AARHUS UNIVERSITY

Methods 4 - 8

Chris Mathys

BSc Programme in Cognitive Science

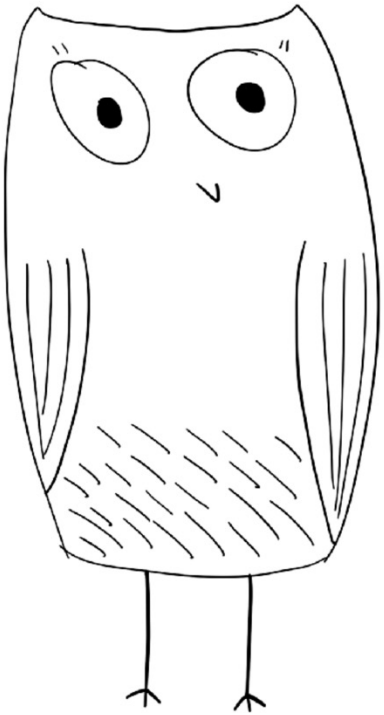
Spring 2024

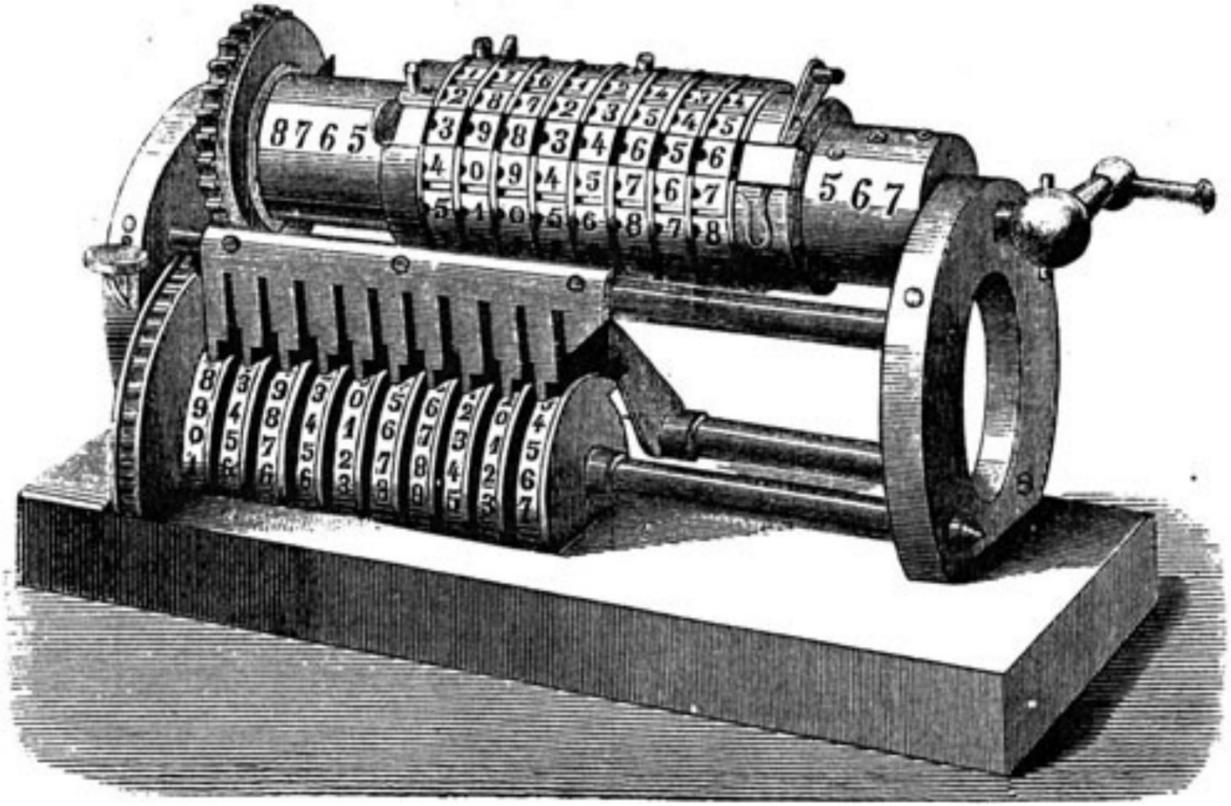
Drawing the Bayesian Owl

1. Theoretical estimand
2. Scientific (causal) model(s)
3. Use 1 & 2 to build statistical model(s)
4. Simulate from 2 to validate 3 yields 1
5. Analyze real data

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

aNAlYzE rEAl DatA

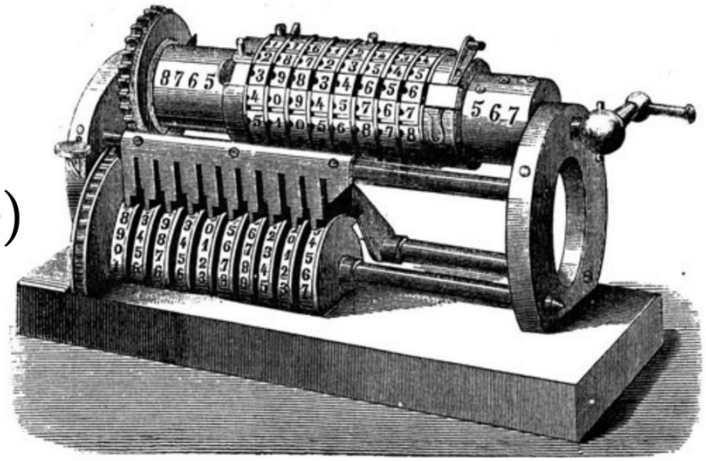




Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Computing the posterior

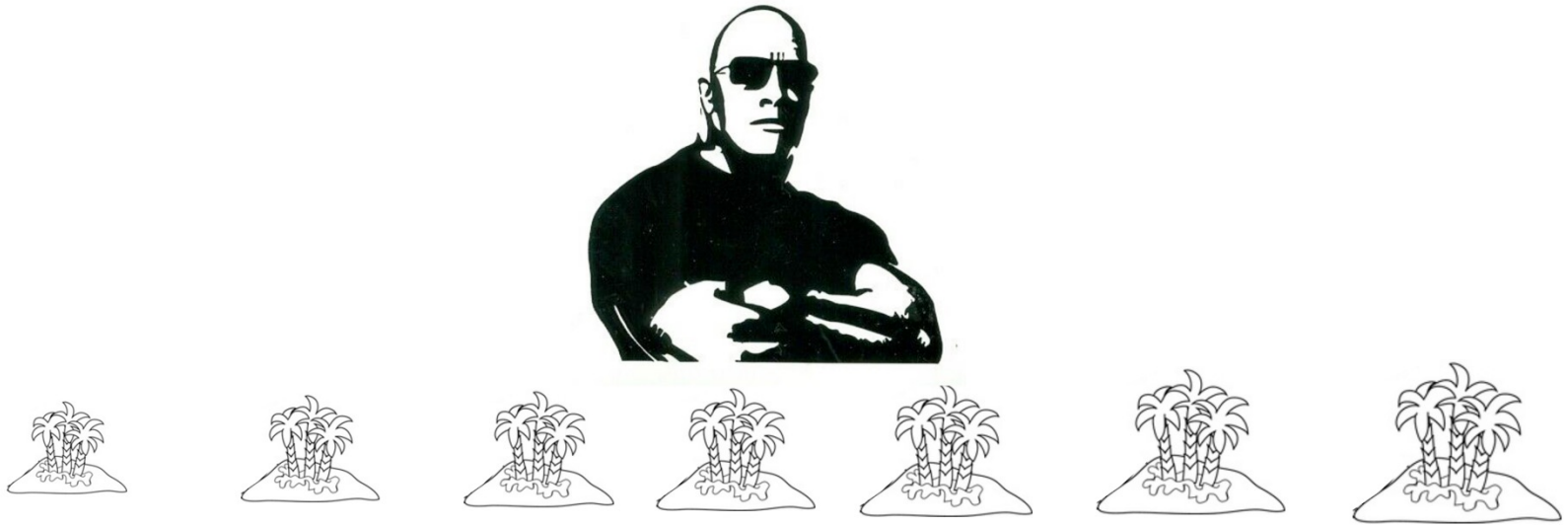
1. Analytical approach (often impossible)
2. Grid approximation (very intensive)
3. Quadratic approximation (limited)
4. Markov chain Monte Carlo (intensive)



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

King Markov

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022



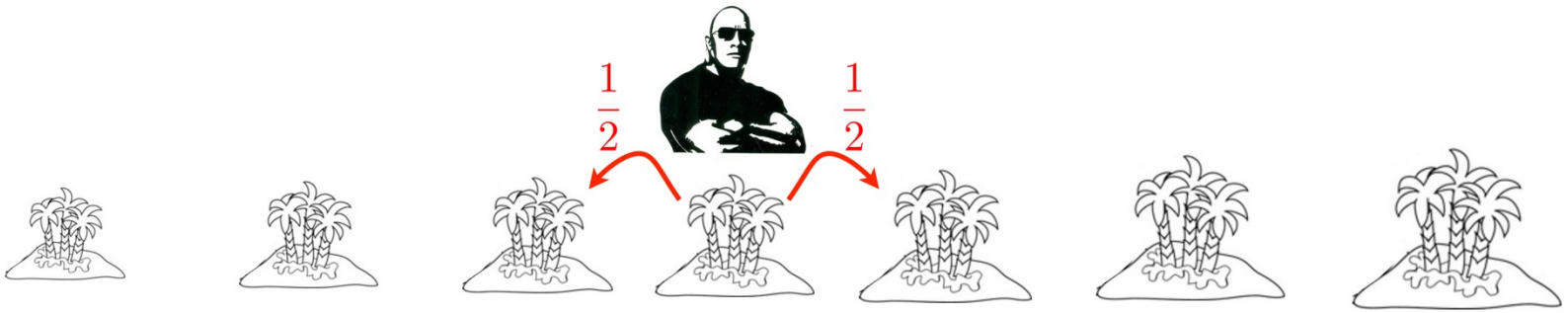
The Metropolis Archipelago

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

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

Here's how he does it...

