Bayesian Estimation with Markov Chain Monte Carlo

January 2011

What we know

$$\Pr(D|\theta)$$

Likelihood

(probability of data, assuming model is true)

What we WANT know

$$\Pr(\theta|D)$$

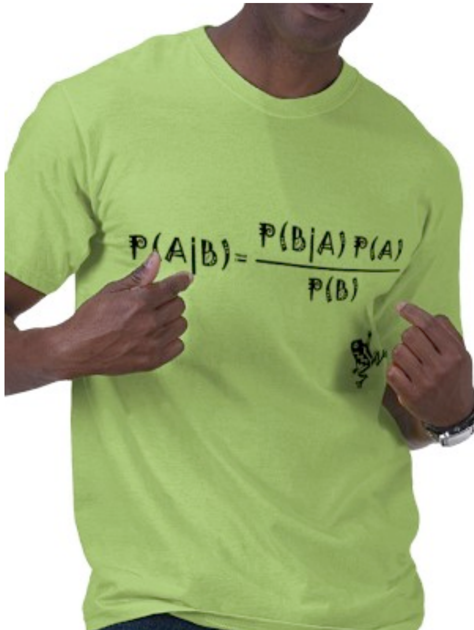
Posterior
(probability of model, given data)

$$\Pr(\theta|D) \neq \Pr(D|\theta)$$

Bayes' theorem

$$\Pr(\theta|D) = \frac{\Pr(D|\theta)\,\Pr(\theta)}{\Pr(D)}$$

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalization}}$$



Likelihood perspective

$$\Pr(\theta|D) = \frac{\Pr(D|\theta)\,\Pr(\theta)}{\Pr(D)}$$

$$\Pr(\theta|D) \propto \Pr(D|\theta)$$

$$L(\theta|D) = \Pr(D|\theta)$$

Maximize likelihood to find model supported by the evidence (*D*). When *n* large, prior won't matter.

Bayesian perspective

$$\Pr(\theta|D) = \frac{\Pr(D|\theta)\,\Pr(\theta)}{\Pr(D)}$$

$$\Pr(\theta|D) \propto \Pr(D|\theta)\,\Pr(\theta)$$

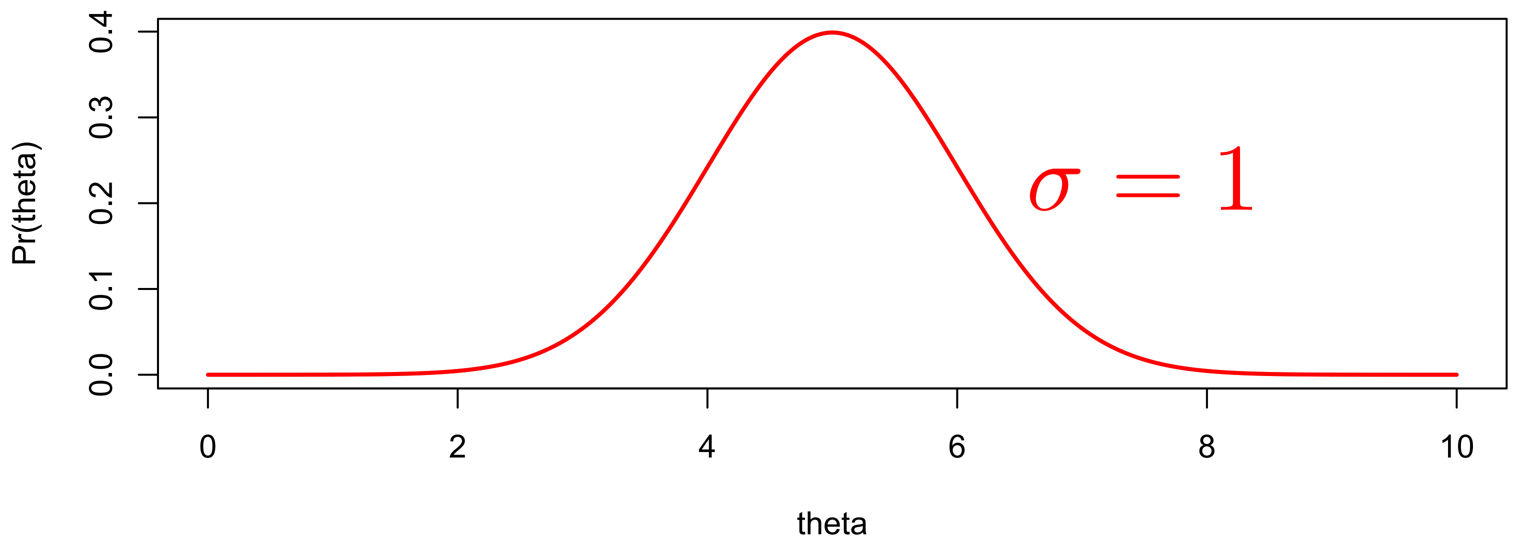
Use prior beliefs with data to update posterior beliefs about the model.

Strong Bayesians

- Must be explicit about prior beliefs, because Bayes' theorem proves prior always affects inference about models.
- Likelihood-only based inference implicitly uses "uninformative" priors.

