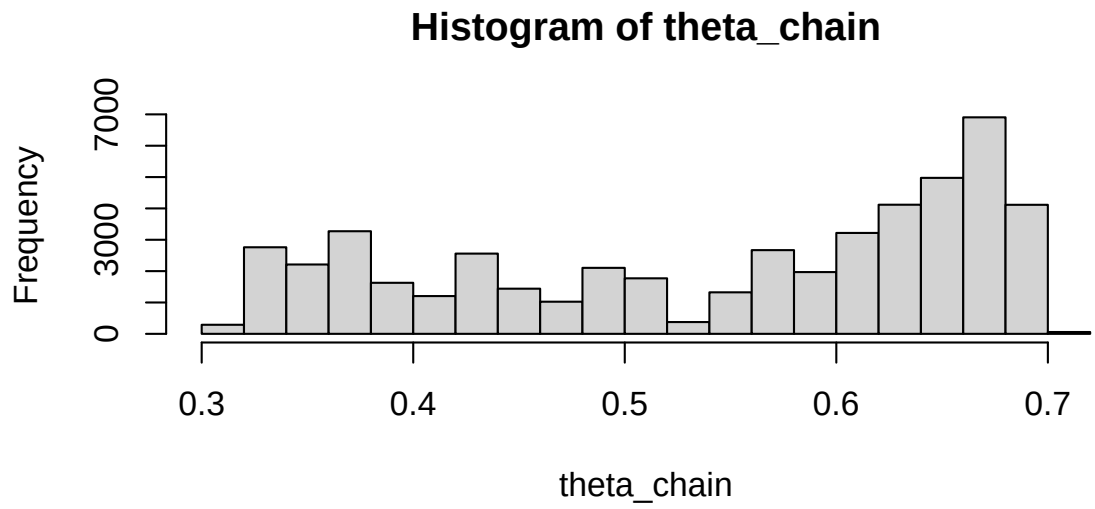

```
plot(theta_chain)
```



- What can go wrong?
  - Step-size is too small

```
# Markov chain
nsamp <- 50000
theta_chain <- rep(NA,nsamp)

#Initialization of Markov chain
```

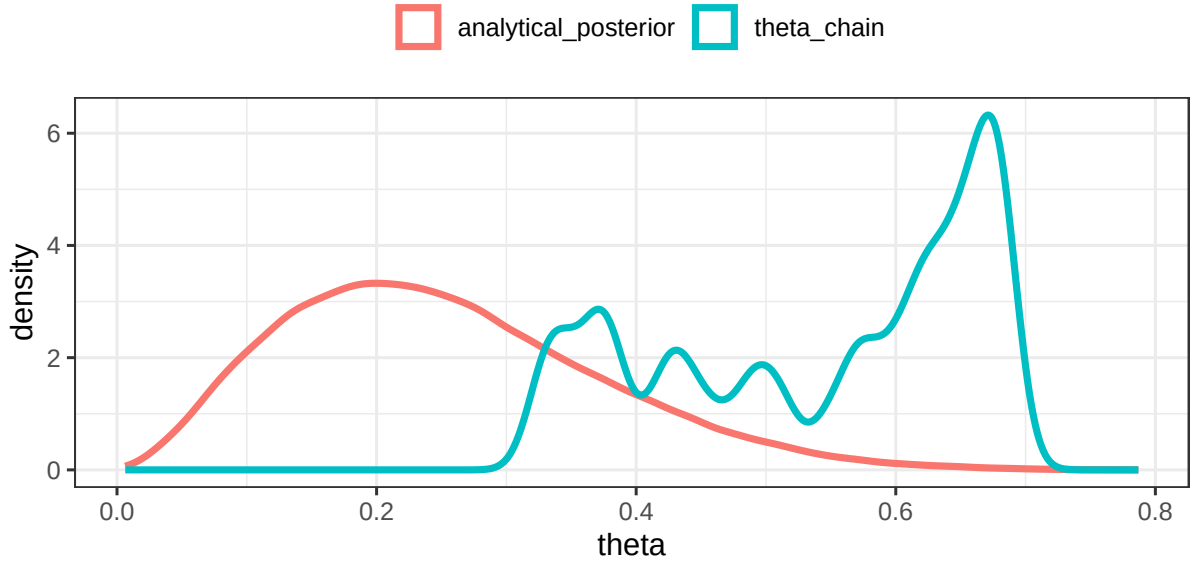
```r
theta_chain[1] <- rbeta(1,1,1)

#Evolution of Markov chain
i <- 1
reject <- 0
step <- 0.0008 # step-size for proposal distribution
while(i<nsamp){
  #Sample from proposal distribution
  proposal_theta <- rnorm(1,theta_chain[i],step)
  if(proposal_theta>0&proposal_theta<1){
    # Compute prior*likelihood
    post_new <- dbinom(k,n,proposal_theta)*dbeta(proposal_theta,a,b)
    post_prev <- dbinom(k,n,theta_chain[i])*dbeta(theta_chain[i],a,b)
    #Compute Hastings ratio
    Hastings_ratio <- (post_new*dnorm(theta_chain[i],proposal_theta,step))/
      (post_prev*dnorm(proposal_theta,theta_chain[i],step))
    p_str <- min(Hastings_ratio,1) # probability of acceptance
    if(p_str>runif(1,0,1)){
      theta_chain[i+1] <- proposal_theta
      i <- i+1
    }else{
      reject <- reject+1
    }
  }
}
hist(theta_chain)
```



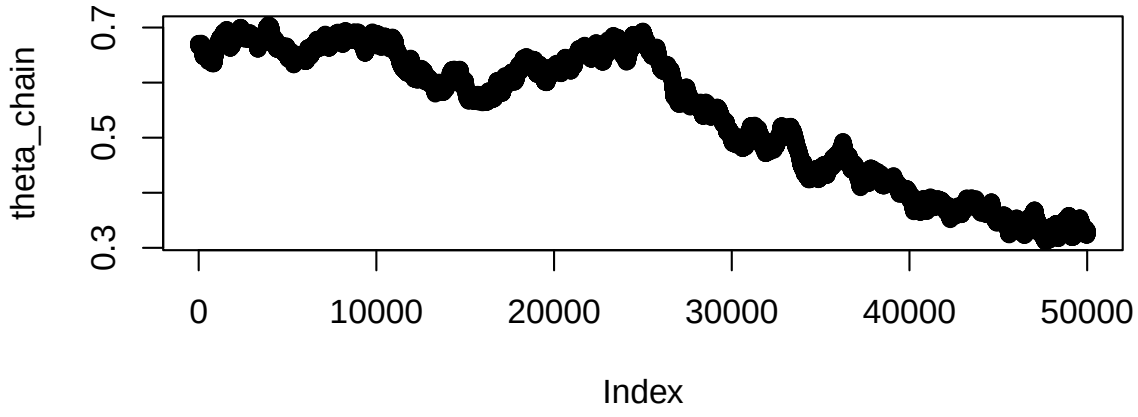

```r
posteriors <- data.frame(analytical_posterior,theta_chain)
```

```
ggplot(melt(posteriors),aes(x=value,colour=variable))+
  geom_density(size=1.2)+theme_bw()+
  xlab("theta")+theme(legend.title = element_blank(),
                      legend.position = "top")
```

```
## No id variables; using all as measure variables
```




```
plot(theta_chain)
```



- Samples are highly correlated - chain varies slowly around the mean
- Chain is stuck - unable to move to high posterior density regions

Let's check the rejection rate,

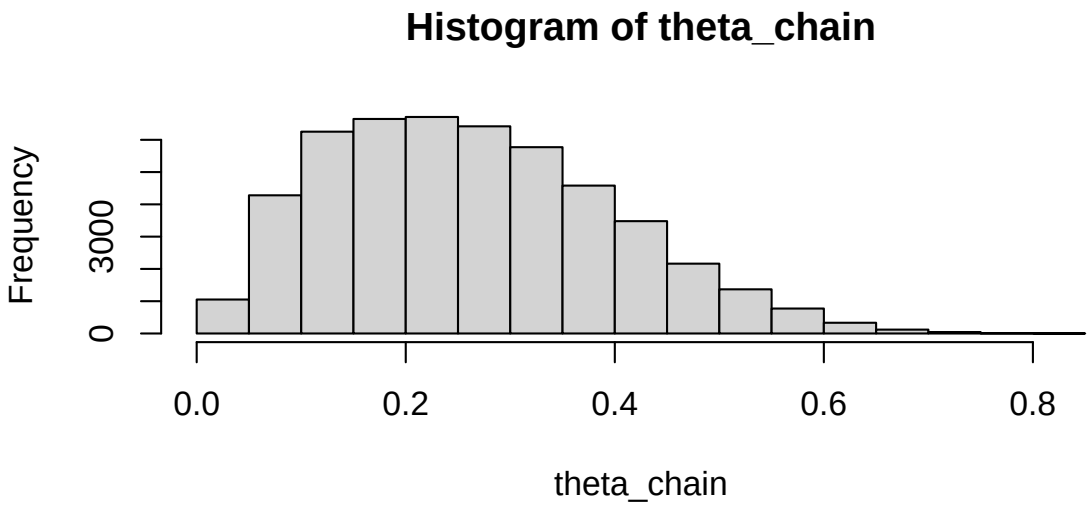
```
reject*100/(reject+nsamp)
```

```
## [1] 0.5113716
```

The rejection rate is too low. We should target a rejection rate of at least 44% (for less than 5 parameters).

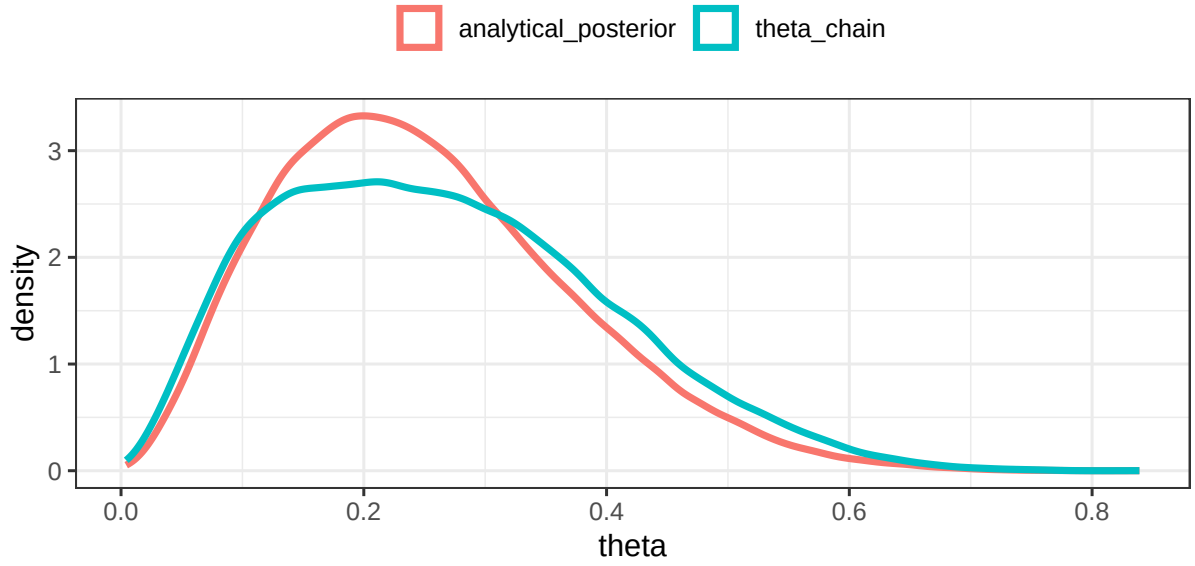
- What else can go wrong?
  - Step-size is too large

```
# Markov chain
nsamp <- 50000
theta_chain <- rep(NA,nsamp)

#Initialization of Markov chain
theta_chain[1] <- rbeta(1,1,1)

#Evolution of Markov chain
i <- 1
reject <- 0
step <- 1 # step-size for proposal distribution
while(i<nsamp){
  #Sample from proposal distribution
  proposal_theta <- rnorm(1,theta_chain[i],step)
  if(proposal_theta>0&proposal_theta<1){
    # Compute prior*likelihood
    post_new <- dbinom(k,n,proposal_theta)*dbeta(proposal_theta,a,b)
    post_prev <- dbinom(k,n,theta_chain[i])*dbeta(theta_chain[i],a,b)
    #Compute Hastings ratio
    Hastings_ratio <- (post_new*dnorm(theta_chain[i],proposal_theta,step))/
      (post_prev*dnorm(proposal_theta,theta_chain[i],step))
    p_str <- min(Hastings_ratio,1) # probability of acceptance
    if(p_str>runif(1,0,1)){
      theta_chain[i+1] <- proposal_theta
      i <- i+1
    }else{
      reject <- reject+1
    }
  }
}
hist(theta_chain)
```