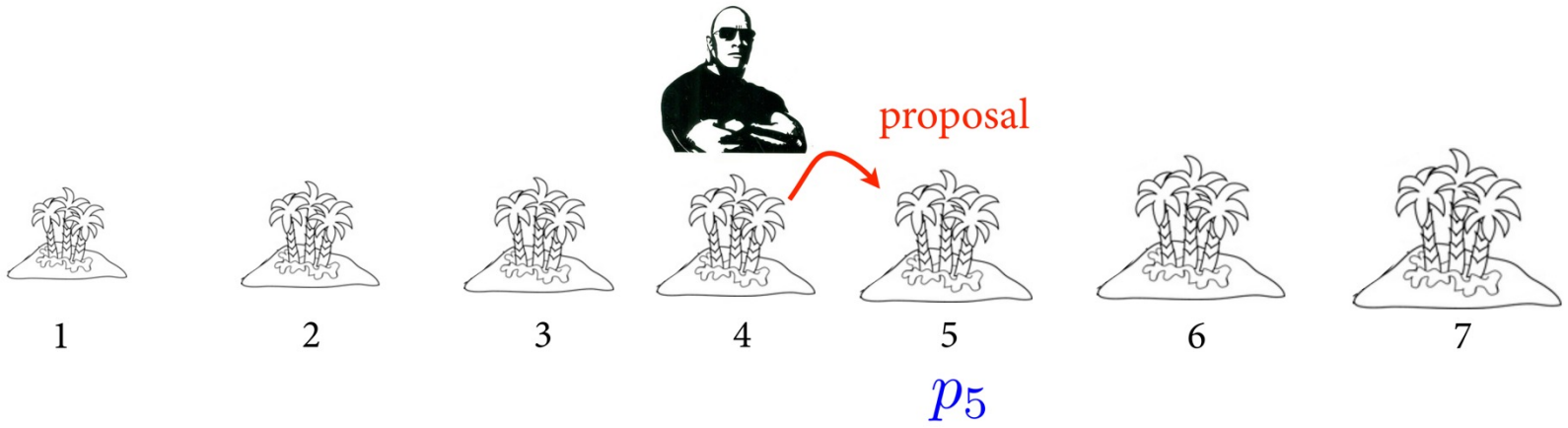
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022



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

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

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

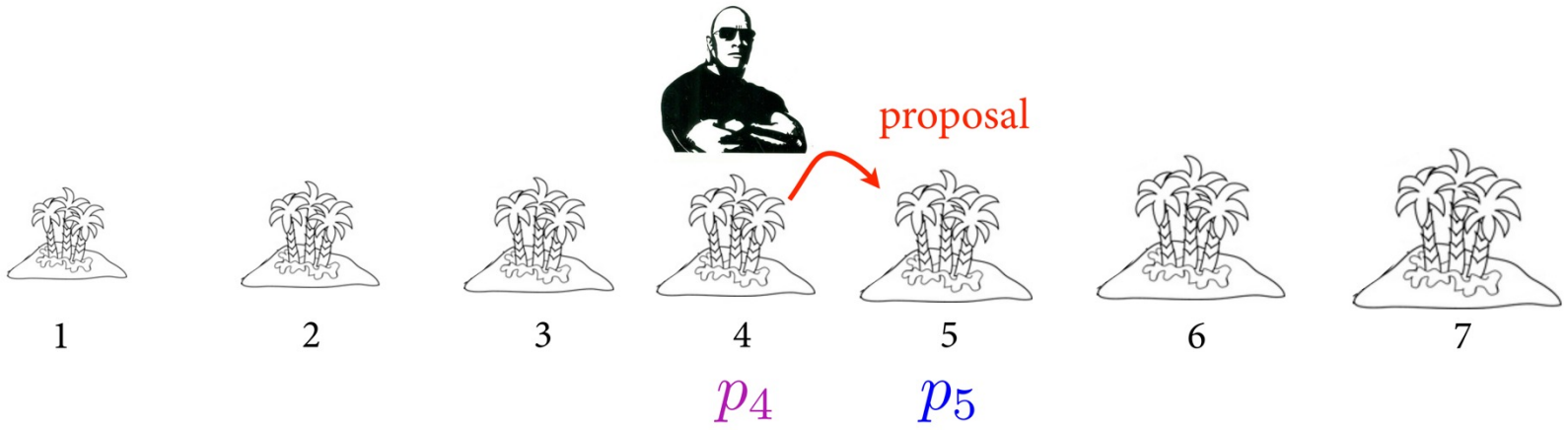


(2) Find population of proposal island.

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

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

(2) Find population of proposal island.



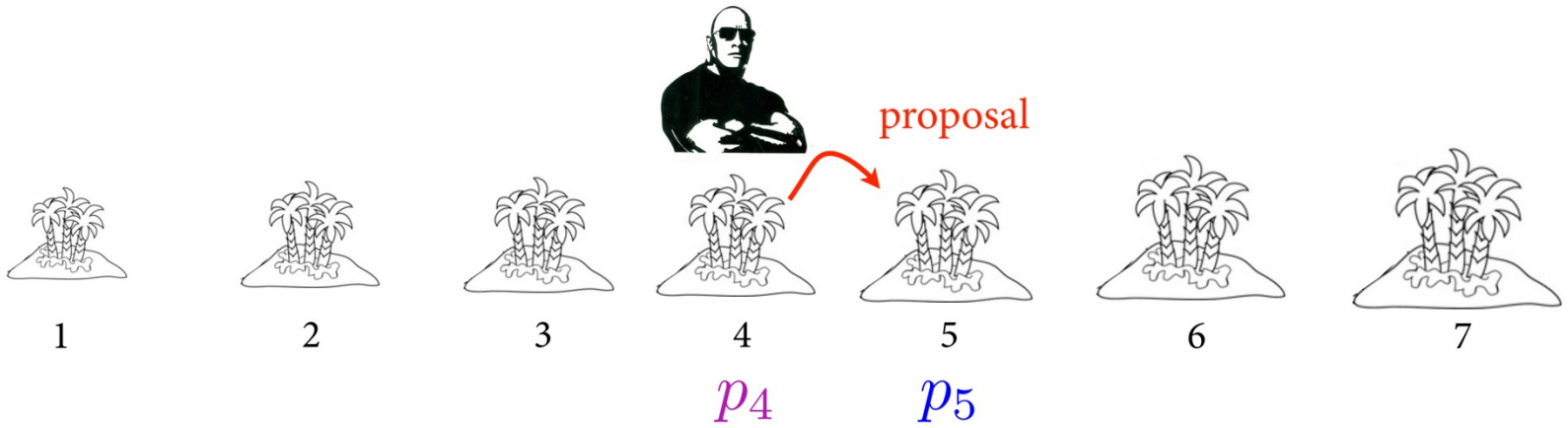
(3) Find population of current island.

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

(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 = $\frac{p_5}{p_4}$

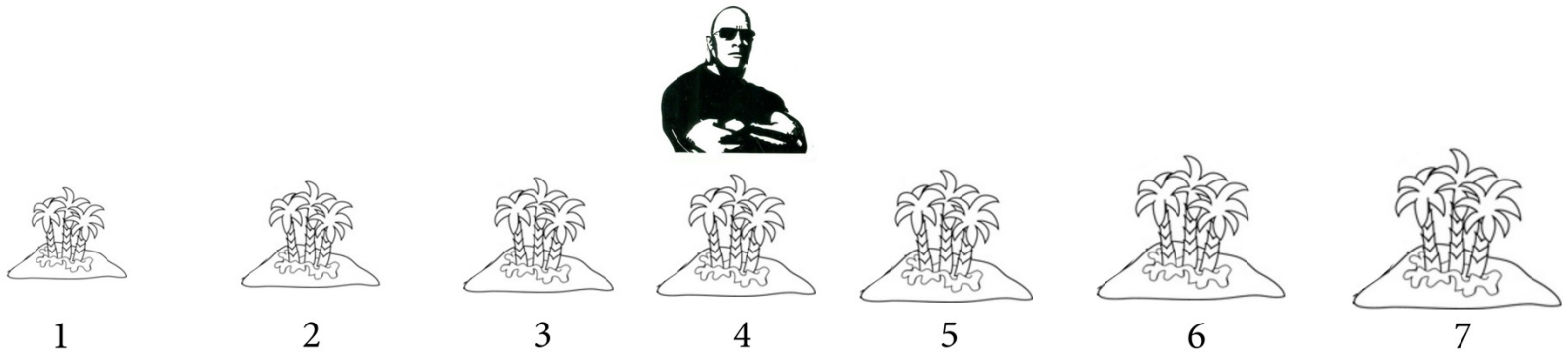
Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

(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)

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

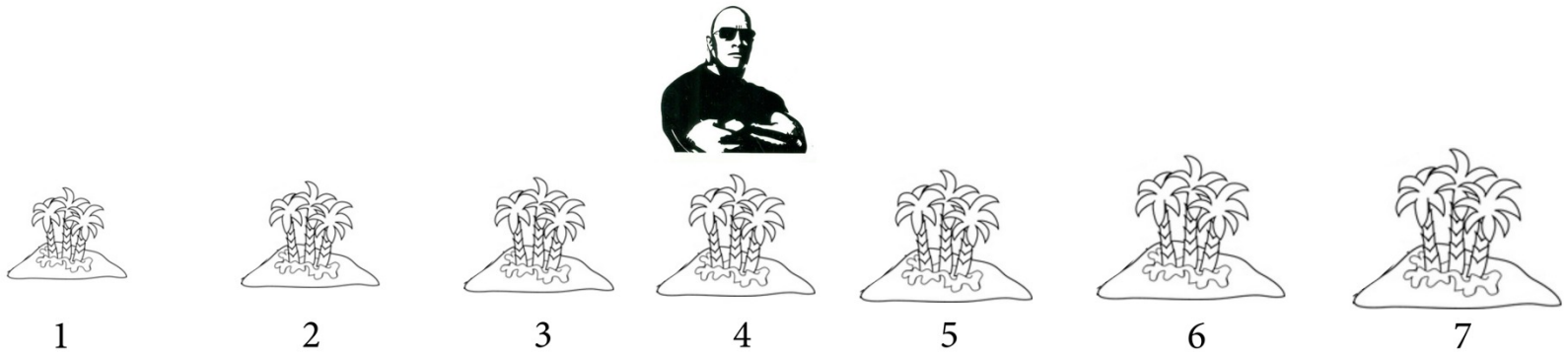
(1) Flip a coin to choose island on left or right.
Call it the "proposal" island.

