Lecture 7: Markov Chain Monte Carlo

Professor Rodrigo Targino

Announcements

- Reading: Chapter 7 of Bayes Rules
- Homework 4 (the last one!): out by the end of the week, deadline on Wednesday, Jun 9
- Last quiz this Friday
- I won't be here next week!
- Tuesday — video Jun 7
- Thursday — Alex Franks Jun 9
- Fri

Monte Carlo estimation

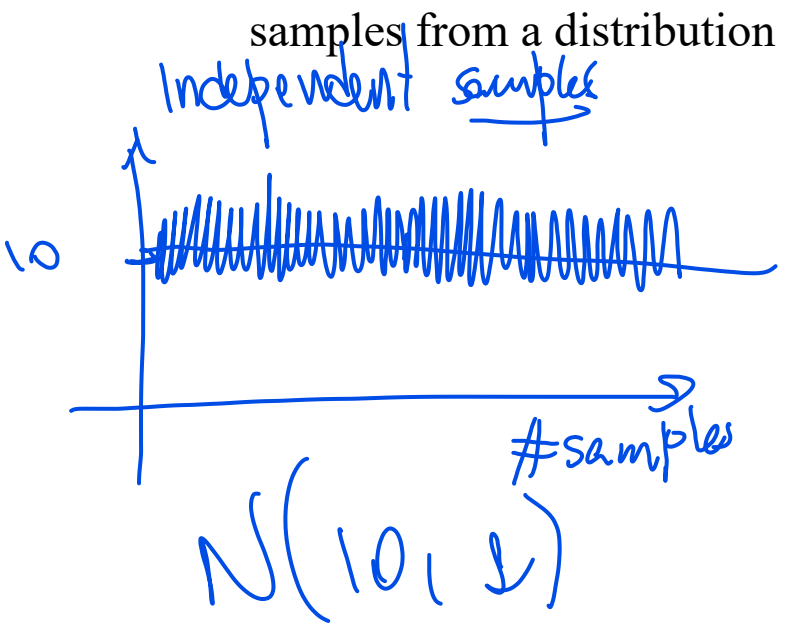
- $\bar{\theta} = \sum_{s=1}^{S} \theta^{(s)}/S \to \mathrm{E}[\theta|y_1, \ldots, y_n]$
- $\sum_{s=1}^{S} \left(\theta^{(s)} - \bar{\theta}\right)^2/(S-1) \to \mathrm{Var}[\theta|y_1, \ldots, y_n]$
- $\#\left(\theta^{(s)} \leq c\right)/S \to \Pr(\theta \leq c|y_1, \ldots, y_n)$
- the α-percentile of $\left\{\theta^{(1)}, \ldots, \theta^{(S)}\right\} \to \theta_\alpha$

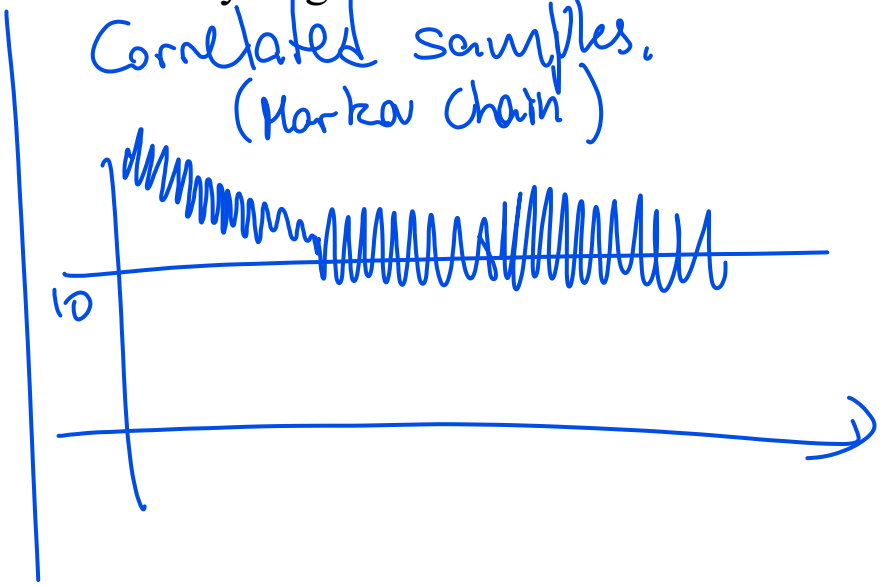
Sampling from the posterior distributions

- The Monte Carlo methods we discussed previously assumed we could easily get samples from the posterior, e.g. with `rnorm`
- In general, sampling from a general probability distribution is hard
- Want to call `rcomplicatedistribution()` but don't have it
  - Inversion sampling is limited
  - Grid sampling is reasonable in 1 or 2 dimensions
- In high dimensions, these approaches aren't sufficient

Markov Chain Monte Carlo

- We want independent random samples, $\theta^{(s)}$ from $p(\theta \mid y_1, \ldots y_n)$
- But there is no good way to get independent samples
- Alternative, create a sequence of **correlated** samples that converge to the correct distribution
- Markov Chain Monte Carlo gives us a way to generate correlated samples from a distribution





Monte Carlo Error

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

$$\mathrm{Var}(\bar{\theta}) = \frac{1}{S^2} \sum_{s=1}^{S} \mathrm{Var}(\theta^{(s)}) = \frac{\mathrm{Var}(\theta \mid y_1, \ldots y_n)}{S}$$

- If the samples are *positively correlated*,

$$\mathrm{Var}(\bar{\theta}) = \frac{1}{S^2} \sum_{s,t} \mathrm{Cov}(\theta^{(s)}, \theta^{(t)}) > \frac{\mathrm{Var}(\theta \mid y_1, \ldots y_n)}{S}$$

- MCMC methods have higher Monte Carlo error due to positive dependence between samples.
- Hope to minimize dependence and thus MC error

Basics of Markov Chains

Markov Chains: Big Picture

- For standard Monte Carlo, we make use of the law of large number to approximate posterior quantities
- The law of large numbers can still apply to random variables that are not independent
- We have a sequence of random variables indexed in time, θ_t
- We'll be using a *discrete-time* Markov Chain: $t \in 0, 1, \ldots T$
- The observations, $\theta^{(t)}$ can be discrete or continous ("discrete-state" or "continuous-state" Markov Chain)

Discrete-state Markov Chains

- Let $\theta^{(t)} \in 1, 2, \ldots M$ be the state space for the Markov Chain
- A sequence is called a markov chain if

$$Pr(\theta^{(t+1)} \mid \theta^{(t)}, \theta^{(t-1)} \ldots \theta^{(1)}) = Pr(\theta^{(t+1)} \mid \theta^{(t)})$$

  for all $t \geq 0$

- The **Markov property**: given the entire past history, $\theta^{(1)}, \ldots \theta^{(t)}$, the most recent $\theta^{(t+1)}$ depends only on the immediate past, $\theta^{(t)}$

The Transition Matrix

- Define $q_{ij} = Pr(\theta^{(t+1)} \mid \theta^{(t)})$ is the transition probability from state i to state j
- The $M \times M$ matrix $Q = (q_{ij})$ is called the *transition matrix* of the Markov Chain

3-state example

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$$

The Transition Matrix

3-state example

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}$$

- The rows of the transition matrix sum to 1
- Note: $Q^n = (q_{ij}^{(n)})$ is the probability of transitioning from i to j in n steps

The limiting distribution

- A regular, irreducible Markov chain has a **limiting probability distribution**
  - Cover definitions of regular and irreducible in PSTAT160 (or related)
- Limit distribution describes the long-run fraction of time the Markov Chain spends in each state
  - *Does not* depend on where the chain starts
- Let $\pi = (\pi_1, \ldots \pi_M)$ be a row vector of probabilities associated with each state, such that $\sum_{i=1}^{M} = \pi_i = 1$
  - The limiting distribution converges to π, which is said to be **stationary** because $\pi Q = \pi$
  - If you sample from the limiting distribution and then transition, the result is still distributed according to the limiting distribution

Markov Chain Example

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

$$\mathbf{Q} = \begin{array}{|l|lll|} \hline & \text{Lower} & \text{Middle} & \text{Upper} \\ \hline \text{Lower} & 0.40 & 0.50 & 0.10 \\ \text{Middle} & 0.05 & 0.70 & 0.25 \\ \text{Upper} & 0.05 & 0.50 & 0.45 \\ \hline \end{array}$$

