# Lecture 7: Markov Chain Monte Carlo

Professor Alexander Franks

2/7/24

# Announcements

- Reading: Chapter 10 + 11

# Markov Chain Monte Carlo

- We want independent random samples, $\theta^{(s)}$ from $p(\theta \mid y_1, \ldots y_n)$
- Generally hard to find efficient Monte Carlo methods with independent samples
- Alternative, create a sequence of **correlated** samples that converge to the posterior distribution

# Discrete-state Markov Chains

- Let $\theta^{(t)} \in 1, 2, \ldots M$ be the state space for the stationary Mark
- A sequence is called a markov chain if
  $Pr(\theta^{(t+1)} = j \mid \theta^{(t)} = i, \theta^{(t-1)} = i_{t-1} \ldots \theta^{(1)} = i_1) = Pr(\theta^{(}$
  for all $t \geq 0$
- $q_{ij} = Pr(\theta^{(t+1)} \mid \theta^{(t)})$ is the transition probability from state $i$ t
- The *Markov property*: given the entire past history, $\theta^{(1)}, \ldots \theta^{(t)}$, t
  recent $\theta^{(t+1)}$ depends only on the immediate past, $\theta^{(t)}$

- The $M \times M$ matrix $Q = (q_{ij})$ is called the *transition matrix*
  of the Markov Chain

# The Transition Matrix

- The $M \times M$ matrix $Q = (q_{ij})$ is called the *transition matrix* of the Markov Chain

3-state example

$$Q = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

- The rows of the transition matrix sum to 1
- Note: $Q^n = (q_{ij}^{(n)})$ is is the probability of transitionining from $i$ to $j$ in $n$ steps
- A Markov Chain is *regular* $Q^n$ has strictly positive entries for some value of $n > 1$
- A Markov Chain is *irreducible* if for any two states $i$ and $j$ it is possible to go from $i$ to $j$ in a finite number of steps (with positive probability)

# The stationary distribution

- A regular, irreducible Markov chain has a *limiting probability distribution*
- What fraction of time does the Markov Chain spend in each state?
- The *stationary distribution* or limiting distribution describes the long-run behavior of the chain
  - The stationary distribution *does not* depend on where the chain starts

# The stationary distribution

- Let $\pi = (\pi_1, \ldots \pi_M)$ be a row vector of probabilities associated with each state, such that $\sum_{i=1}^{M} = \pi_i = 1$
- $\pi$ is stationary if $\pi Q = \pi$
  - If you sample from the stationary distribution and then transtion, the result is still distributed according to the stationary distribution

# Markov Chain Example

- Sociologists often study social mobility using a Markov chain.
- In this example, the state space if {low income, middle income, and high income} of families
- Let $\mathbf{Q}$ be the transition matrix from parents income to childrens income:

| | Lower | Middle | Upper |
|---|---|---|---|
| Lower | 0.40 | 0.50 | 0.10 |
| Middle | 0.05 | 0.70 | 0.25 |
| Upper | 0.05 | 0.50 | 0.45 |

# Multi-step Transition Probabilities

2-step transition probabilities

$$\mathbf{Q}^2 = \mathbf{Q} \times \mathbf{Q} = \begin{vmatrix} 0.1900 & 0.6000 & 0.2100 \\ 0.0675 & 0.6400 & 0.2925 \\ 0.0675 & 0.6000 & 0.3325 \end{vmatrix}$$

4-step transition probabilities

$$\mathbf{Q}^4 = \mathbf{Q}^2 \times \mathbf{Q}^2 = \begin{vmatrix} 0.0908 & 0.6240 & 0.2852 \\ 0.0758 & 0.6256 & 0.2986 \\ 0.0758 & 0.6240 & 0.3002 \end{vmatrix}$$

# Multi-step Transition Probabilities

4-step transition probabilities

$$\mathbf{Q}^4 = \mathbf{Q}^2 \times \mathbf{Q}^2 = \begin{vmatrix} 0.0908 & 0.6240 & 0.2852 \\ 0.0758 & 0.6256 & 0.2986 \\ 0.0758 & 0.6240 & 0.3002 \end{vmatrix}$$

8-step transition probabilities

$$\mathbf{Q}^8 = \mathbf{Q}^4 \times \mathbf{Q}^4 = \begin{vmatrix} 0.0772 & 0.6250 & 0.2978 \\ 0.0769 & 0.6250 & 0.2981 \\ 0.0769 & 0.6250 & 0.2981 \end{vmatrix}$$

# The stationary distribution

$$\mathbf{Q}^{\infty} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \pi = \begin{vmatrix} \pi_1 & \pi_2 & \pi_3 \\ \pi_1 & \pi_2 & \pi_3 \\ \pi_1 & \pi_2 & \pi_3 \end{vmatrix}$$

```r
Q <- matrix(c(0.4, 0.05, 0.05,
              0.5, 0.7, 0.5,
              0.1, 0.25, 0.45), ncol=3)
p <- eigen(t(Q))$vectors[, 1]
stationary_probs <- p/sum(p)
stationary_probs
```

```
[1] 0.07692308 0.62500000 0.29807692
```

```r
stationary_probs %*% Q
```

```
           [,1]  [,2]      [,3]
[1,] 0.07692308 0.625 0.2980769
```

# Markov Chain Monte Carlo

- Incredible idea: create a Markov Chain with the desired limiting distribution
  - Want the limiting distribution to be the posterior distribution
- Unlike the previous examples, we will mostly work with *infinite* state space
- Want $p(\theta^{(t+1)} \mid \theta^{(t)})$ to have limiting distribution $p(\theta \mid y)$
  - If we run the random walk for long enough, $\theta^{(t)}$ will be distributed approximately according to $p(\theta \mid y)$

# Detailed Balance

- Assume a Markov Chain $\theta_t$, with stationary distribution $\pi(s)$, and let

$$p(s, s') = Pr(\theta_{t+1} = s \mid \theta_t = s')$$

The chain is said to be reversible with respect to $\pi$ or to satisfy detailed balance with respect to $\pi$ if

$$\pi(s')p(s, s') = \pi(s)p(s', s) \quad \forall s, s'$$

## Detailed Balance

- If $\pi(s)$ and $p(s, s')$ satisfy detailed balance then $\pi$ is the unique stationary distribution of the Markov Chain.
- Goal: devise a transition kernel $p(s, s')$ for which $\pi(\theta = s \mid y) = \pi(s)$ satisfies detailed balance
- Detailed balance is a stronger condition than required for a stationary distribution

# Monte Carlo CLT

- Reminder: $\overline{\theta} = \sum_{s=1}^{S} \theta^{(s)} / S$ and $S$ is the number of samples.
- If the samples are independent,

$$p(\overline{\theta}) \rightarrow N\left(\theta, \frac{\mathrm{Var}(\theta \mid y_1, \ldots y_n)}{S}\right)$$

- If the samples are from a Markov Chain with $\theta_1$ from the stationary distribution, then

$$p(\overline{\theta}) \rightarrow N\left(\theta, \frac{\overline{\sigma}^2}{S}\right)$$

where $\overline{\sigma}^2 = Var(\theta \mid y_1, \ldots y_n) + 2\sum_{s=1}^{\infty} Cov(\theta_1, \theta_s)$

# The Independence Sampler

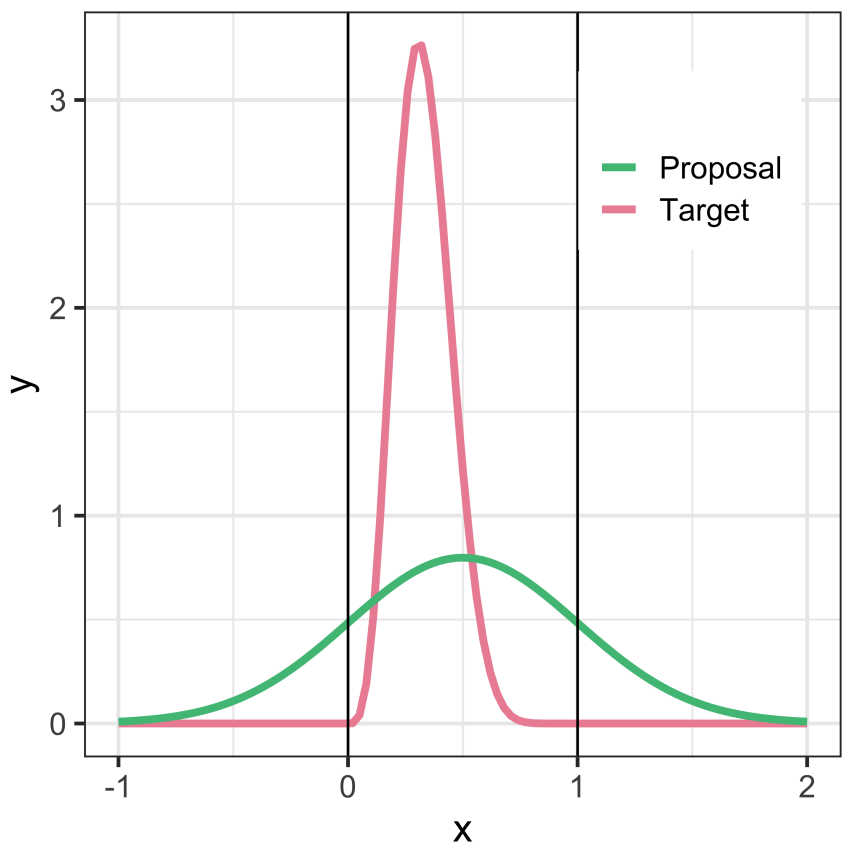
1. Initialize $\theta_0$ to be the starting point
2. Generate the candidate $\theta^*$ from a proposal distribution, $J(\theta^*)$ which is indepednent of $\theta_t$
3. Compute $r = \min(1, \frac{p(\theta^*|y)}{p(\theta_t|y)})$
4. Set $\theta_{t+1} \leftarrow \theta^*$ with probability $r$.

- Generate a uniform random number $u \sim Unif(0, 1)$
- If $u < r$ we accept $\theta^*$ as our next sample
- Else $\theta_{t+1} \leftarrow \theta_t$ (we do not update the sample this time)

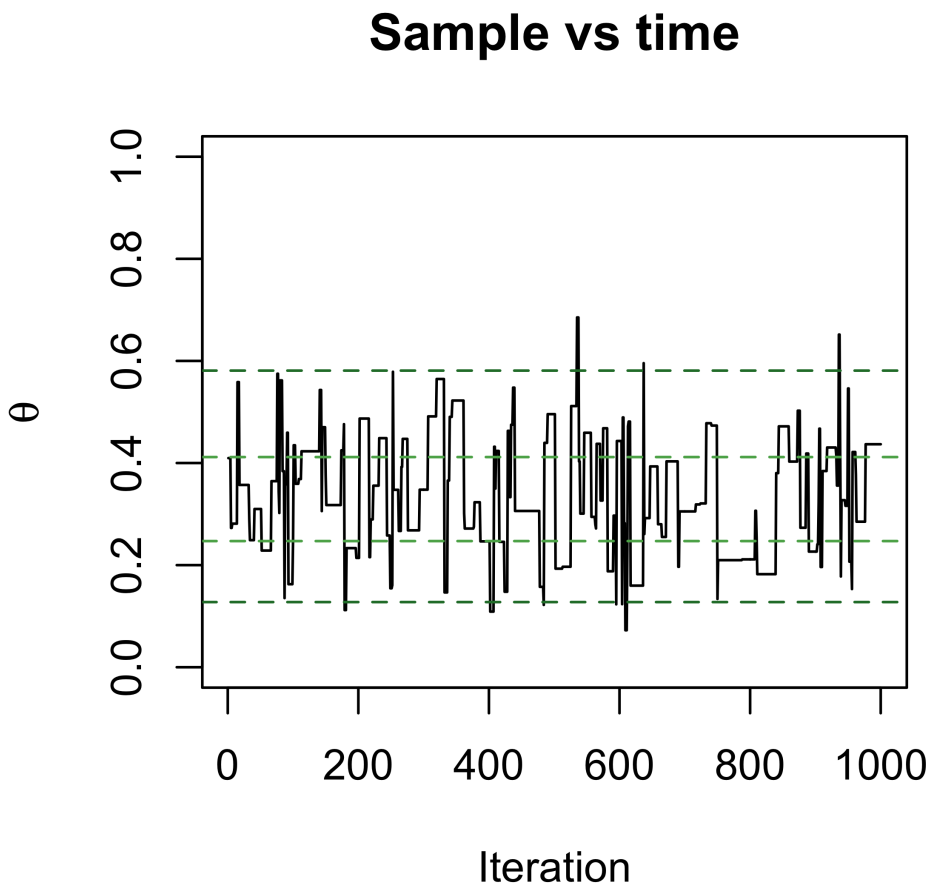
# Intuition

# An Example

- Let $P(\theta \mid y)$ be a Beta(5, 10) posterior distribution
- Propose from a distribution $J(\theta^*) \sim N(0.5, 1)$