(2) Find population of proposal island.

(3) Find population of current island.

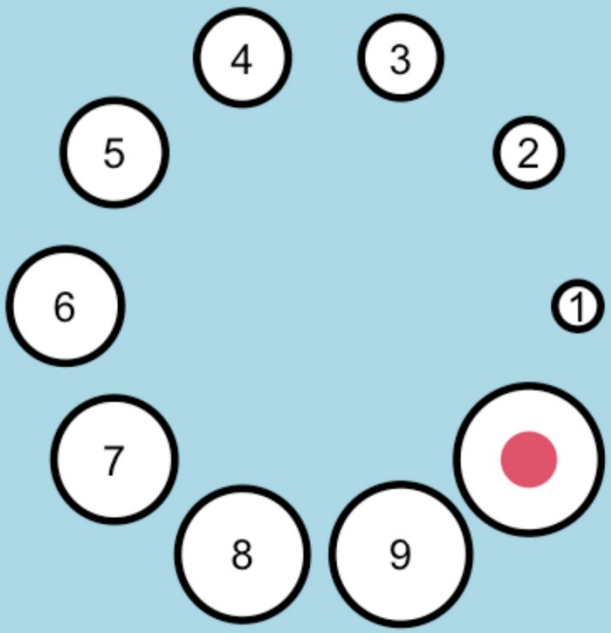
(4) Move to proposal, with probability = $\frac{p_5}{p_4}$

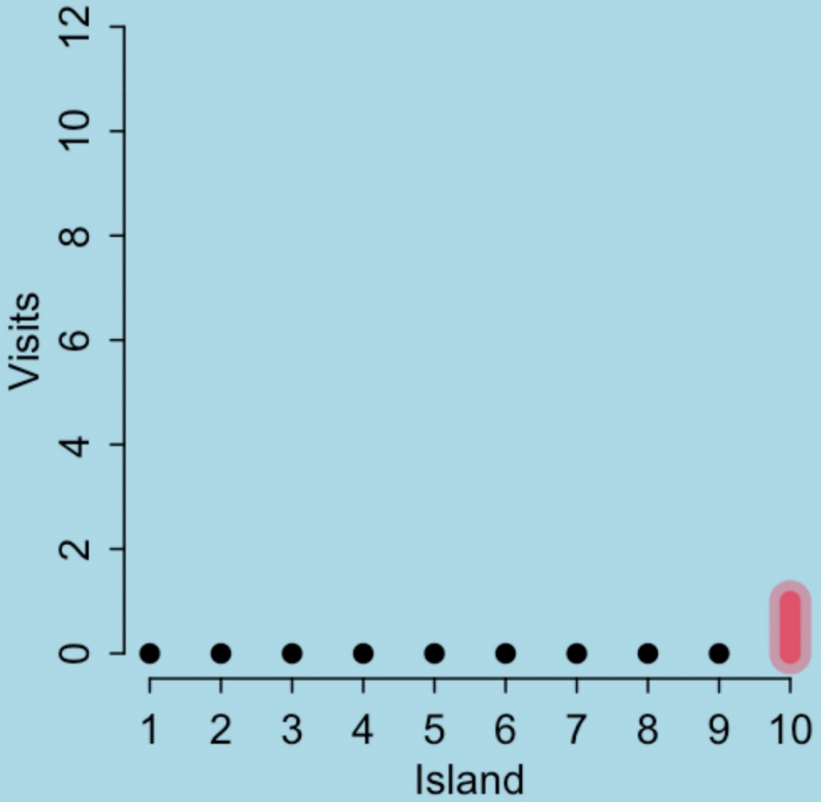
(5) Repeat from (1)



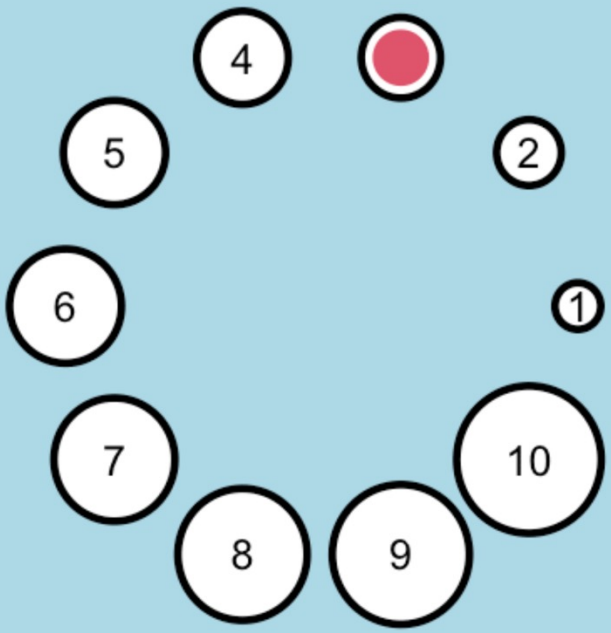
This procedure ensures visiting each island in proportion to its population, *in the long run*.

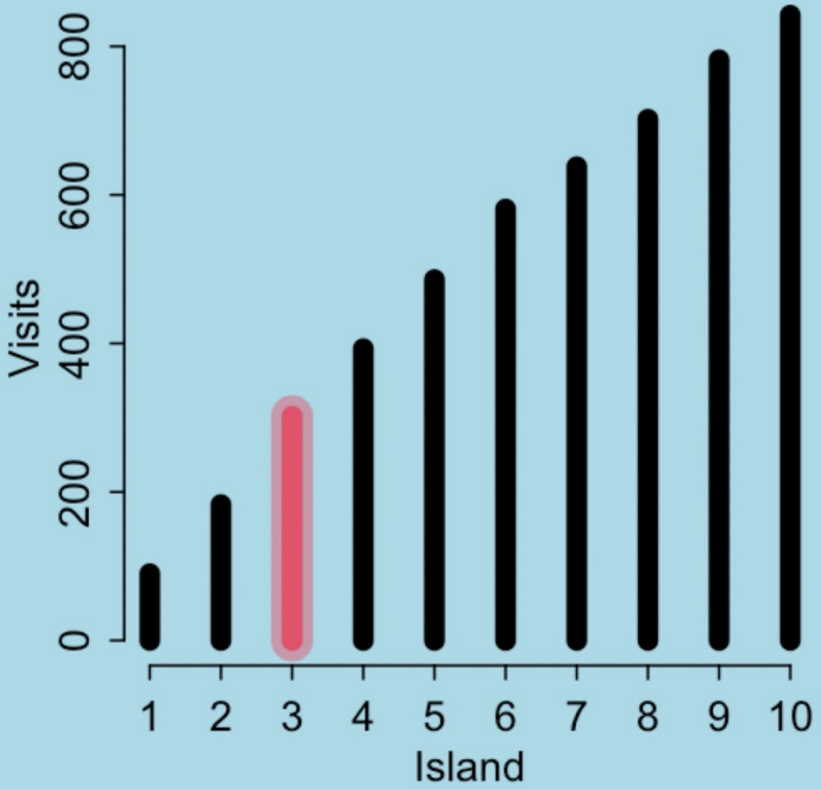
Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022





Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022





Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Markov chain Monte Carlo

Usual use: Draw samples from a posterior distribution

“Islands”: parameter values

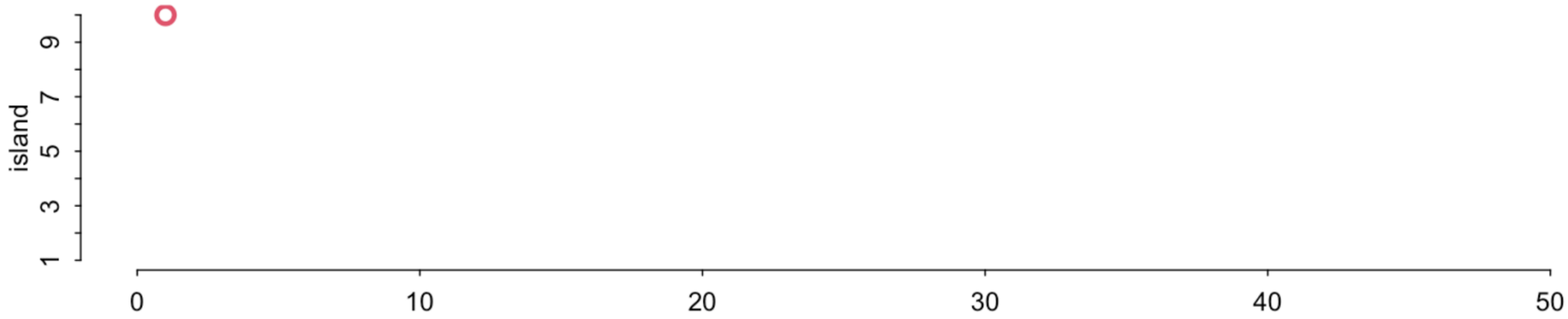
“Population size”: posterior probability

Visit each parameter value in proportion to its posterior probability

Any number of dimensions (parameters)

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

“Markov chain Monte Carlo”