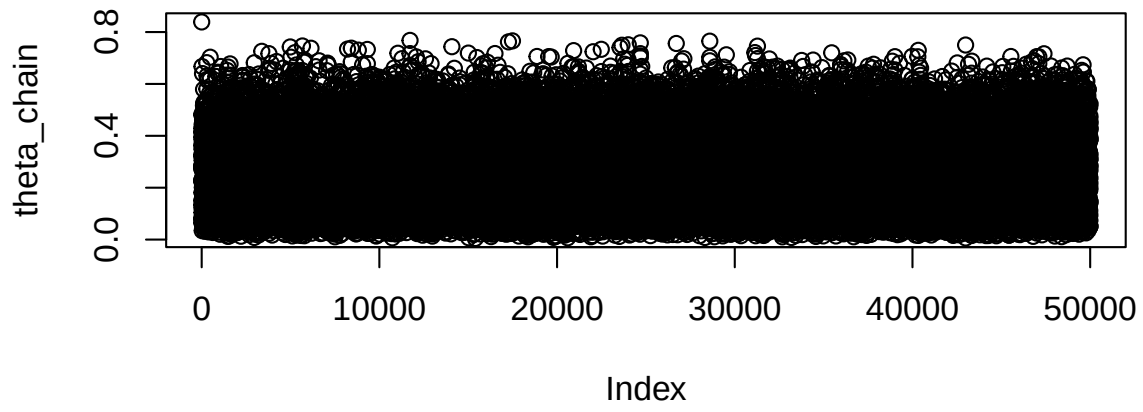



```
posteriors <- data.frame(analytical_posterior,theta_chain)
ggplot(melt(posteriors),aes(x=value,colour=variable))+
  geom_density(size=1.2)+theme_bw()+
  xlab("theta")+theme(legend.title = element_blank(),
                      legend.position = "top")
```

```
## No id variables; using all as measure variables
```




```
plot(theta_chain)
```




```
reject*100/(reject+nsamp)
```

```
## [1] 60.22022
```

- Rejection rate increases to 60%
- Sampler becomes slower
- Inefficiency increase with an increase in the number of parameters
- There are some proposals to resolve this issue.
  - Adaptive MCMC: step-size is not fixed

Can we estimate marginal likelihood using our simple MCMC sampler?

```
# Markov chain
nsamp <- 50000
theta_chain <- rep(NA,nsamp)

#Initialization of Markov chain
theta_chain[1] <- rbeta(1,1,1)

#Evolution of Markov chain
i <- 1
reject <- 0
step <- 0.08 # step-size for proposal distribution
ML <- 0
while(i<nsamp){
  #Sample from proposal distribution
  proposal_theta <- rnorm(1,theta_chain[i],step)
  if(proposal_theta>0&proposal_theta<1){
```

```
    # Compute prior*likelihood
    post_new <- dbinom(k,n,proposal_theta)*dbeta(proposal_theta,a,b)
    post_prev <- dbinom(k,n,theta_chain[i])*dbeta(theta_chain[i],a,b)
    #Compute Hastings ratio
    Hastings_ratio <- (post_new*dnorm(theta_chain[i],proposal_theta,step))/
      (post_prev*dnorm(proposal_theta,theta_chain[i],step))
    p_str <- min(Hastings_ratio,1) # probability of acceptance
    if(p_str>runif(1,0,1)){
      theta_chain[i+1] <- proposal_theta
      lkl <- dbinom(k,n,proposal_theta)*dbeta(proposal_theta,a,b)/
        dnorm(proposal_theta,theta_chain[i],step)
      ML <- ML+lkl
      i <- i+1
    }else{
      reject <- reject+1
    }
  }
}
estimated_ML <- ML/nsamp
estimated_ML
```

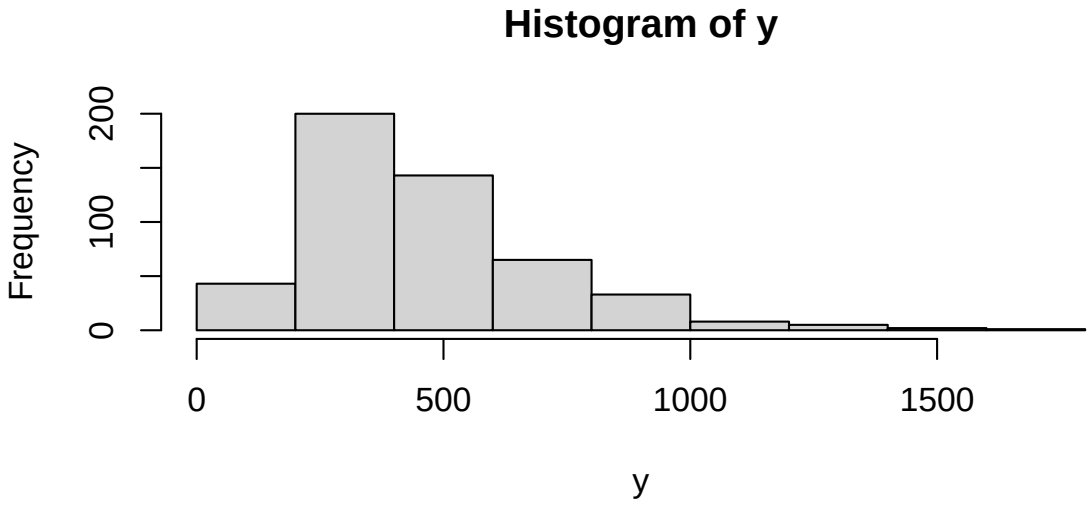
```
## [1] 0.09208308
```

```
analytical_ML <- 1/11
analytical_ML
```

```
## [1] 0.09090909
```

Example. A lognormal model with non-conjugate priors

You are given reading time data consisting of 500 observations,



You assume that the data come from a lognormal distribution with mean μ and variance σ.

Priors:

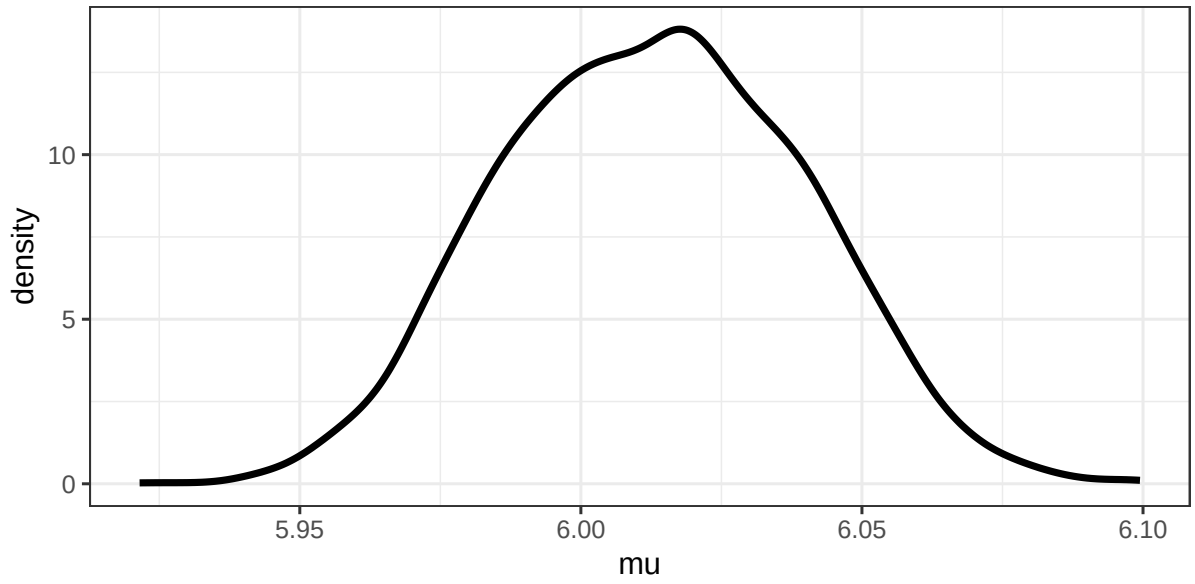
$\mu \sim Normal(10, 6)$

$\sigma \sim Normal_+(0, 2)$

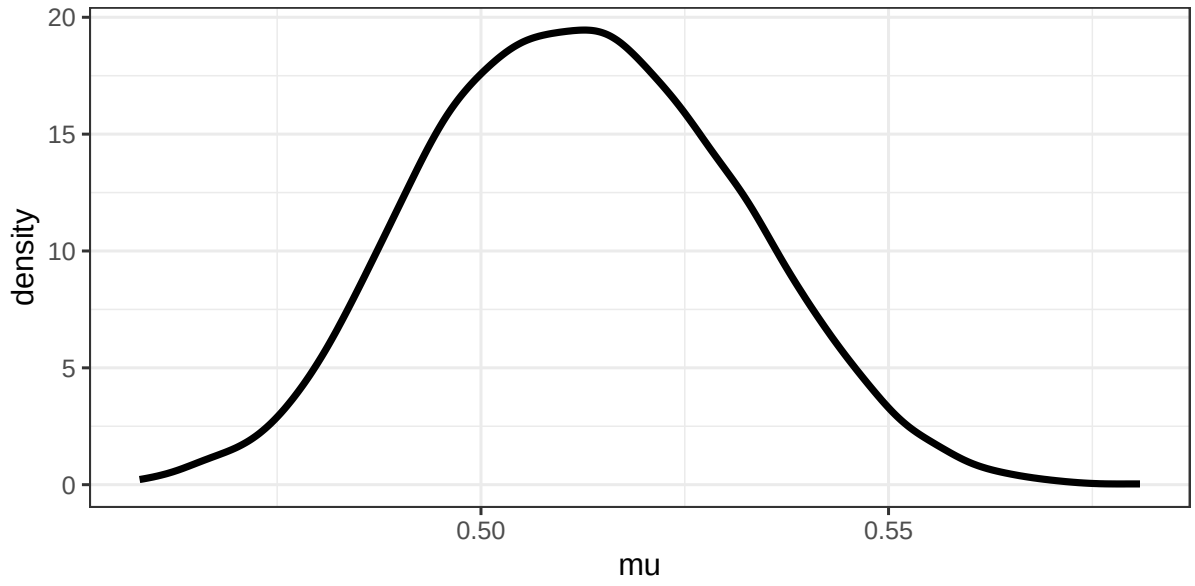
Let us estimate the posterior distributions of μ and σ using our simple Metropolis-Hastings sampler.

```
# Markov chain
nsamp <- 6000
mu_chain <- rep(NA,nsamp)
sigma_chain <- rep(NA,nsamp)

#Initialization of Markov chain
mu_chain[1] <- rnorm(1,10,6)
sigma_chain[1] <- rtruncnorm(n=1,mean=0,sd=2,a=0)

#Evolution of Markov chain
i <- 1
reject <- 0
step <- 0.1 # step-size for proposal distribution
while(i<nsamp){
  #Sample from proposal distribution
  proposal_mu <- rnorm(1,mu_chain[i],step)
  proposal_sigma <- rtruncnorm(n=1,mean=sigma_chain[i],sd=step,a=0)
  # Compute prior*likelihood
  post_new <- sum(dlnorm(y,proposal_mu,proposal_sigma,log = TRUE))+
    dnorm(proposal_mu,10,6,log = TRUE)+
    log(dtruncnorm(x=proposal_sigma,mean=0,sd=2,a=0,b=Inf))
  post_prev <- sum(dlnorm(y,mu_chain[i],sigma_chain[i],log = TRUE))+
    dnorm(mu_chain[i],10,6,log = TRUE)+
    log(dtruncnorm(x=sigma_chain[i],mean=0,sd=2,a=0,b=Inf))
  #Compute Hastings ratio
  Hastings_ratio <-
    exp((post_new+dnorm(mu_chain[i],proposal_mu,step,log=TRUE)+
       log(dtruncnorm(x=sigma_chain[i],mean=proposal_sigma,sd=step,a=0)))-
    (post_prev+dnorm(proposal_mu,mu_chain[i],step,log=TRUE)+
       log(dtruncnorm(x=proposal_sigma,mean=sigma_chain[i],sd=step,a=0))))
  p_str <- min(Hastings_ratio,1) # probability of acceptance
  if(p_str>runif(1,0,1)){
    mu_chain[i+1] <- proposal_mu
    sigma_chain[i+1] <- proposal_sigma
    i <- i+1
  }else{
    reject <- reject+1
```

```
  }
}

posteriors <- data.frame(mu_chain,sigma_chain)
ggplot(posteriors[-(1:2000),],aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")
```