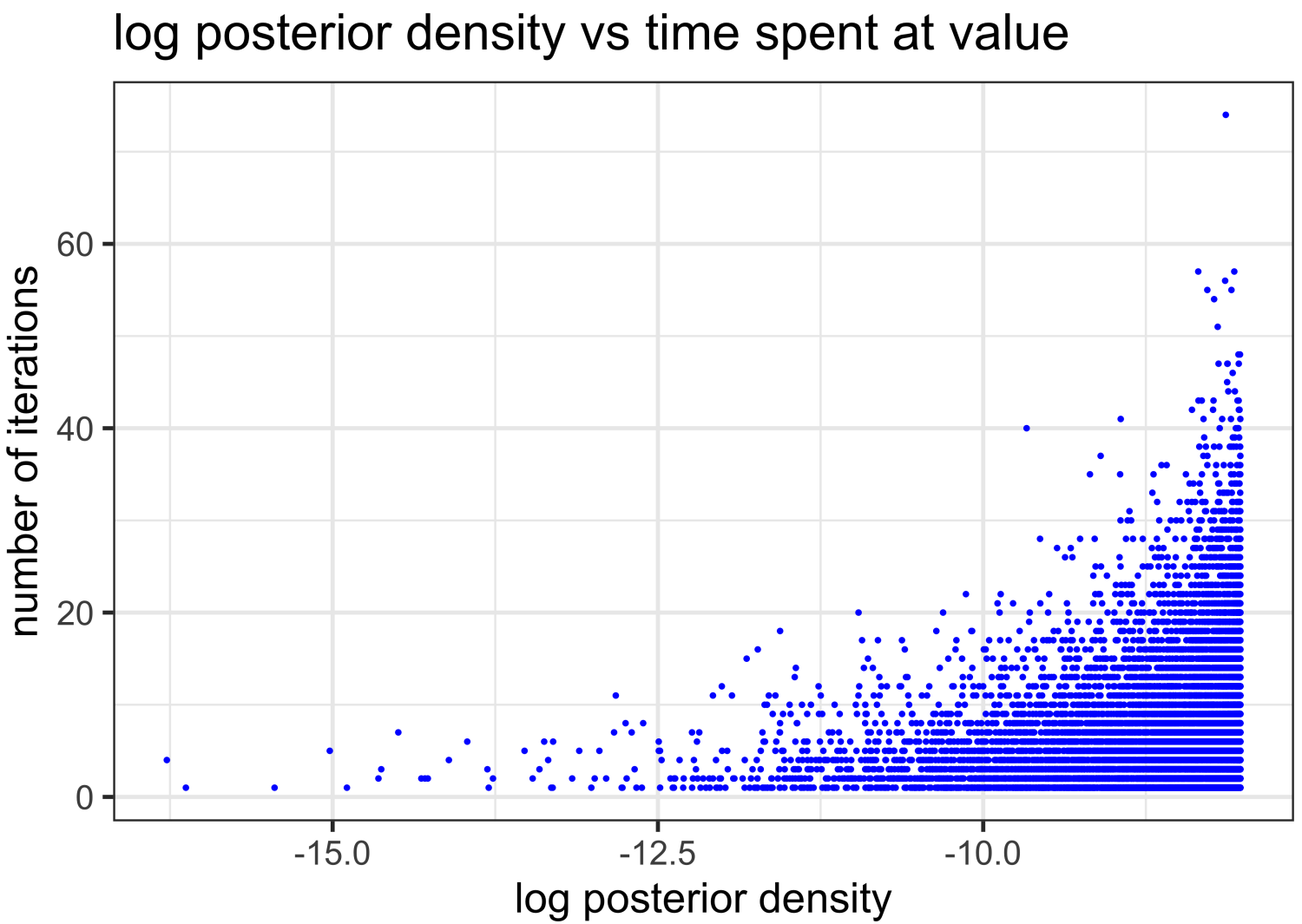


# Independence Sampler



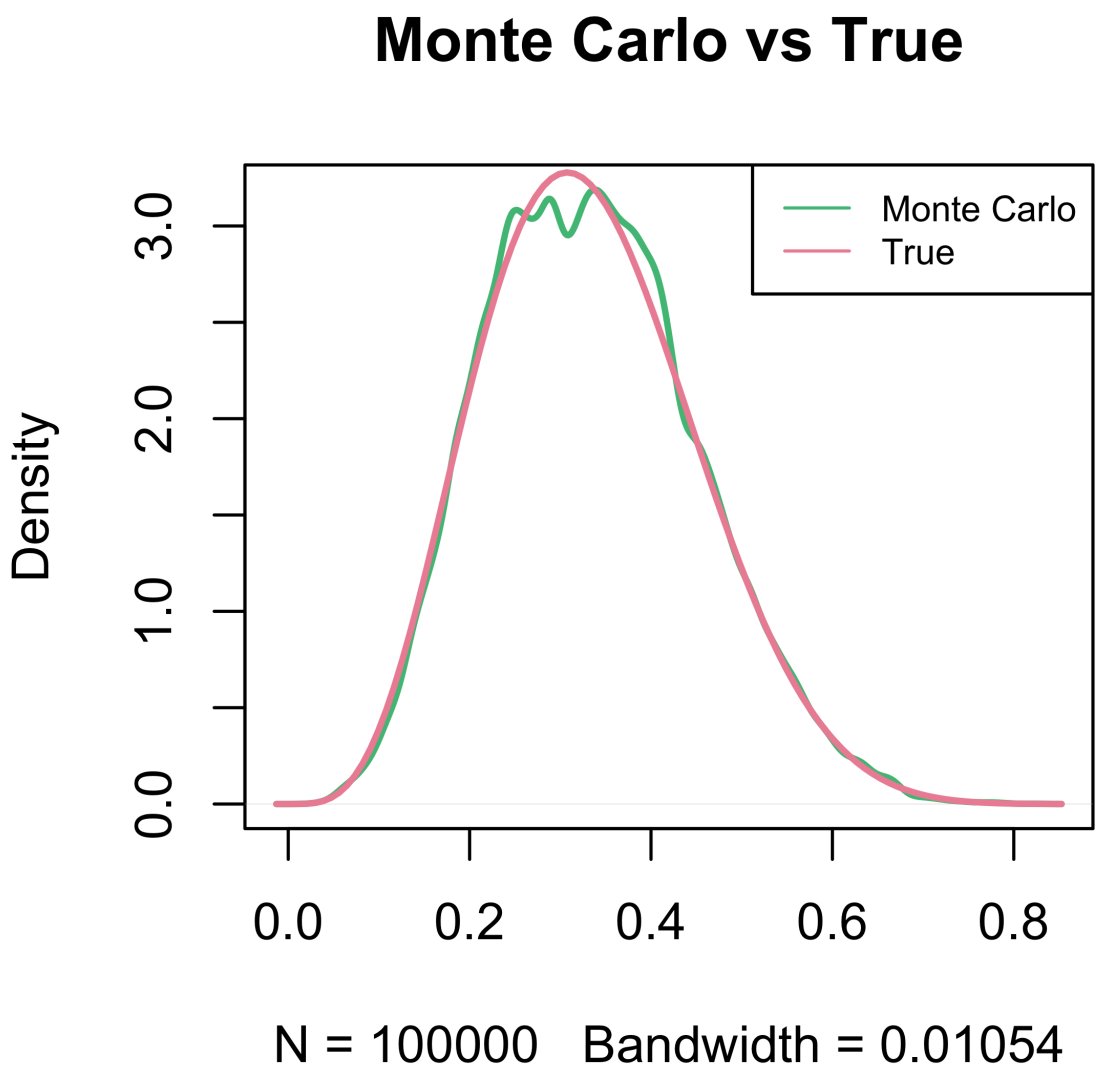

Note and source of confusion: samples are correlated over time for the “independence sampler”

# Weighting by waiting




Where did the sampler get stuck? Where does it quickly leave?

# Independence Sampler



# Metropolis Algorithm

- Let $p(\theta \mid y)$ be the posterior distribution
- The goal is to define a transition kernel, $p(\theta_{t+1} \mid \theta_t)$ such that $p(\theta \mid y)$ is the stationary distribution for the proposal

# The Metropolis Algorithm

- Generalize the independence sampler
  - $J(\theta^* \mid \theta_t) = J(\theta^*)$
- Metropolis: Allow the proposal distribution to depend on the most recent sample
  - $J(\theta^* \mid \theta_t)$, e.g. $\theta^* \sim N(\theta_t, 1)$
- Metropolis sampler: a “moving” proposal distribution

# The Metropolis Algorithm

# Metropolis-Hastings Algorithm

- The Metropolis-*Hastings* algorithm allows us to use non-symmetric proposals
- The Hastings correction is needed when $J(\theta^* \mid \theta_t) \neq J(\theta^t \mid \theta^*)$

$$r = \min(1, \frac{p(\theta^* \mid y)}{p(\theta_t \mid y)} \frac{J(\theta^t \mid \theta^*)}{J(\theta^* \mid \theta_t)})$$

- For symmetric proposals $\frac{J(\theta^t|\theta^*)}{J(\theta^*|\theta_t)} = 1$