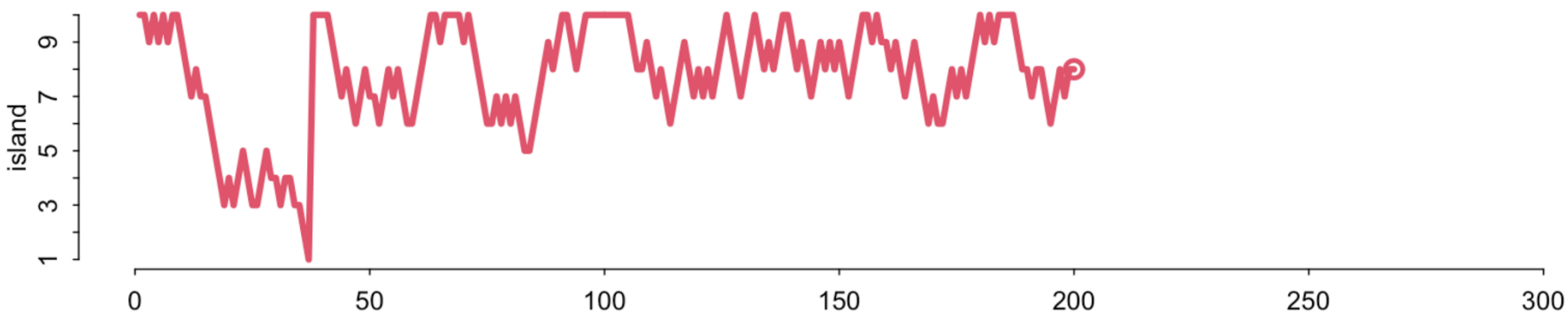
Chain: Sequence of draws from distribution

Markov chain: History doesn’t matter, just where you are now

Monte Carlo: Random simulation

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

“Markov chain Monte Carlo”



Metropolis algorithm: Simple version of *Markov chain Monte Carlo* (MCMC)

Easy to write, very general, often inefficient

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953)

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953)

Equation of state calculations by fast computing machines

…, AW **Rosenbluth**, MN **Rosenbluth**… - The journal of …, **1953** - aip.scitation.org

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

☆ Save Cite Cited by 47206 Related articles All 29 versions

Equa

NICHOLAS METR

ER,

EI

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

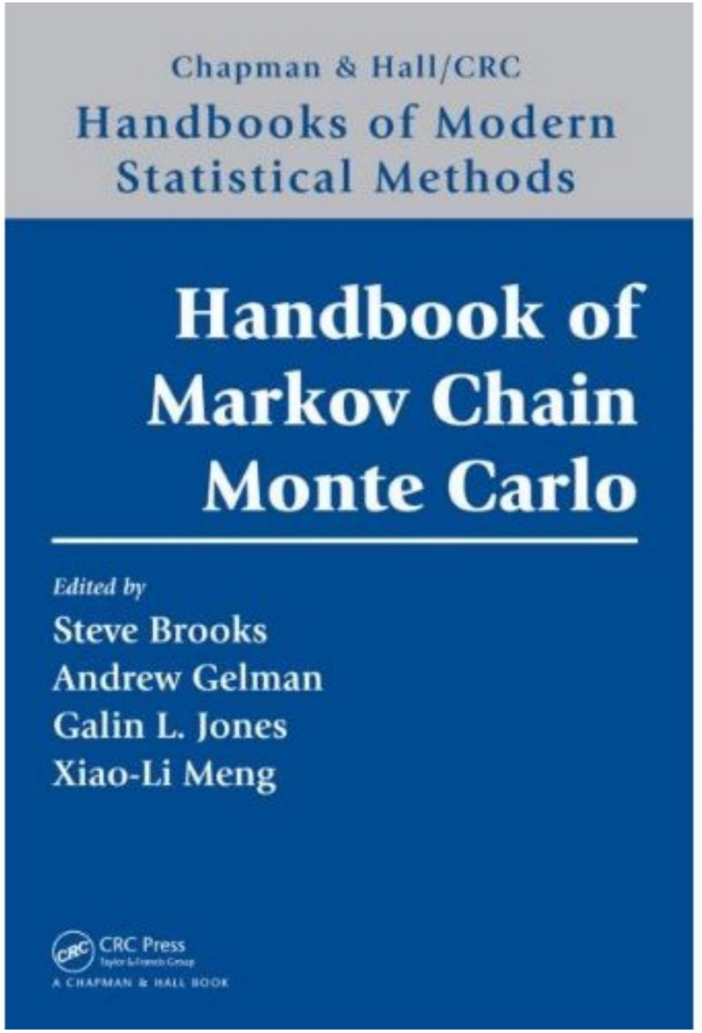
MCMC is diverse

Metropolis has yielded to newer, more efficient algorithms

Many innovations in the last decades

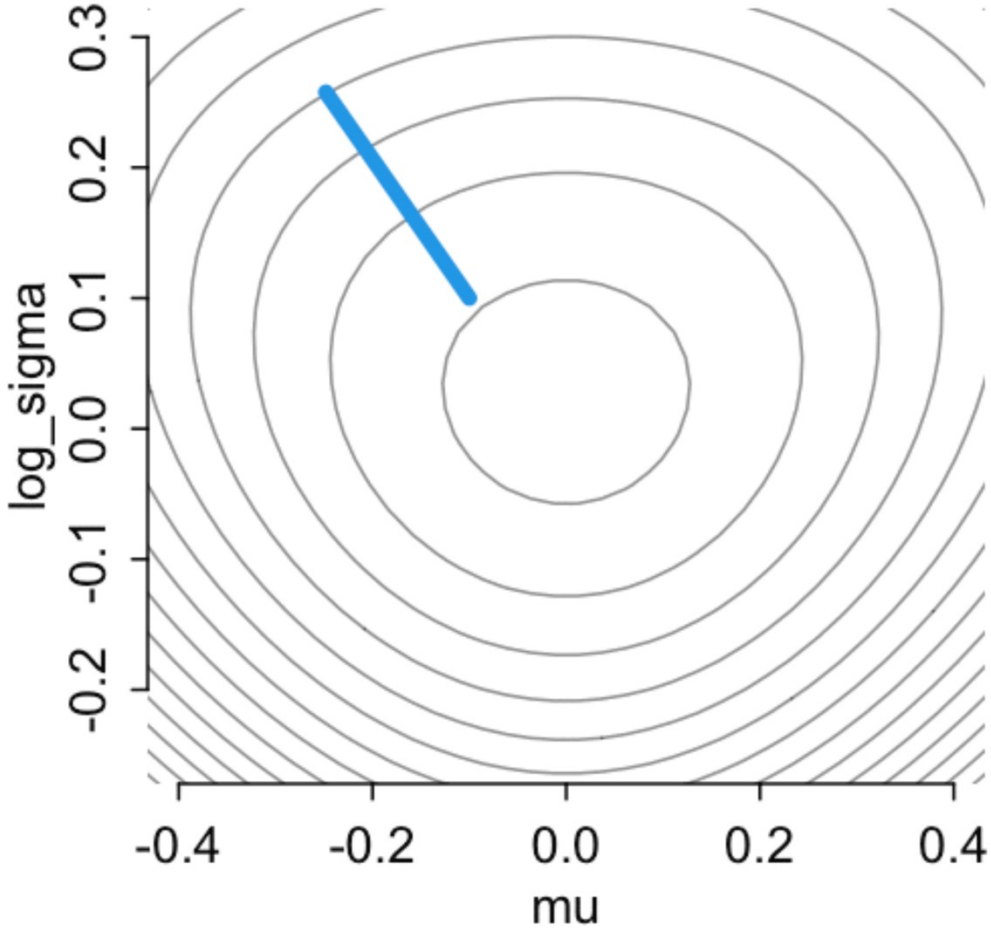
Best methods use ***gradients***

We'll use Hamiltonian Monte Carlo



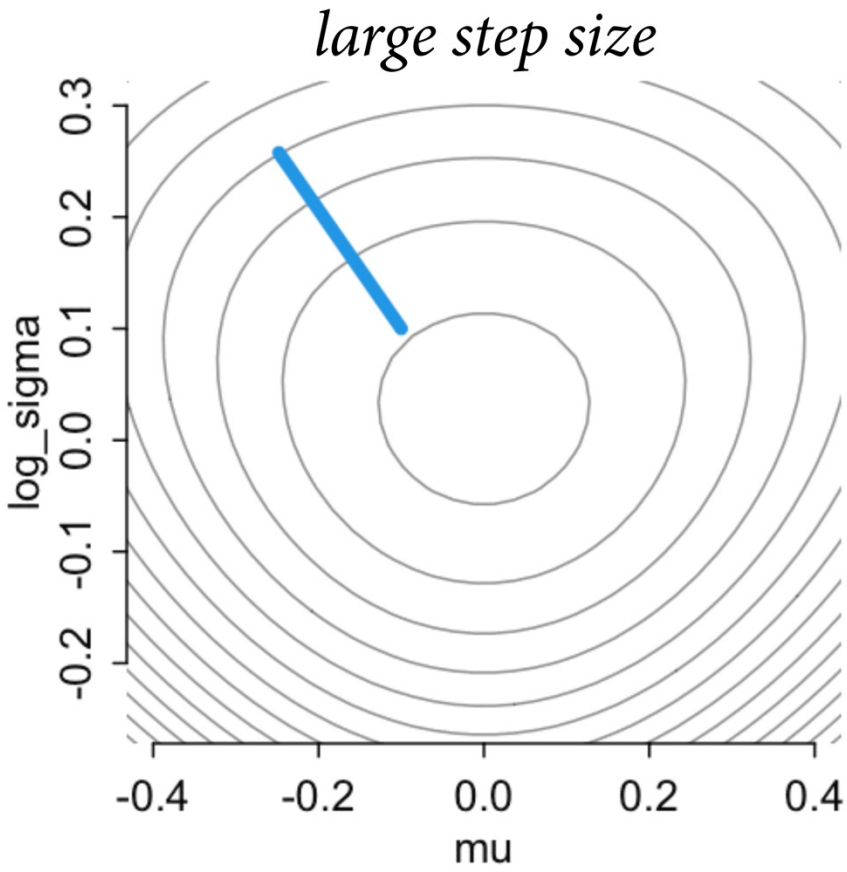
Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