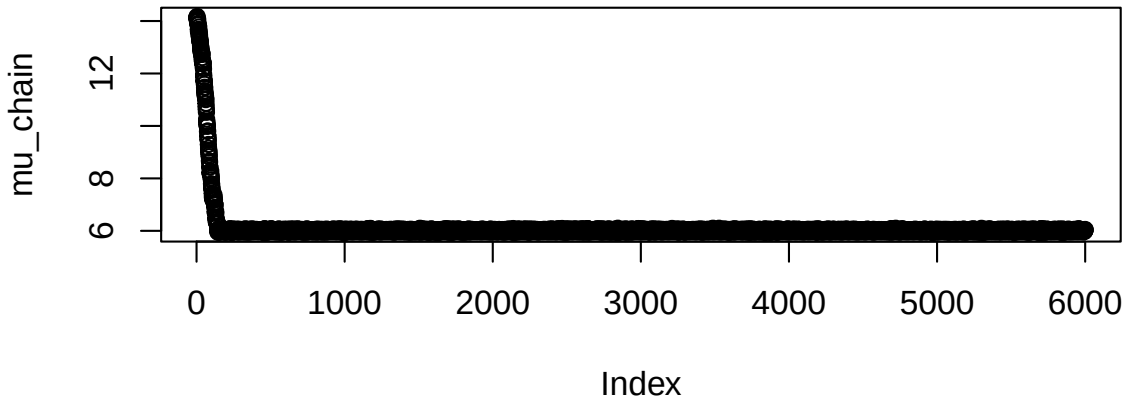

```
ggplot(posteriors[-(1:2000),],aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")
```



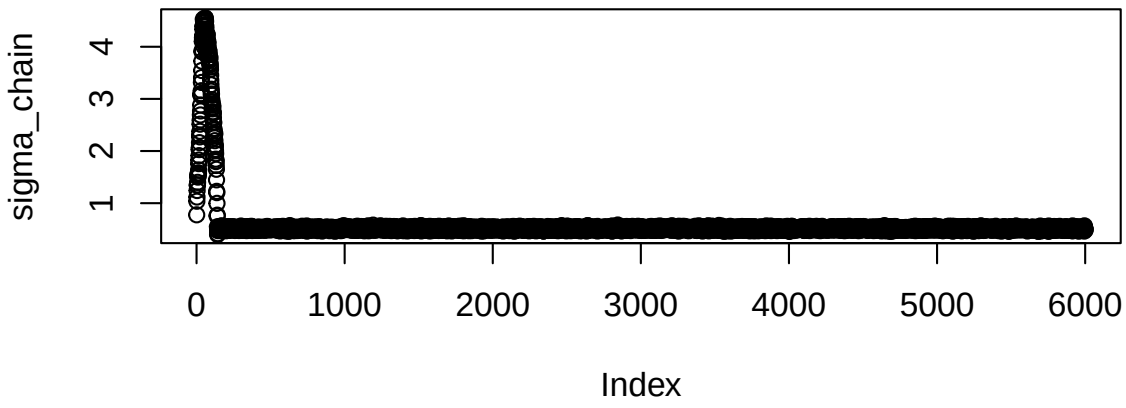

```
#Rejection rate
reject*100/(reject+nsamp)
```

```
## [1] 93.49939
```

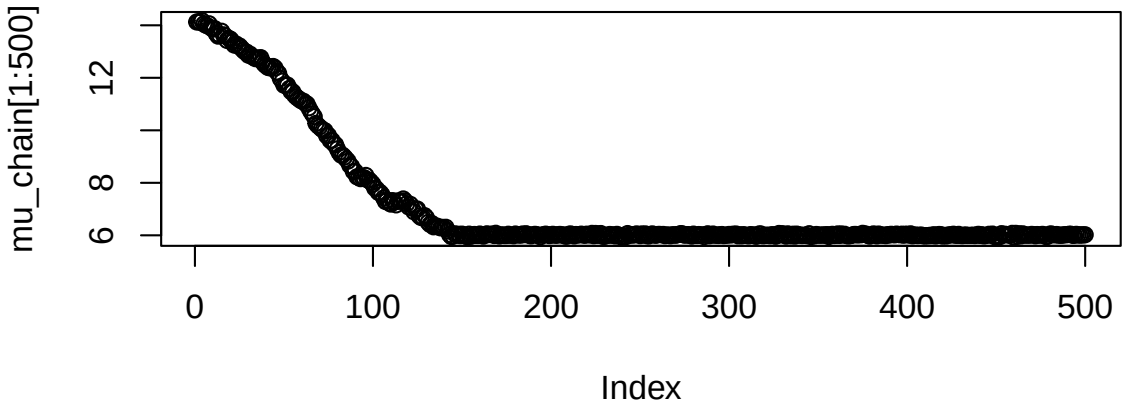
```
#Chain inspection
plot(mu_chain)
```



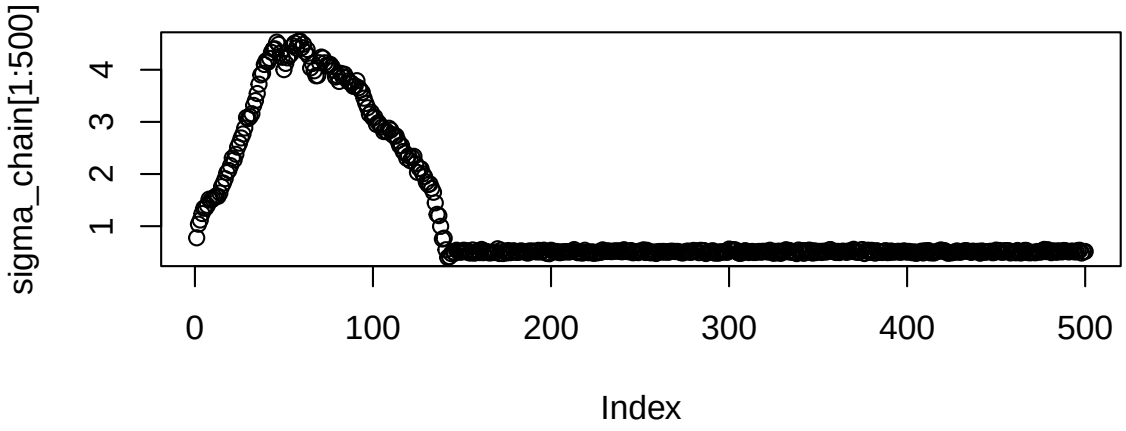

```
plot(sigma_chain)
```




```
#Zooming in
plot(mu_chain[1:500])
```



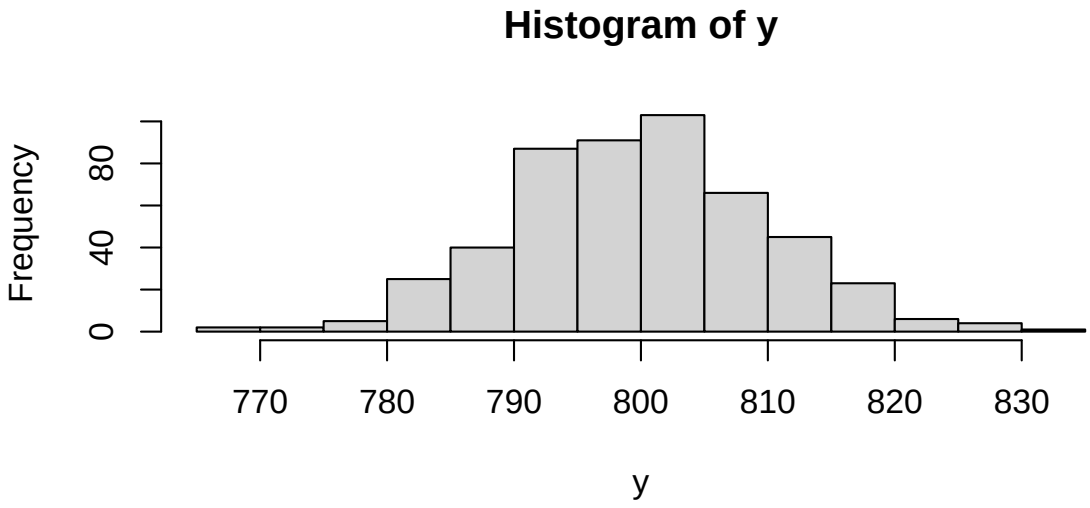

```
plot(sigma_chain[1:500])
```



7 *Recap: Metropolis algorithms*

Example. A Normal model

You are given 500 data points that are assumed to come from a normal distribution with mean μ and variance σ^2,

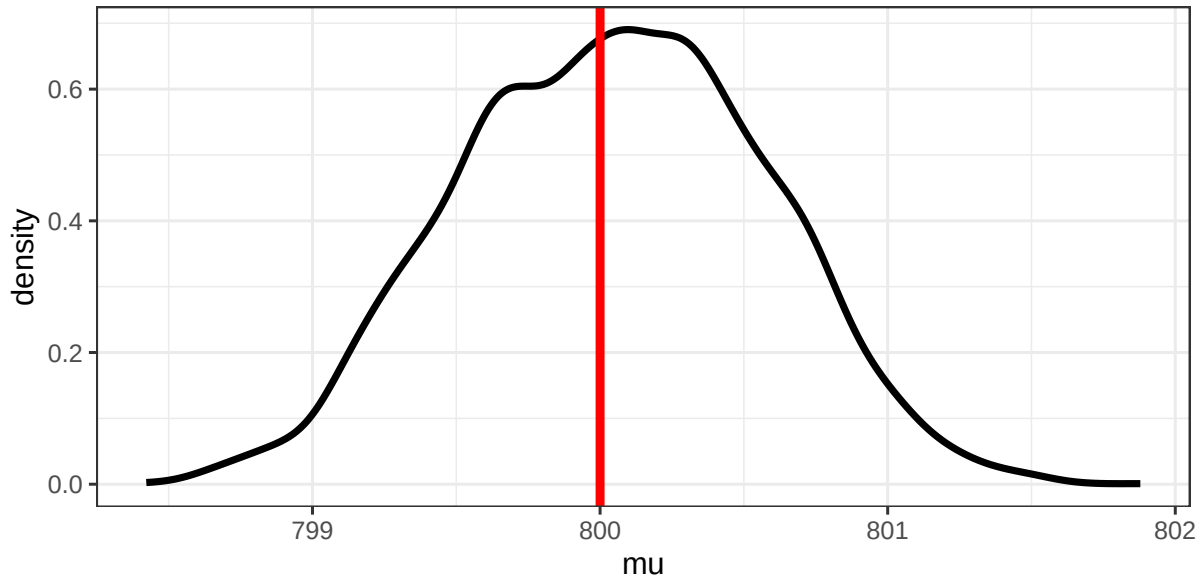


Let y_i be i^{th} data point,
$y_i \sim Normal(\mu, \sigma^2)$
$\mu \sim Normal(\mu_0 = 1000, \sigma_0^2 = 10000)$
$\sigma^2 \sim InverseGamma(\alpha = 21, \beta = 2000)$
Let us estimate the posterior distributions of μ and σ using our simple Metropolis-Hastings sam-