# Detailed Balance Proof

# Aside: Arianna Rosenbluth

https://en.wikipedia.org/wiki/Arianna_W._Rosenbluth

# An Example

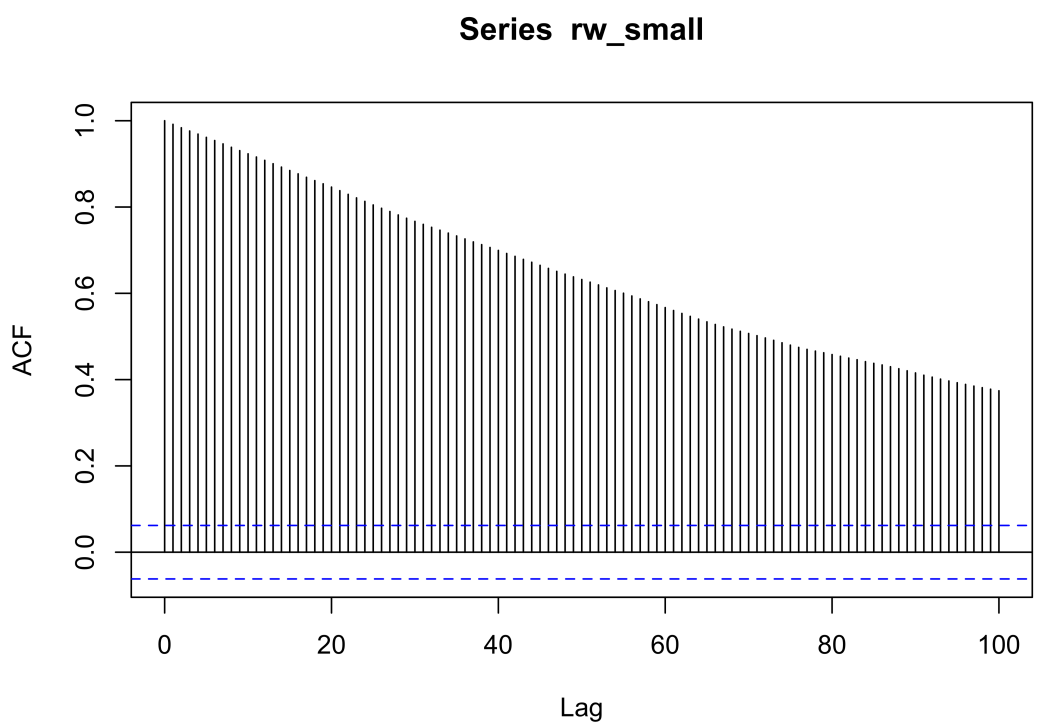
# Small variance proposal

$\tau^2 = 0.01$ ("small" jump)

```
1 acf(rw_small, lag=100)
```

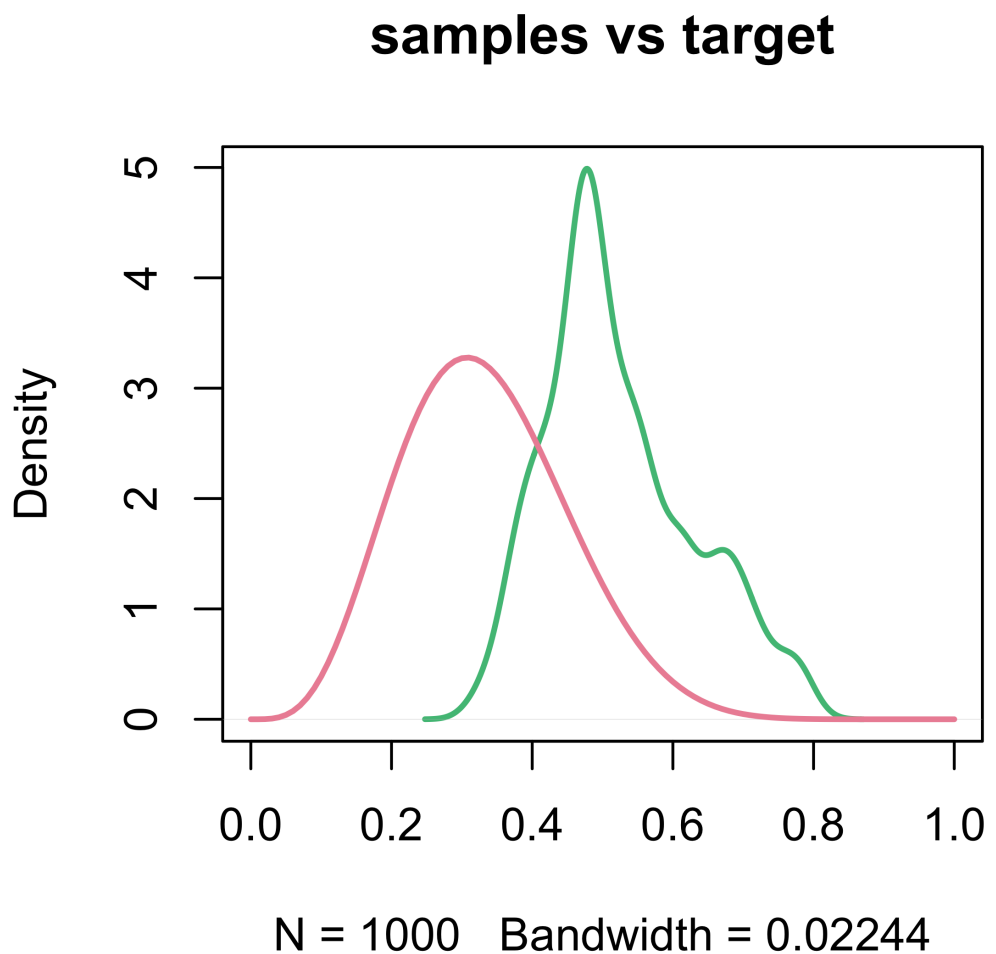
```
1 print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
2               effectiveSize(rw_small), rejectionRate(as.mcmc(rw_small))))
```

```
[1] "Effective sample size: 4.19, Rejection Rate: 0.05"
```



# Small variance proposal

$\tau^2 = 0.01$ (“small” jump)



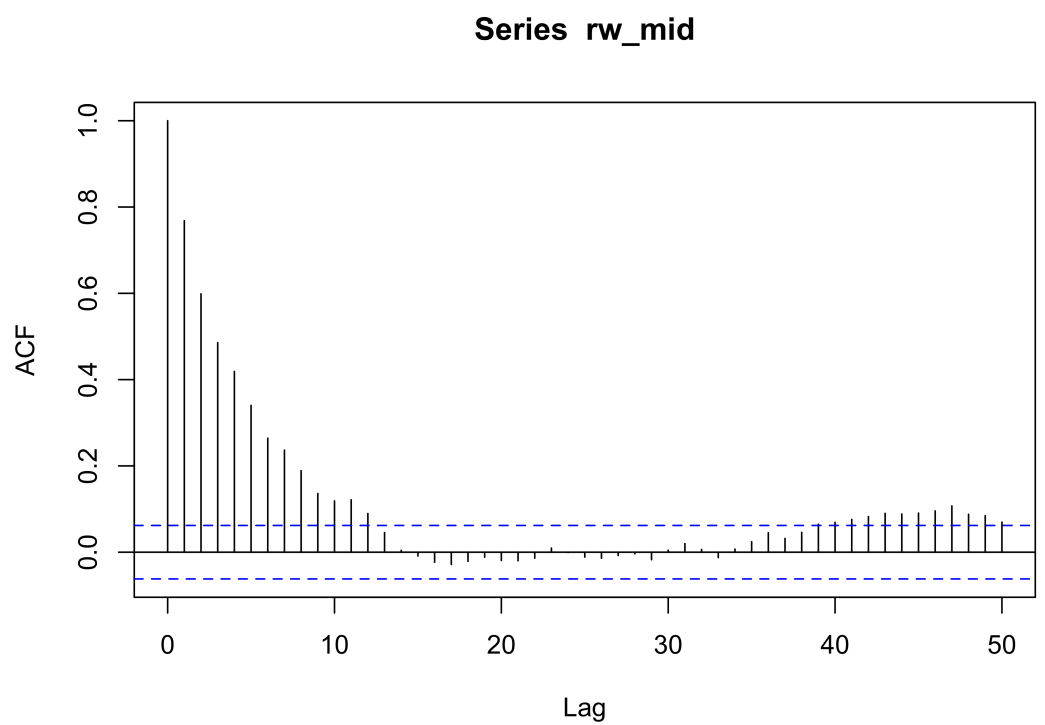
# The Metropolis Algorithm

## $\tau^2 = 0.1$ ("medium" proposal variance)

```
1 acf(rw_mid, lag=50)
```

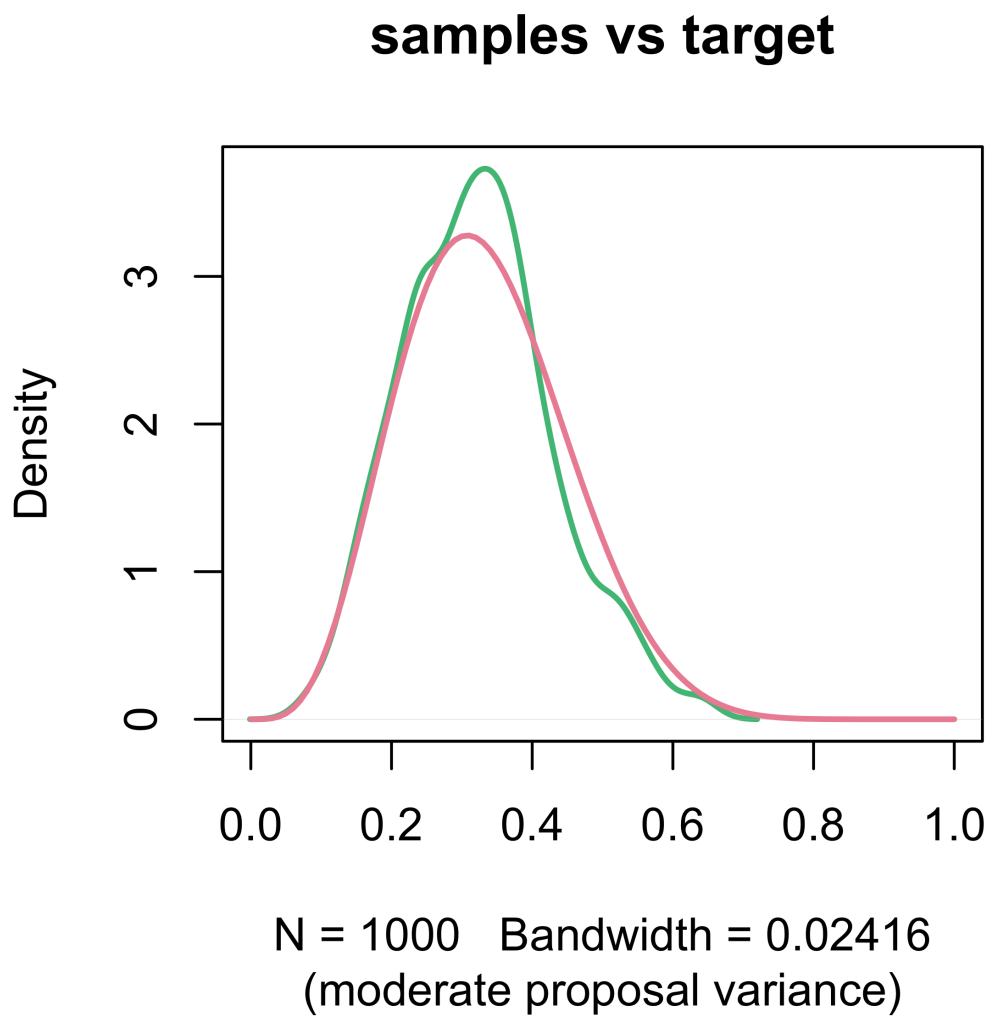
```
1 print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
2               effectiveSize(rw_mid), rejectionRate(as.mcmc(rw_mid))))
```

```
[1] "Effective sample size: 110.21, Rejection Rate: 0.26"
```



# The Metropolis Algorithm

$\tau^2 = 0.1$ (“medium” proposal variance)



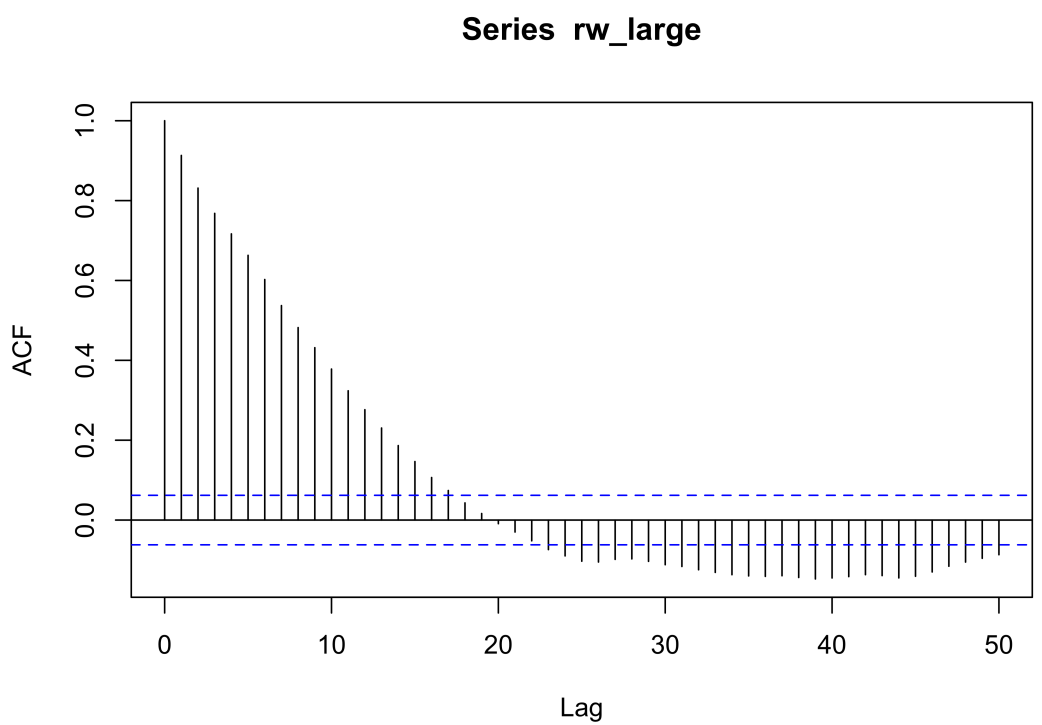
# The Metropolis Algorithm

## $\tau^2 = 2$ ("large" jump)

```
1 acf(rw_large, lag=50)
```

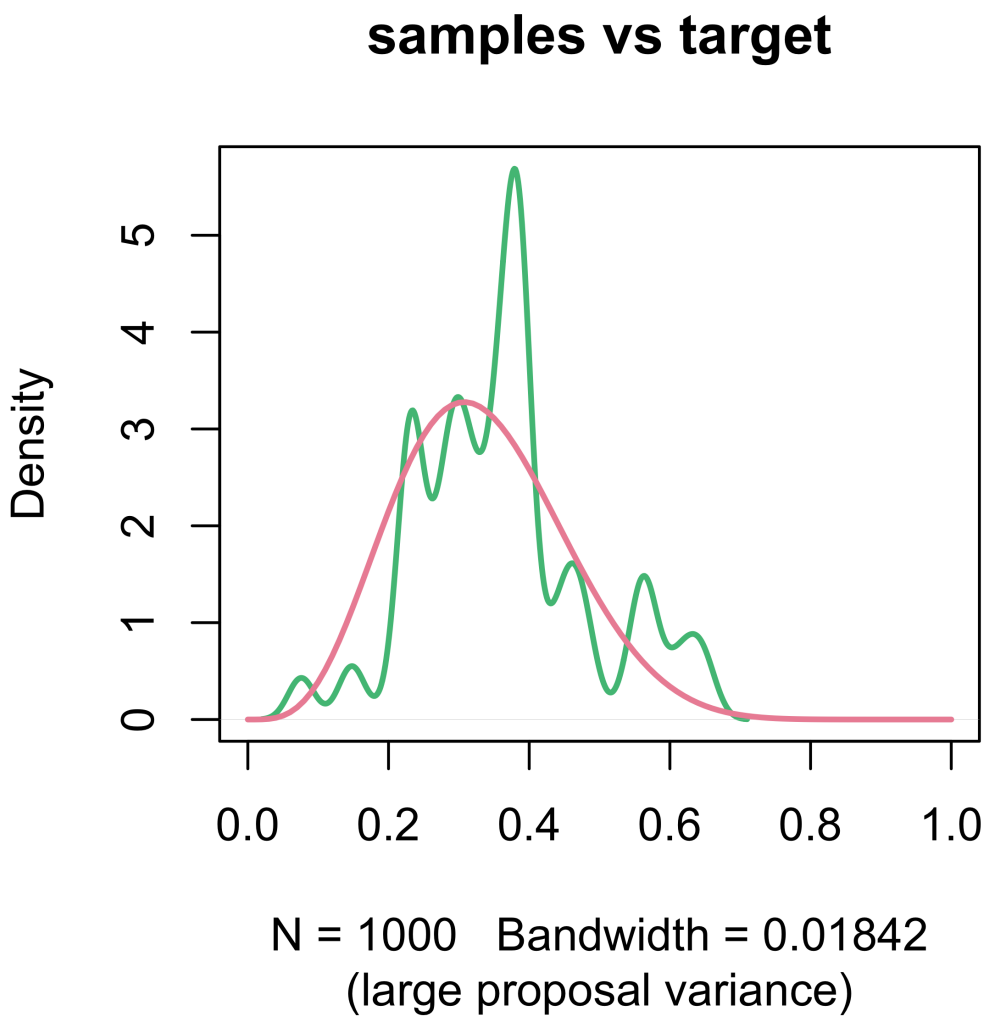
```
1 print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
2               effectiveSize(rw_large), rejectionRate(as.mcmc(rw_large))))
```

```
[1] "Effective sample size: 50.78, Rejection Rate: 0.94"
```