Multi-step Transition Probabilities

2-step transition probabilities

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

4-step transition probabilities

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

Multi-step Transition Probabilities

4-step transition probabilities

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

8-step transition probabilities

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

The limiting distribution

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

- The equation $\pi Q = \pi$ implies that the (row) vector π is a left eigenvector of Q with eigenvalue equal to 1
- Reminder: $Ax = \lambda x$ implies that x, a column vector, is a (right) eigenvector with eigenvalue λ

The limiting distribution

```
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
```

```
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 $p(\theta^{(t+1)} \mid \theta^{(t)})$ is constructed correctly, and we run the chain long enough, $\theta^{(t)}$ will be distributed approximately according to $p(\theta \mid y)$

The Independence Sampler

- The Metropolis algorithm tells us how to construct a transition matrix with the correct limiting distribution
    - The Independence Sampler is a special case of the Metropolis algorithm
- Sample from a proposal, $J(\theta)$. Best if $J(\theta)$ is close to $p(\theta \mid y)$.
- If $p(\theta \mid y) > 0$ then we need $J(\theta) > 0$
- At each iteration we have a choice:
    - Accept the new proposed sample
    - Or keep the previous sample for another iteration

The Independence Sampler

1. Initialize θ_0 to be the starting point for you Markov Chain
2. Choose a *proposal distribution*, $J(\theta^*)$
    - Propose a candidate value for the next sample
    - Best performance if density is very similar to target
3. Generate the candidate θ^* from the proposal distribution, J
4. Compute $r = \min(1, \frac{p(\theta^*|y)}{p(\theta_t|y)})$
5. Set $\theta_{t+1} \leftarrow \theta^*$ with probability r
    - Generate a uniform random number $u \sim Unif(0, 1)$
    - If $u < r$ we accept θ^* as our next sample
    - Else $\theta_{t+1} \leftarrow \theta_t$ (we do not update the sample this time)

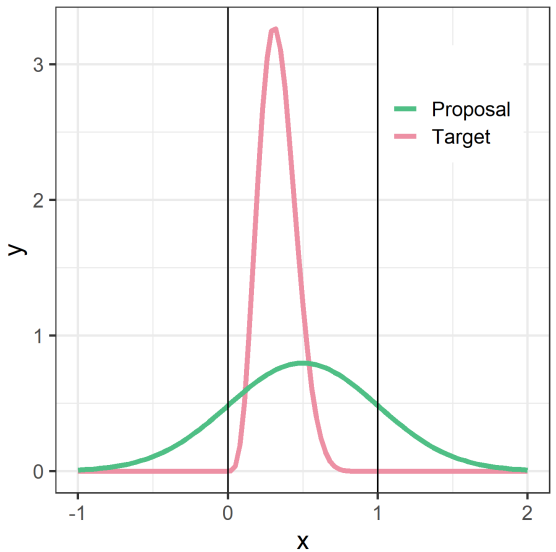
Intuition

- If $p(\theta^* \mid y) > p(\theta_t \mid y)$ accept with probability 1
  - The proposed sample has higher posterior density than the previous sample
  - Always accept if we increase the posterior probability density
- If $p(\theta^* \mid y) < p(\theta_t \mid y)$ accept with probability $r < 1$
  - Accept with probability less than 1 if probability density would decrease
  - Relative frequency of θ^* vs θ_t in our samples should be $\frac{p(\theta^*|y)}{p(\theta_t|y)}$

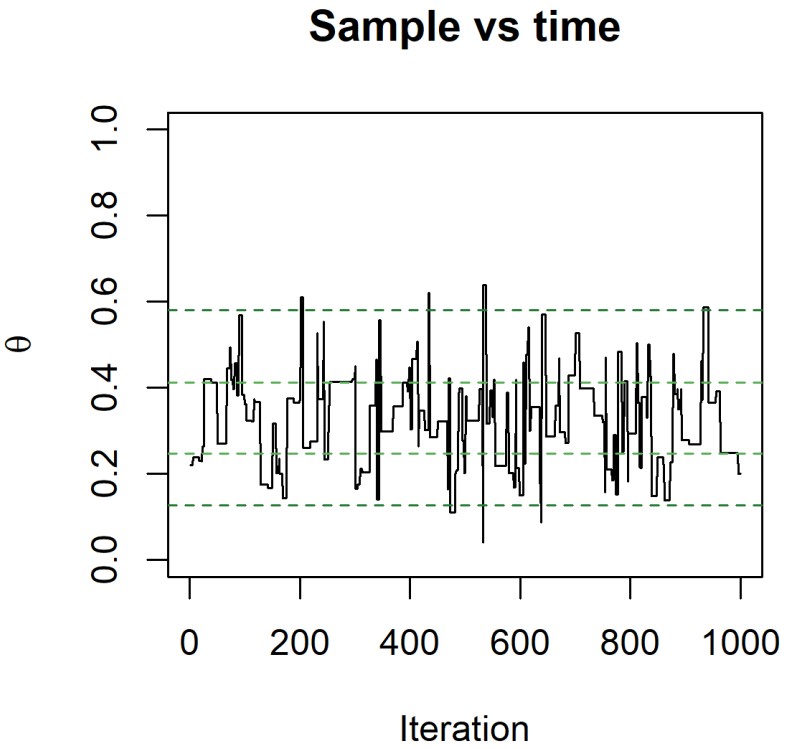
An Example

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

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.
## ℹ Please use `linewidth` instead.
```