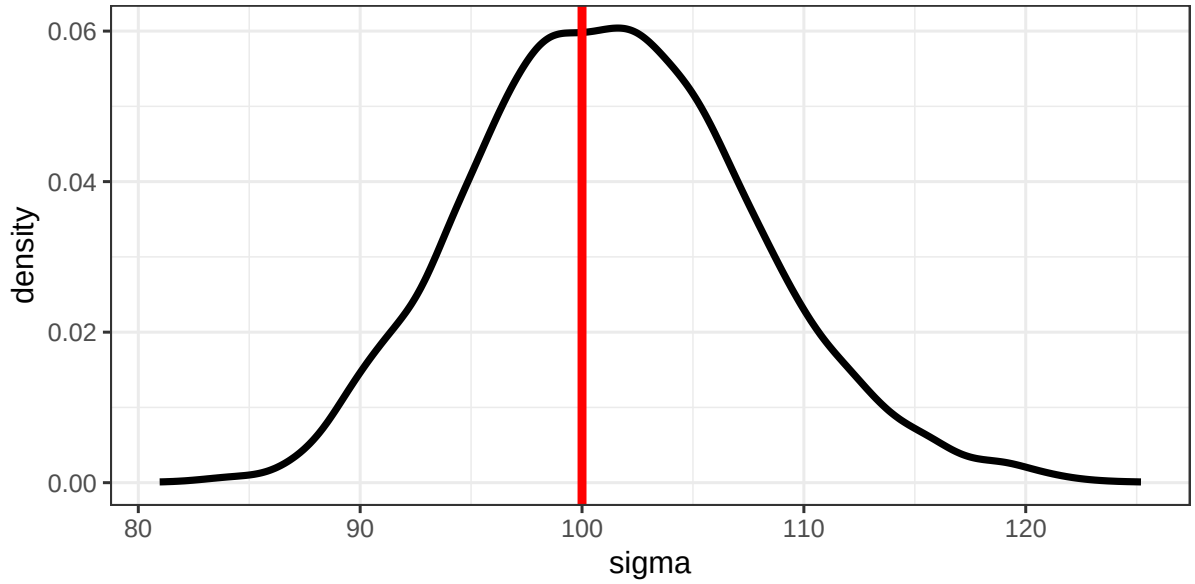
pler.

```
# Markov chain
nsamp <- 8000
mu_chain <- rep(NA,nsamp)
sigma_chain <- rep(NA,nsamp)

#Initialization of Markov chain
mu_chain[1] <- rnorm(1,1000,100)
sigma_chain[1] <- rinvgamma(1,21,2000)

#Evolution of Markov chain
i <- 1
reject <- 0
step <- 4 # step-size for proposal distribution

while(i<nsamp){
  #Sample from proposal distribution
  proposal_mu <- rnorm(1,mu_chain[i],step)
  proposal_sigma <- rtruncnorm(n=1,mean=sigma_chain[i],sd=step,a=0)
  # Compute prior*likelihood
  post_new <- sum(dnorm(y,proposal_mu,sqrt(proposal_sigma),log = TRUE))+
    dnorm(proposal_mu,1000,100,log = TRUE)+
    dinvgamma(proposal_sigma,21,2000,log=TRUE)
  post_prev <- sum(dnorm(y,mu_chain[i],sqrt(sigma_chain[i]),log = TRUE))+
    dnorm(mu_chain[i],1000,100,log = TRUE)+
    dinvgamma(sigma_chain[i],21,2000,log=TRUE)
  #Compute Hastings ratio
  Hastings_ratio <-
    exp((post_new+dnorm(mu_chain[i],proposal_mu,step,log=TRUE)+
       log(dtruncnorm(x=sigma_chain[i],mean=proposal_sigma,sd=step,a=0)))-
    (post_prev+dnorm(proposal_mu,mu_chain[i],step,log=TRUE)+
       log(dtruncnorm(x=proposal_sigma,mean=sigma_chain[i],sd=step,a=0))))
  p_str <- min(Hastings_ratio,1) # probability of acceptance
  if(p_str>runif(1,0,1)){
    mu_chain[i+1] <- proposal_mu
    sigma_chain[i+1] <- proposal_sigma
    i <- i+1
  }else{
    reject <- reject+1
  }
}

posteriors <- data.frame(mu_chain,sigma_chain)
```

```
ggplot(posteriors[-(1:2000),],aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept=800,size=1.5,color="red")
```




```
ggplot(posteriors[-(1:2000),],aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("sigma")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept = 100,size=1.5,color="red")
```




```
metropolis_posterior <- posteriors

#Rejection rate
reject*100/(reject+nsamp)
```

```
## [1] 87.30219
```

Key features

- Collection of samples from one iteration to the other is a **Markov process**.
- **Rejection sampling** based on relative posterior density
  - Tends to reject large proportion of proposed samples
- **Random walk** in parameter space

Possible improvements

- Can we avoid rejection sampling?
  - Gibbs sampling
- Can we make random walk less random?
  - Hamiltonaian Monte Carlo

8 Gibbs sampling

8.1 Idea

Suppose the data y come from a model having two parameters θ_1 and θ_2,

$y \sim f(\theta_1, \theta_2)$

Our goal is to estimate the posterior distributions for θ_1 and θ_2. We can write the joint posterior for θ_1 and θ_2 using Bayes' rule,

$p(\theta_1, \theta_2|y) = \frac{p(y|\theta_1,\theta_2)p(\theta_1)p(\theta_2)}{p(y)}$

We cannot analytically derive the posterior distribution $p(\theta_1, \theta_2|y)$ because we cannot solve the denominator $p(y)$.

But in some cases, we can derive the **conditional posterior** distributions

$p(\theta_1|\theta_2, y)$ and $p(\theta_2|\theta_1, y)$

- $p(\theta_1|\theta_2, y)$ means posterior distribution of θ_1 when θ_2 and y are known.
- $p(\theta_2|\theta_1, y)$ means posterior distribution of θ_2 when θ_1 and y are known.

The idea is to sample from the conditional posteriors of θ_1 and θ_2 using Markov process.

8.2 Method

- In each iteration, sample from the conditional posterior of θ_i such that the known values of parameters θ_{-i} come from the current state of the Markov chain
- Do not reject any samples
- In our example, to sample from the posterior distributions of θ_1 and θ_2,
  - Initiate chains for parameters θ_1 and θ_2
  - In each iteration,
    - Draw a sample from $p(\theta_1|\theta_2, y)$ where θ_2 = current state of θ_2 chain
    - Draw a sample from $p(\theta_2|\theta_1, y)$ where θ_1 = current state of θ_1 chain

Steps:

- Set $\theta_1 = u_0$ and $\theta_2 = v_0$
  - Current state: (u_0, v_0)
- Sample $u_1 \sim p(\theta_1|\theta_2 = v_0, y)$
  - Current state: (u_1, v_0)
- Sample $v_1 \sim p(\theta_2|\theta_1 = u_1, y)$
  - Current state: (u_1, v_1)
- Sample $u_2 \sim p(\theta_1|\theta_2 = v_1, y)$

  - Current state: (u_2, v_1)

- Sample $v_2 \sim p(\theta_2|\theta_1 = u_2, y)$

  - Current state: (u_2, v_2)

- Repeat this process until you have collected enough samples

- $(u_0, v_0), (u_1, v_1), (u_2, v_2)$... satisfies the property of being a Markov chain.

8.3 Implementation

Example. A Normal model with semi-conjugate priors

You are given 500 data points, $y_1, y_2, ..., y_{500}$ that are assumed to come from a normal distribution with mean μ and variance σ^2,

Let y_i be i^{th} data point,

$y_i \sim Normal(\mu, \sigma^2)$

$\mu \sim Normal(\mu_0, \sigma_0^2)$

$\sigma^2 \sim InverseGamma(\alpha, \beta)$

We can derive conditional posterior distributions for μ and σ^2,

$\mu|\sigma^2, y \sim Normal(\frac{n\sigma_0^2\bar{y}+\sigma^2\mu_0}{n\sigma_0^2+\sigma^2}, \frac{\sigma^2\sigma_0^2}{n\sigma_0^2+\sigma^2})$

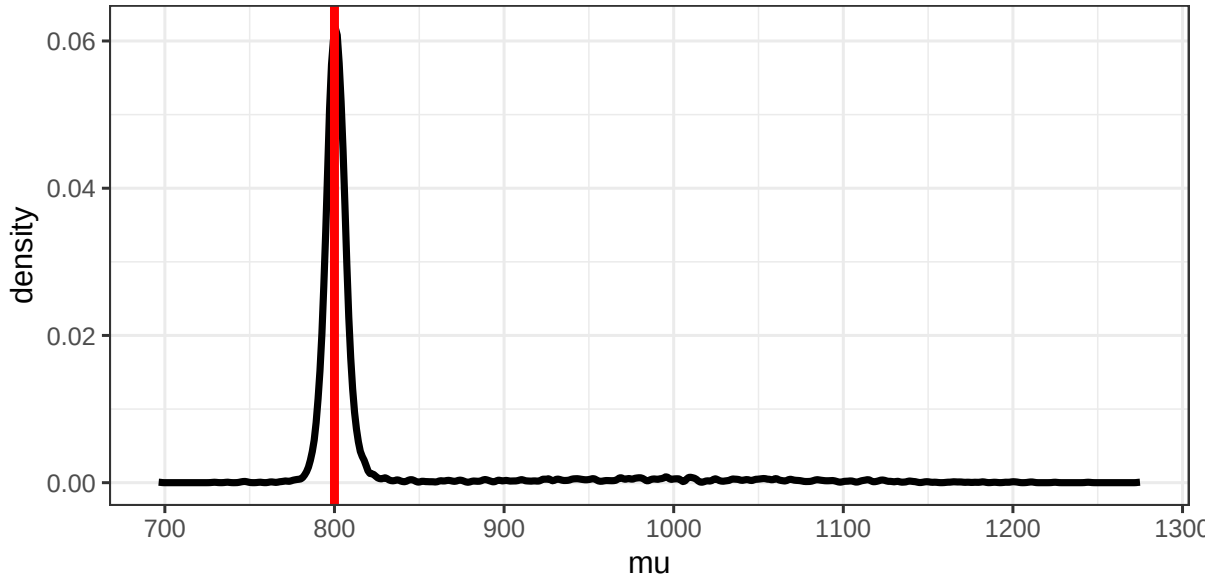
$\sigma^2|\mu, y \sim InverseGamma(\frac{n}{2}+\alpha, \frac{n}{2}((\mu-\bar{y})^2 + \sum_{i=1}^{n}(y_i-\bar{y})^2/n) + \beta$

Where n is the total number of data points.