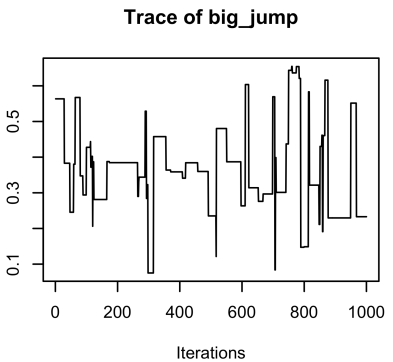


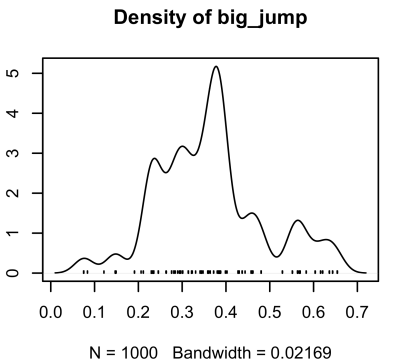
# The Metropolis Algorithm

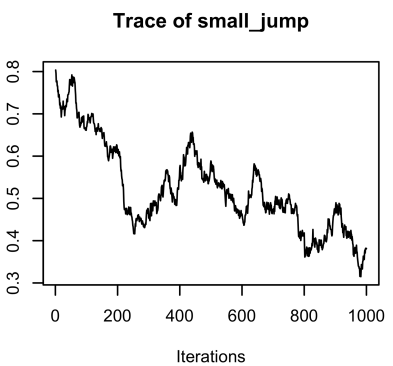
$\tau^2 = 2$ (“large” proposal variance)

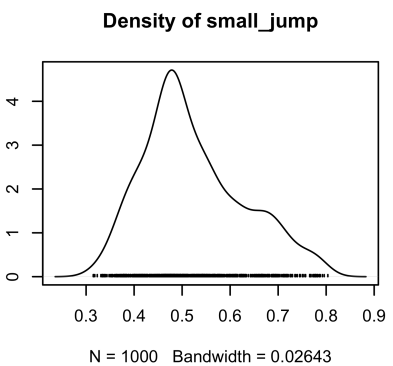


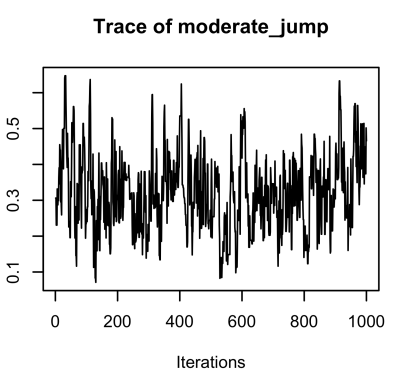
# Small, Moderate and Large Proposal variance

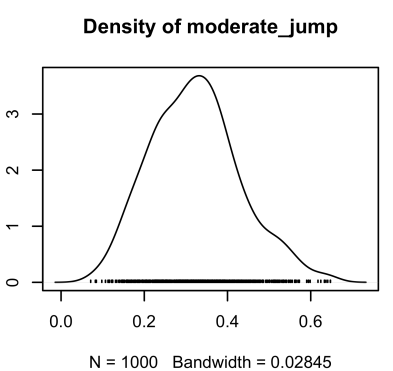


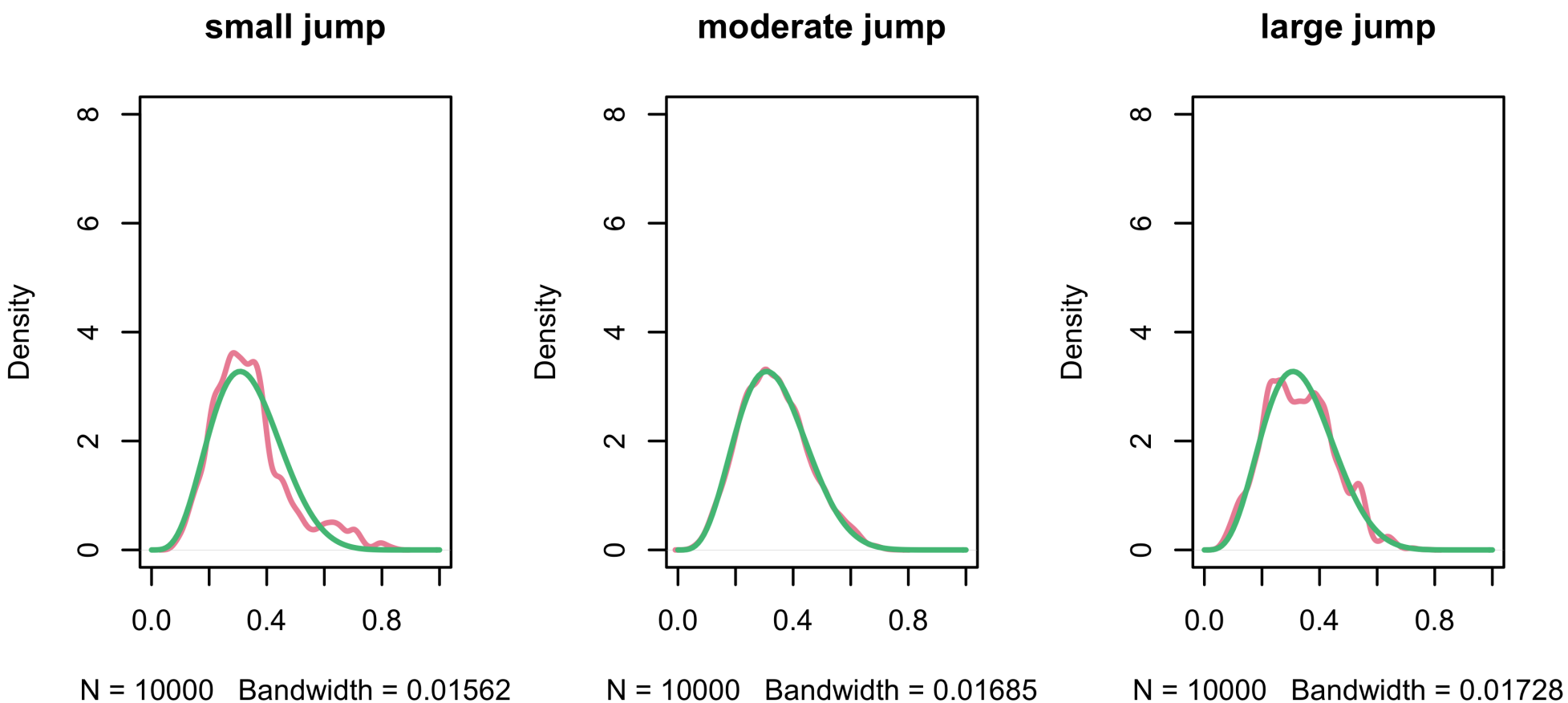



# 10,000 Samples

```
[1] "Effective sample size (small proposal variance): 13.75"
[1] "Effective sample size (medium proposal variance): 983.72"
[1] "Effective sample size (large proposal variance): 448.31"
```



# MCMC diagnostics

- Diagnose the chain(s) performance by examining:
  - Rejection rate
  - Autocorrelation
  - Effective sample size
  - Rhat

# MCMC diagnostics

- **Rejection rate**: if rejection rate is high, the proposal density is proposing “too far away” from the current sample
- Means we are often keeping the previous sample
- Sampler is “sticky”. Traceplots look like cityscapes.

# MCMC diagnostics

- **Autocorrelation**: if samples are highly correlated, the proposal density is proposing “too close” to the current sample
  - Highly correlated implies the Markov chain is mixing slowly
- The mixing time of a Markov chain is the time until the Markov chain is “close” to its limiting distribution.

# MCMC diagnostics

- **Effective sample size:** correlated chain of samples is equivalent to this number of independent samples
  - High rejection rate implies a lot of duplicate samples so effective size is smaller than number of iterations
  - High autocorrelation means neighboring samples are very similar (even if not exactly the same)

Reminder: $\overline{\sigma}^2 = Var(\theta \mid y_1, \ldots y_n) + 2\sum_{s=1}^{\infty} Cov(\theta_1, \theta_s)$

$$n_{\text{eff}} = \frac{mn}{1+2\sum_{t=1}^{\infty}\rho_t}$$

# Metropolis Algorithm

- In the Beta example, the accept ratio, $\min(1, \frac{p(\theta^*|y)}{p(\theta_t|y)})$ is zero when $\theta^* > 1$ or $\theta^* < 0$
- If $\tau^2$ (proposal variance) too large, you will often reject your proposal
  - Proposing far from your current location may move you too far out of the high density areas
  - This makes for a "sticky" chain (stay at current sample for a long time)
- $\tau^2$ too small, the chain explore the parameter space slowly

# Rhat

- Important to assess convergence by checking whether the chains “mix”
- Visual check: look at traceplots of multi-chains
- Quantitative check: look at within- and between- chain variation

# Rhat

- Assume $m$ chains and $n$ draws per chain
- Let $B = \frac{1}{m-1} \sum_{j=1}^{m} \left( \bar{\theta}_{\cdot j} - \bar{\theta}_{\cdot \cdot} \right)^2$ be the between chain variance
- Let $W = \frac{1}{m} \sum_{j=1}^{m} s_j^2$ be the average within chain variance

Then we define $\widehat{\mathrm{var}}^{+}(\psi \mid y) = \frac{n-1}{n} W + B$ and the Rhat as $\widehat{R} = \sqrt{\frac{\widehat{\mathrm{var}}^{+}(\psi|y)}{W}} = \sqrt{1 + B/W}$

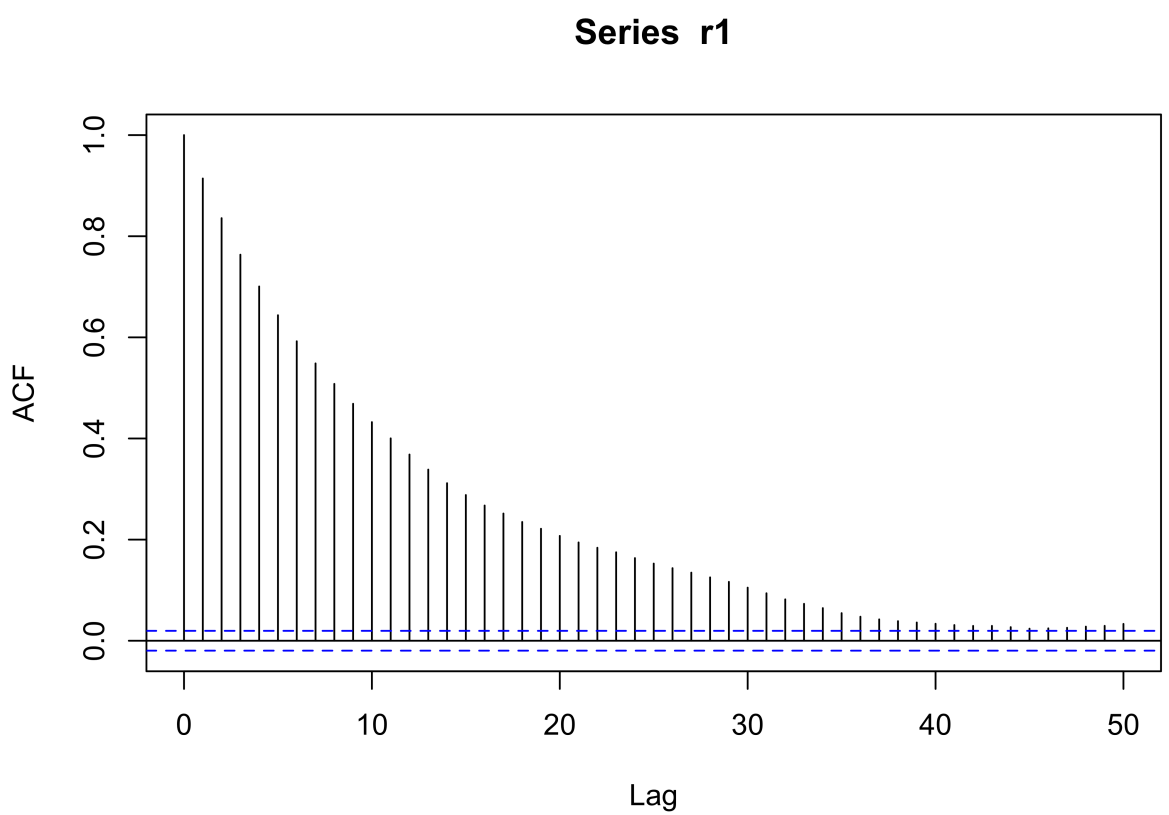
# Demo: Metropolis-Hastings

# The Metropolis Algorithm

```
1 acf(r1, lag=50)
```

```
1 print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
2               effectiveSize(r1), rejectionRate(as.mcmc(r1))))
```

```
[1] "Effective sample size: 448.31, Rejection Rate: 0.92"
```