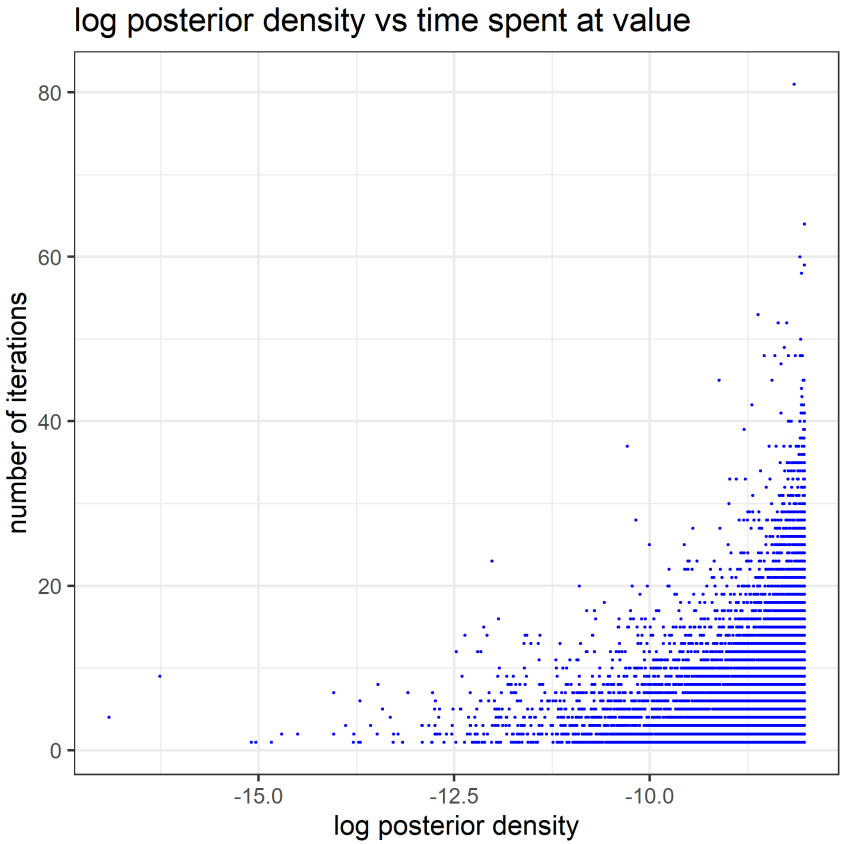


Independence Sampler



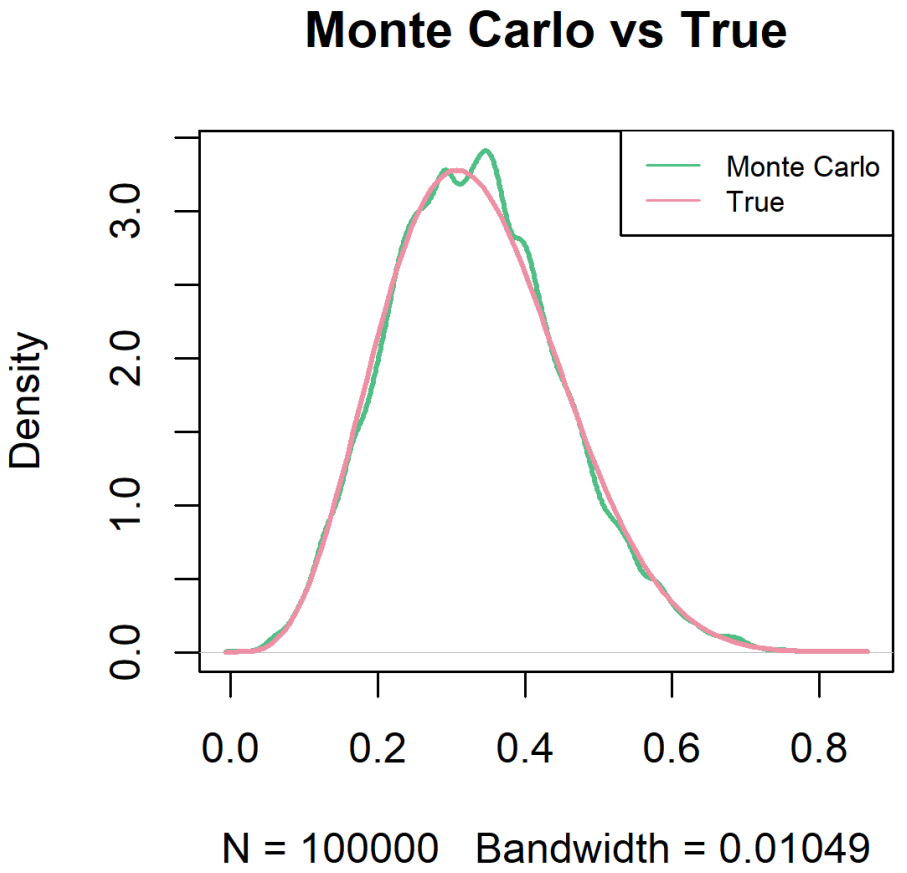
Note and source of confusion: samples are correlated over time for the "independence sampler".

Weighting by waiting



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

Independence Sampler



The Metropolis Algorithm

- Generalize the previous special case
- Allow the proposal distribution to depend on the most recent sample
    - Sometimes called an "Independence sampler":
      $J(\theta^*)$, e.g. $\theta^* \sim N(0.5, 1)$
    - Metropolis: $J(\theta^* \mid \theta_t)$, e.g. $\theta^* \sim N(\theta_t, 1)$
- Independence sampler: "Independence" refers to the proposal being fixed (the samples are **not** independent)!
- Metropolis sampler: a "moving" proposal distribution

The Metropolis Algorithm

1. Initialize θ_0 to be the starting point for you Markov Chain
2. Choose a proposal distribution, $J(\theta^* \mid \theta_t)$
    - Propose a candidate value for the next sample
    - Must have symmetry: $J(\theta^* \mid \theta_t) = J(\theta_t \mid \theta^*)$
3. Generate the candidate θ^* from the proposal distribution, J
4. Compute $r = \min(1, \frac{p(\theta^*|y)}{p(\theta_t|y)})$
5. Set $\theta_{t+1} \leftarrow \theta^*$ with probability r
    - Generate a uniform random number $u \sim Unif(0, 1)$
    - If $u < r$ we accept θ^* as our next sample
    - Else $\theta_{t+1} \leftarrow \theta_t$ (we do not update the sample this time)

Aside: Arianna Rosenbluth

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

https://www.nytimes.com/2021/02/09/science/arianna-wright-dead.html

Metropolis Algorithm

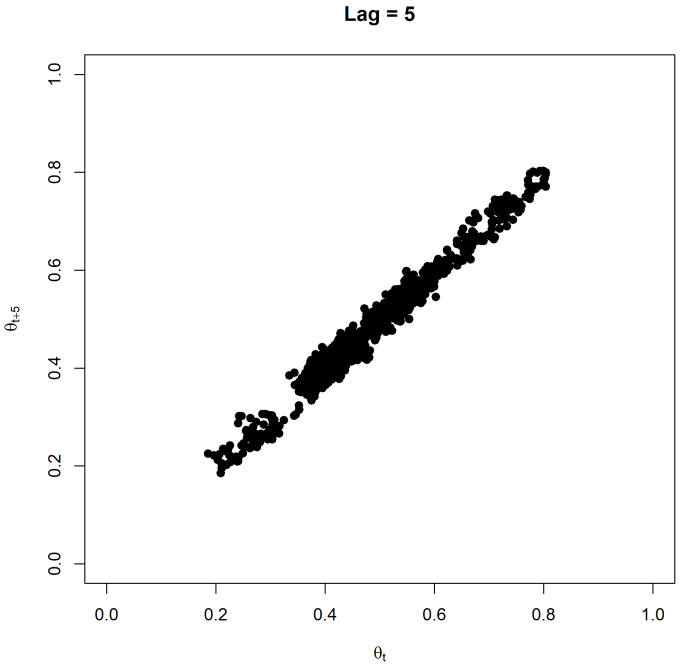
- Let $P(\theta \mid y)$ be a Beta(5, 10) posterior distribution
- 1-d sampling: lets try sampling from the Beta using the Metropolis algorithm
- Initialize θ_0 to 0.9
  - Note that the probability of drawing a value larger than 0.9 from a Beta(5, 10) is smaller than 1e-8
  - Our initial value is far from the high posterior density
  - In the long run this won't matter
- Define transition kernel $J(\theta_{t+1} \mid \theta_t)$ as $\theta^* \sim N(\theta_t, \tau^2)$
  - How does choice of τ^2 effect performance of MC sampler?

Demo

Autocorrelation of the Markov Chain

$\tau^2 = 0.01$ ("small" proposal variance)