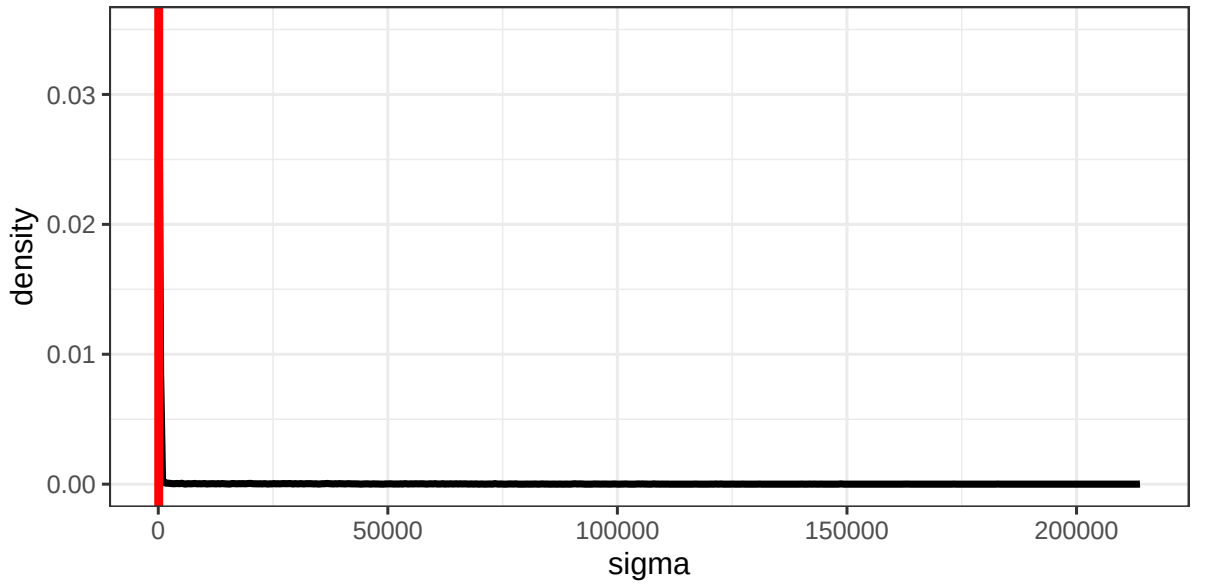
Let us estimate the posterior distributions of μ and σ using a Gibbs sampler.

```
#Priors
# mu ~ Normal(m,s)
m <- 1000
s <- 100
# sigma^2 ~ InverseGamma(a,b)
a <- 21
b <- 2000
# Data
y <- y
n <- length(y)

# Function for drawing sample from conditional posteriors

mu_sample <- function(sigma,y,n,m,s,a,b){
  mu_p <- ((n*(s^2)*mean(y))+((sigma^2)*m))/((n*(s^2))+(sigma^2))
  sigma_p <- sqrt(((sigma^2)*(s^2))/((n*(s^2))+(sigma^2)))
  mu.samp <- rnorm(1,mean=mu_p,sd=sigma_p)
  mu.samp
}

sigma_sample <- function(mu,y,n,m,s,a,b){
```

```
  a_p <- (n/2)+a
  b_p <- ((n/2)*(((mu-mean(y))^2)+var(y)))+b
  sigma.sq <- rinvgamma(1,a_p,b_p)
  sigma.sq
}

# Gibbs sampler

# Markov chain
nsamp <- 10000
mu_chain <- rep(NA,nsamp)
sigma_chain <- rep(NA,nsamp)

#Initialization of Markov chain
mu_chain[1] <- rnorm(1,1000,100)
sigma_chain[1] <- rinvgamma(1,21,2000)

#Evolution of Markov chain
i <- 1
while(i<nsamp){
  #Sample from conditional posterior of mu
  proposal_mu <- mu_sample(sigma=sigma_chain[i],y,n,m,s,a,b)
  #Update the chains
  mu_chain[i+1] <- proposal_mu
  #Sample from conditional posterior of sigma
  proposal_sigma <- sigma_sample(mu=mu_chain[i+1],y,n,m,s,a,b)
  #Update the chains
  sigma_chain[i+1] <- proposal_sigma
  i <- i+1
}

posteriors <- data.frame(mu_chain,sigma_chain)
ggplot(posteriors[-(1:5000),],aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept=800,size=1.5,color="red")
```



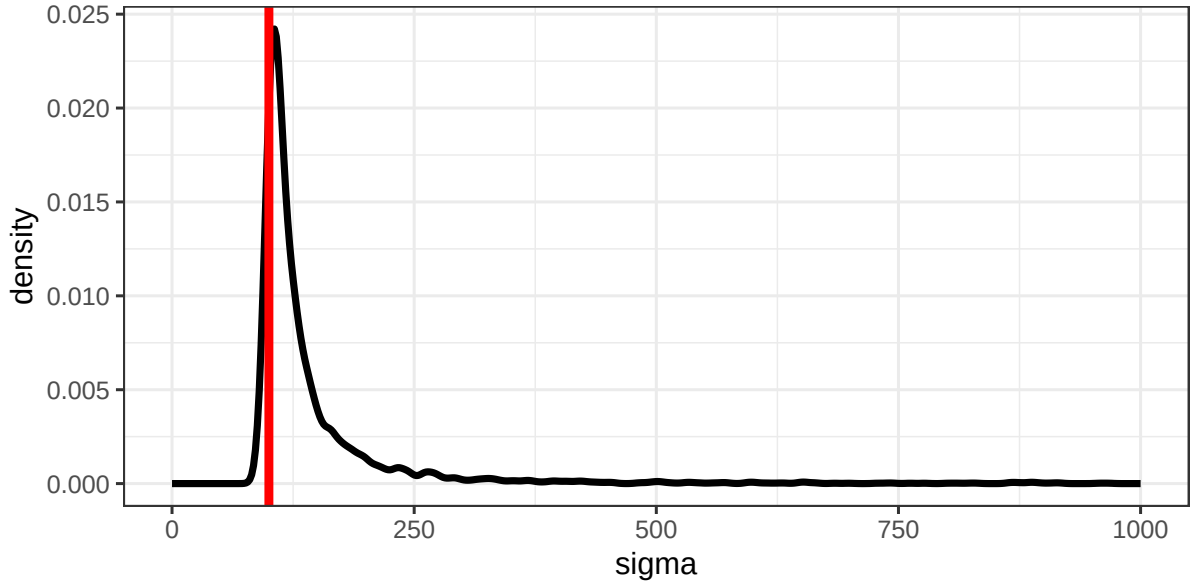

```
ggplot(posteriors[-(1:5000),],aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("sigma")+
  theme(legend.title = element_blank(),
                       legend.position = "top")+
  geom_vline(xintercept = 100,size=1.5,color="red")
```




```
ggplot(posteriors[-(1:5000),],aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("sigma")+
```

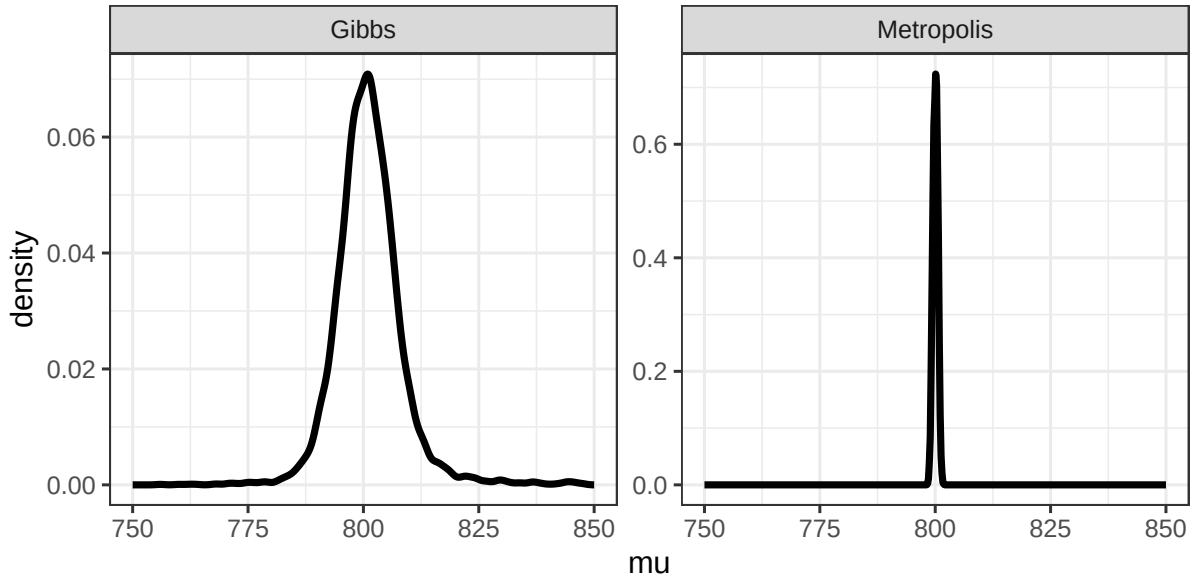
```
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept = 100,size=1.5,color="red")+
  scale_x_continuous(limits = c(0,1000))
```

```
## Warning: Removed 536 rows containing non-finite values (‘stat_density()‘).
```



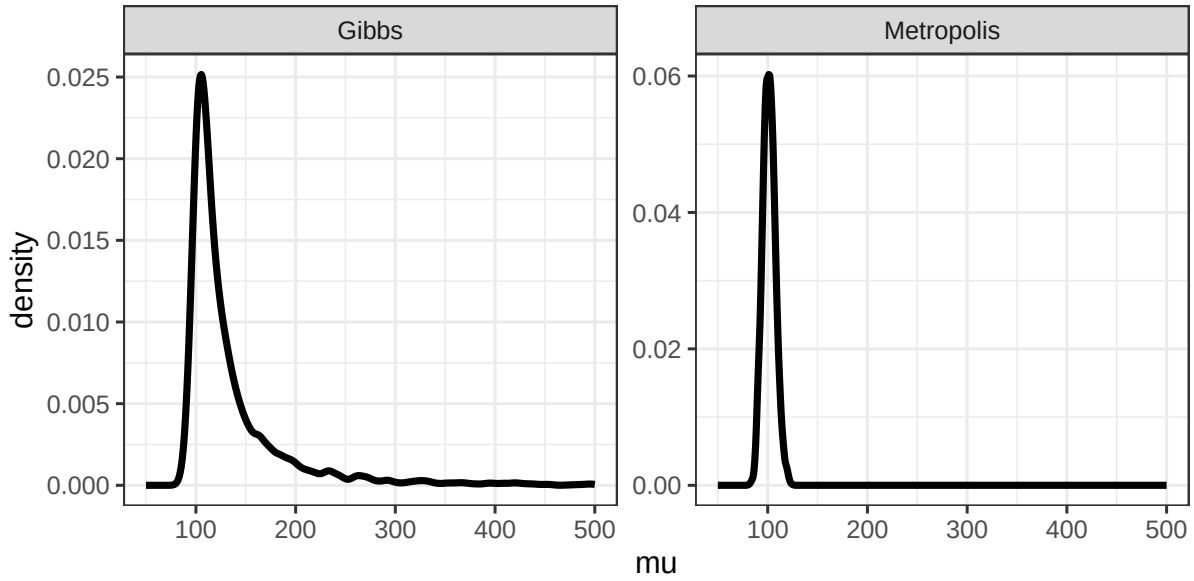

```
gibbs_posterior <- posteriors
gibbs_posterior$algorithm <- "Gibbs"
metropolis_posterior$algorithm <- "Metropolis"
gibbs_metro <- rbind(gibbs_posterior[-(1:4000),],metropolis_posterior[-(1:2000),])

ggplot(gibbs_metro,aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  facet_wrap(~algorithm,scales = "free")+
  scale_x_continuous(limits = c(750,850))
```

```
## Warning: Removed 563 rows containing non-finite values (‘stat_density()‘).
```




```
ggplot(gibbs_metro,aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                       legend.position = "top")+
  facet_wrap(~algorithm,scales = "free")+
  scale_x_continuous(limits = c(50,500))
```

```
## Warning: Removed 674 rows containing non-finite values (‘stat_density()‘).
```



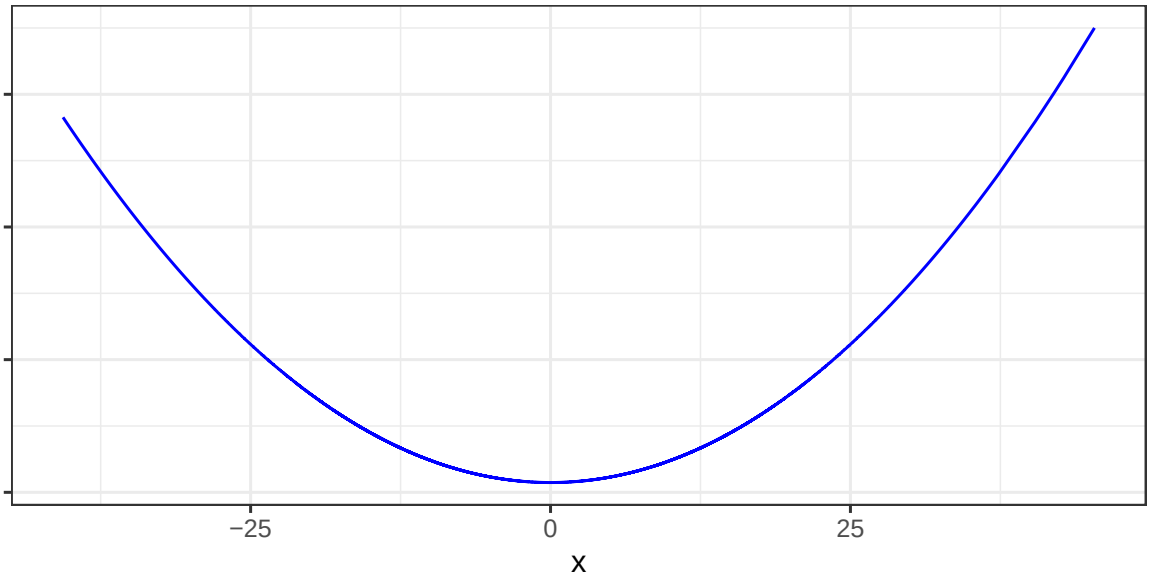
8.4 *Limitations*

- We cannot derive conditional posterior densities in most cases
- Restricts the choice of prior distributions