# Computational Considerations

- For very small values of $p(\theta \mid y)$, numerical underflow is a problem
- Can resolve this by working on the log scale

```
1 dnorm(1000) / dnorm(1001)
```

```
[1] NaN
```

```
1 dnorm(1000, log=TRUE) - dnorm(1001, log=TRUE)
```

```
[1] 1000.5
```

- Compute $l = \min(0, \log(p(\theta^* \mid y)) - \log(p(\theta_t \mid y)))$
- If $\log(u) < l$ we accept $\theta^*$ as our next point

# MCMC in the normal model

- Modeling wing length of different specifies of midge (small, two-winged flies)
- Reminder: $Y_i \sim N(\mu, \sigma^2)$
- $P(\mu \mid \sigma^2), \mu \sim N(\mu_0, \frac{\sigma^2}{\kappa_0})$
- $P(\sigma^2), \sigma^2 \sim \text{Inv-Gamma}(\nu_0/2, \nu_0/2\sigma_0^2)$
- Parameterize in terms of $\theta = (\mu, \log(\sigma))$? Why parameterize this way?
- Let $J(\theta^* \mid \theta_t) = N\left(\begin{pmatrix} \mu^{(t)} \\ \log(\sigma^{(t)}) \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\right)$

# MCMC in the normal model

Example: midge wing length

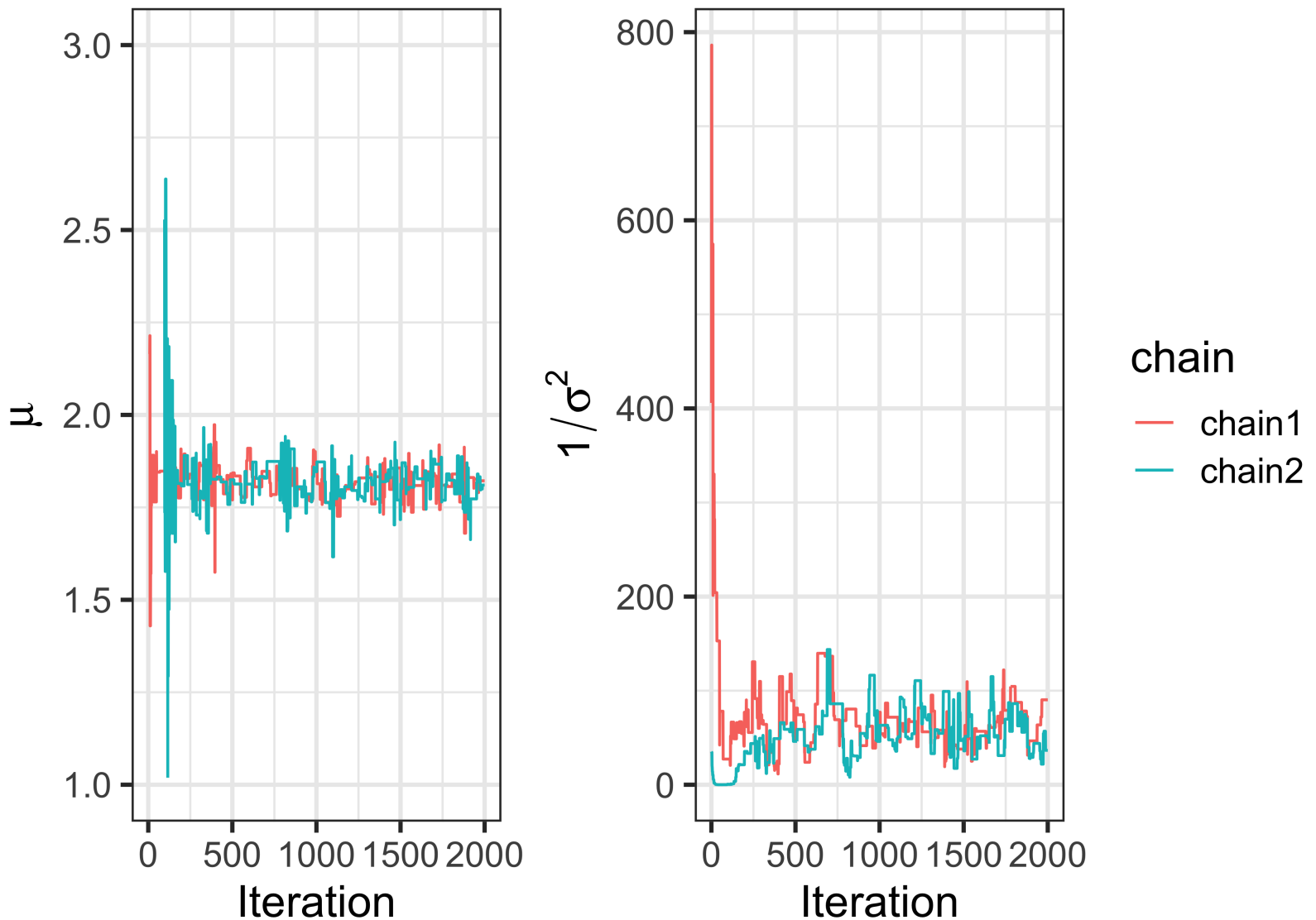
- Modeling wing length of different specifies of midge (small, two-winged flies)
- From prior studies: mean wing length close to 1.9mm with sd close to 0.1mm
- $\mu_0 = 1.9, \sigma_0^2 = 0.01$
- Choose $\kappa_0 = \nu_0 = 1$
- $(\bar{y}, s^2) = (1.804, 0.0169)$
- We will run 2 separate chains at different starting locations

# MCMC in the normal model

```r
#Random walk in theta space.
rw_metrop_multi <- function(theta_0, burnin, maxit, taus=c(0.25, 0.25)){

    ## Accept/reject thresholds. We will do these in log-space, note that
    thresh <- -rexp(burnin + maxit)
    for(i in 1:(burnin + maxit)){

        ## Propose new mu and sigma
        theta_p <- c(theta_t[1] + mu_props[i], theta_t[2] + log_sigma_props

        # Log-rejection ratio
        l_metrop <- log_posterior(theta_p) -
          log_posterior(theta_t)
        ## Accept/reject step
        theta_t <- if(l_metrop > thresh[i]) theta_p else theta_t

        ## Save the draw
        theta_out[i, ] <- theta_t
```

# Demo: MCMC for the normal model

# MCMC for multivariate distributions

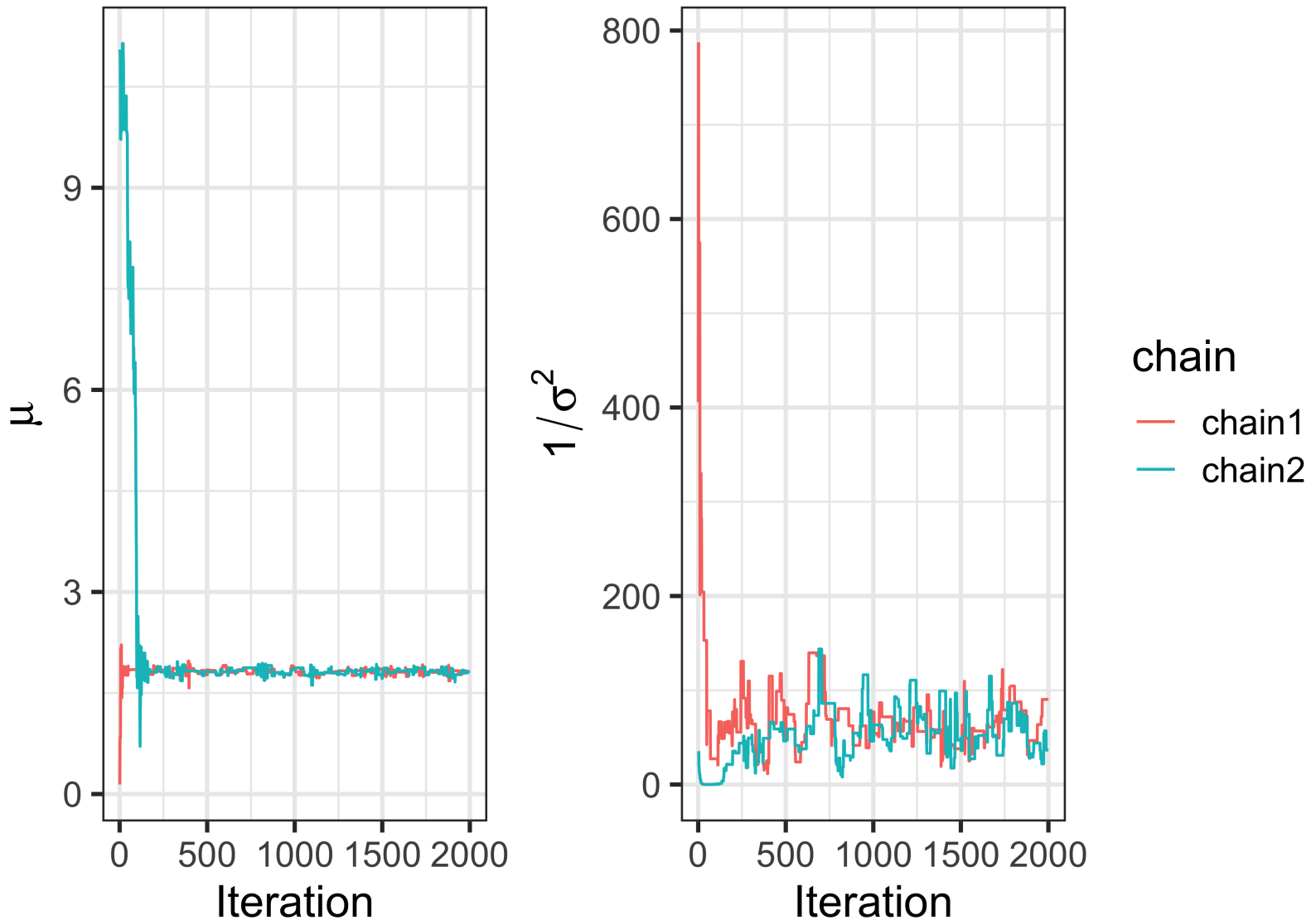


# MCMC for multivariate distributions

- We will run 2 separate chains at different starting locations
- For chain 1: $J(\theta^* \mid \theta_t) = N\left(\begin{pmatrix} \mu^{(t)} \\ \log(\sigma^{(t)}) \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$
- For chain 2:

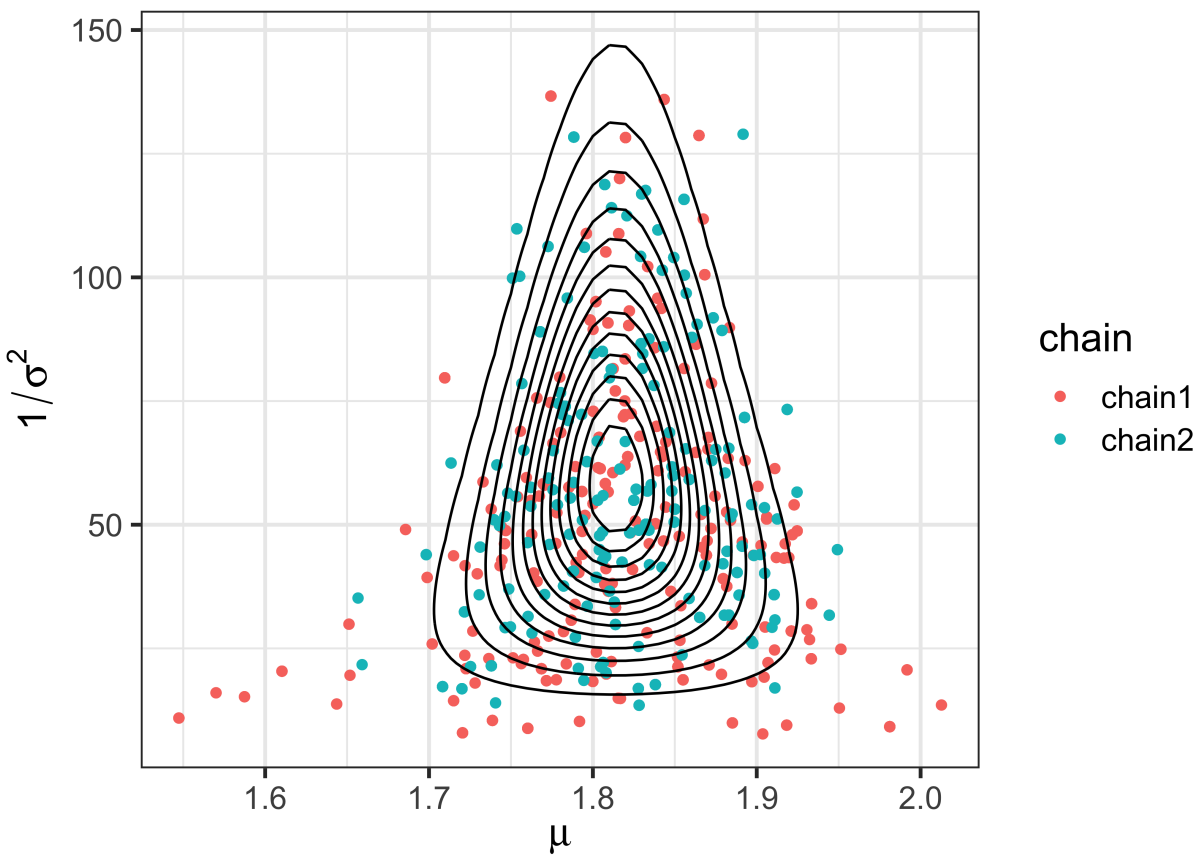
$$J(\theta^* \mid \theta_t) = N\left(\begin{pmatrix} \mu^{(t)} \\ \log(\sigma^{(t)}) \end{pmatrix}, \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}\right)$$

# MCMC for multivariate distributions



# MCMC for multivariate distributions

```
# A tibble: 2 × 3
  chain  EF_mu EF_prec
  <fct>  <dbl>   <dbl>
1 chain1 439.     72.6
2 chain2  34.4    45.9
```



# Metropolis Sampling

Try out the Metropolis algorithm at:

https://chi-feng.github.io/mcmc-demo/app.html

- Choose “Random Walk MH” algorithm
- Experiment by sampling from different target distribution
- Try different proposal variances by changing “Proposal $\sigma$”

# Difficulties with MCMC

# Hamiltonian Monte Carlo

- A clever parameter expansion strategy for MCMC
- Imagine a particle on a frictionless surface. The location of the marble is the current value of $\theta_t$
- The negative posterior density is the “height” of the surface
- Each iteration we flick the marble with some velocity in a random direction. This imparts the marble which momentum (an additional parameters)
- Important caveat: cannot be applied with discrete parameters!

# HMC

- For Metropolis-Hastings we only need to be able to evaluate the posterior at each location
- For HMC we need the gradient (derivative) of the posterior as well
- In physics the Hamiltonian is the sum of the kinetic energies, plus the potential energy of the particles
  - As our proposal, we randomly sample a momentum for the particle and update its position accordingly
  - Can think of HMC as the MH algorithm with a very clever jumping/proposal rule

# HMC

- For each parameter $\theta_j$ in your target space, HCM adds a 'momentum' variable $\phi_j$
- Both $\theta$ and $\phi$ are updated together in a new Metropolis algorithm
- $p(\theta, \phi \mid y) = p(\phi)p(\theta \mid y)$
- In additional to the log posterior, HMC also requires the **gradient** of the log-posterior density.
- $\phi$ is usually given a $N_d(0, M)$ distribution, where $M$ is typically chosen to be diagonal

# HMC

Equations of motion of the Hamiltonian, $H(\theta, \phi)$, described by partial derivatives:

$$\frac{d\theta_i}{dt} = \frac{\partial H}{\partial \phi_i}$$
$$\frac{d\phi_i}{dt} = -\frac{\partial H}{\partial \theta_i}$$

where

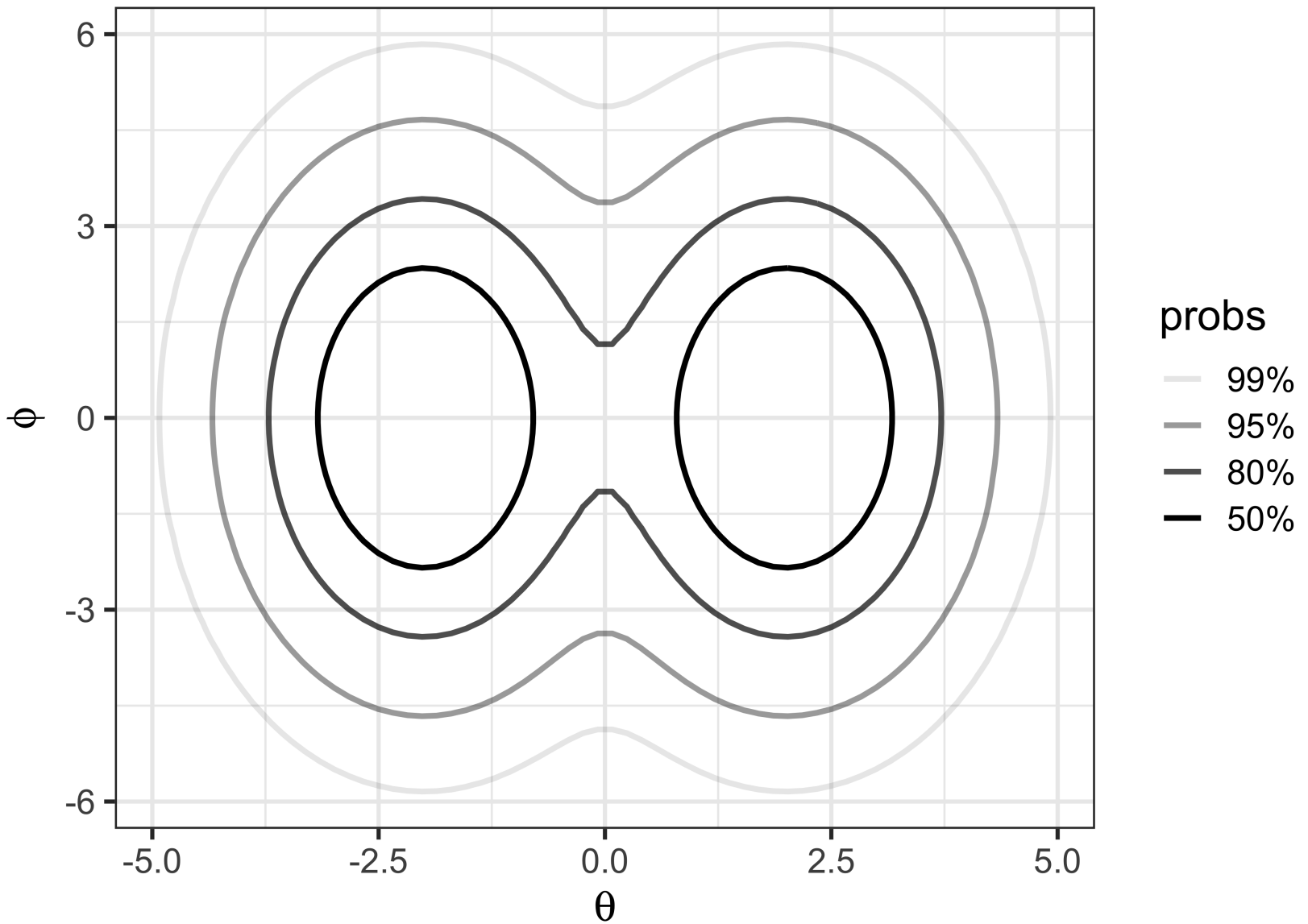
$$H(\theta, \phi) = U(\theta) + K(\phi)$$

- $U(\theta)$ is the potential energy ($-logp(\theta \mid y)$)
- $K(\phi)$ is the kinetic energy, usually $K(\phi) = \phi^T M^{-1} \phi / 2$

# Some key Properties

- Hamiltonian Dynamics are reversible
- Dynamics keeps the Hamiltonian invariant, i.e. $dH/dt = 0$.
  - Acceptance probability of 1 if we can simulate dynamics exactly!

# HMC Intuition



# HMC

- Initialize parameters $\theta_0$
- For t in 1:T:
  1. Draw $\phi_t \sim N_d(0, M)$
  2. Update $\theta_t$ according to the Hamiltonian dynamics for a fixed amount of time
  3. Take the $\theta_t$ as the new sample

Problem: usually the Hamiltonian dynamics can't be solved exactly

# The Leapfrog Algorithm

- If we could simulate the Hamiltonian dynamics exactly then we'd accept every proposed sample
- We can't. Numericaly approximate the dynamics with the "Leapfrog Algorithm"
- For the HMC we have to fix $L$ and $\epsilon$. $L\epsilon$ is the length of time that we simulate the dynamics.
- For numerical approximation, repeat L times:

  1. $\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}$
  2. $\theta \leftarrow \theta + \epsilon M^{-1}\phi$
  3. $\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}$

# The Full HMC Algorithm

Initialize $\theta_0$ and for each iteration:

1. Set $\theta^* \leftarrow \theta_{t-1}$ and set $\phi^* \leftarrow \phi^{t-1} \sim N_d(0, M)$
2. For L iterations:
    1. $\phi^* \leftarrow \phi^* + \frac{1}{2}\epsilon\frac{d \log p(\theta^*|y)}{d\theta}$
    2. $\theta^* \leftarrow \theta^* + \epsilon M^{-1}\phi^*$
    3. $\phi^* \leftarrow \phi^* + \frac{1}{2}\epsilon\frac{d \log p(\theta^*|y)}{d\theta}$
3. $\theta_t \leftarrow \theta^*$ with probability $r = \min(1, \frac{p(\theta^*|y)p(\phi^*)}{p(\theta_{t-1}|y)p(\phi_{t-1})})$

# HMC

Try out HMC at:

https://chi-feng.github.io/mcmc-demo/app.html

- Choose “HamiltonianMC” algorithm
- Experiment by sampling from different target distributions
- Compare to the Random Walk Metropolis

# Paramter tuning

HMC can be tuned in three places:

1. the probability distribution for the momentum variables $\phi$ (which, in our implementation requires specifying the diagonal elements of a covariance matrix, that is, a scale parameter for each of the d dimensions of the parameter vector),
2. the scaling factor $\epsilon$ of the leapfrog steps,
3. the number of leapfrog steps L per iteration.

# No-U-Turn Sampling and Stan

# Stan

## Tips, tricks and diagnostics

- $\hat{R}$ statistics
- Divergences
- Shinystan

# Divergences

- Divergences occur when the simulated Hamiltonian trajectory departs from the true trajectory (leapfrog isn't accurate)
- Sometimes it helps to reduce the step size or increase acceptance rate (`adapt_delta`)
- Reparameterization can be very helpful!

# Example: Neal's Funnel

- Exemplifies the difficulties of sampling from some hierarchical models
- The probability contours are shaped like ten-dimensional funnels.

# Example: Neal's Funnel

```
parameters {
  real y;
  vector[9] x;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

# Neal's Funnel

```
Running MCMC with 4 sequential chains...