Plot θ^t vs θ^{t+5} for all values of t

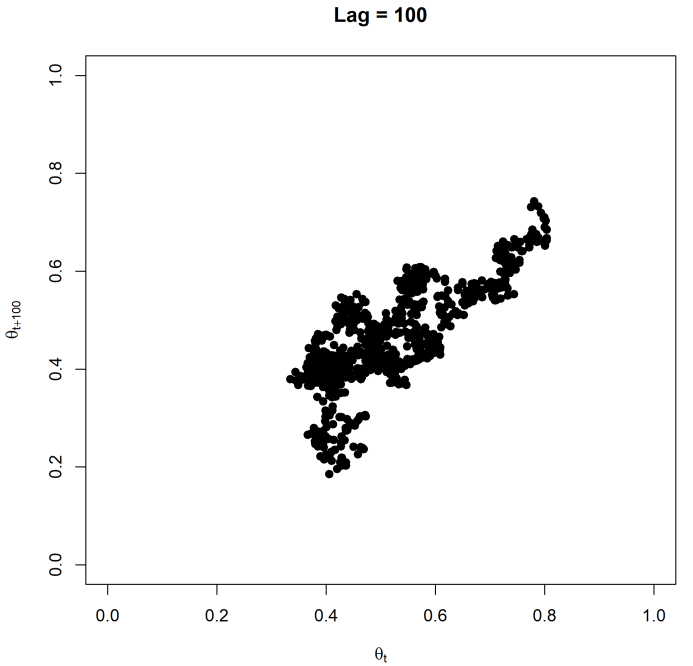


Correlation = 0.9791817

Autocorrelation of the Markov Chain

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

Plot θ^t vs θ^{t+100} for all values of t

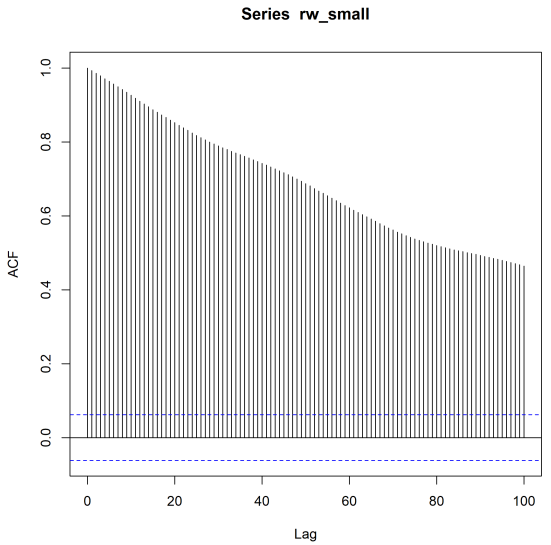


Correlation = 0.6388372

The Metropolis Algorithm

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

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

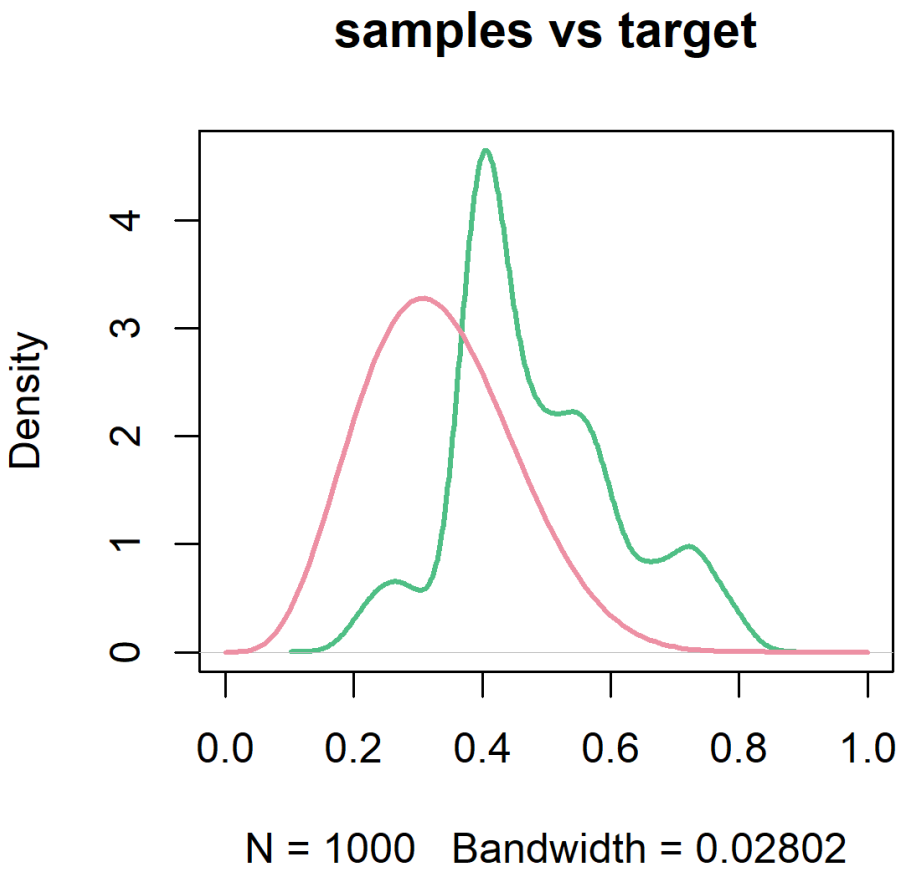



```
print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
              effectiveSize(rw_small), rejectionRate(as.mcmc(rw_sm
```

```
## [1] "Effective sample size: 3.57, Rejection Rate: 0.03"
```

The Metropolis Algorithm

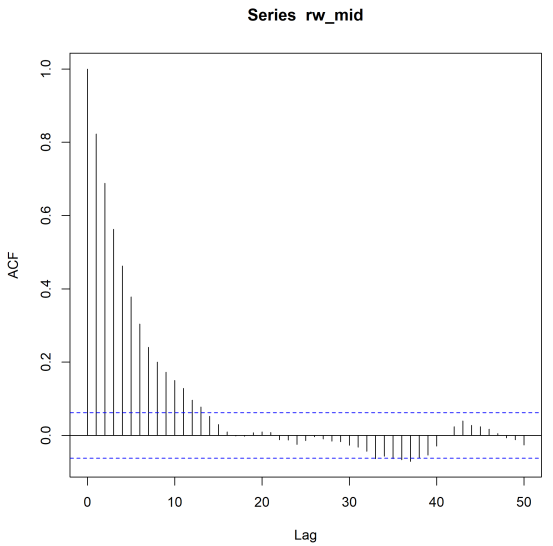
$\tau^2 = 0.01$ ("small" jump)



The Metropolis Algorithm

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

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

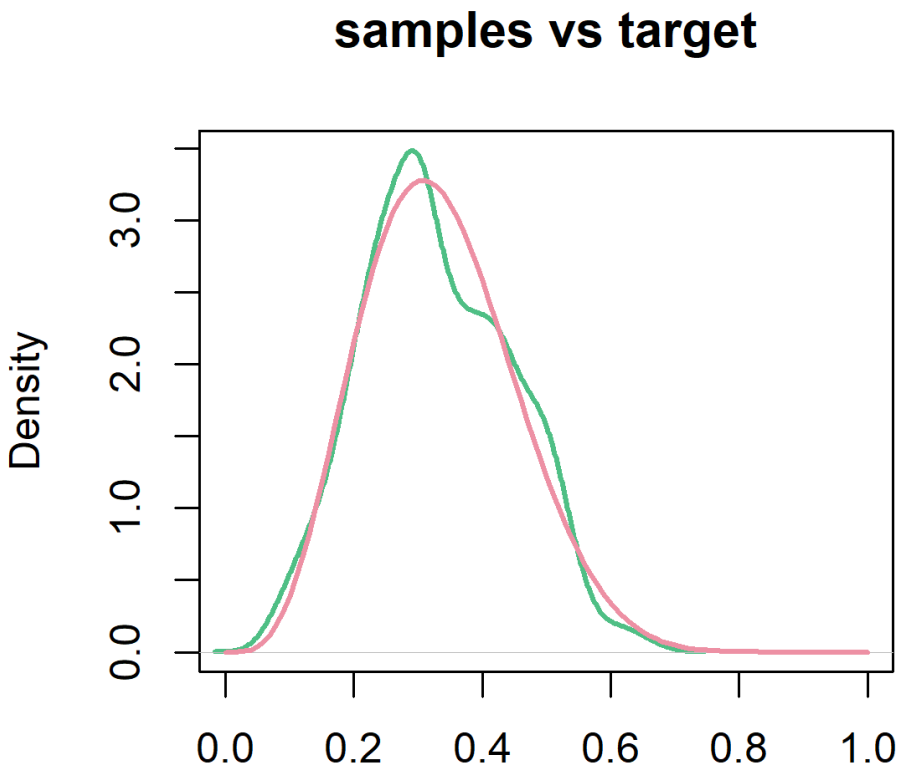



```
print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
              effectiveSize(rw_mid), rejectionRate(as.mcmc(rw_mid)
```

```
## [1] "Effective sample size: 97.36, Rejection Rate: 0.24"
```

The Metropolis Algorithm

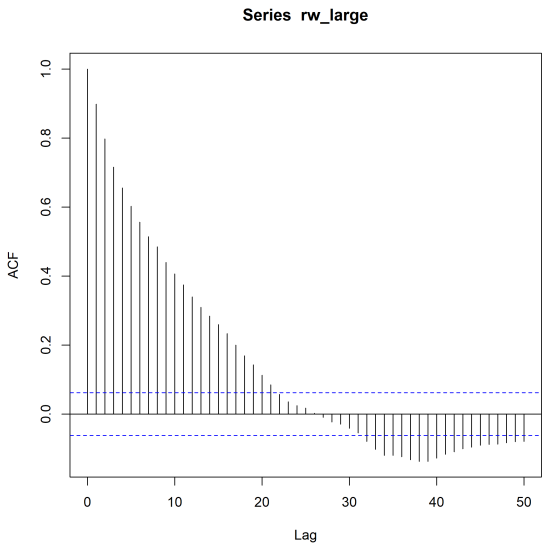
$\tau^2 = 0.1$ ("medium" proposal variance)



The Metropolis Algorithm

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

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

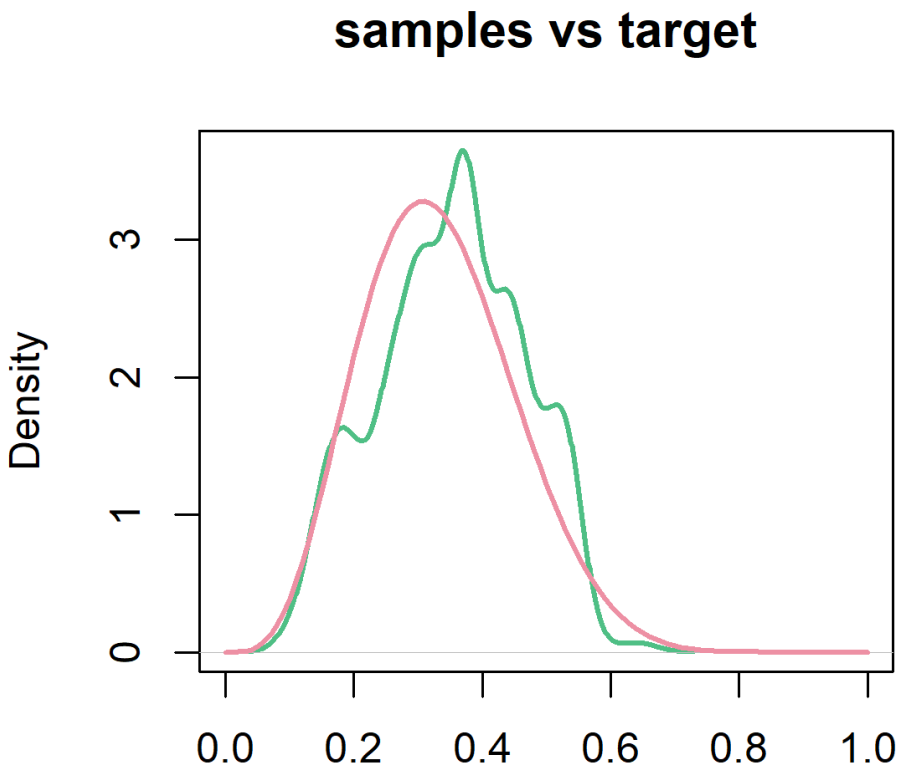



```
print(sprintf("Effective sample size: %.2f, Rejection Rate: %.2f",
              effectiveSize(rw_large), rejectionRate(as.mcmc(rw_la
```