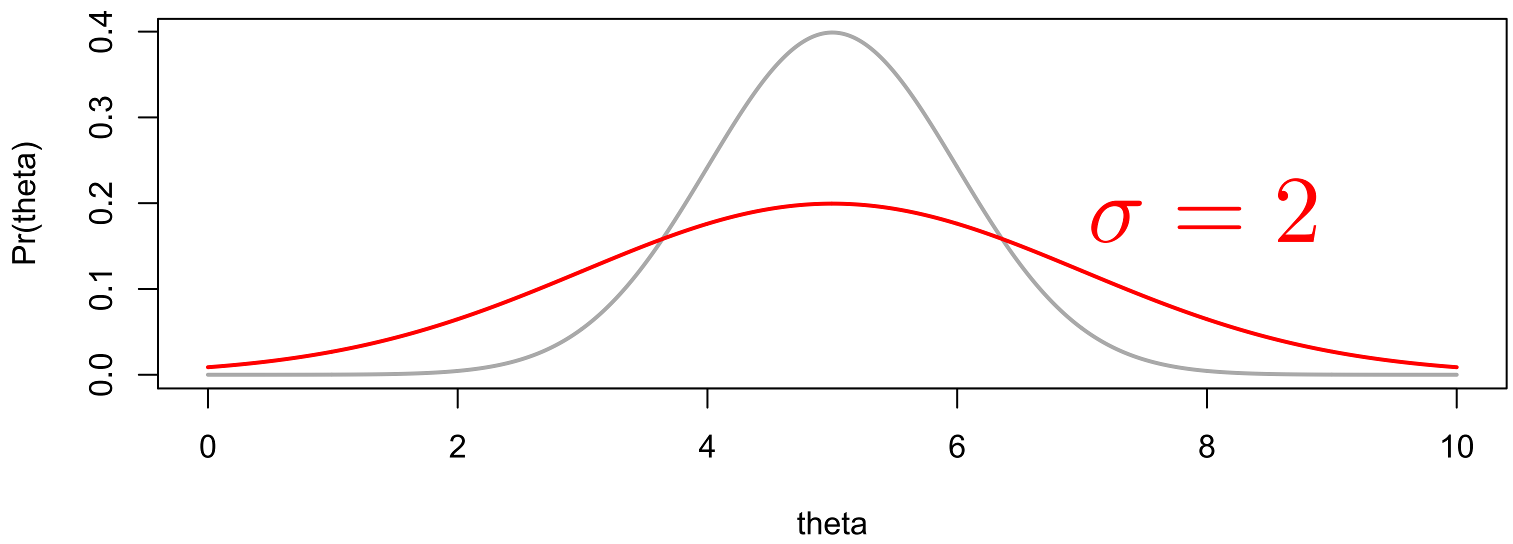
Uninformative Priors

- An “uninformative” prior assigns equally low probability to all values of Θ.



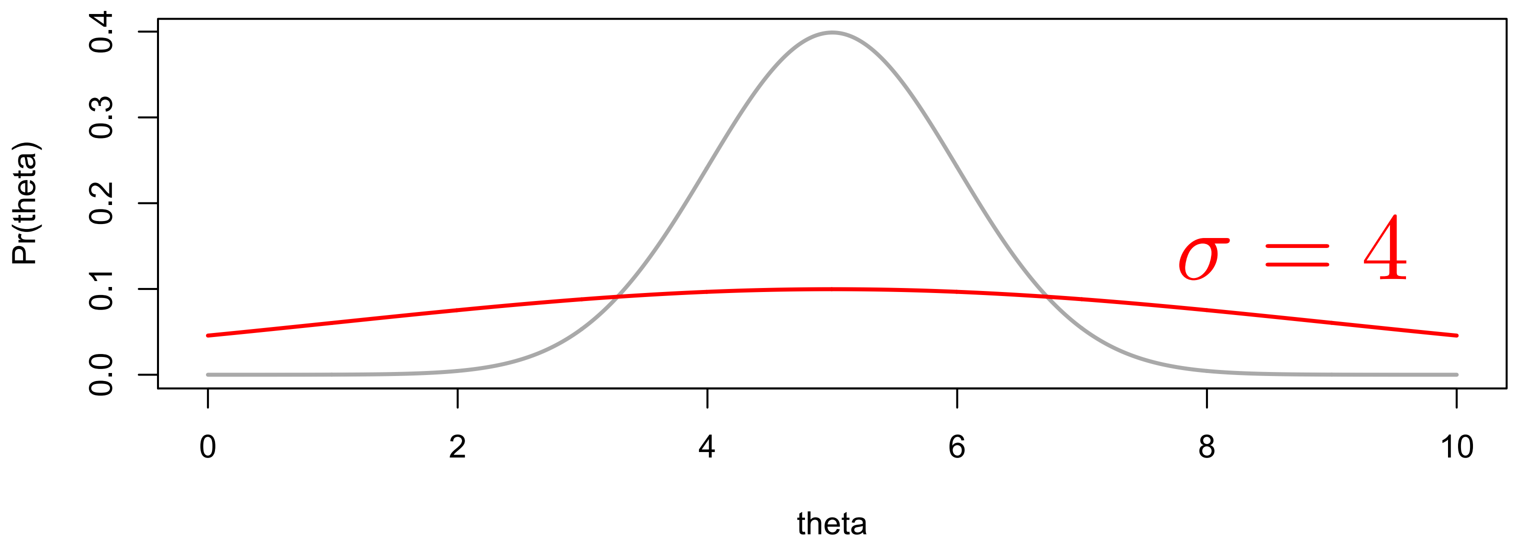
Uninformative Priors

- An "uninformative" prior assigns equally low probability to all values of Θ.



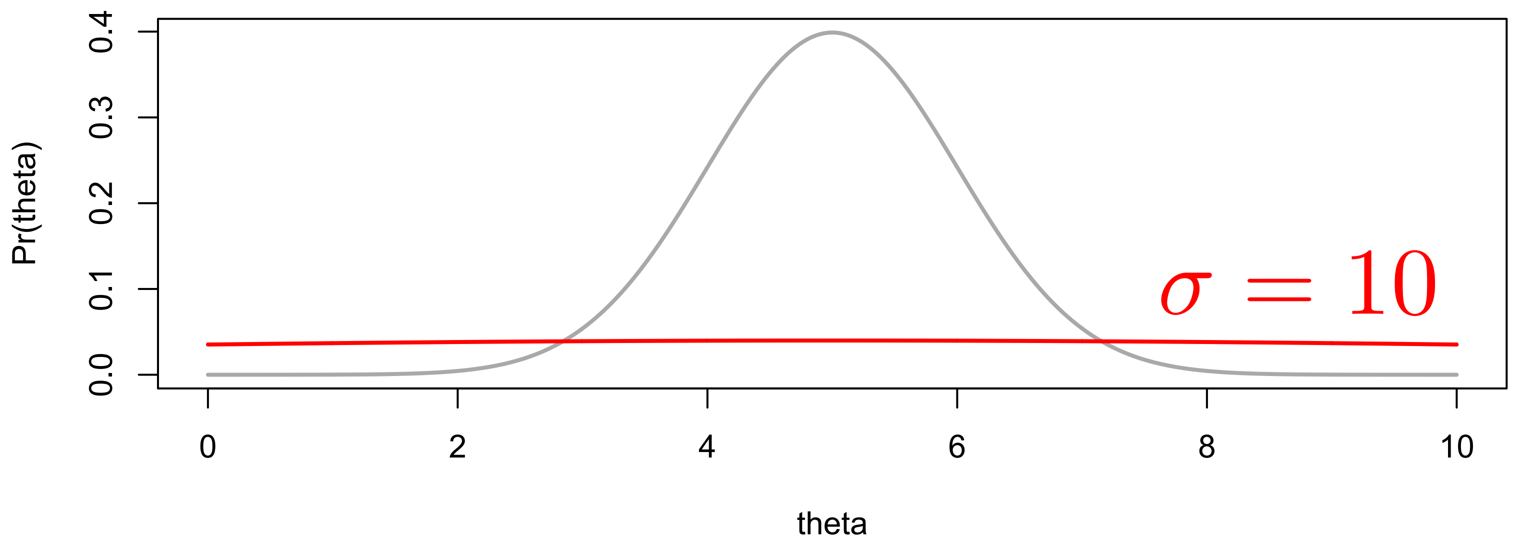
Uninformative Priors

- An "uninformative" prior assigns equally low probability to all values of Θ.