Chain 1 finished in 0.1 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.1 seconds.
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.1 seconds.
Total execution time: 0.6 seconds.
```

| variable | mean | median | sd | mad | q5 | q95 | rhat | ess_bulk | ess_tail |
|---|---|---|---|---|---|---|---|---|---|
| lp__ | -4.90 | -3.80 | 11.72 | 11.76 | -26.31 | 11.77 | 1.36 | 9 | NA |
| y | 0.07 | -0.15 | 2.45 | 2.63 | -3.37 | 4.53 | 1.36 | 9 | 5 |
| x[1] | -0.09 | -0.01 | 5.32 | 0.78 | -4.43 | 4.09 | 1.16 | 5080 | 421 |
| x[2] | -0.11 | -0.07 | 4.15 | 0.85 | -4.69 | 3.76 | 1.26 | 4585 | 478 |
| x[3] | 0.01 | 0.01 | 4.71 | 0.91 | -4.50 | 4.57 | 1.13 | 4244 | 356 |
| x[4] | 0.03 | 0.01 | 5.62 | 0.89 | -4.37 | 4.58 | 1.14 | 4588 | 420 |
| x[5] | 0.04 | 0.07 | 5.27 | 0.81 | -4.49 | 4.40 | 1.30 | 4561 | 447 |
| x[6] | -0.13 | -0.01 | 4.69 | 0.83 | -4.57 | 4.22 | 1.22 | 4497 | 441 |
| x[7] | -0.07 | 0.01 | 4.57 | 0.79 | -4.22 | 4.12 | 1.11 | 3171 | 337 |
| x[8] | -0.03 | 0.05 | 5.29 | 0.83 | -4.66 | 4.41 | 1.26 | 4920 | 423 |

# Neal's Funnel

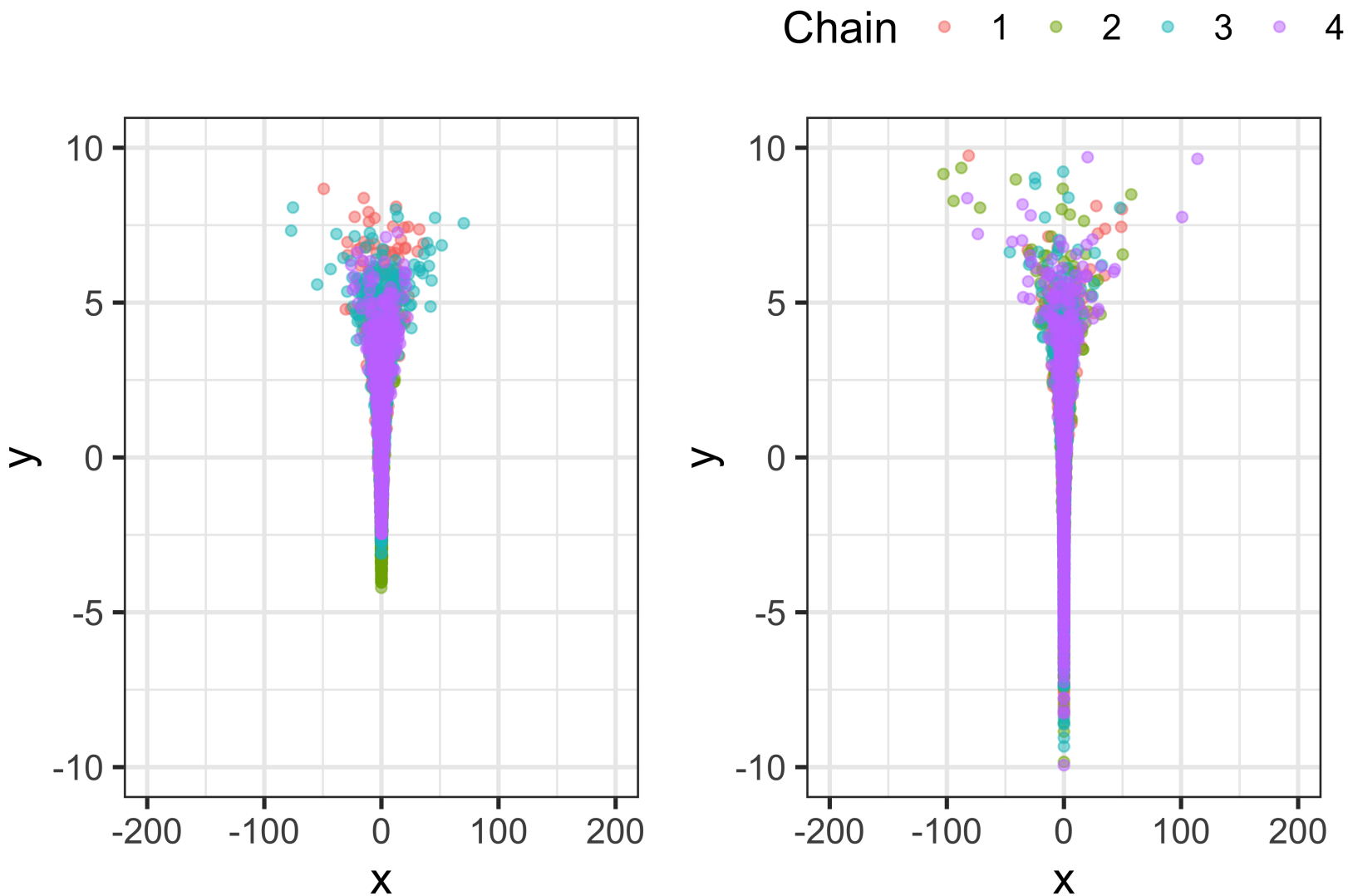
```
parameters {
  real y_raw;
  vector[9] x_raw;
}
transformed parameters {
  real y;
  vector[9] x;

  y = 3.0 * y_raw;
  x = exp(y/2) * x_raw;
}
model {
  y_raw ~ std_normal(); // implies y ~ normal(0, 3)
  x_raw ~ std_normal(); // implies x ~ normal(0, exp(y/2))
}
```

# Neal's Funnel

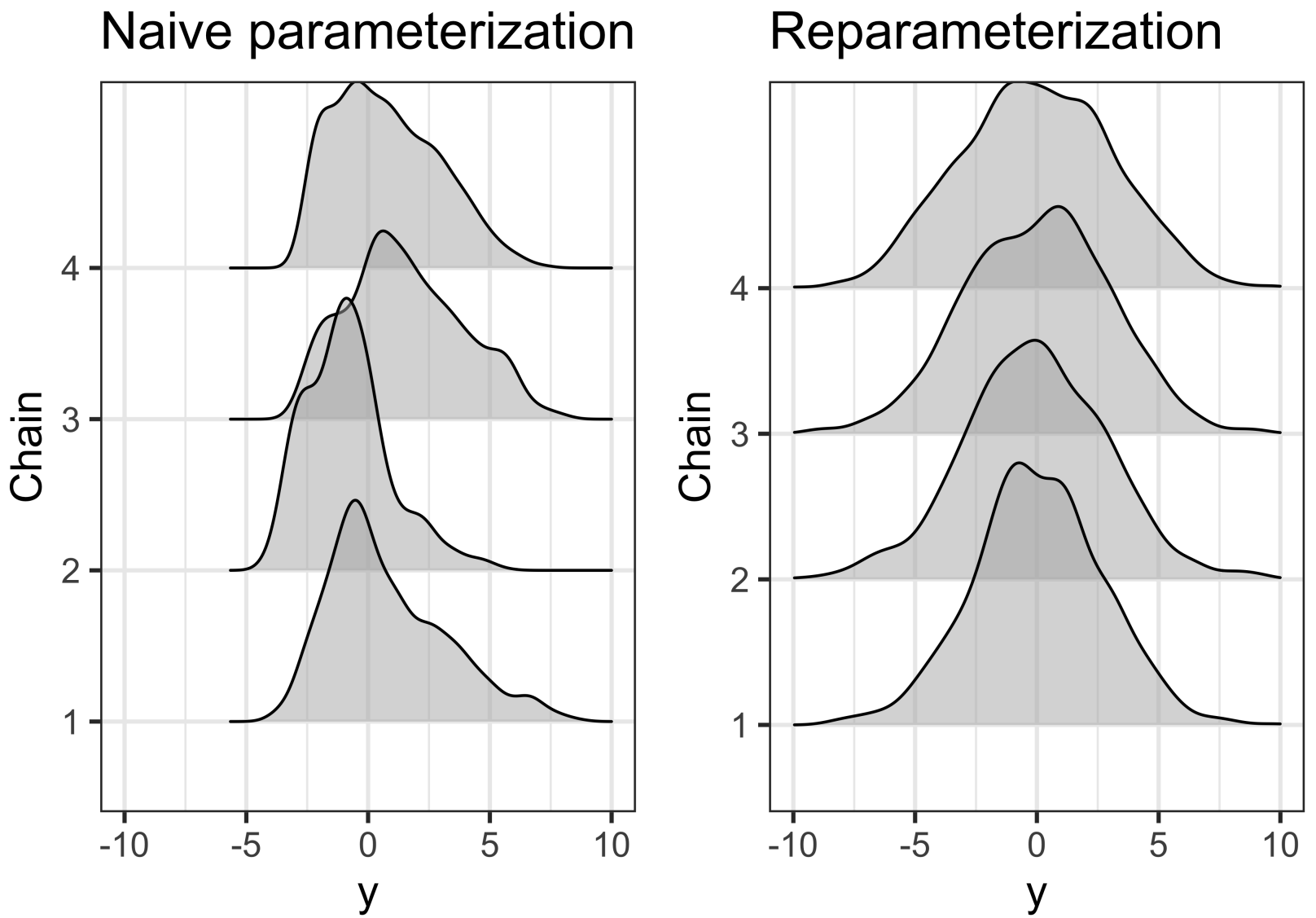
```
Running MCMC with 4 sequential chains...

Chain 1 finished in 0.1 seconds.
Chain 2 finished in 0.1 seconds.
Chain 3 finished in 0.1 seconds.
Chain 4 finished in 0.1 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.1 seconds.
Total execution time: 0.6 seconds.
```

```
Running MCMC with 4 sequential chains...

Chain 1 finished in 0.0 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.0 seconds.
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.5 seconds.
```

# Neal's Funnel

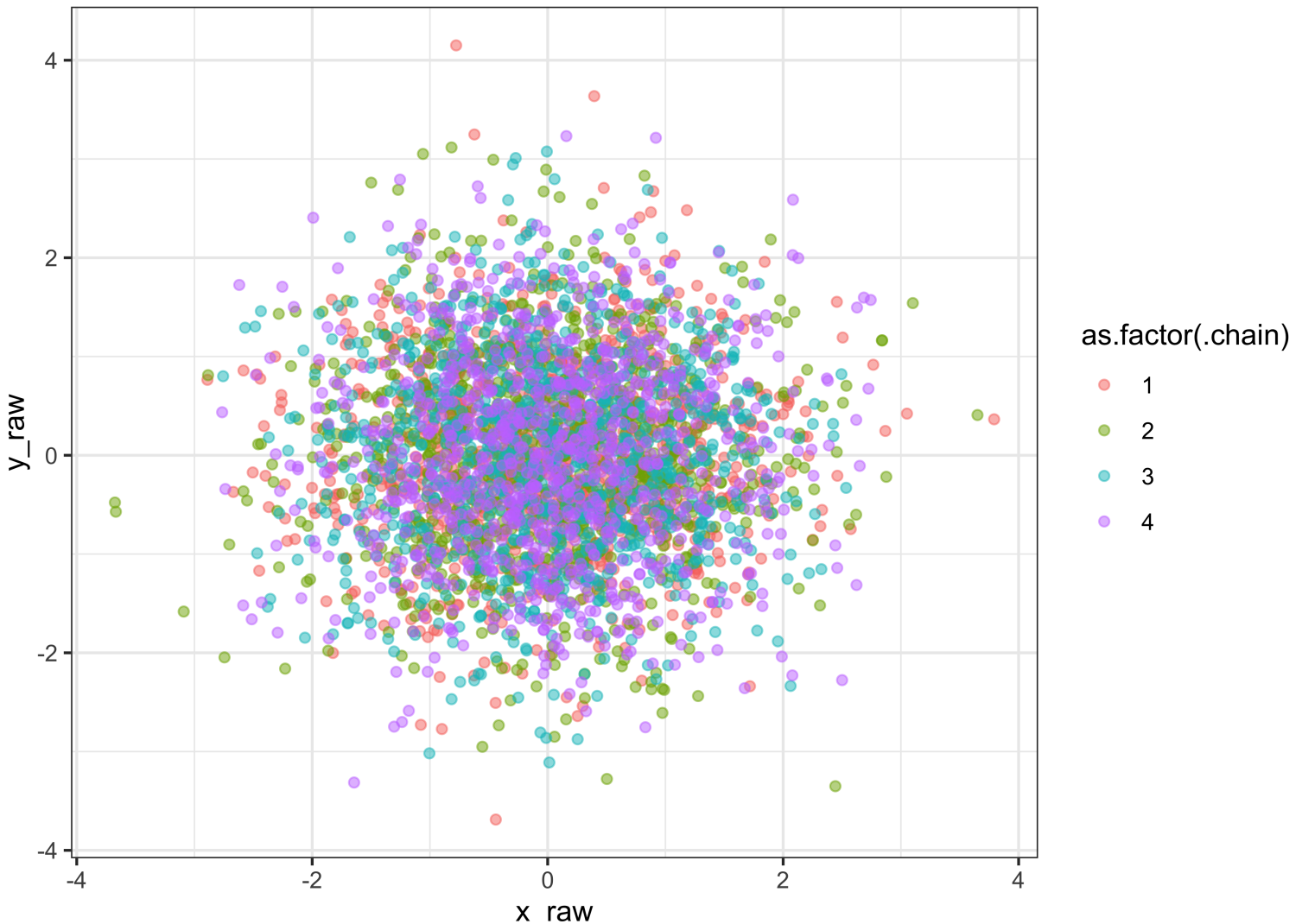


# Neal's Funnel



# Neal's Funnel

Sampling in the reparameterzied space is *much* easier!!



# The Gibbs Sampler

- Suppose that the parameter vector $\theta$ can be divided into $d$ subvectors.
- For sample $s = 1, \ldots, S$:
  - For $j \in 1, \ldots, d$
    - Draw a value from the conditional distribution of $\theta_j$ given all the other parameters,

$$\theta_j^s \sim p(\theta_j | \theta_{-j}^{(s-1)}, y),$$

where $\theta_{-j}$ consists of updated parameters for all parameters preceding $j$ and the previous iteration's values for all succeeding parameters,

$$\theta_{-j}^{s-1} = \left(\theta_1^s, \ldots, \theta_{s-1}^t, \theta_{j-1}^{s-1}, \ldots, \theta_d^{s-1}\right).$$

# The Gibbs Sampler

- To identify the full conditional distributions:
  1. Start with the full posterior
  2. For each parameter, $\theta_j$, remove all multiplicative factors which constant in $\theta_j$
  3. Identify the kernel of the resultant conditional density (or otherwise devise a strategy for sampling)

# Gibbs Sampling the Normal Model

# The Gibbs Sampler

- Assume $p(\mu) \sim N(\mu_0, \tau^2)$ and $\frac{1}{\sigma^2} \sim \text{Gamma}(a, b)$
- $p(\mu \mid \sigma^2, y)$ is a normal distribution (conjugacy)
  - Sample $\mu^{(s)} \sim \text{Normal}(\mu_n, \tau_n^2)$
  - $\tau_n^2 = \frac{\sigma^{2\,(s-1)}}{n+\kappa_0}$
- $p(\frac{1}{\sigma^2} \mid \mu, y)$ is a gamma distribution (conjugacy)
  - Sample

    $(\frac{1}{\sigma^2})^{(s)} \sim \text{Gamma}(a + n/2, b + \frac{\sum (d_i^{(s)})^2}{2} + \frac{\kappa_0}{2}(\mu^{(s)} - \mu_0$
  - $d_i^{(s)} = (y_i - \mu^{(s)})$

# Gibbs Sampler