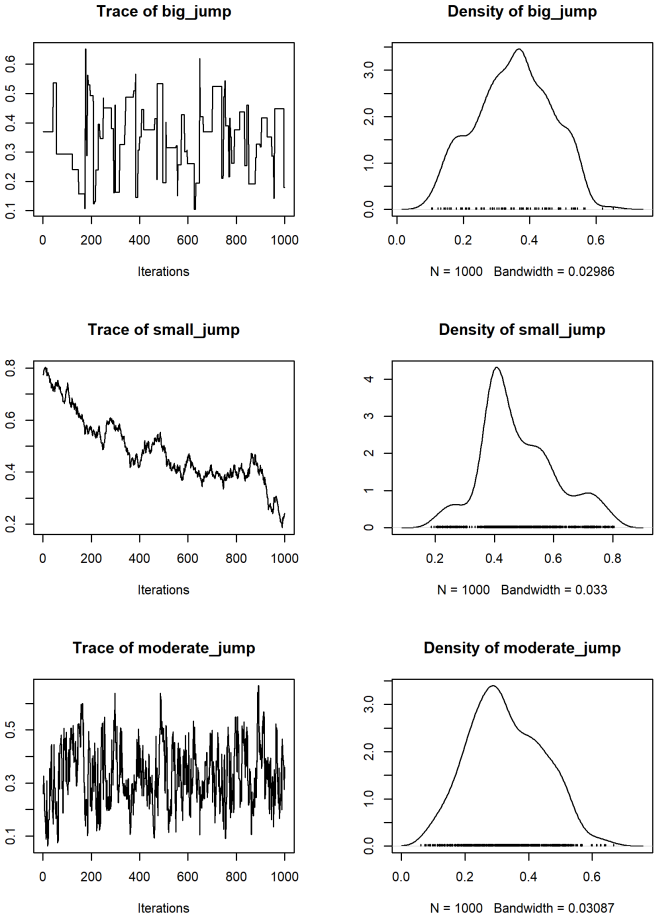
```
## [1] "Effective sample size: 42.27, Rejection Rate: 0.92"
```

The Metropolis Algorithm

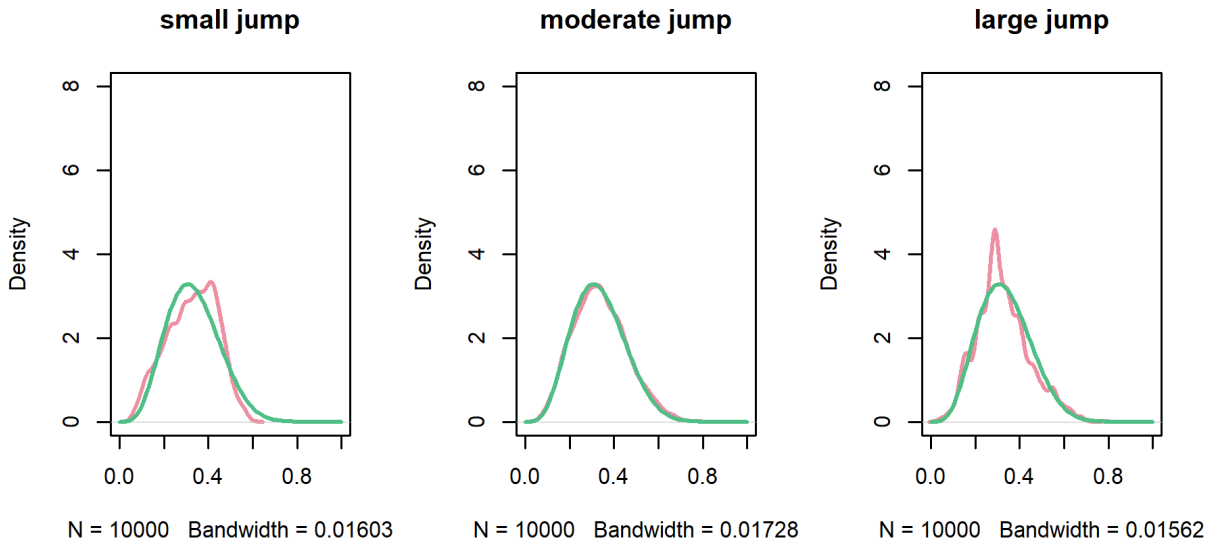
$\tau^2 = 2$ ("large" proposal variance)



Small, Moderate and Large Proposal variance



10,000 Samples




```
## [1] "Effective sample size (small proposal variance): 20.64"

## [1] "Effective sample size (medium proposal variance): 936.65"

## [1] "Effective sample size (large proposal variance): 527.44"
```

MCMC diagnostics

- Diagnose the chain performance by examining:
  - Rejection rate
  - Autocorrelation
  - Effective sample size

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

- **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)

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.

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 τ^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)
- τ^2 too small, the chain explore the parameter space slowly
- A rule of thumb is to aim for 30%-40% acceptance rate for random walk samplers.
    - This balances "stickiness" and slow convergence

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

MCMC for multivariate distributions

- Modeling wing length of different specifies of midge (small, two-winged flies)
- Reminder: $Y_i \sim N(\mu, \sigma^2)$
- $P(\mu \mid \sigma^2)$, $\mu \sim N(\mu_0, \frac{\sigma^2}{\kappa_0})$
- $P(\sigma^2) \propto \frac{1}{\sigma^2}$ (improper prior)

MCMC for multivariate distributions

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 $\kappa_0 = 1$
- We will run 2 separate chains at different starting locations
- $J(\mu*, \log(\sigma*)) \sim N((\mu^t, \log(\sigma^t)), \begin{pmatrix} \tau_\mu^2 & 0 \\ 0 & \tau_{log\sigma}^2 \end{pmatrix})$

Initialization and Convergence

- In the long run, it doesn't matter where you initialize your sampler
- In practice, we can only run an algorithm for a finite amount of time
    - Need to check with the sampler has converged to the limiting distribution
    - Exclude samples close to the initial values since these are unlikely to be representative samples
    - We call the time to convergence **burn in** and throw away and samples generated during this time.
- How do we know when a sampler has converged to the limiting distribution?

Running multiple chains

- How do we know when a sampler has converged to the limiting distribution?
    - Hard to know for sure.
- Idea: run multiple chains at very different initial locations
    - If the chains are very far apart we haven't converged
    - If the chains end up in the same place, we have confidence that its reached convergence

Diagnosing mixing with Rhat

$$B = \frac{n}{m-1} \sum_{j=1}^{m} \left(\bar{\psi}_{.j} - \bar{\psi}_{..} \right)^2$$

$$W = \frac{1}{m} \sum_{j=1}^{m} \boldsymbol{s}_j^2$$

Demo

Multiple Chains in Stan

```
library(cmdstanr)
y <- c(1.64, 1.7, 1.72, 1.74, 1.82, 1.82, 1.82, 1.9, 2.08)

# compile stan model.  This may take a minute.
stan_model <- cmdstan_model(stan_file="normal_model.stan")

## data is a list, arguments must match the datablock in the stan
stan_fit <- stan_model$sample(data=list(N=length(y), y=y, k0=1),
                              chains = 2,
                              num_warmup=200,
                              num_samples = 1000,
                              refresh=200)
```