Uninformative Priors

- An “uninformative” prior assigns equally low probability to all values of Θ.



Uninformative Priors

- An "uninformative" prior assigns equally low probability to all values of Θ.

$$\Pr(\theta|D) = \frac{\Pr(D|\theta)k_1}{\Pr(D)}$$

$$\Pr(\theta|D) = \Pr(D|\theta)\frac{k_1}{k_2} \propto \Pr(D|\theta)$$

Strong Bayesians

- Must be explicit about prior beliefs, because Bayes' theorem proves prior always affects inference about models.
- Likelihood-only based inference implicitly uses "uninformative" priors.
- Likelihood response: Hard to justify priors. What is the prior probability that God exists? (Sober 2008)
- In any event, as n gets large,
  Bayesian results ≈ non-Bayesian results.

Weak Bayesians

- Bayes' theorem requires priors, and when we can justify a prior, or use an uninformative prior, we can do useful things:
  - Bayesian estimation more capable than likelihood maximization. Many HIERARCHICAL models are hard or impossible to MLE, straightforward to BE.

Weak Bayesians

- Bayes' theorem requires priors, and when we can justify a prior, or use an uninformative prior, we can do useful things:
  - Bayesian estimation more capable
  - Bayesian estimation gives equivalent of confidence intervals (credible intervals) with no extra work. MLE CI's often require a lot of work.

Weak Bayesians

- Bayes' theorem requires priors, and when we can justify a prior, or use an uninformative prior, we can do useful things:
  - Bayesian estimation more capable
  - Bayesian estimation gives equivalent of confidence intervals with no extra work.
  - Can use previous information in inference. e.g.: observe a black hole, try to estimate mass, incorporate previous estimates.

Bayesian estimation

- MLE: Only specify distribution of data

$$y_i \sim \mathcal{N}(\mu, \sigma)$$

- Bayes: Also specify beliefs about distribution of parameters

$$y_i \sim \mathcal{N}(\mu, \sigma)$$

$$\sigma \sim \text{inv-gamma}(s_0, r_0)$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0)$$

Bayesian estimation

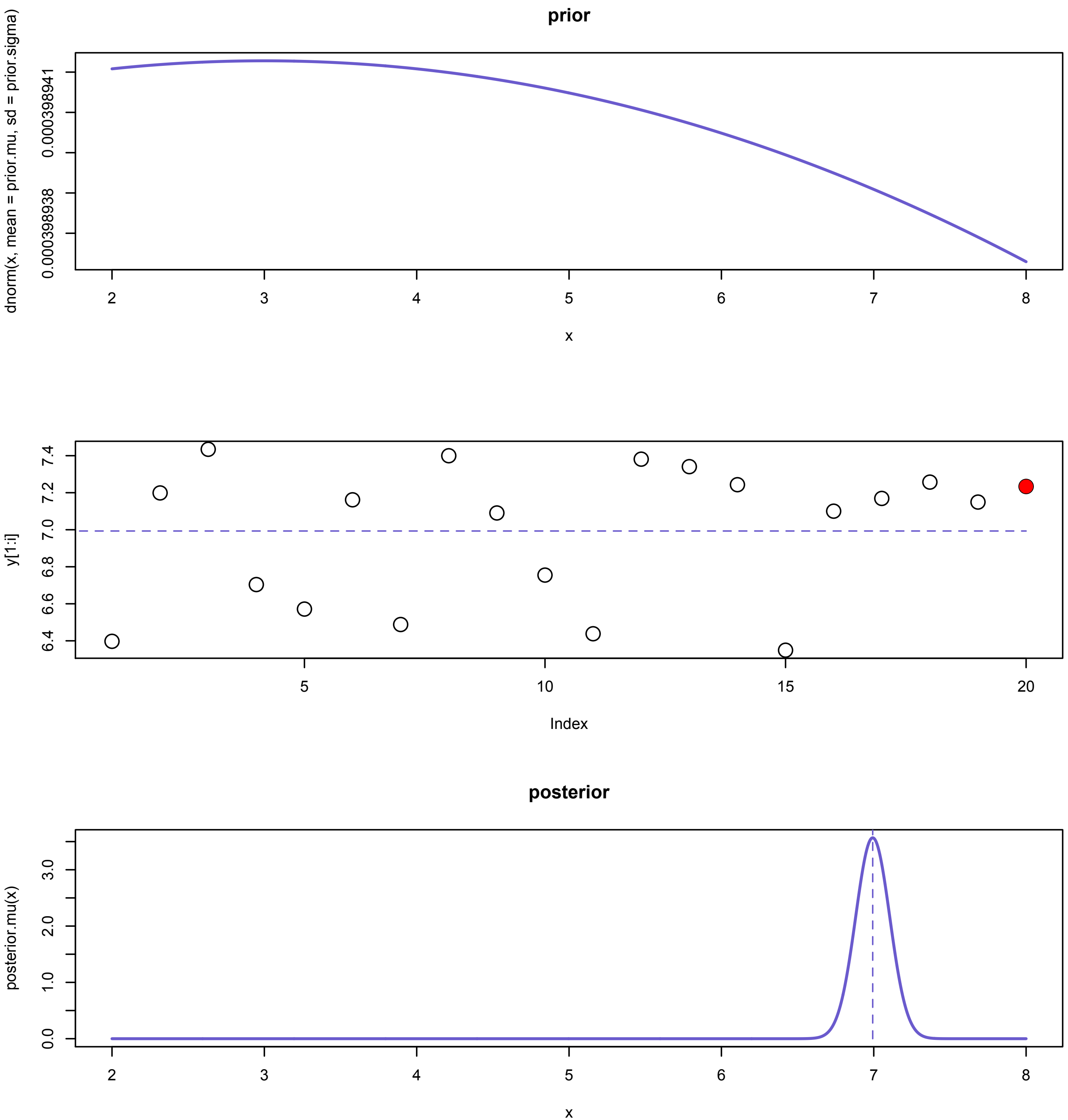
- ANY model fit with MLE can be fit in Bayesian fashion.
- Task: Compute the posterior, $\Pr(\Theta | D)$.
- Two main ways:
  (1) Do the math
  (2) Markov Chain Monte Carlo (MCMC)