9 *Hamiltonian Monte Carlo (HMC)*

9.1 *Idea*

- Exploration of parameter space is informed by the geometry of the posterior distribution
- MCMC using Hamiltonian dynamics
- Differs from Metropolis algorithms in 'how a new proposal is generated'
  - Random walk: randomly jump from the current location to a new location in the parameter space
  - Hamiltonian slide: slide along the posterior density space from the current location to a new location
    * Metropolis algorithms do not use posterior density in proposal generation, Hamiltonian Monte Carlo does
    * How to achieve this?
    * Hamiltonian dynamics

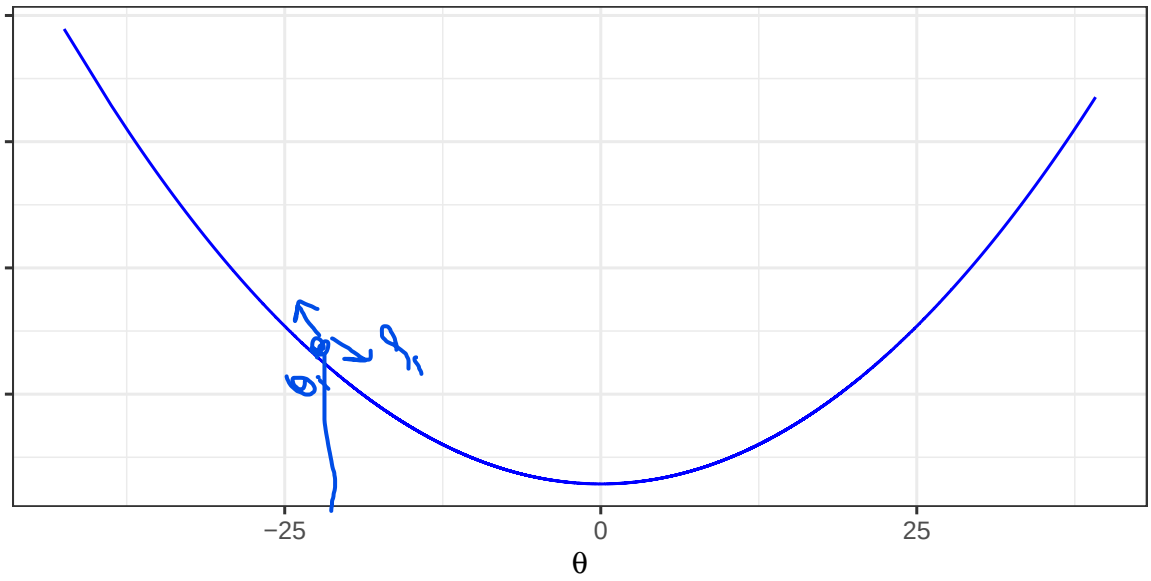


9.2 *Hamiltonian dynamics*

- A frictionless particle of mass m sliding over a frictionless surface

- Total energy of the system at a point x is the sum of the potential and the kinetic energy of the particle at that point,
  - Say, H is the total energy, V is the potential energy and T is the kinetic energy
  - $H(x, \rho) = V(x) + T(\rho)$
    - $T = \rho^2/2m$ where ρ is the momentum
  - The potential energy of the particle $V(x)$ is a function of position x
    - Potential energy increases with increase in height
  - If you leave the particle at rest, the particle will slide down such that its potential energy will decrease and kinetic energy will increase
- How position x and momentum ρ change over time t?
  - $\frac{\delta x}{\delta t} = \frac{\delta H}{\delta \rho} = \frac{\delta T(\rho)}{\delta \rho}$
  - $\frac{\delta \rho}{\delta t} = -\frac{\delta H}{\delta x} = -\frac{\delta V(x)}{\delta x}$
  - These are called Hamiltonian's equations.
  - Given initial position and initial momentum of the particle at time t_0, we can determine its position and momentum at time $t_0 + T$

9.3 *Using Hamiltonian dynamics for MCMC*



- A particle of mass m moves in *negative log posterior density* space. The potential energy function, $V(\theta)$, is negative log of (unnormalized) posterior density function
  - $V(\theta) = -\log p(y|\theta)p(\theta)$

- In each iteration, the particle is pushed from its current position θ_i in random direction with some momentum ρ_i.
  - After time T has elapsed, the particle acquires new position θ_j and new momentum ρ_j
  - New position and new momentum can be calculated using Hamiltonian's equations
  - In practice, Hamiltonian's equations must be numerically approximated by discretizing time. This is done by splitting the time interval T into small intervals of size ϵ.
    - ϵ is called step-size parameter
    - Given initial momentum ρ, momentum after ϵ time will be $\rho + \delta\rho$.
    - Since ϵ is very small, we can write $\delta\rho$ as $\frac{\delta\rho}{\delta t}\epsilon$
    - Thus momentum after ϵ time can be written as $\rho + \frac{\delta\rho}{\delta t}\epsilon = \rho - \frac{\delta V(\theta)}{\delta\theta}\epsilon$
    - Let us call $\frac{\delta V(\theta)}{\delta\theta}$ gradient for parameter θ
  - Leapfrog method to discretize Hamiltonian's equations
    - $\rho \leftarrow \rho - \frac{\epsilon}{2}\frac{\delta V(\theta)}{\delta\theta}$ (Half step update for momentum)
    - $\theta \leftarrow \theta + \epsilon\frac{\delta T(\rho)}{\delta\rho}$ (Full step update for position)
    - $\rho \leftarrow \rho - \frac{\epsilon}{2}\frac{\delta V(\theta)}{\delta\theta}$ (Half step update for momentum)
    - These three steps are repeated L times such that $T = L\epsilon$
- The total energy of the particle is recorded in the positions θ_i and θ_j
  - Total energy in position θ_i, $H(\theta_i, \rho_i) = V(\theta_i) + T(\rho_i)$
  - Total energy in position θ_j, $H(\theta_j, \rho_j) = V(\theta_j) + T(\rho_j)$
    - $V(\theta) = -\log(p(y|\theta)p(\theta))$
    - $T(\rho) = \rho^2/2m$
- The system wants to remain in a lower energy state
  - Boltzmann distribution: the system occupies state i with probability p_i, $p_i \propto e^{-\frac{H_i}{kT}}$
    - H_i is the energy of the system in state i, T is the temprature of the system
  - The probability of transitioning from position i to position j, $p(i \to j) = e^{\frac{H_i - H_j}{kT}}$
- Accept the new position θ_j with probability $min(1, exp(H(\theta_i, \rho_i) - H(\theta_j, \rho_j)))$

9.4 *Step-by-step procedure*

- Define potential energy function, i.e., negative log of posterior density function
  - $V(\theta) = -log(p(y|\theta)p(\theta))$
- Define gradient function, $\frac{\delta V}{\delta\theta}$
- Choose a distribution for proposing a new momentum in each iteration
  - E.g., $\rho \sim MultivariateNormal(0, M)$

- Choose internal parameters: number of samples, number of leapfrog steps, step size
- In each iteration:
  - Generate a random momemntum value, $\rho \sim multiNormal(0, M)$
  - Record current position and current momentum
  - Calcualte total energy of the system at the current position
  - Take L leapfrog steps each in time ϵ
  - Record new position and new momentum after time $L\epsilon$
  - Calculate total energy of the system at the new position
  - Accept the new position with probability $min(1, exp(H_{new} - H_{current})$

9.5 Implementation

Example. A Normal model with normal priors on μ and σ

You are given 500 data points, $y_1, y_2, ..., y_{500}$ that are assumed to come from a normal distribution with mean μ and variance σ^2,

Let y_i be i^{th} data point,

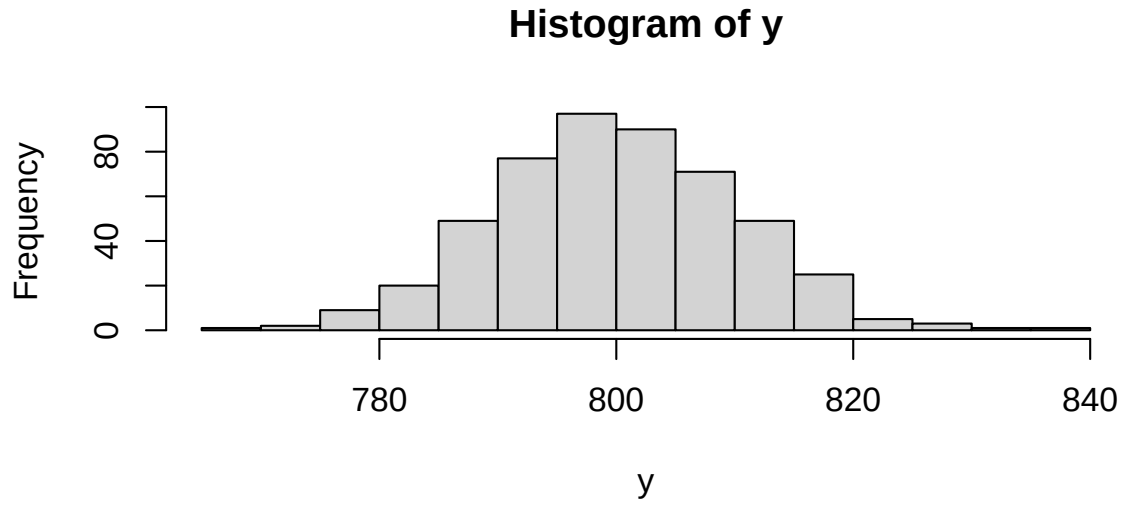
Model:

$$y_i \sim Normal(\mu, \sigma)$$

$$\mu \sim Normal(m, s)$$

$$\sigma \sim Normal(a, b)$$

Data



Step 1: Define gradient functions

We have to derive $\frac{\delta V}{\delta \mu}$ and $\frac{\delta V}{\delta \sigma}$ where V is the negative log of (un-normalized) posterior density.