```
gibbs <- function(mu_cur, prec_cur, burnin, maxit)

  nsamp <- maxit + burnin
  mu_samps <- numeric(nsamp)
  prec_samps <- numeric(nsamp)
  mu_samps[1] <- mu_cur
  prec_samps[1] <- prec_cur

  for(i in 2:nsamp) {
    ## Sample mu given current precision
    mu_cur <- rnorm(1, mun, sqrt(1/prec_cur * 1/(k0 + n)))

    ## Sample precision given current mu
    prec_cur <- rgamma(1, n/2+1, sum((y - mu_cur)^2)/2 + k0/2*(mu_cur - mu0

    mu_samps[i] <- mu_cur
    prec_samps[i] <- prec_cur
  }
## ...
```

# Gibbs Sampler Code

```
    ## ... cont
    ## Chop off the first part of the chain -- this reduces dependence on t
    if(burnin == 0) {
      list(mu_samps = mu_samps, theta_samps=theta_samps)
    }
    else {
      list(mu_samps=mu_samps[-(1:burnin), ], prec_samps=prec_samp[-(1:burni
    }
}
```

# The Gibbs Sampler

Animations

# Gibbs Sampler: Normal Hierarchical Model

# Gibbs Sampler

# Gibbs: a special case of MH

# Gibbs Sampler: Finite Mixture Models

## Reminder: mixture of binomials example

```r
set.seed(123)
n <- 25
phi <- 0.4
size <- 10
z <- ifelse(rbinom(100, 1, phi), "Experience", "No Experience")
y <- rbinom(n, size=size, prob=ifelse(z=="Experience", 0.7, 0.3))
```

# Gibbs Sampler: Binomial Mixture

```r
## Initialize all
run_binomial_gibbs <- function() {
experience_vec <- rbinom(n, 1, 0.5)
theta_cur <- c(0.1, 0.9)
phi_cur <- 0.5
nsamps <- 10000

theta_samps <- matrix(0, nrow=nsamps, ncol=2)
colnames(theta_samps) <- c("theta1", "theta2")
phi_samps <- numeric(nsamps)
experience_samps <- matrix(0, nrow=nsamps, ncol=n)

for(i in 1:nsamps) {

  ## Update experience indicators
  experience_prob <- (phi_cur*dbinom(y, 10, prob=theta_cur[2])) / ((1-phi_cur)*dbinom(y, 10, prob=theta_cur[1
  experience_vec <- 1*(runif(n) < experience_prob)
    ## Update mixture weights
  phi_cur <- rbeta(1, sum(experience_vec)+1, n-sum(experience_vec)+1)
  phi_samps[i] <- phi_cur

  y0 <- y[experience_vec == 0]
  n0 <- sum(experience_vec == 0)
  y1 <- y[experience_vec == 1]
  n1 <- sum(experience_vec)

  ## Update theta
```

# Stan version

```
library(cmdstanr)
library(tidyverse)
library(tidybayes)

sm <- cmdstan_model("binomial_mix.stan")
data_list=list(y=y, N=length(y), K=2)
stan_res <- sm$sample(data=data_list, iter_warmup=10000, iter_sampling=2000
```

```
Running MCMC with 4 sequential chains...

Chain 1 Iteration:     1 / 30000 [  0%]  (Warmup)
Chain 1 Iteration:  1000 / 30000 [  3%]  (Warmup)
Chain 1 Iteration:  2000 / 30000 [  6%]  (Warmup)
Chain 1 Iteration:  3000 / 30000 [ 10%]  (Warmup)
Chain 1 Iteration:  4000 / 30000 [ 13%]  (Warmup)
Chain 1 Iteration:  5000 / 30000 [ 16%]  (Warmup)
Chain 1 Iteration:  6000 / 30000 [ 20%]  (Warmup)
Chain 1 Iteration:  7000 / 30000 [ 23%]  (Warmup)
Chain 1 Iteration:  8000 / 30000 [ 26%]  (Warmup)
Chain 1 Iteration:  9000 / 30000 [ 30%]  (Warmup)
Chain 1 Iteration: 10000 / 30000 [ 33%]  (Warmup)
Chain 1 Iteration: 10001 / 30000 [ 33%]  (Sampling)
Chain 1 Iteration: 11000 / 30000 [ 36%]  (Sampling)
```

```r
stan_res$summary()
```

```
# A tibble: 7 × 10
  variable    mean  median     sd    mad       q5     q95  rhat ess_bulk
ess_tail
  <chr>      <num>   <num>  <num>  <num>    <num>   <num> <num>    <num>
<num>
1 lp__     -60.1   -59.7   1.38   1.08   -62.7   -58.6     1.00   25068.
30303.
2 prop[1]    0.409   0.407 0.112  0.114    0.228   0.597  1.00   37382.
34151.
3 prop[2]    0.591   0.593 0.112  0.114    0.403   0.772  1.00   37382.
34151.
4 lambda[…   0.627   0.523 0.451  0.380    0.111   1.50   1.00   24698.
22267.
5 lambda[…   1.37    1.15  0.975  0.832    0.245   3.27   1.00   24203.
22220.
```

```r
stan_res %>% spread_draws(theta[K]) %>%
  ggplot() + geom_line(aes(x=.iteration, y=theta, col=as.factor(.chain))) +
  facet_wrap(~K)
```

# Quantile intervals

```r
quantile(gibbs_results[[1]][, "theta1"], c(0.025, 0.5, .975))
```

```
     2.5%       50%     97.5% 
0.2399146 0.3738561 0.7823448 
```

```r
quantile(gibbs_results[[1]][, "theta2"], c(0.025, 0.5, .975))
```

```
     2.5%       50%     97.5% 
0.3040817 0.7194903 0.8339195 
```

```r
quantile(gibbs_results[[1]][, "phi_samps"], c(0.025, 0.5, .975))
```

```
     2.5%       50%     97.5% 
0.2355142 0.5119565 0.7879399 
```

# Rhat

```
rstan::Rhat(cbind(gibbs_results[[1]][, "theta1"],gibbs_results[[2]][, "thet
```

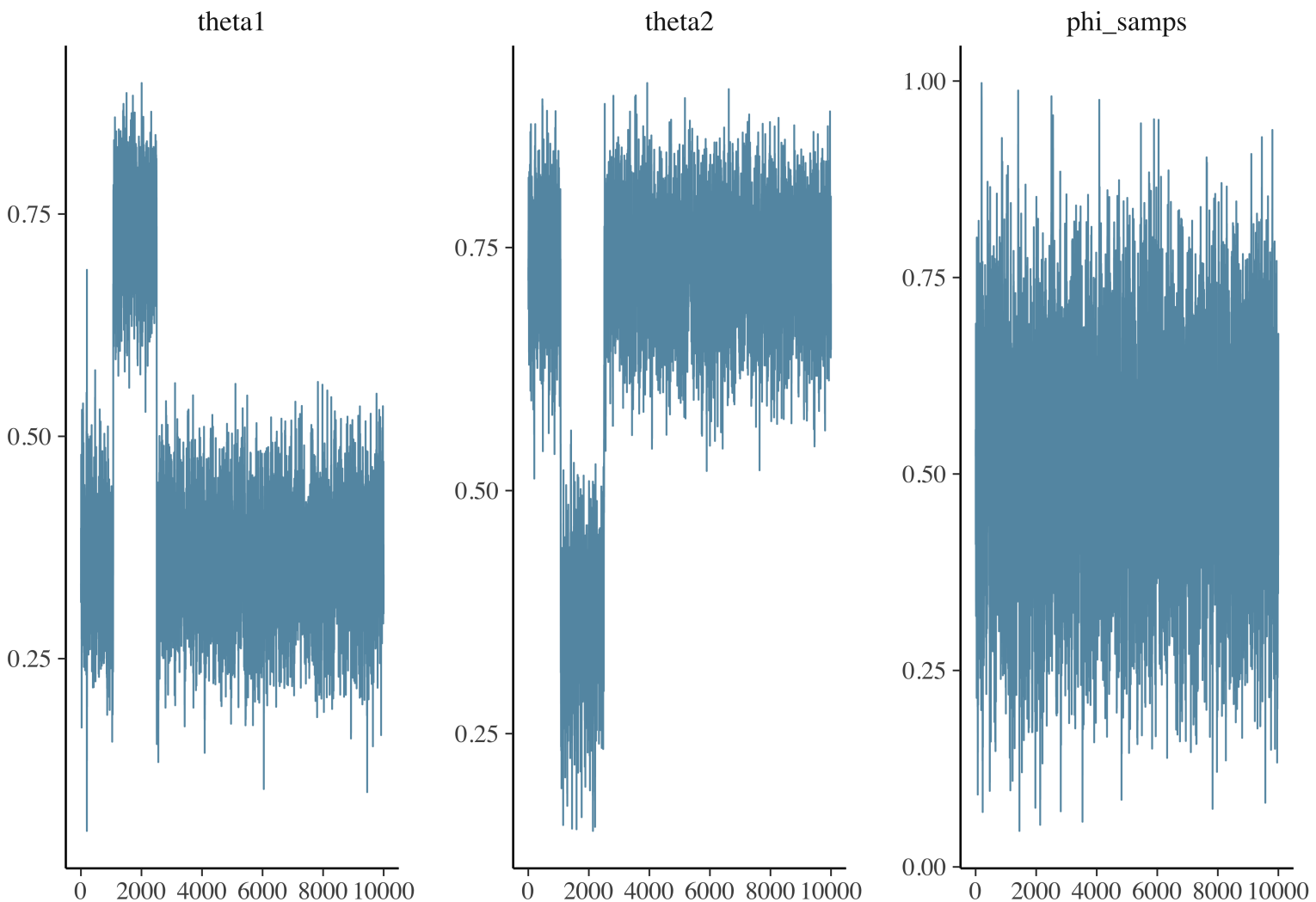
```
[1] 1.175579
```

```
rstan::Rhat(cbind(gibbs_results[[1]][, "theta2"],gibbs_results[[2]][, "thet
```

```
[1] 1.174507
```

# Traceplots

```
1  bayesplot::mcmc_trace(gibbs_results[[1]])
```



# Correcting label switching

```r
quantile(apply(gibbs_results[[1]][, 1:2], 1, max), c(0.05, 0.5, 0.95))
```

```
       5%       50%       95% 
0.6328924 0.7315352 0.8209664 
```

```r
quantile(apply(gibbs_results[[1]][, 1:2], 1, min), c(0.05, 0.5, 0.95))
```

```
       5%       50%       95% 
0.2570862 0.3606355 0.4630786 
```