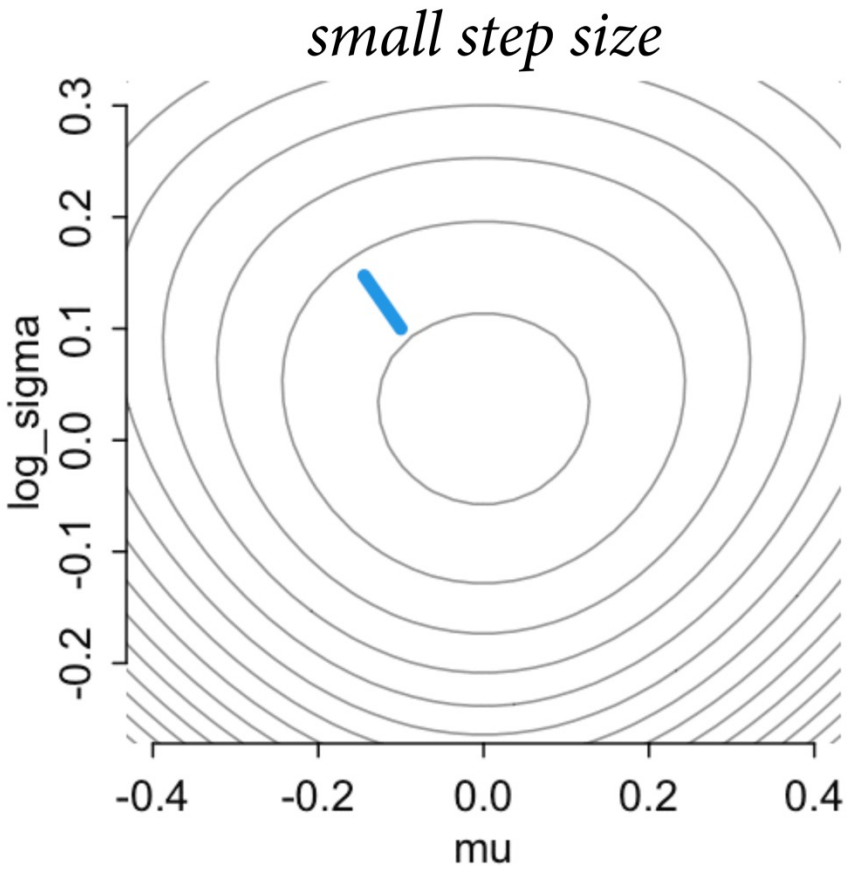
Basic Rosenbluth (aka Metropolis) algorithm



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

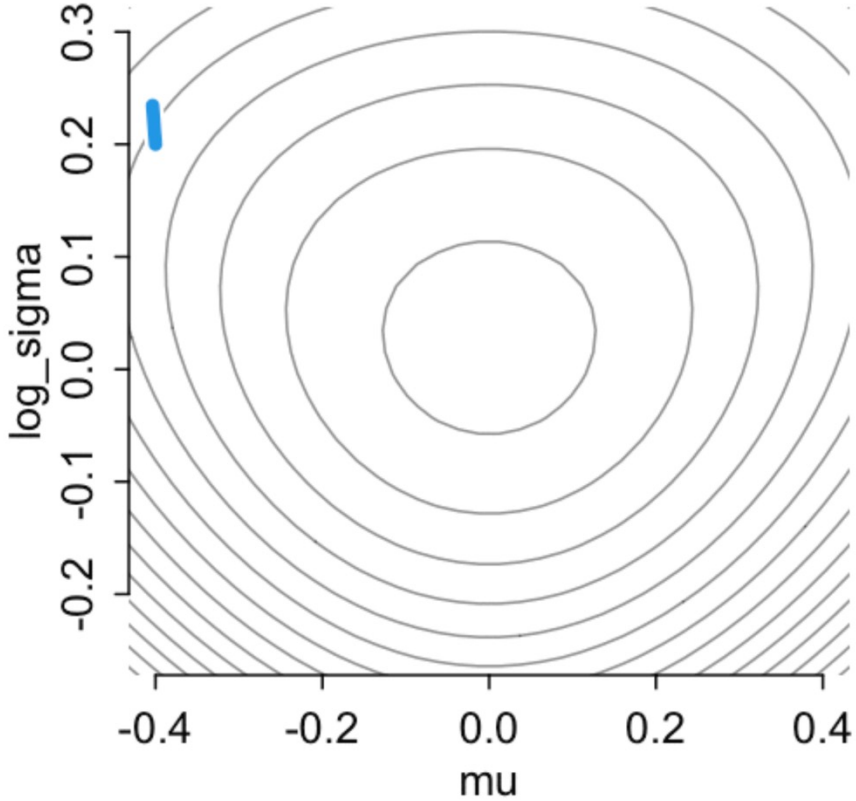
Basic Rosenbluth (aka Metropolis) algorithm





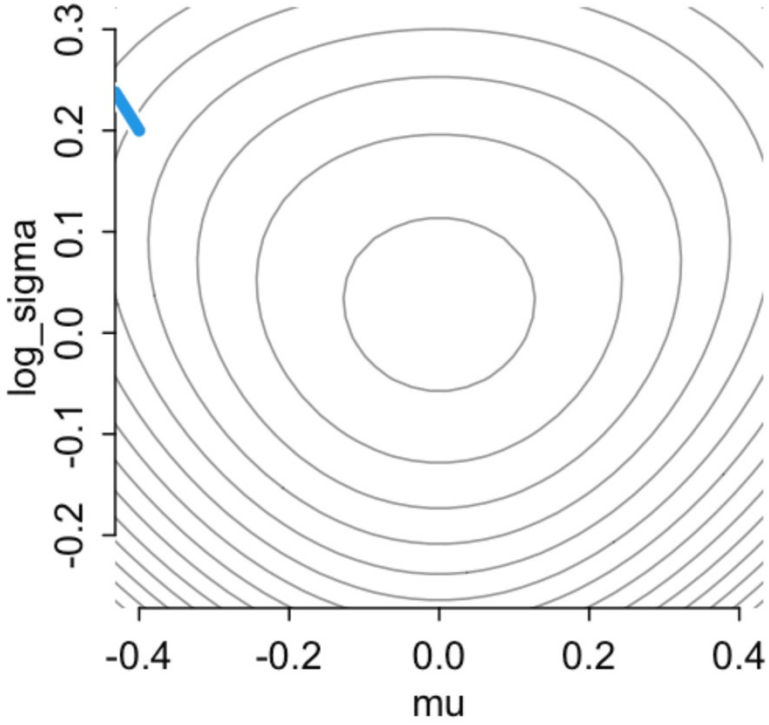
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

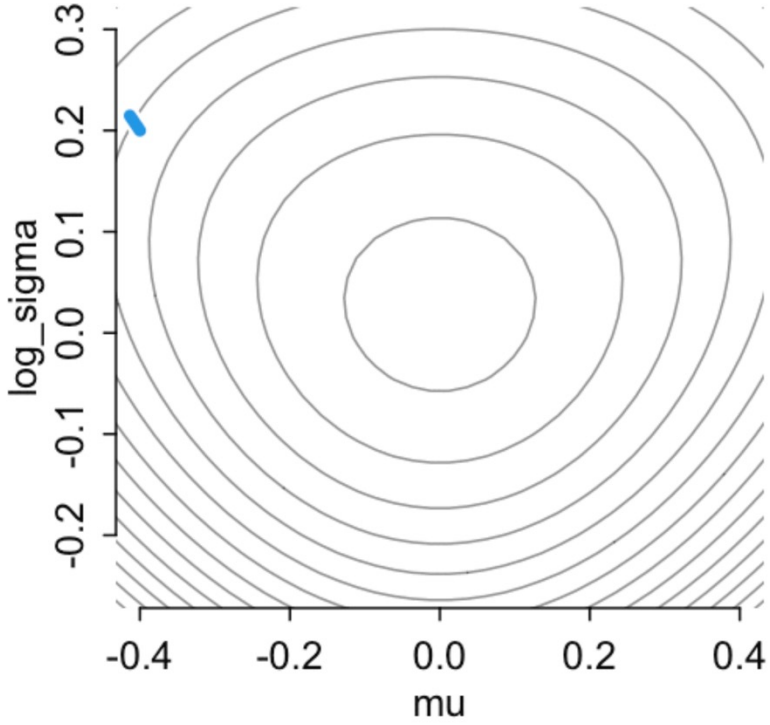
Hamiltonian Monte Carlo



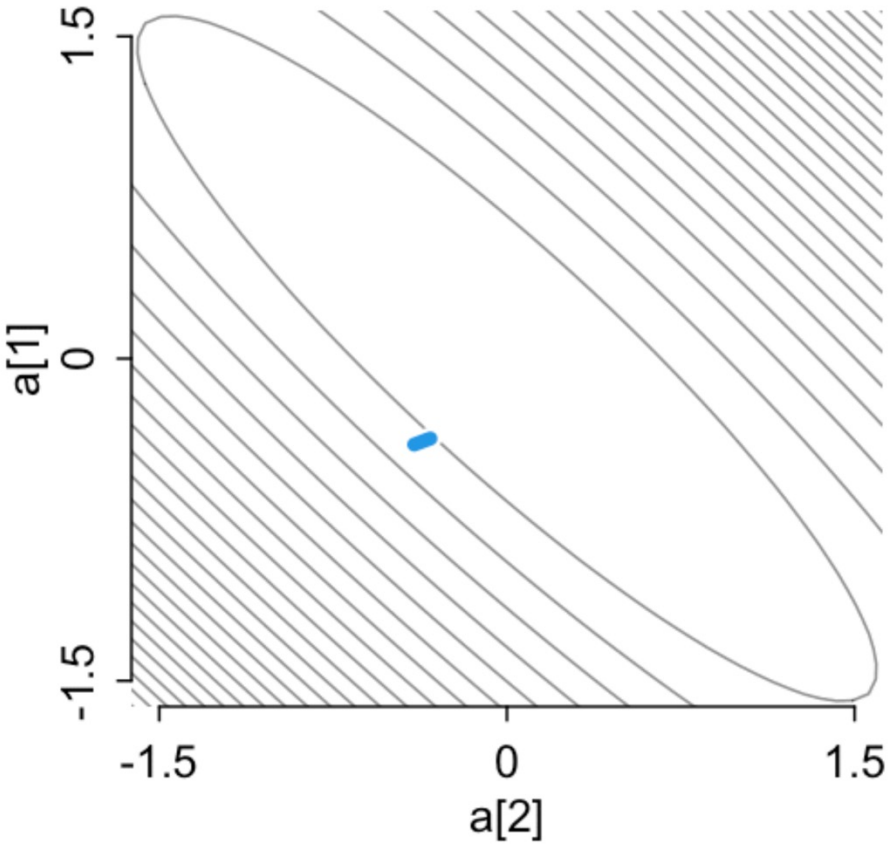
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Hamiltonian Monte Carlo





Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Pages 276–278

Overthinking: Hamiltonian Monte Carlo in the raw. The HMC algorithm needs five things to go: (1) a function `U` that returns the negative log-probability of the data at the current position (parameter values), (2) a function `grad_U` that returns the *gradient* of the negative log-probability at the current position, (3) a step size `epsilon`, (4) a count of leapfrog steps `L`, and (5) a starting position `current_q`. Keep in mind that the position is a vector of parameter values and that the gradient also needs to return a vector of the same length. So that these `U` and `grad_U` functions make more sense, let's present them first, built custom for the 2D Gaussian example. The `U` function just expresses the log-posterior, as stated before in the main text:

$$\sum_i \log p(y_i|\mu_y, 1) + \sum_i \log p(x_i|\mu_x, 1) + \log p(\mu_y|0, 0.5) + \log p(\mu_x, 0, 0.5)$$

So it's just four calls to `dnorm` really:

R code 9.5

```
# U needs to return neg-log-probability
U <- function( q , a=0 , b=1 , k=0 , d=1 ) {
    muy <- q[1]
    mux <- q[2]
    U <- sum( dnorm(y,muy,1,log=TRUE) ) + sum( dnorm(x,mux,1,log=TRUE) ) +
        dnorm(muy,a,b,log=TRUE) + dnorm(mux,k,d,log=TRUE)
    return( -U )
}
```

Now the gradient function requires two partial derivatives. Luckily, Gaussian derivatives are very clean. The derivative of the logarithm of any univariate Gaussian with mean a and standard deviation b with respect to a is:

$$\frac{\partial \log \mathrm{N}(y|a, b)}{\partial a} = \frac{y - a}{b^2}$$

And since the derivative of a sum is a sum of derivatives, this is all we need to write the gradients:

$$\frac{\partial U}{\partial \mu_x} = \frac{\partial \log \mathrm{N}(x|\mu_x, 1)}{\partial \mu_x} + \frac{\partial \log \mathrm{N}(\mu_x|0, 0.5)}{\partial \mu_x} = \sum_i \frac{x_i - \mu_x}{1^2} + \frac{0 - \mu_x}{0.5^2}$$

And the gradient for μ_y has the same form. Now in code form:

R code 9.6

```
# gradient function
# need vector of partial derivatives of U with respect to vector q
U_gradient <- function( q , a=0 , b=1 , k=0 , d=1 ) {
    muy <- q[1]
    mux <- q[2]
    G1 <- sum( y - muy ) + (a - muy)/b^2 #dU/dmuy
    G2 <- sum( x - mux ) + (k - mux)/d^2 #dU/dmux
    return( c( -G1 , -G2 ) ) # negative bc energy is neg-log-prob
}
# test data
set.seed(7)
y <- rnorm(50)
x <- rnorm(50)
x <- as.numeric(scale(x))
y <- as.numeric(scale(y))
```

The gradient function above isn't too bad for this model. But it can be terrifying for a reasonably complex model. That is why tools like Stan build the gradients dynamically, using the model definition. Now we are ready to visit the heart. To understand some of the details here, you should read Radford Neal's chapter in the *Handbook of Markov Chain Monte Carlo*. Armed with the log-posterior and gradient functions, here's the code to produce FIGURE 9.6:

R code 9.7

```
library(shape) # for fancy arrows
Q <- list()
Q$q <- c(-0.1,0.2)
pr <- 0.3
plot( NULL , ylab="muy" , xlab="mux" , xlim=c(-pr,pr) , ylim=c(-pr,pr) )
step <- 0.03
L <- 11 # 0.03/28 for U-turns --- 11 for working example
n_samples <- 4
path_col <- col.alpha("black",0.5)
points( Q$q[1] , Q$q[2] , pch=4 , col="black" )
for ( i in 1:n_samples ) {
    Q <- HMC2( U , U_gradient , step , L , Q$q )
    if ( n_samples < 10 ) {
      for ( j in 1:L ) {
        K0 <- sum(Q$ptraj[j,]^2)/2 # kinetic energy
        lines( Q$traj[j:(j+1),1] , Q$traj[j:(j+1),2] , col=path_col , lwd=1+2*K0 )
      }
      points( Q$traj[1:L+1,] , pch=16 , col="white" , cex=0.35 )
      Arrows( Q$traj[L,1] , Q$traj[L,2] , Q$traj[L+1,1] , Q$traj[L+1,2] ,
          arr.length=0.35 , arr.adj = 0.7 )
      text( Q$traj[L+1,1] , Q$traj[L+1,2] , i , cex=0.8 , pos=4 , offset=0.4 )
    }
    points( Q$traj[L+1,1] , Q$traj[L+1,2] , pch=ifelse( Q$accept==1 , 16 , 1 ) ,
        col=ifelse( abs(Q$dH)>0.1 , "red" , "black" ) )
}
```

The function `HMC2` is built into `rethinking`. It is based upon one of Radford Neal's example scripts.[153] It isn't actually too complicated. Let's tour through it, one step at a time, to take the magic away. This function runs a single trajectory, and so produces a single sample. You need to use it repeatedly to build a chain. That's what the loop above does. The first chunk of the function chooses random momentum—the flick of the particle—and initializes the trajectory.

R code 9.8

```
HMC2 <- function (U, grad_U, epsilon, L, current_q) {
  q = current_q
  p = rnorm(length(q),0,1) # random flick - p is momentum.
  current_p = p
  # Make a half step for momentum at the beginning
  p = p - epsilon * grad_U(q) / 2
  # initialize bookkeeping - saves trajectory
  qtraj <- matrix(NA,nrow=L+1,ncol=length(q))
  ptraj <- qtraj
  qtraj[1,] <- current_q
  ptraj[1,] <- p
```

Then the action comes in a loop over leapfrog steps. `L` steps are taken, using the gradient to compute a linear approximation of the log-posterior surface at each point.

R code 9.9

```
  # Alternate full steps for position and momentum
  for ( i in 1:L ) {
    q = q + epsilon * p # Full step for the position
    # Make a full step for the momentum, except at end of trajectory
    if ( i!=L ) {
        p = p - epsilon * grad_U(q)
        ptraj[i+1,] <- p
    }
```

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Calculus is a superpower

Hamiltonian Monte Carlo needs gradients

How does it get them? Write them yourself or…

Auto-diff: Automatic differentiation

Symbolic derivatives of your model code

Used in many machine learning approaches;
"Backpropagation" is special case

$$J = \begin{bmatrix} \frac{\partial B_x}{\partial x} & \frac{\partial B_y}{\partial x} & \frac{\partial B_z}{\partial x} \\ \frac{\partial B_x}{\partial y} & \frac{\partial B_y}{\partial y} & \frac{\partial B_z}{\partial y} \\ \frac{\partial B_x}{\partial z} & \frac{\partial B_y}{\partial z} & \frac{\partial B_z}{\partial z} \end{bmatrix}$$

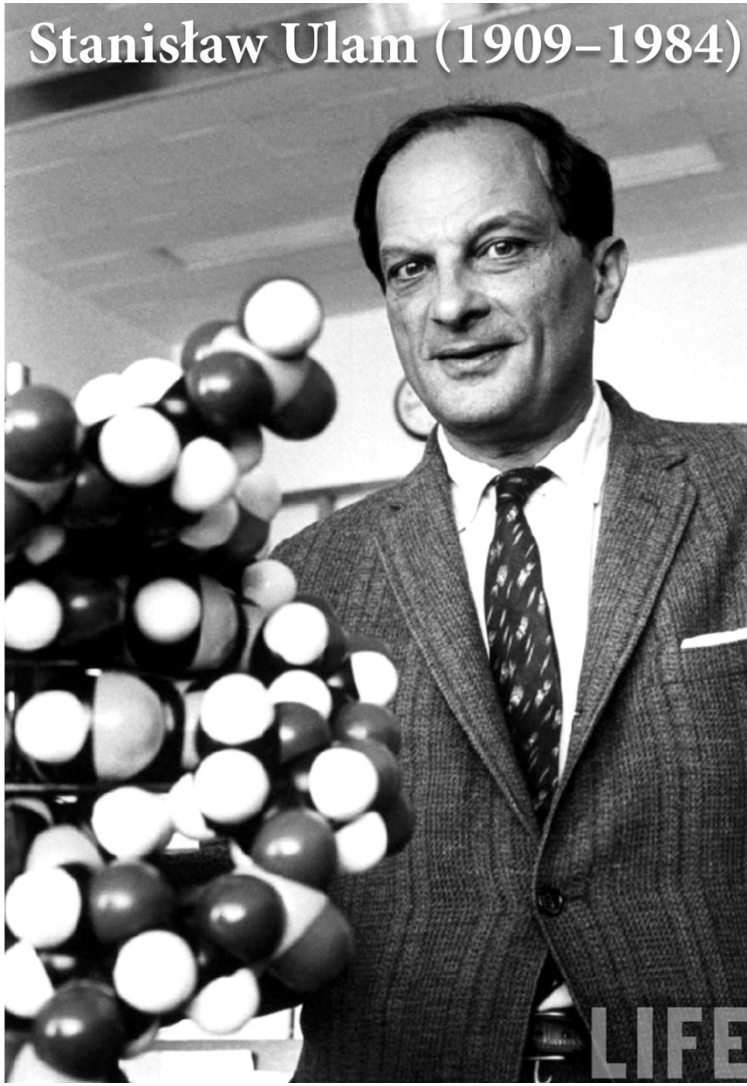
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

mc-stan.org

Stan

About Stan

Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Thousands of users rely on Stan for statistical modeling, data analysis, and prediction in the social, biological, and physical sciences, engineering, and business.