Bayesian estimation

- Two main ways:
  (1) Do the math
- Computing posterior often requires doing a difficult integral:

$$\Pr(\theta|D) = \frac{\Pr(D|\theta)\,\Pr(\theta)}{\int \Pr(\theta)\,\Pr(D|\theta)}$$

bayesian updating animation.r



Bayesian estimation

- Two main ways:
  (2) Markov Chain Monte Carlo (MCMC)
- MCMC: Procedures for simulating random draws from posterior, where each draw is dependent on only the previous draw.
- Sequence of draws is called a "chain."
- If our chain meets some simple conditions, then guaranteed to converge to posterior.

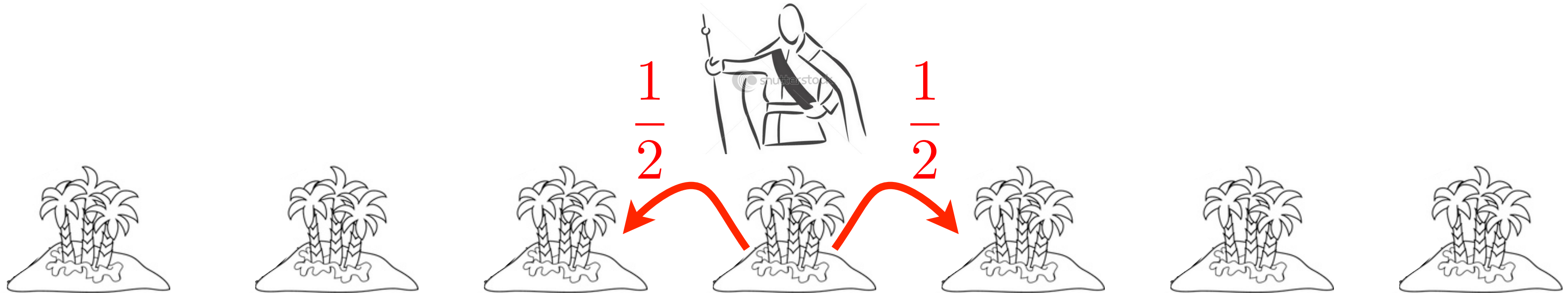
Markov chain in the island chain

King Markov

The Metropolis Archipelago

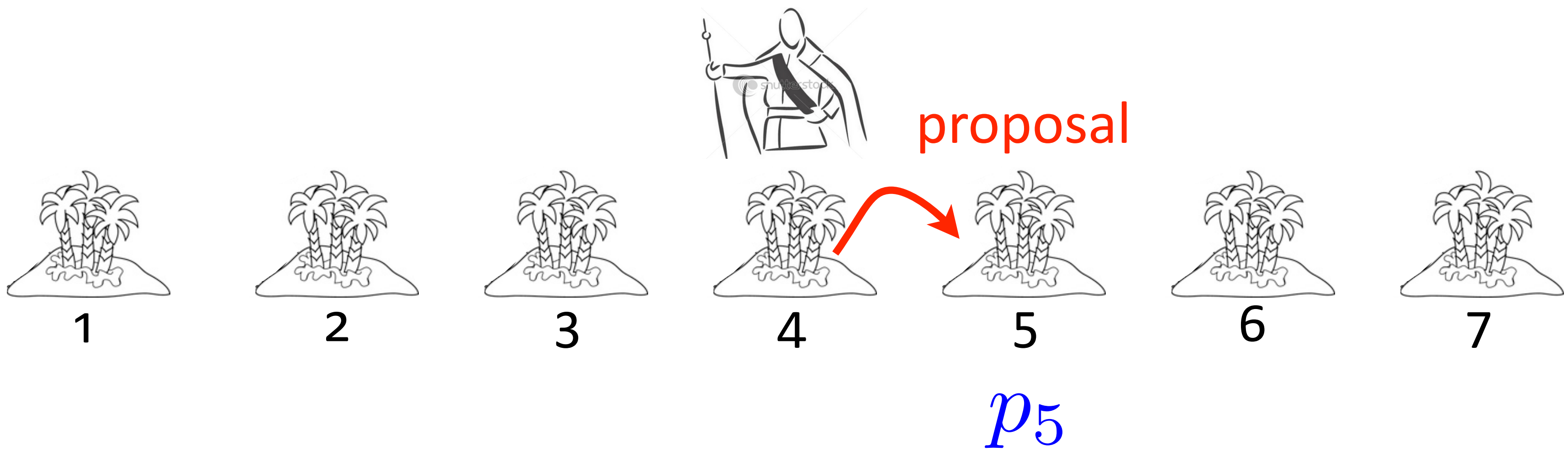
Contract: King Markov must visit each island in proportion to its population size.

Here’s how he does it...



(1) Flip a coin to choose island on left or right.
Call it the “proposal” island.