$V = -\log(p(y|\mu,\sigma)p(\mu)p(\sigma))$

$V = -\log p(y|\mu,\sigma) - \log p(\mu|m,s) - \log p(\sigma|a,b)$

$V = -\log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{y-\mu}{\sigma})^2} - \log \frac{1}{s\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mu-m}{s})^2} - \log \frac{1}{b\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\sigma-a}{b})^2}$

Since y is a vector of n data points,

$V = (\sum_{i=1}^{n} - \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{y_i-\mu}{\sigma})^2}) - \log \frac{1}{s\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mu-m}{s})^2} - \log \frac{1}{b\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\sigma-a}{b})^2}$

$V = n\log\sigma + \sum_{i=1}^{n} \frac{1}{2}(\frac{y_i-\mu}{\sigma})^2 + (n+2)\log\sqrt{2\pi} + \log sb + \frac{1}{2}(\frac{\mu-m}{s})^2 + \frac{1}{2}(\frac{\sigma-a}{b})^2$

$\frac{\delta V}{\delta \mu} = \frac{n\mu - \sum_{i=1}^{n} y_i}{\sigma^2} + \frac{\mu-m}{s^2}$

$\frac{\delta V}{\delta \sigma} = \frac{n}{\sigma} - \frac{\sum_{i=1}^{n}(y_i-\mu)^2}{\sigma^3} + \frac{\sigma-a}{b^2}$

```
#Gradient functions
gradient <- function(mu,sigma,y,n,m,s,a,b){
  grad_mu <- (((n*mu)-sum(y))/(sigma^2))+((mu-m)/(s^2))
  grad_sigma <- (n/sigma)-(sum((y-mu)^2)/(sigma^3))+((sigma-a)/(b^2))
  return(c(grad_mu,grad_sigma))
}
```

Step 2: Define potential energy function

$V = -\log(p(y|\mu,\sigma)p(\mu)p(\sigma))$

$V = -(\log p(y|\mu,\sigma) + \log p(\mu) + \log p(\sigma))$

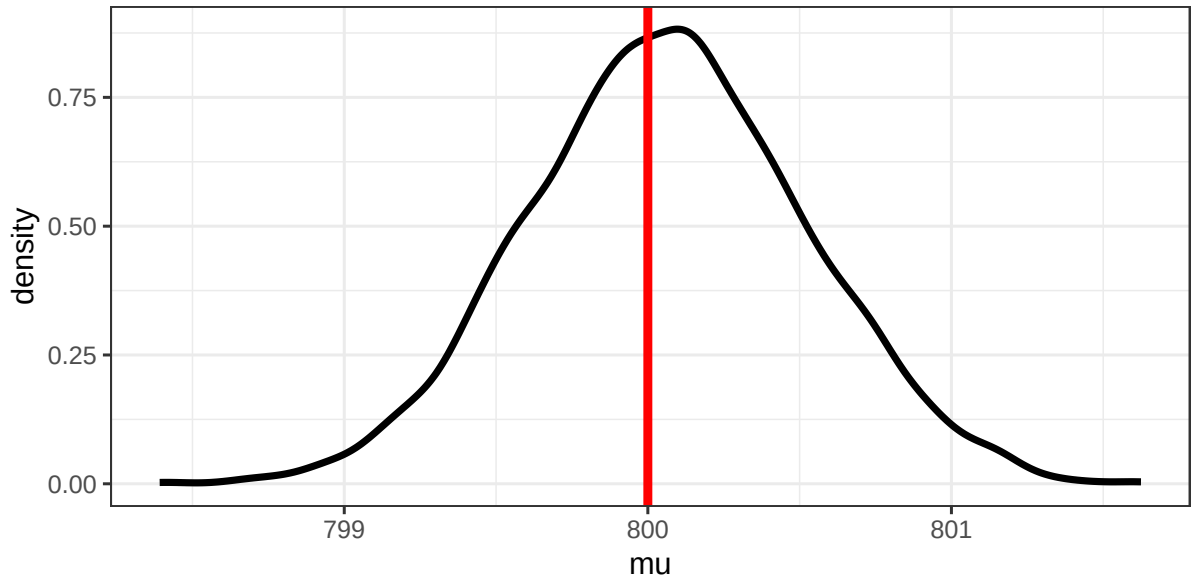
```
#Potential energy function
V <- function(mu,sigma,y,n,m,s,a,b){
  nlpd <- -(sum(dnorm(y,mu,sigma,log=T))+dnorm(mu,m,s,log=T)+dnorm(sigma,a,b,log=T))
  nlpd
}
```

Step 3: Implement HMC sampler

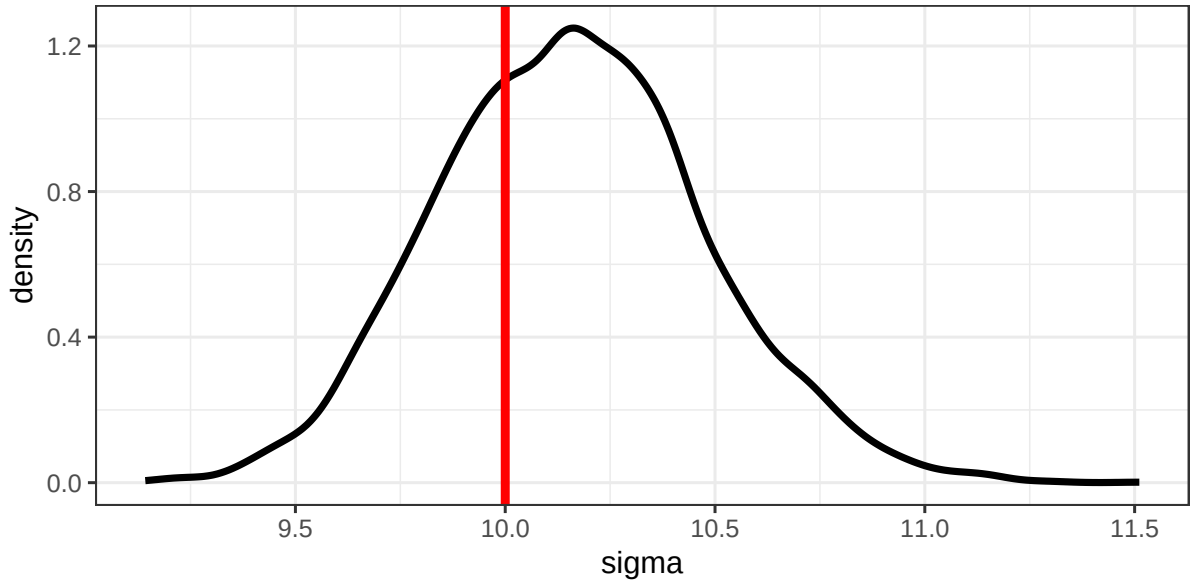
```
#Data
y <- y
n <- length(y)

#Model parameters
m <- 1000
s <- 100
a <- 10
b <- 2

# HMC sampler
# Internal parameters
# Step size
step <- 0.02
# Number of leapfrog steps
```

```r
L <- 20

# Markov chain
nsamp <- 8000
mu_chain <- rep(NA,nsamp)
sigma_chain <- rep(NA,nsamp)
reject <- 0

#Initialization of Markov chain
mu_chain[1] <- rnorm(1,1000,10)
sigma_chain[1] <- rnorm(1,10,1)

#Evolution of Markov chain
i <- 1
while(i < nsamp){
  q <- c(mu_chain[i],sigma_chain[i])   # Current position of the particle
  p <- rnorm(length(q),0,1)            # Generate random momentum at the current position
  current_q <- q
  current_p <- p
  current_V = V(current_q[1],current_q[2],y,n,m,s,a,b) # Current potential energy
  current_T = sum(current_p^2)/2                       # Current kinetic energy
  # Take L leapfrog steps
  for(l in 1:L){
    # Change in momentum in 'step/2' time
    p <- p-((step/2)*gradient(q[1],q[2],y,n,m,s,a,b))
    # Change in position in 'step' time
    q <- q + step*p
    # Change in momentum in 'step/2' time
    p <- p-((step/2)*gradient(q[1],q[2],y,n,m,s,a,b))
  }
  proposed_q <- q
  proposed_p <- p
  proposed_V = V(proposed_q[1],proposed_q[2],y,n,m,s,a,b)  # Proposed potential energy
  proposed_T = sum(proposed_p^2)/2                         # Proposed kinetic energy
  accept.prob <- min(1,exp(current_V+current_T-proposed_V-proposed_T))
  # Accept/reject the proposed position q
  if(accept.prob>runif(1,0,1)){
    mu_chain[i+1] <- proposed_q[1]
    sigma_chain[i+1] <- proposed_q[2]
    i <- i+1
  }else{
    reject <- reject+1
  }
}
```

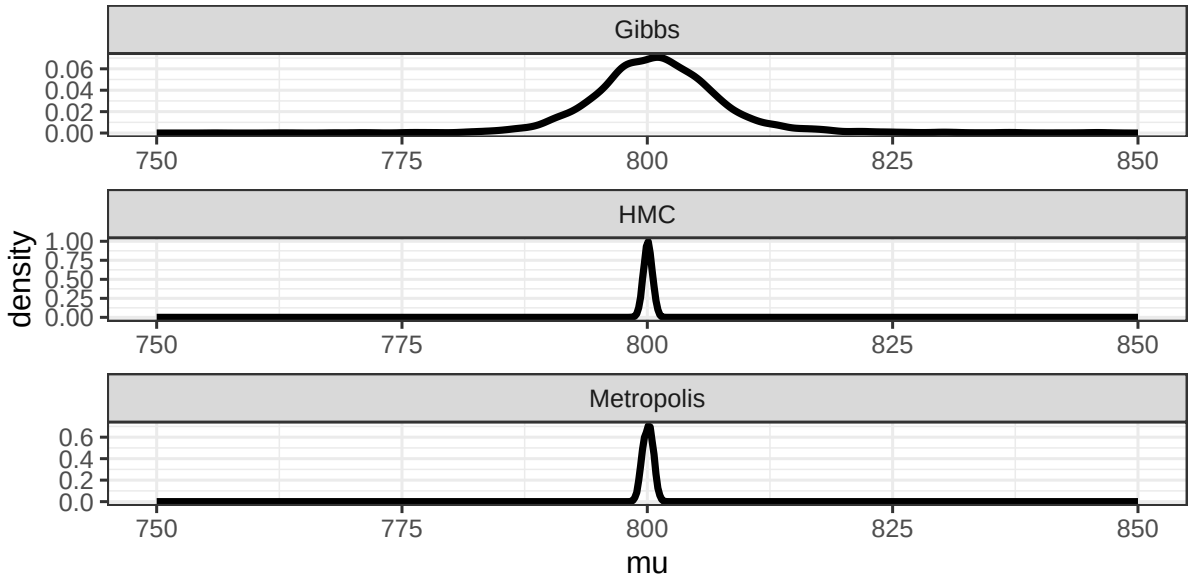
```
posteriors <- data.frame(mu_chain,sigma_chain)
ggplot(posteriors[-(1:2000),],aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept=800,size=1.5,color="red")
```



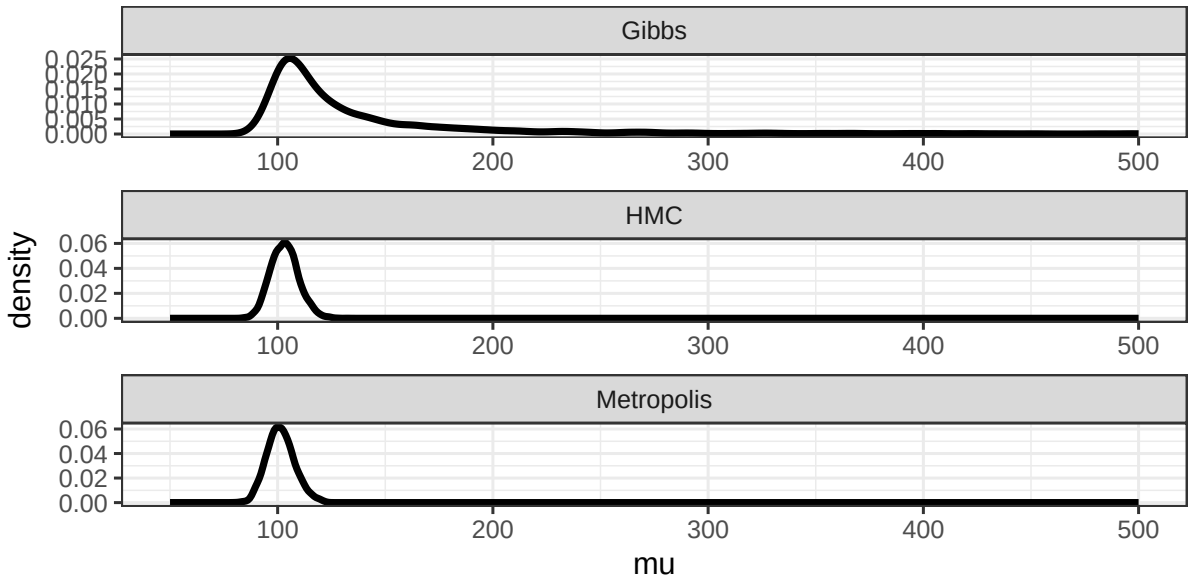

```
ggplot(posteriors[-(1:2000),],aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("sigma")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  geom_vline(xintercept = 10,size=1.5,color="red")
```




```
HMC_posterior <- posteriors
HMC_posterior$sigma_chain <- HMC_posterior$sigma_chain^2
gibbs_posterior$algorithm <- "Gibbs"
metropolis_posterior$algorithm <- "Metropolis"
HMC_posterior$algorithm <- "HMC"

gibbs_metro_hmc <- rbind(gibbs_posterior[-(1:6000),],metropolis_posterior[-(1:4000),],HMC_posterior[-(1:20

ggplot(gibbs_metro_hmc,aes(x=mu_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                      legend.position = "top")+
  facet_wrap(~algorithm,scales = "free",nrow = 3)+
  scale_x_continuous(limits = c(750,850))
```