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Example: Divorce data

```
library(rethinking)
data(WaffleDivorce)
d <- WaffleDivorce

dat <- list(
    D = standardize(d$Divorce),
    M = standardize(d$Marriage),
    A = standardize(d$MedianAgeMarriage)
)

f <- alist(
        D ~ dnorm(mu,sigma),
        mu <- a + bM*M + bA*A,
        a ~ dnorm(0,0.2),
        bM ~ dnorm(0,0.5),
        bA ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    )

mq <- quap( f , data=dat )
```

$$
\begin{aligned}
D_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha + \beta_M M_i + \beta_A A_i \\
\alpha &\sim \text{Normal}(0,0.2) \\
\beta_M &\sim \text{Normal}(0,0.5) \\
\beta_A &\sim \text{Normal}(0,0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Example: Divorce data

```
f <- alist(
        D ~ dnorm(mu,sigma),
        mu <- a + bM*M + bA*A,
        a ~ dnorm(0,0.2),
        bM ~ dnorm(0,0.5),
        bA ~ dnorm(0,0.5),
        sigma ~ dexp(1)
    )

mq <- quap( f , data=dat )

library(cmdstanr)
mHMC <- ulam( f , data=dat )
```

$$
\begin{aligned}
D_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha + \beta_M M_i + \beta_A A_i \\
\alpha &\sim \text{Normal}(0,0.2) \\
\beta_M &\sim \text{Normal}(0,0.5) \\
\beta_A &\sim \text{Normal}(0,0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Example: Divorce data

```
f <- alist(
        D ~ dnorm(mu,sigma),
        mu <- a + bM*M + bA*A,
```

```
> precis(mHMC)
            mean   sd  5.5% 94.5% n_eff Rhat4
a           0.00 0.10 -0.16  0.16  1632     1
bM         -0.06 0.17 -0.32  0.21  1137     1
bA         -0.61 0.17 -0.86 -0.34  1160     1
sigma       0.83 0.09  0.70  0.99  1504     1
>
```

```
mq
```

```
library(cmdstanr)
mHMC <- ulam( f , data=dat )
```

$$D_i \sim \text{Normal}(\mu_i, \sigma)$$
$$\mu_i = \alpha + \beta_M M_i + \beta_A A_i$$
$$\alpha \sim \text{Normal}(0,0.2)$$
$$\beta_M \sim \text{Normal}(0,0.5)$$
$$\beta_A \sim \text{Normal}(0,0.5)$$
$$\sigma \sim \text{Exponential}(1)$$

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Pure Stan approach

ulam() just builds Stan code

Stan code is portable, runs on anything

Learn Stan, work in any scripting language

```
// stancode(mHMC)
data{
    // the observed variables
    vector[50] D;
    vector[50] A;
    vector[50] M;
}
parameters{
    // the unobserved variables
    real a;
    real bM;
    real bA;
    real<lower=0> sigma;
}
model{
    // compute the log posterior probability
    vector[50] mu;
    sigma ~ exponential( 1 );
    bA ~ normal( 0 , 0.5 );
    bM ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 0.2 );
    for ( i in 1:50 ) {
        mu[i] = a + bM * M[i] + bA * A[i];
    }
    D ~ normal( mu , sigma );
}
```

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

```
// stancode(mHMC)
data{
    // the observed variables
    vector[50] D;
    vector[50] A;
    vector[50] M;
}
parameters{
    // the unobserved variables
    real a;
    real bM;
    real bA;
    real<lower=0> sigma;
}
model{
    // compute the log posterior probability
    vector[50] mu;
    sigma ~ exponential( 1 );
    bA ~ normal( 0 , 0.5 );
    bM ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 0.2 );
    for ( i in 1:50 ) {
        mu[i] = a + bM * M[i] + bA * A[i];
    }
    D ~ normal( mu , sigma );
}
```

Must declare the type of each observed variable so Stan can catch errors and know what operations are allowed

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

```
// stancode(mHMC)
data{
    // the observed variables
    vector[50] D;
    vector[50] A;
    vector[50] M;
}
parameters{
    // the unobserved variables
    real a;
    real bM;
    real bA;
    real<lower=0> sigma;
}
model{
    // compute the log posterior probability
    vector[50] mu;
    sigma ~ exponential( 1 );
    bA ~ normal( 0 , 0.5 );
    bM ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 0.2 );
    for ( i in 1:50 ) {
        mu[i] = a + bM * M[i] + bA * A[i];
    }
    D ~ normal( mu , sigma );
}
```

Must declare the type of each observed variable so Stan can catch errors and know what operations are allowed

Unobserved variables also need checks and constraints. Declared here.

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

```
// stancode(mHMC)
data{
    // the observed variables
    vector[50] D;
    vector[50] A;
    vector[50] M;
}
parameters{
    // the unobserved variables
    real a;
    real bM;
    real bA;
    real<lower=0> sigma;
}
model{
    // compute the log posterior probability
    vector[50] mu;
    sigma ~ exponential( 1 );
    bA ~ normal( 0 , 0.5 );
    bM ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 0.2 );
    for ( i in 1:50 ) {
        mu[i] = a + bM * M[i] + bA * A[i];
    }
    D ~ normal( mu , sigma );
}
```

Must declare the type of each observed variable so Stan can catch errors and know what operations are allowed

Unobserved variables also need checks and constraints. Declared here.

Declare the distributional parts of the model, sufficient to compute posterior probability

In big models, this part can be very complex

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Pure Stan approach

Save Stan code as own file

```
mHMC_stan <- cstan( file="08_mHMC.stan"
, data=dat )
```

Extract samples and proceed as usual

```
post <- extract.samples(mHMC_stan)
```

```
// stancode(mHMC)
data{
    // the observed variables
    vector[50] D;
    vector[50] A;
    vector[50] M;
}
parameters{
    // the unobserved variables
    real a;
    real bM;
    real bA;
    real<lower=0> sigma;
}
model{
    // compute the log posterior probability
    vector[50] mu;
    sigma ~ exponential( 1 );
    bA ~ normal( 0 , 0.5 );
    bM ~ normal( 0 , 0.5 );
    a ~ normal( 0 , 0.2 );
    for ( i in 1:50 ) {
        mu[i] = a + bM * M[i] + bA * A[i];
    }
    D ~ normal( mu , sigma );
}
```