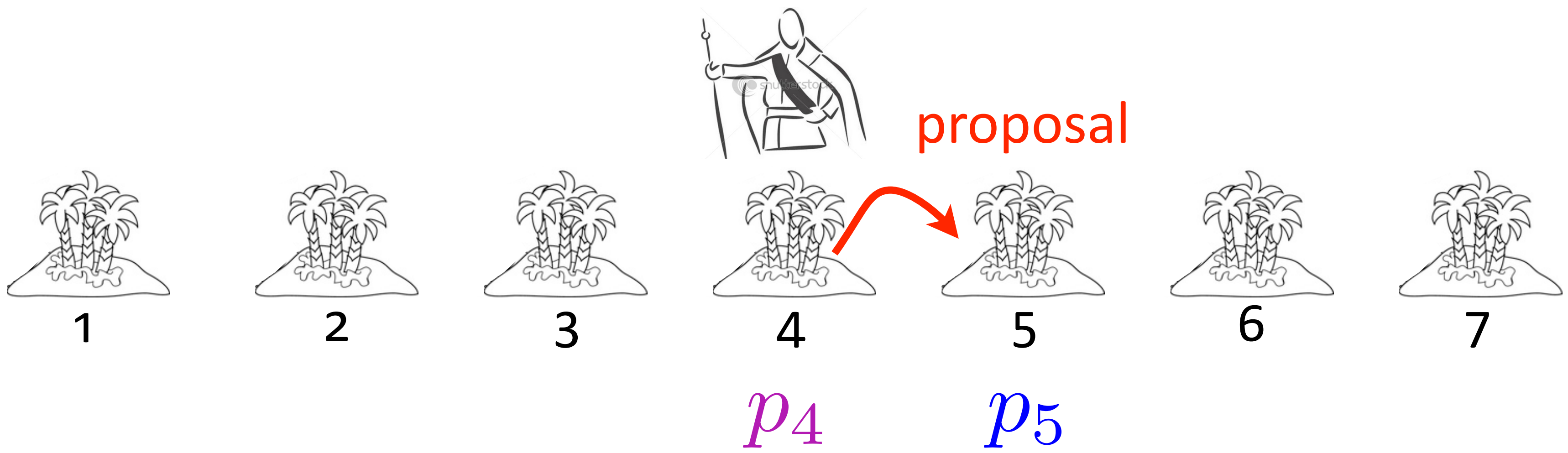
(1) Flip a coin to choose island on left or right.
Call it the “proposal” island.



(2) Find population of proposal island.

(1) Flip a coin to choose island on left or right.
Call it the "proposal" island.

(2) Find population of proposal island.

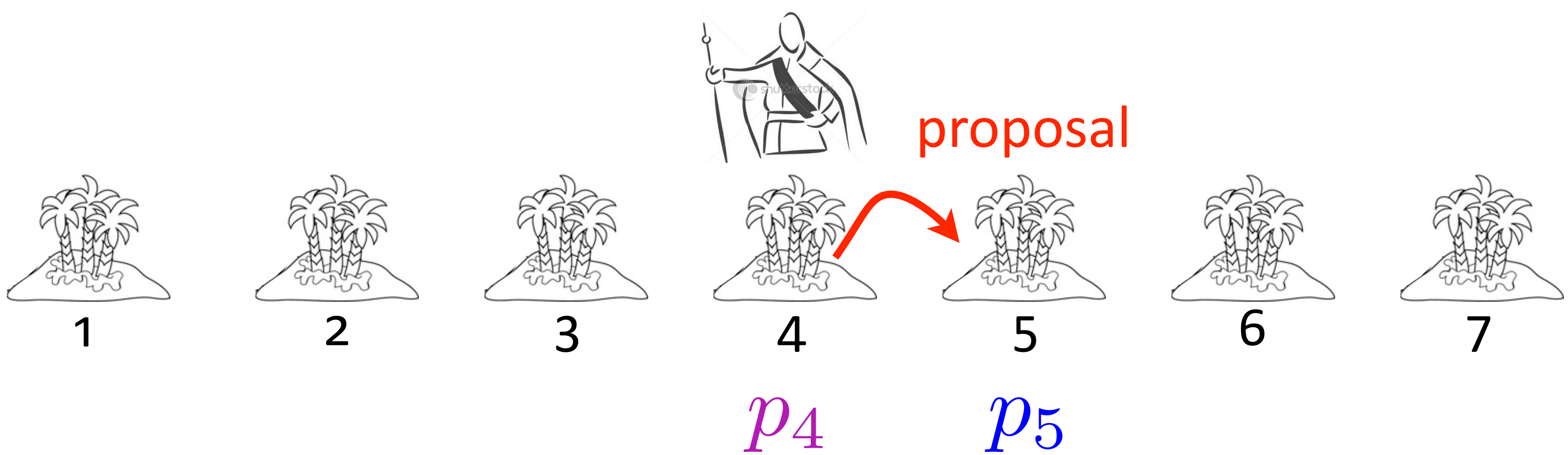


(3) Find population of current island.

(1) Flip a coin to choose island on left or right.
Call it the "proposal" island.

(2) Find population of proposal island.

(3) Find population of current island.



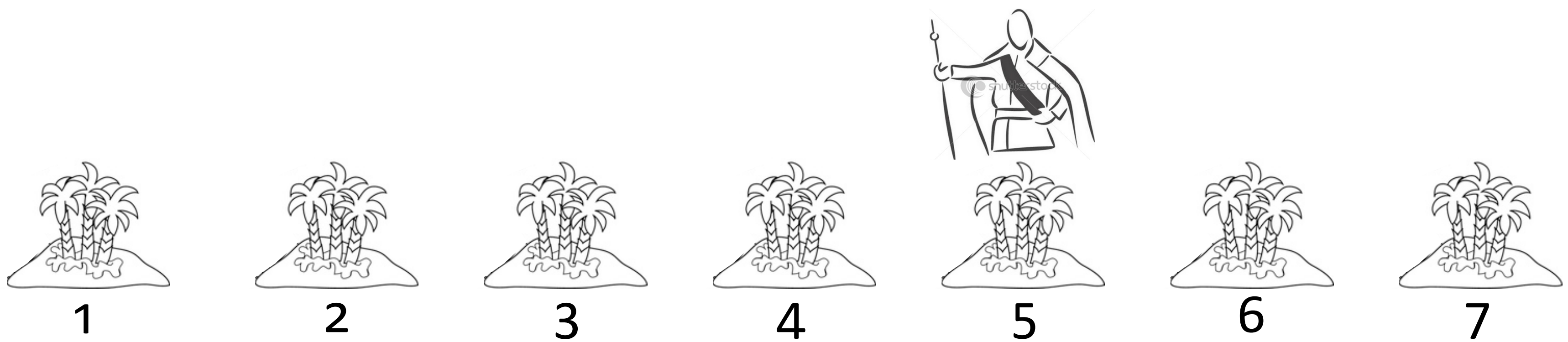
(4) Move to proposal, with probability = $\frac{p_5}{p_4}$

(1) Flip a coin to choose island on left or right. Call it the "proposal" island.

(2) Find population of proposal island.

(3) Find population of current island.

(4) Move to proposal, with probability = $\frac{p_5}{p_4}$



(5) Repeat from (1).