Multiple Chains in Stan

```
## This is cmdstanr version 0.5.3

## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr

## - CmdStan path: C:/Users/targi/OneDrive/Documentos/.cmdstan/cmdstan-2.32

## - CmdStan version: 2.32.2

## Running MCMC with 2 sequential chains...
##
## Chain 1 finished in 0.0 seconds.
## Chain 2 finished in 0.0 seconds.
##
## Both chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.6 seconds.
```

```
SAMPLING FOR MODEL 'normal_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 9e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 1: Iteration: 201 / 1000 [ 20%]  (Sampling)
Chain 1: Iteration: 400 / 1000 [ 40%]  (Sampling)
Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.015009 seconds (Warm-up)
Chain 1:                0.02576 seconds (Sampling)
Chain 1:                0.040769 seconds (Total)
Chain 1:
```

Stan Samples

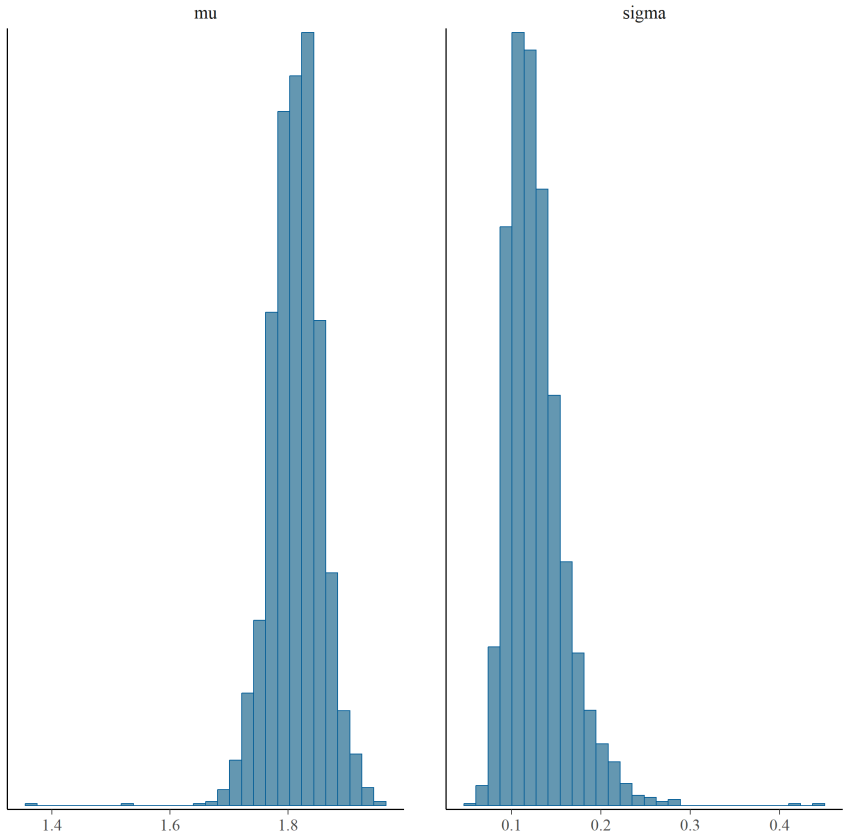
```
draws <- stan_fit$draws(format="df")
draws
```

```
## # A draws_df: 1000 iterations, 2 chains, and 3 variables
##    lp__  mu sigma
## 1    18 1.8 0.124
## 2    18 1.9 0.116
## 3    17 1.9 0.121
## 4    18 1.8 0.111
## 5    18 1.8 0.112
## 6    18 1.8 0.114
## 7    17 1.8 0.097
## 8    18 1.8 0.117
## 9    18 1.8 0.117
## 10   18 1.8 0.120
## # ... with 1990 more draws
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

Plotting Stan Draws

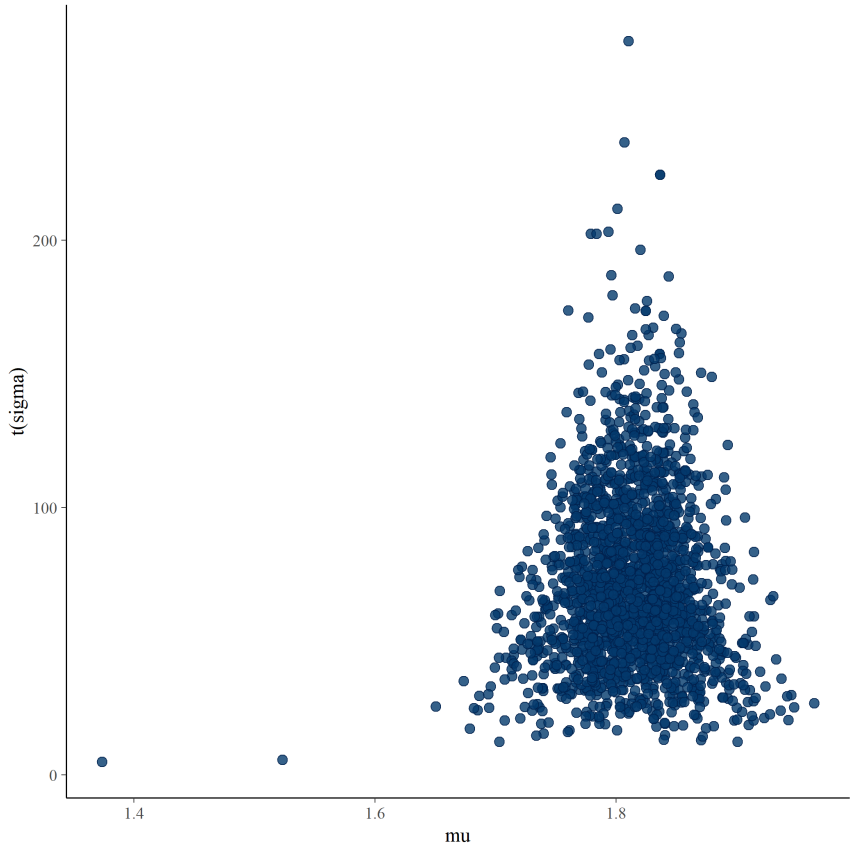
```
mcmc_hist(draws, pars=c("mu", "sigma"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



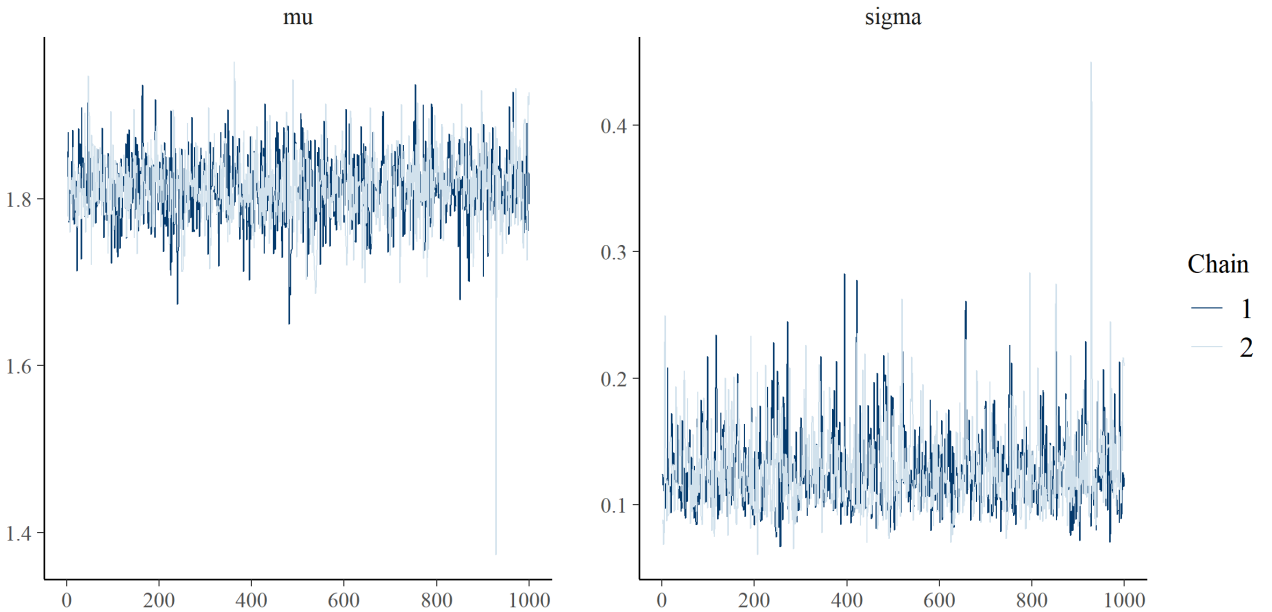
Plotting Stan Draws

```
mcmc_scatter(draws,
             pars=c("mu", "sigma"),
             transformations=list(sigma=function(x) 1/x^2))
```