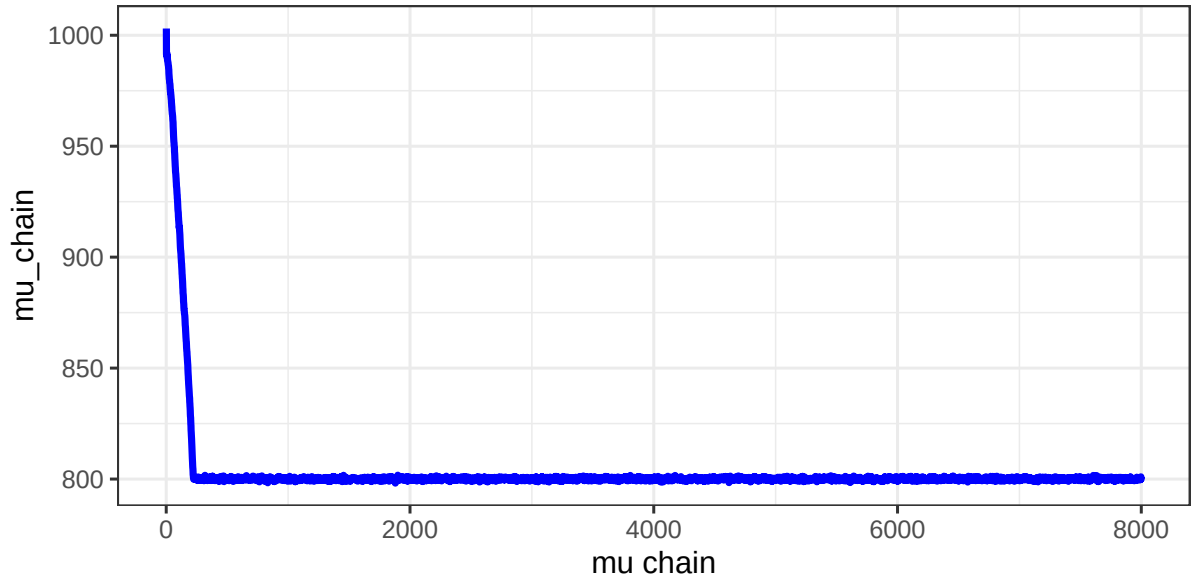



```
ggplot(gibbs_metro_hmc,aes(x=sigma_chain))+
  geom_density(size=1.2)+
  theme_bw()+xlab("mu")+
  theme(legend.title = element_blank(),
                       legend.position = "top")+
  facet_wrap(~algorithm,scales = "free",nrow = 3)+
  scale_x_continuous(limits = c(50,500))
```

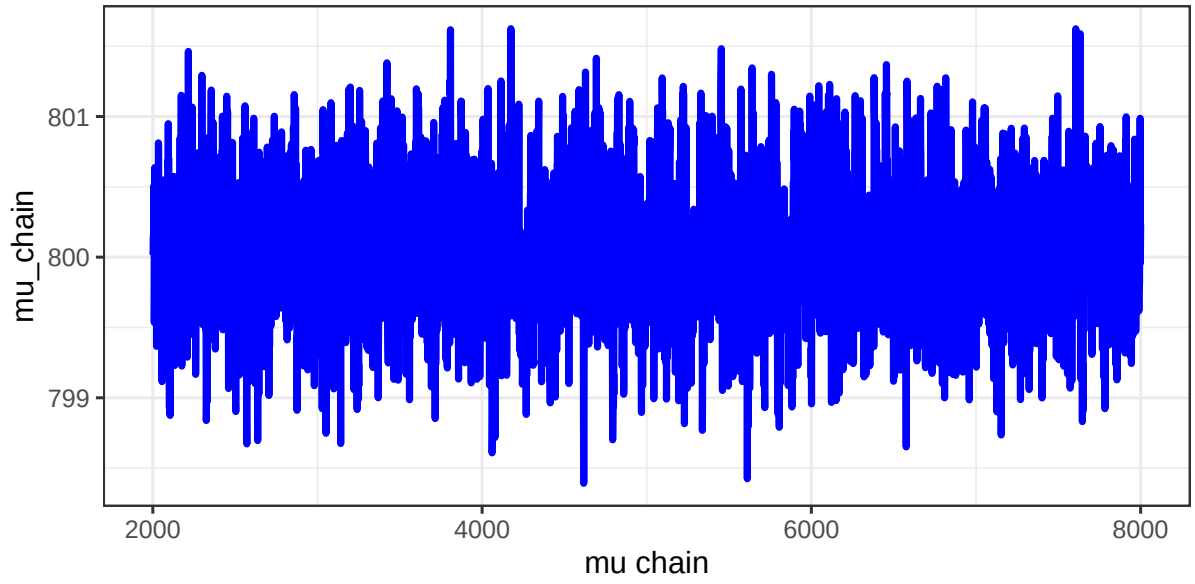



```
# Inspect chains
posteriors <- data.frame(mu_chain,sigma_chain)
```

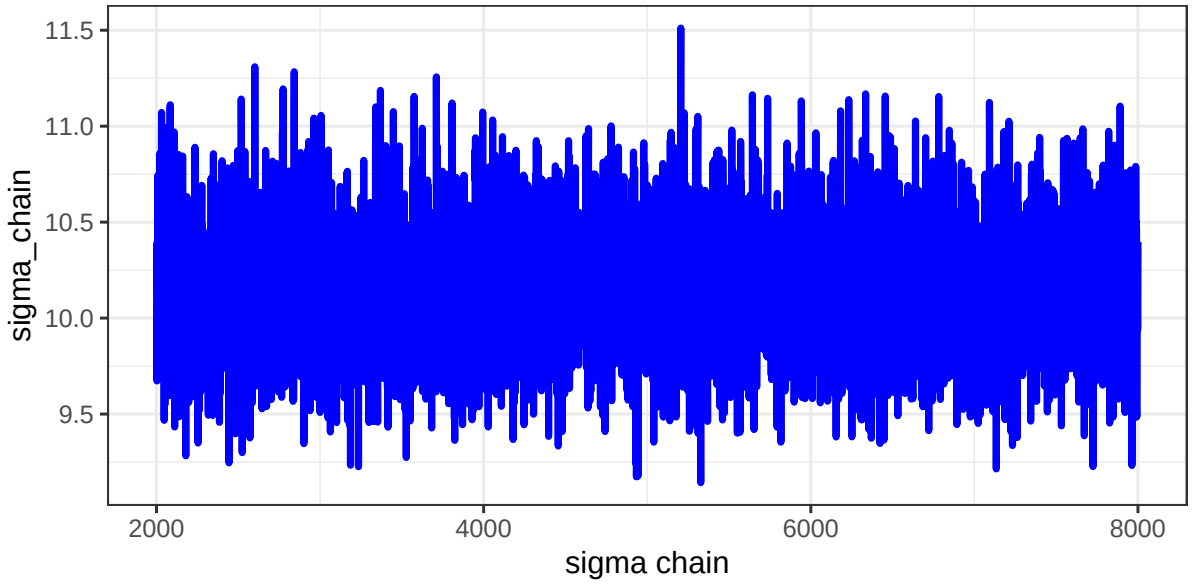
```
posteriors$id <- 1:nsamp
ggplot(posteriors,aes(x=id,y=mu_chain))+
  geom_line(size=1.2,color="blue")+
  theme_bw()+xlab("mu chain")
```




```
ggplot(posteriors[-c(1:2000),],aes(x=id,y=mu_chain))+
  geom_line(size=1.2,color="blue")+
  theme_bw()+xlab("mu chain")
```




```
ggplot(posteriors[-c(1:2000),],aes(x=id,y=sigma_chain))+
  geom_line(size=1.2,color="blue")+
  theme_bw()+xlab("sigma chain")
```



10 Parameter estimation using brms/stan

- There are packages in R and Python can estimate parameters for you using one of posterior simulation algorithms introduced here.
- You only need to define your likelihood and the priors in a given syntax; the algorithm (defined for the package) will start drawing samples from the posterior.
- A popular one is Rstan/pystan/brms package, which uses a Hamiltonian Monte Carlo algorithm for sampling.
- We will talk about them in the next lecture; you can read chapter 3 of the book "An Introduction to Bayesian Data Analysis for Cognitive Science (https://vasishth.github.io/bayescogsci/book/)" for reference.

10.1 Implementation of the previous example using brms

```
library(brms)

# We need data in a dataframe format
dat <- data.frame(y=y,obs=1:length(y))
head(dat)
```

```
##          y obs
## 1 791.4674   1
## 2 797.0719   2
## 3 786.8546   3
## 4 798.3879   4
```

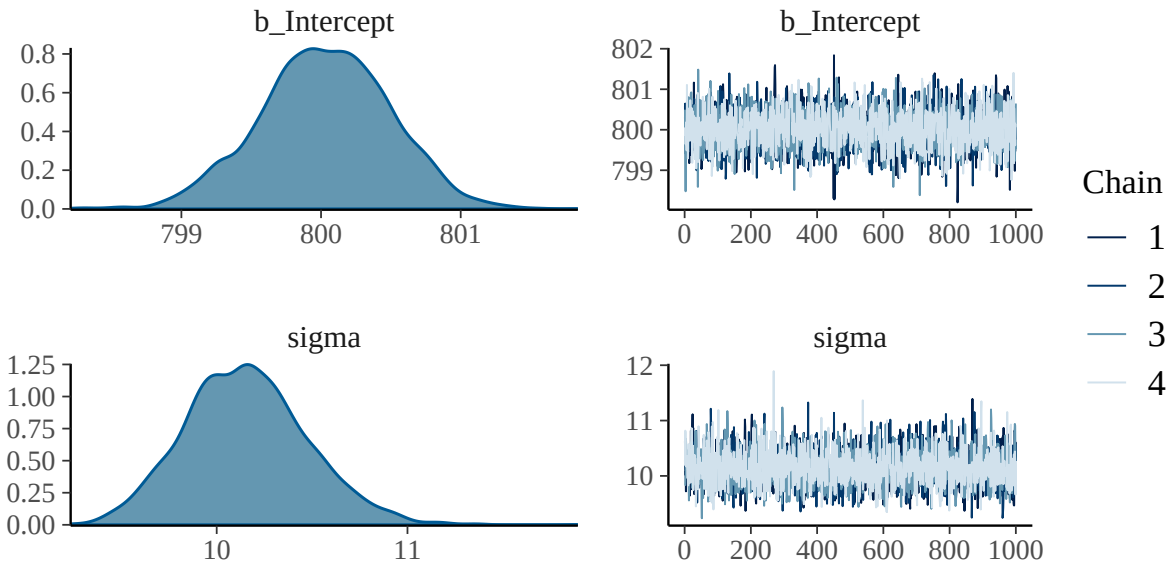
```
## 5 786.7599   5
## 6 801.9058   6
```

```
# brm function estimates model parameters using HMC

fit_normal <- brm(y~1, data=dat,
                  family = gaussian(),   # likelihood
                  prior = c(
                    prior(normal(1000, 100), class = Intercept),
                    prior(normal(10,2), class=sigma)
                  ),
                  chains = 4,
                  iter = 2000,
                  warmup = 1000,
                  cores=4)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
plot(fit_normal)
```




```
posterior_summary(fit_normal)
```

```
##                 Estimate  Est.Error         Q2.5        Q97.5
## b_Intercept   800.030335 0.46607394   799.111208   800.900274
## sigma          10.153867 0.31946035     9.565223    10.817231
## lprior         -9.151311 0.02643087    -9.221978    -9.122592
## lp__        -1874.865172 1.05129548 -1877.580053 -1873.862596
```