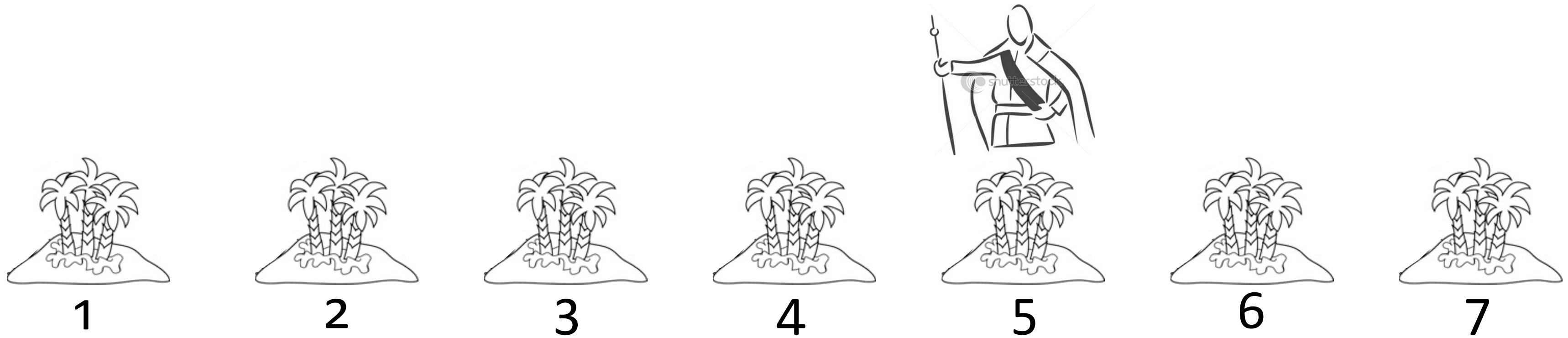
(1) Flip a coin to choose island on left or right.
Call it the "proposal" island.

(2) Find population of proposal island.

(3) Find population of current island.

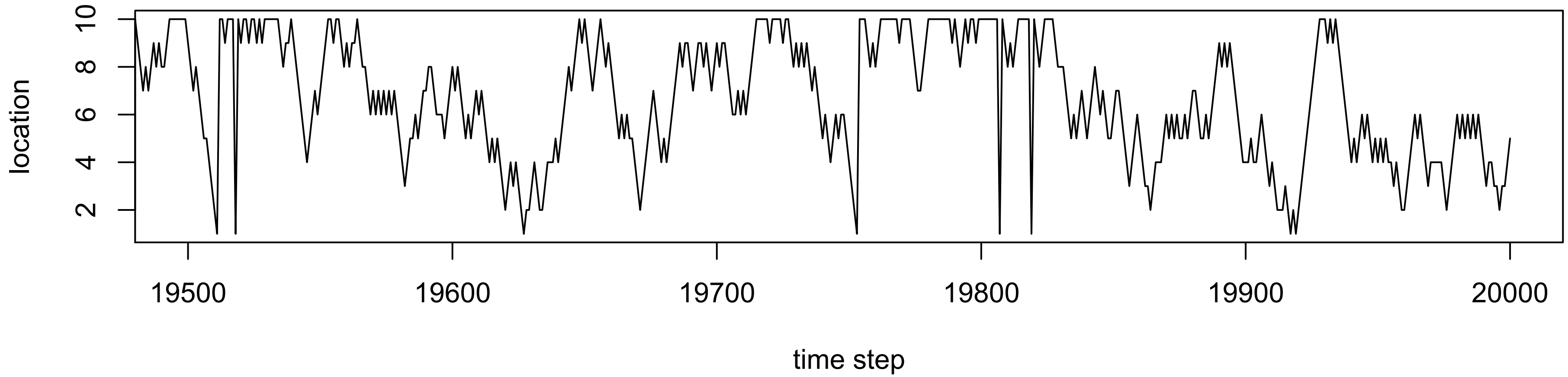
(4) Move to proposal, with probability = $\frac{p_5}{p_4}$

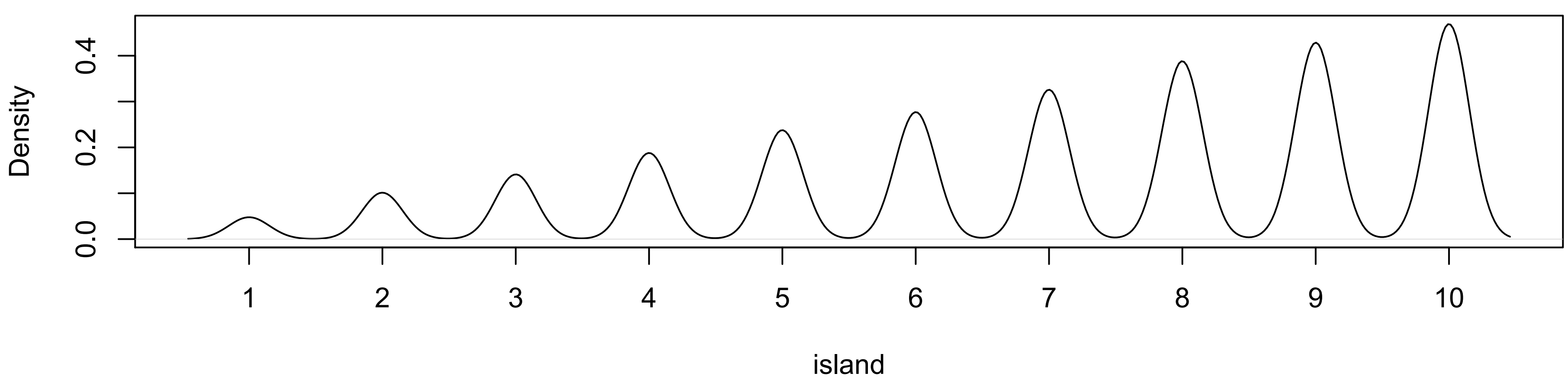
(5) Repeat from (1).



This procedure ensures visiting each island in proportion to its population, *in the long run*.

King Markov Island Chain.r





Metropolis algorithm

- Markov's strategy is an example of the **Metropolis algorithm**, a MCMC method.
- Islands = values of Θ
- Population = $\Pr(D|\Theta)\Pr(\Theta) \propto \Pr(\Theta|D)$
- As long as proposals are symmetric, (chance look left = chance look right) always works.