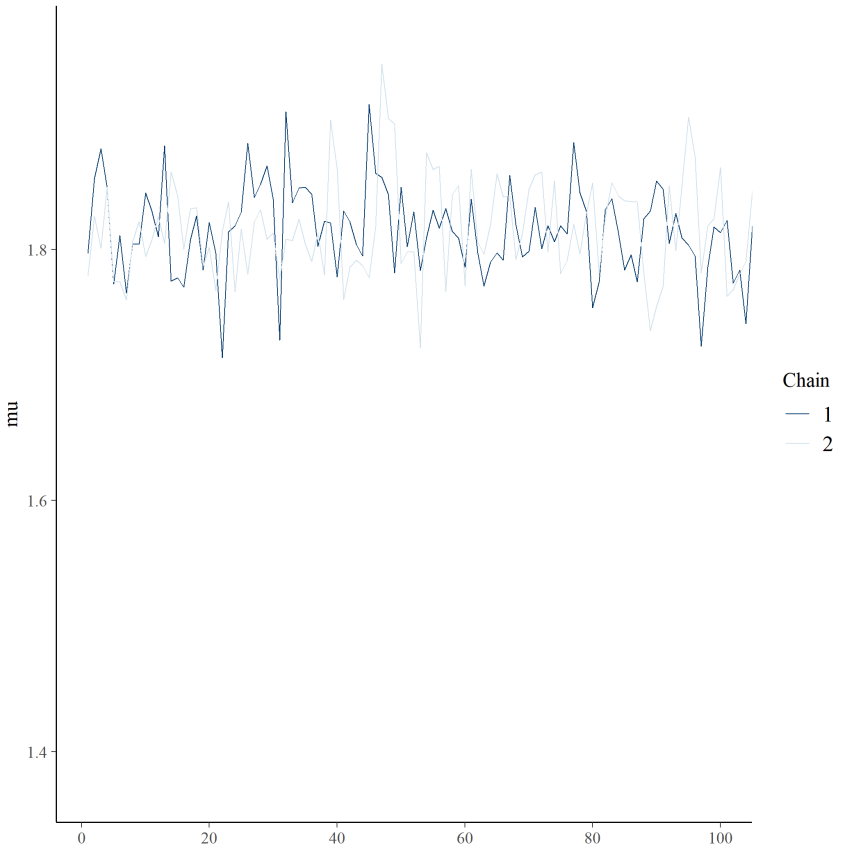
MCMC Diagnostics: Traceplots

```
mcmc_trace(draws, pars=c("mu", "sigma"))
```



MCMC Diagnostics: Traceplots

```
mcmc_trace(draws, pars=c("mu"), window = c(1, 100))
```



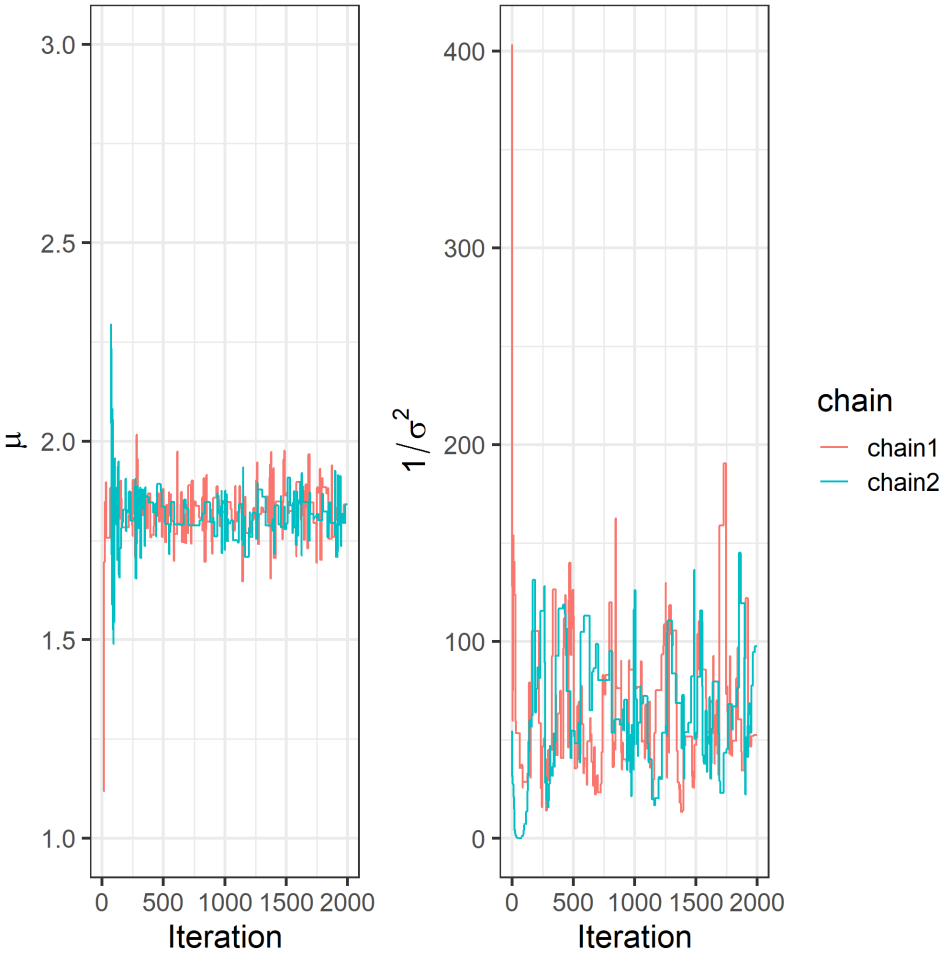
MCMC Diagnostics: Effective Sample Size

Efficiency can be expressed as (effective samples) / (total iterations)

```
stan_fit$summary() %>% glimpse
```

```
## Rows: 3
## Columns: 10
## $ variable <chr> "lp__", "mu", "sigma"
## $ mean     <dbl> 17.3069496, 1.8133395, 0.1276222
## $ median   <dbl> 17.637750, 1.813905, 0.121679
## $ sd       <dbl> 1.10547734, 0.04267468, 0.03291669
## $ mad      <dbl> 0.74352390, 0.03940751, 0.02741624
## $ q5       <dbl> 15.18689500, 1.74382850, 0.08801157
## $ q95      <dbl> 18.3171150, 1.8799200, 0.1894261
## $ rhat     <dbl> 1.001449, 1.003787, 1.000893
## $ ess_bulk <dbl> 901.7438, 1201.9776, 868.8390
## $ ess_tail <dbl> 988.1744, 732.6349, 1024.0068
```