Metropolis algorithm

- A better example: Estimate mean of a normal distribution.

$$y_i \sim \mathcal{N}(\mu, \sigma)$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0)$$

metropolis1.r

```r
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Define prior

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)
```

Starting parameter values

```
num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

```r
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Determine number of samples

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Initialize empty chain of samples

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Set width of proposal distribution

```r
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Generate samples from chain

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Record current parameter value

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Generate proposal value

```r
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

$\Pr(D|\text{prop.mu})\Pr(\text{prop.mu})$

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Pr(D|k.mu)Pr(k.mu)

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

$$\frac{\Pr(D|\text{prop.mu})\Pr(\text{prop.mu})}{\Pr(D|\text{k.mu})\Pr(\text{k.mu})}$$

```
prior.mu <- function(theta) dnorm( theta , mean=7 , sd=1000 , log=TRUE )

k.mu <- 8
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
step <- 1/10
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu

    prop.mu <- k.mu + rnorm( 1 , mean=0 , sd=step)

    pr.prop <- sum( dnorm( y , mean=prop.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(prop.mu)
    pr.here <- sum( dnorm( y , mean=k.mu , sd=k.sigma , log=TRUE ) ) + prior.mu(k.mu)
    pr.accept <- exp( pr.prop - pr.here )

    k.mu <- ifelse( runif(1) < pr.accept , prop.mu , k.mu )
}
```

Accept proposal or not

Interpreting the chain

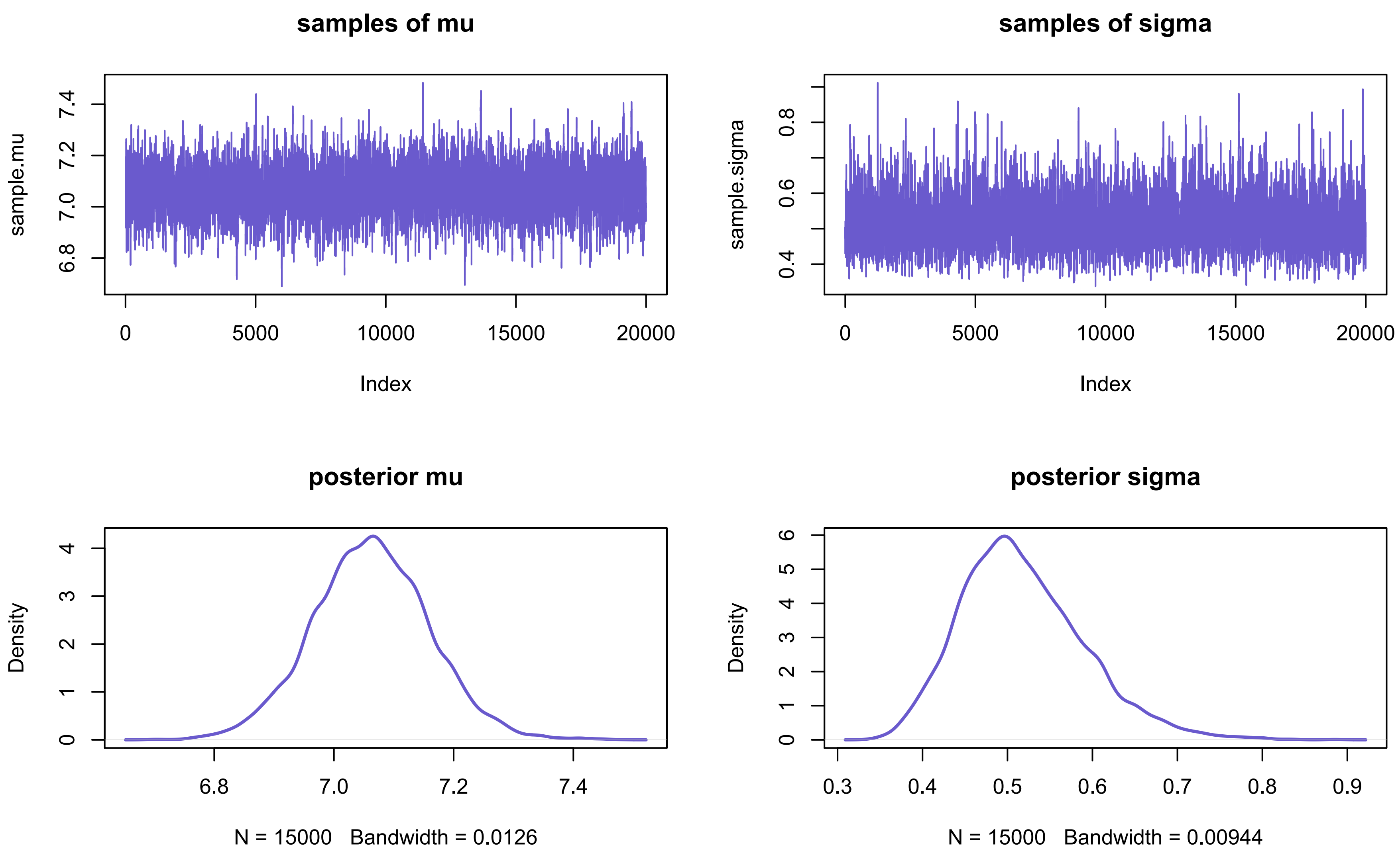
- Plot the chain: Did it converge?
- Remove “burn in” and plot density
- Plot the chain: Is it autocorrelated?
- Compute credible intervals
- “Thinning” the chain is usually unnecessary, but saves memory

metropolis2.r

$$y_i \sim \mathcal{N}(\mu, \sigma)$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0)$$

$$\sigma \sim \text{inv-gamma}(s_0, r_0)$$



Gibbs Sampling

- BUGS, OpenBUGS, JAGS. wtf?
- GS is Gibbs Sampling, a MCMC algorithm, based on Metropolis
- Gibbs Sampling uses posterior of each parameter to sample each parameter – always accepts proposal
- Requires being able to compute:

$$\Pr(\theta_1|D,\theta_2,\theta_3,\ldots,\theta_n) = \Pr(D|\theta_1,\theta_2,\theta_3,\ldots,\theta_n)\Pr(\theta_1)/\Pr(D)$$

Gibbs Sampling

- GS requires being able to compute:

$$\Pr(\theta_1 | D, \theta_2, \theta_3, \ldots, \theta_n) = \Pr(D | \theta_1, \theta_2, \theta_3, \ldots, \theta_n) \Pr(\theta_1) / \Pr(D)$$