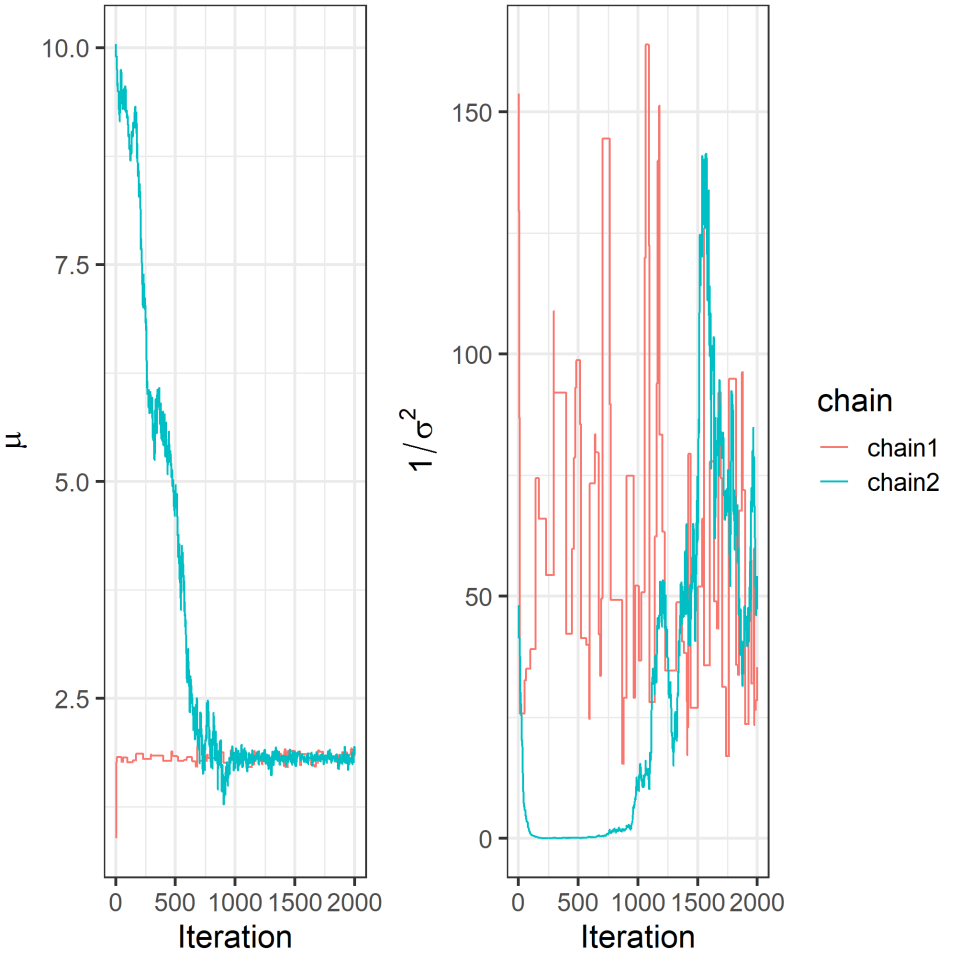
MCMC for multivariate distributions



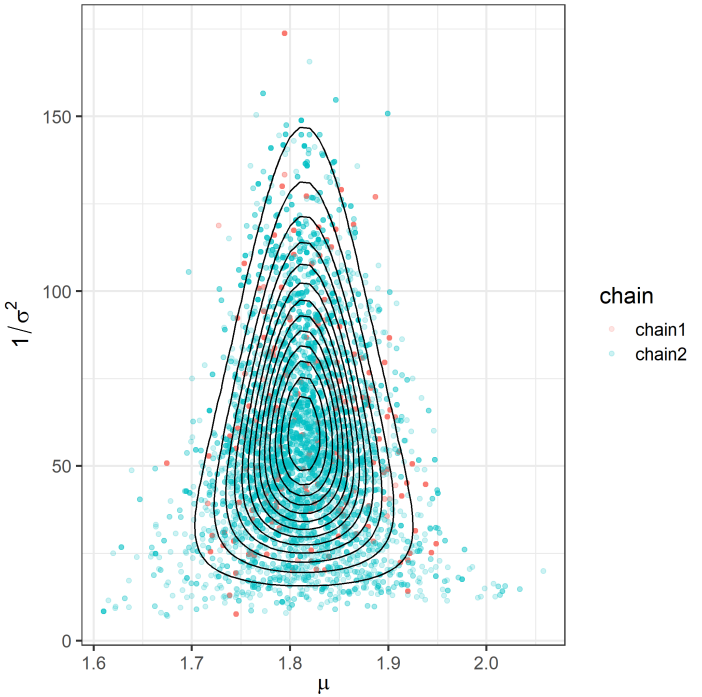
MCMC for multivariate distributions

- 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 271.    140.
## 2 chain2  11.4    33.2
```

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 σ"

Gibbs Sampling

The Gibbs Sampler

- The Gibbs sampler is actually special case of the MH sampler
  - This is not obvious or immediately apparent
- Idea: break the problem down into many smaller sampling problems
- Iteratively update each parameter in the full parameter vector by doing getting lower-dimensional sample

The Gibbs Sampler

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

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

    where θ_{-j}^{t-1} has consists of updated parameters for all parameters preceding j and the previous iteration's values for all succeeding parameters,

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

The Gibbs Sampler

- To identify the full conditional distributions:

  1. Write down the full posterior
  2. For each parameter, θ_j, remove all multiplicative constants that don't have a θ_j in them.
  3. Identify the type of distribution to sample from

- Assuming conjugate prior distributions, the full conditionals in the normal model are:

  - $p(\mu \mid \sigma^2, y)$ is a normal distribution
  - $p(\frac{1}{\sigma^2} \mid \mu, y)$ is a gama distribution

The Gibbs Sampler

Gibbs Sampling a Bivariate Normal

Demo

Gibbs Sampler

- Advantages
  - proposals, $p(\theta^{(t+1)} \mid \theta^{(t)})$, are never rejected.
  - don't need to choose the proposal density distribution or tune parameters of the density
- Disadvantages
  - It can be difficult to derive the full-conditional distributions (unless, for example, the prior distributions are chosen to be conjugate)
  - When the parameters of the posterior distribution are highly correlated:
  - Hard to traverse diagonals and consequently:
    - autocorrelation of the samples is high
    - effective sample size is low

Challenges in MCMC

- Modern models often have *many* parameters. Large models pose a challenge for MCMC.
- When there are thousands or more parameters
  - MCMC may take a long time to converge to the limiting distribution
  - In Metropolis-Hastings we have many tuning parameters for the proposal distribution
  - Gibbs sampling has no tuning parameters, but does not work well for highly correlated posterior distributions (see demo: banana + Gibbs)
- In general, MCMC is slow relative to optimization methods

Summary of MCMC

What is Monte Carlo and MCMC:

- MCMC is *not* a model
- It does *not* generate more information
- Provides *dependendent* approximate samples from $p(\theta \mid y)$
- Samples can be used to summarise $p(\theta \mid y)$ (approximate integrals)

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

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

```
## [1] NaN
```

```
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 θ^* as our next point

MCMC for multivariate distributions

- 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 $J(\theta^* \mid \theta_t) = N\left(\begin{pmatrix} \mu^{(t)} \\ \log(\sigma^{(t)}) \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\right)$

Challenges in MCMC

- Modern models often have *many* parameters. Large models pose a challenge for MCMC.
- When there are thousands or more parameters
    - MCMC may take a long time to conververge to the stationary distribution
    - In Metropolis-Hastings we have many tuning parameters for the proposal distribution
    - Gibbs sampling has no tuning parameters, but does not work well for highly correlated posterior distributions
- In general, MCMC is very slow relative to optimization methods

Modern MCMC

- Gibbs and Metropolis samplers have a "random walk" behavior
  - Induces autocorrelation
  - Makes it difficult to explore the posterior space
- Hamiltonian Monte Carlo (HMC) borrows an idea from physics to reduce this problem

HMC

- Imagine a marble on a frictionless surface. The location of the marble is the current value of θ_t
- The negative posterior density is the "height" of the surface
- Each iteration we flick the marble with some velocity in a random direction
- Regions of high posterior density are like "wells"

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
    - Determines where the marble rolls
- 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 marble and update its position accordingly
    - Can think of HMC as the MH algorithm with a very clever jumping/proposal rule

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

Approximate Inference

- MCMC can be very slow in high dimensional problems
- Idea: find a distribution that is easy to sample from which closely approximate $p(\theta \mid y)$
- A couple of examples
  - Laplace Approximation
  - Variational Bayes

Laplace Approximation to the Posterior

- Approximate the posterior distribution using a multivariate normal distribution
- When we have a lot of i.i.d. observations, the posterior will be approximately normal
- Center the normal at the mode of the posterior
- Compute the (co)variance of the normal by computing the second derivative / hessian of the posterior at the mode

Laplace Approximation

- Approximate the posterior distribution using a normal distribution
- When we have a lot of i.i.d. observations, the posterior will be approximately normal
- Center the normal at the mode of the posterior
- Compute the (co)variance of the normal by computing the second derivative / hessian of the posterior at the mode

Laplace Approximation

- Let $\tilde{\theta}$ be the mode of the of the posterior distribution
- Use a Taylor Series approximation the log-posterior around the mode is
  - $\log p(\theta \mid y) \approx \log p(\tilde{\theta} \mid y) - 1/2(\theta - \tilde{\theta})H(\theta - \tilde{\theta})$
  - $H = \frac{d^2}{d\theta^2} log p(\theta \mid y)$
  - Note, linear term falls out because derivative at the mode is zero
- $p(\theta \mid y) \approx N(\tilde{\theta}, I(\theta)^{-1})$

Finding the mode of the posterior distribution

- Calculus
  - Take the log
  - Differentiate, set to zero and solve
- Computational
  - `optim` in R for one dimensional posteriors
  - `optimise` in R for multivariate p

Variational Bayes

- Let θ be d dimensional parameter vector with posterior $p(\theta)$
- We search for a distribution that `best" approximates(p(\theta))` in some sense
- Kullback-Leibler divergence:

$$KL(q||p) = E_q\left[\log\left(\frac{q(\theta)}{p(\theta)}\right)\right]$$

- Intuitively, there are three cases
  - If q is high and p is high then we are happy :)
  - If q is high and p is low then we pay a price :(
  - If q is low then we don't care (because of the expectation) :|
- Searching for the best q over all distributions is hard. We restrict ourselves to a class of distributions parametrized by ν: $q_\nu(\theta)$
- Finding the best q_ν when $q_\nu(\theta) = \prod_i q_\nu(\theta_i)$ is reasonably easy! Mean Field Approximation!