- A lot better than having to compute:

$$\Pr(\theta_1, \theta_2, \theta_3, \ldots, \theta_n | D) = \Pr(D | \theta_1, \theta_2, \theta_3, \ldots, \theta_n) \Pr(\theta_1, \theta_2, \theta_3, \ldots, \theta_n) / \Pr(D)$$

Gibbs Sampling

- GS requires being able to compute:

$$\Pr(\theta_1|D, \theta_2, \theta_3, ..., \theta_n) = \Pr(D|\theta_1, \theta_2, \theta_3, ..., \theta_n)\Pr(\theta_1)/\Pr(D)$$

- Metropolis only requires:

$$\Pr(D|\theta_1, \theta_2, \theta_3, ..., \theta_n)\Pr(\theta_1)$$

- GS more efficient (never rejects a proposal, no need to tune proposals), but needs more information.

gibbs.r

$$y_i \sim \mathcal{N}(\mu, \sigma)$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma_0)$$

$$\sigma \sim \text{inv-gamma}(s_0, r_0)$$

```
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

```r
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Define prior beliefs

```
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)
```

Starting guesses for parameters

```
num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

```
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Initialize Markov chains

```r
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Generate samples from chains

```r
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Record current parameter values

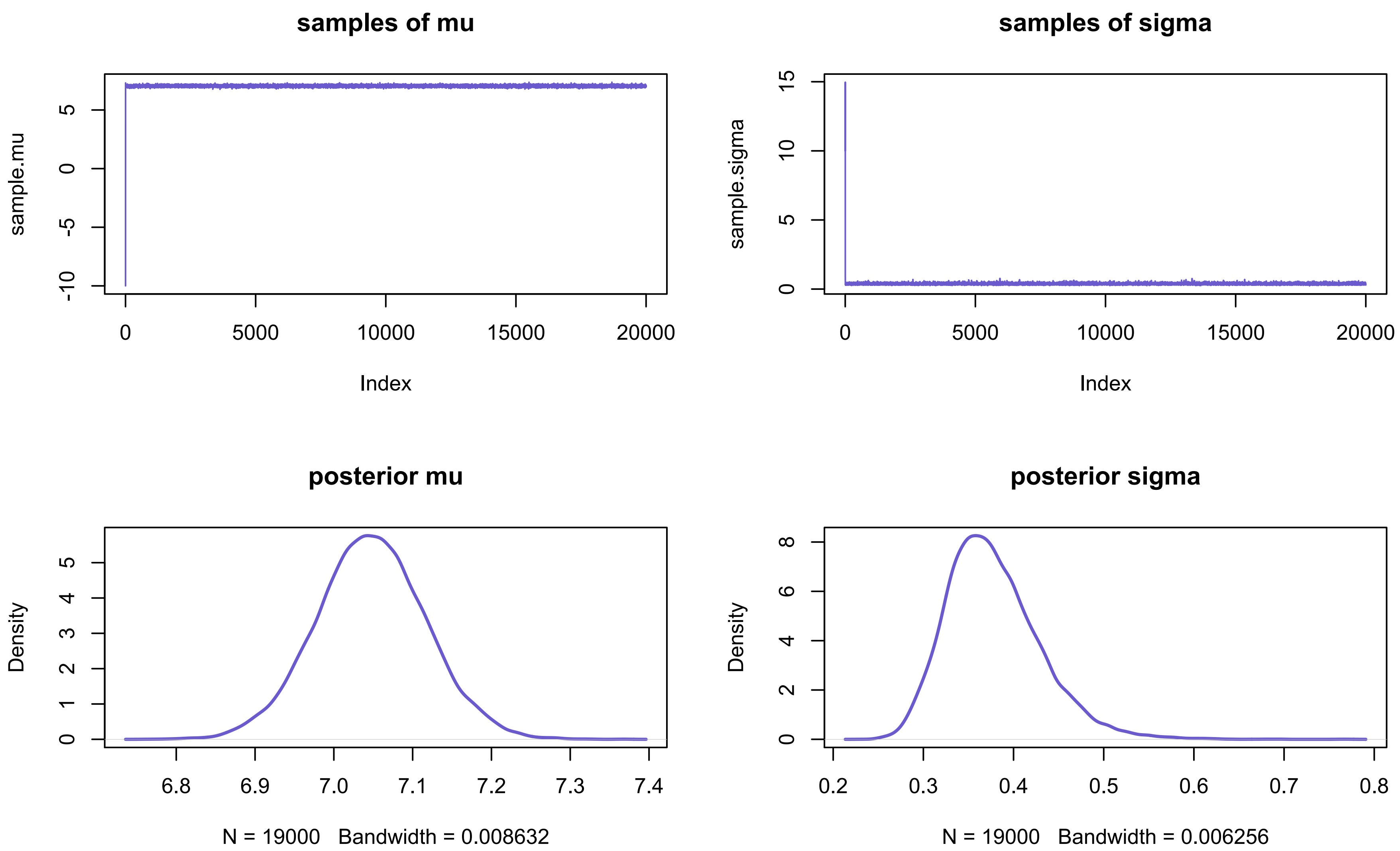
```r
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Sample sigma from posterior

```
mu0 <- 7
sigma0 <- 1000
shape0 <- 0.001
rate0 <- 0.001

k.mu <- mean(y)
k.sigma <- sd(y)

num.samples <- 20000
sample.mu <- rep(0,num.samples)
sample.sigma <- rep(0,num.samples)
for ( i in 1:num.samples ) {
    sample.mu[i] <- k.mu
    sample.sigma[i] <- k.sigma

    k.tau <- rgamma( 1 , shape=shape0 + length(y)/2 , rate=rate0 + sum( (y-k.mu)^2 )/2 )
    k.sigma <- sqrt(1/k.tau)
    k.mu <- rnorm( 1 , mean= ( mu0/sigma0^2 + sum(y)/k.sigma^2 ) / (1/sigma0^2 + length(y)/k.sigma^2) ,
sd=sqrt(1/(1/sigma0^2 + length(y)/k.sigma^2)) )

}
```

Sample mu from posterior

gibbs.r



samples of mu
sample.mu
Index
samples of sigma
sample.sigma
Index
posterior mu
Density
N = 19000 Bandwidth = 0.008632
posterior sigma
Density
N = 19000 Bandwidth = 0.006256

MCMC

- Next two days:
- OpenBUGS/JAGS/etc can automate defining and sampling the chain. Lets you focus on the structure of the model, instead of the details of the code.
- Own code: almost always (much!) faster than OpenBUGS
- OpenBUGS: Easier to use