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

Drawing the Markov Owl

Complex machinery, but lots of diagnostics

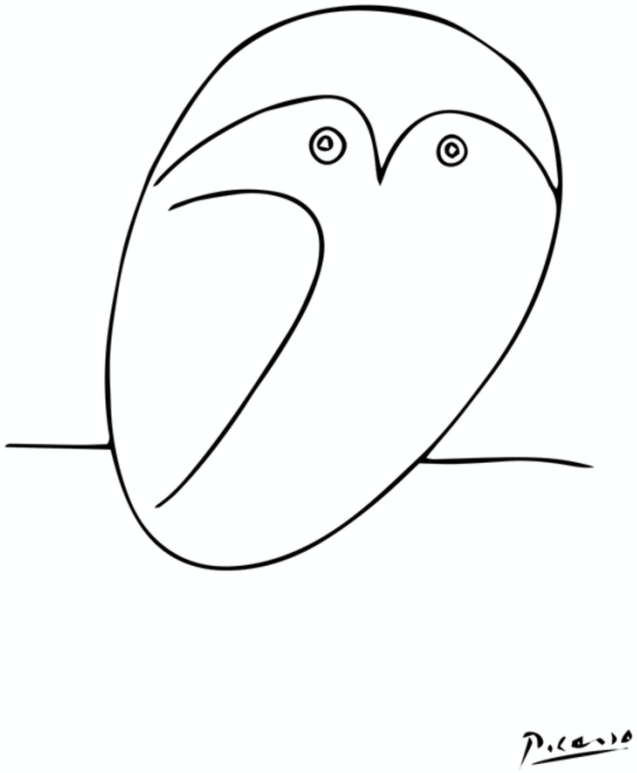
(1) Trace plots

(2) Trace rank plots

(3) R-hat convergence measure

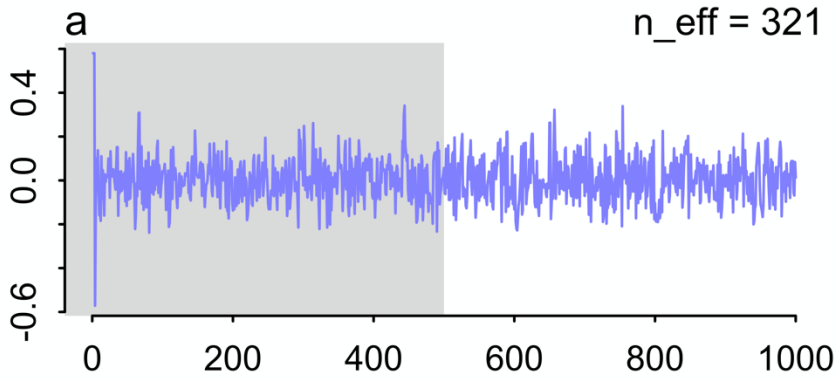
(4) Number of effective samples

(5) Divergent transitions



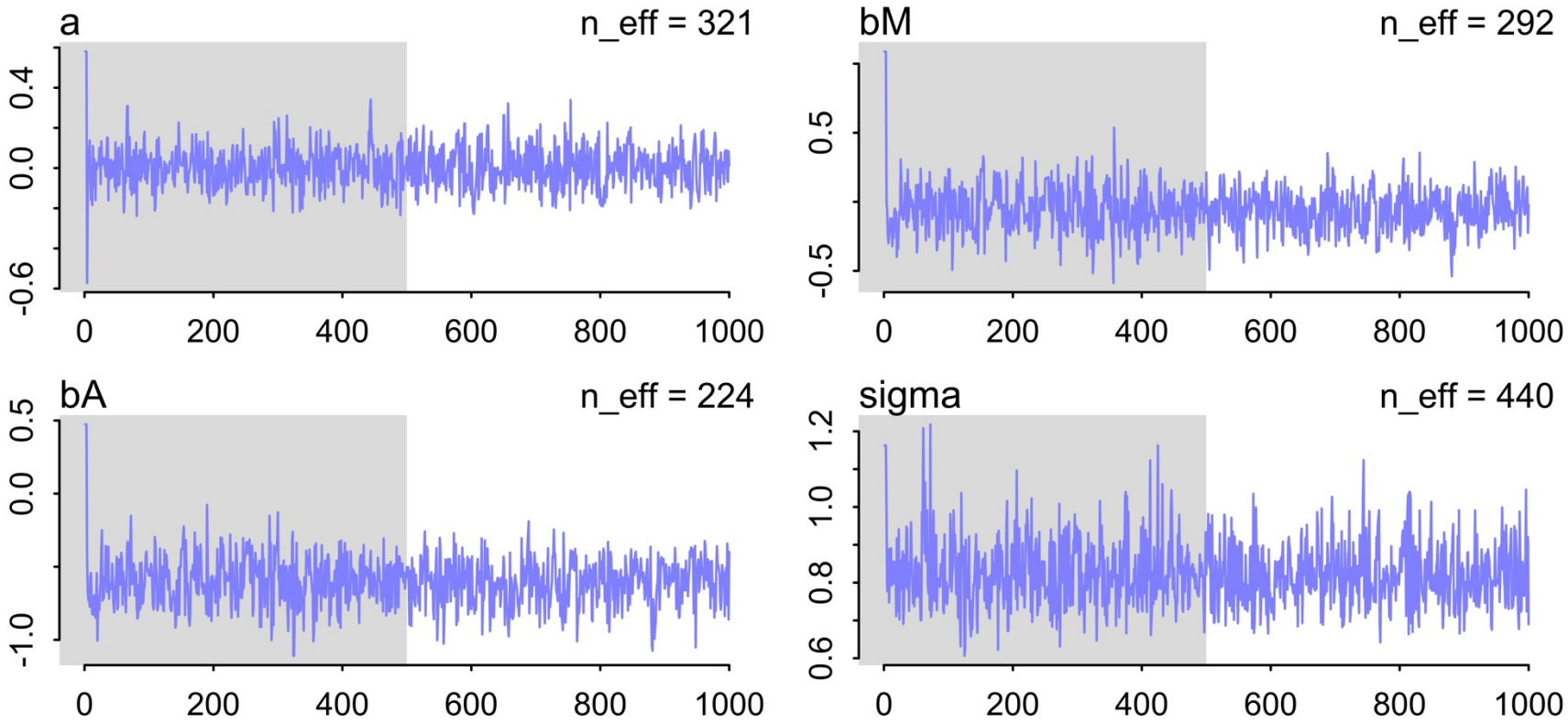
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Need more than 1 chain to check convergence

Convergence: Each chain explores the right distribution and every chain explores the same distribution

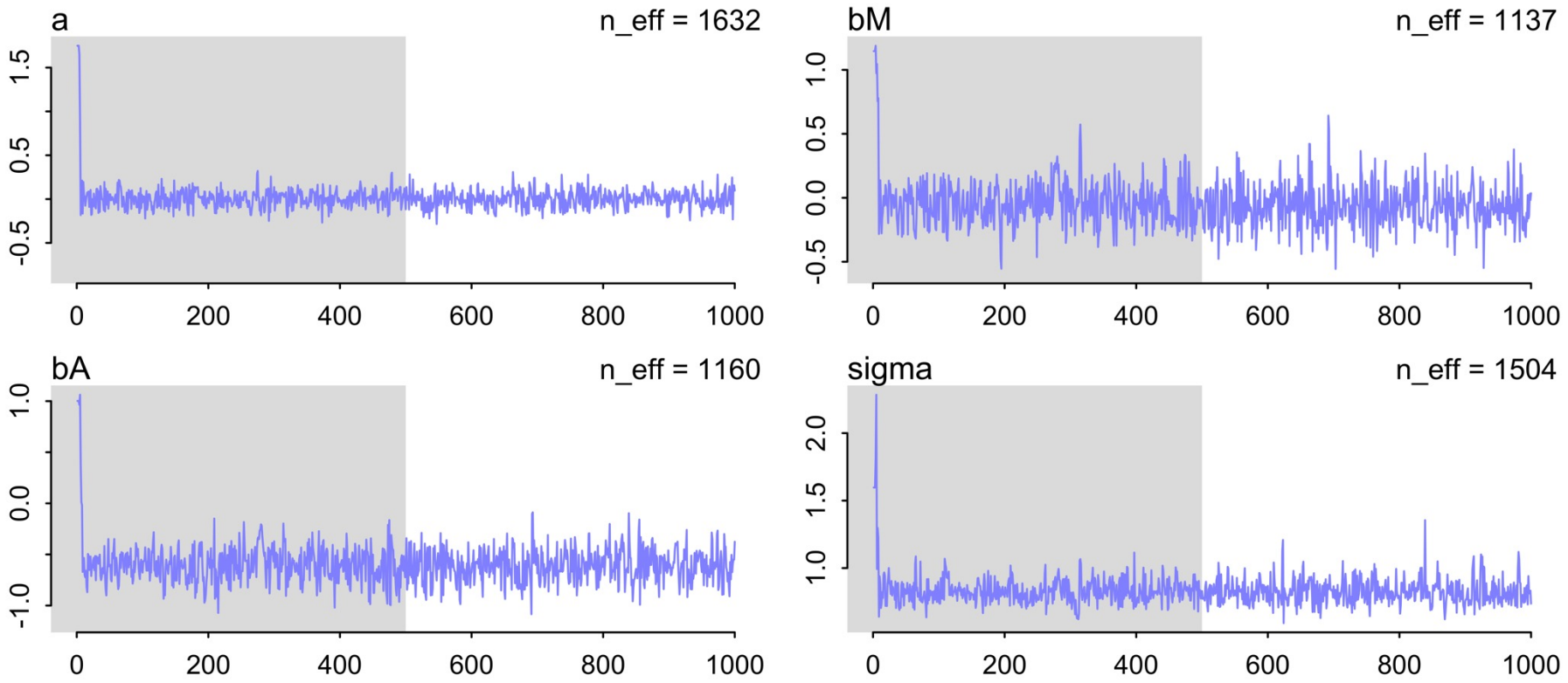
```
library(cmdstanr)
mHMC <- ulam( f , data=dat )

mHMC <- ulam( f , data=dat , chains=4 , cores=4 )
```

$$
\begin{aligned}
D_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha + \beta_M M_i + \beta_A A_i \\
\alpha &\sim \text{Normal}(0,0.2) \\
\beta_M &\sim \text{Normal}(0,0.5) \\
\beta_A &\sim \text{Normal}(0,0.5) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

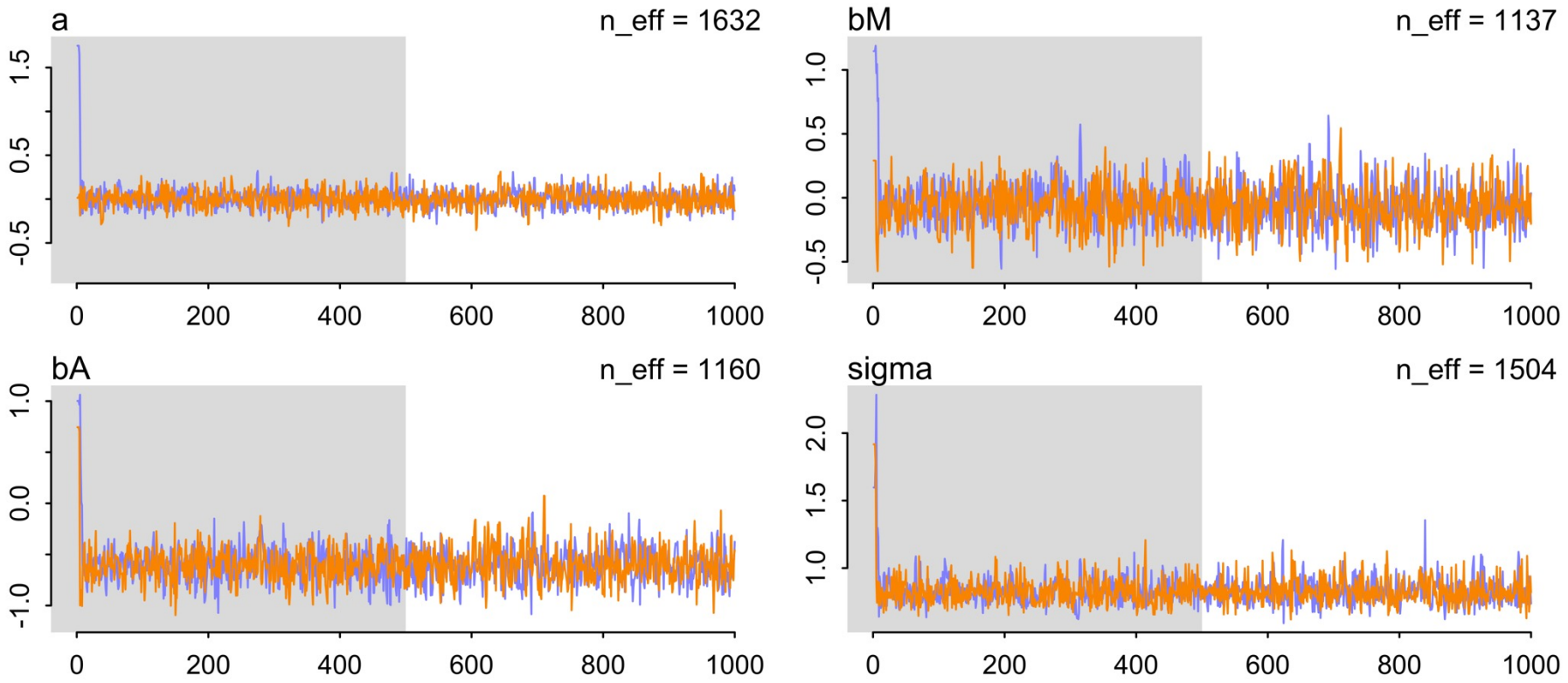
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



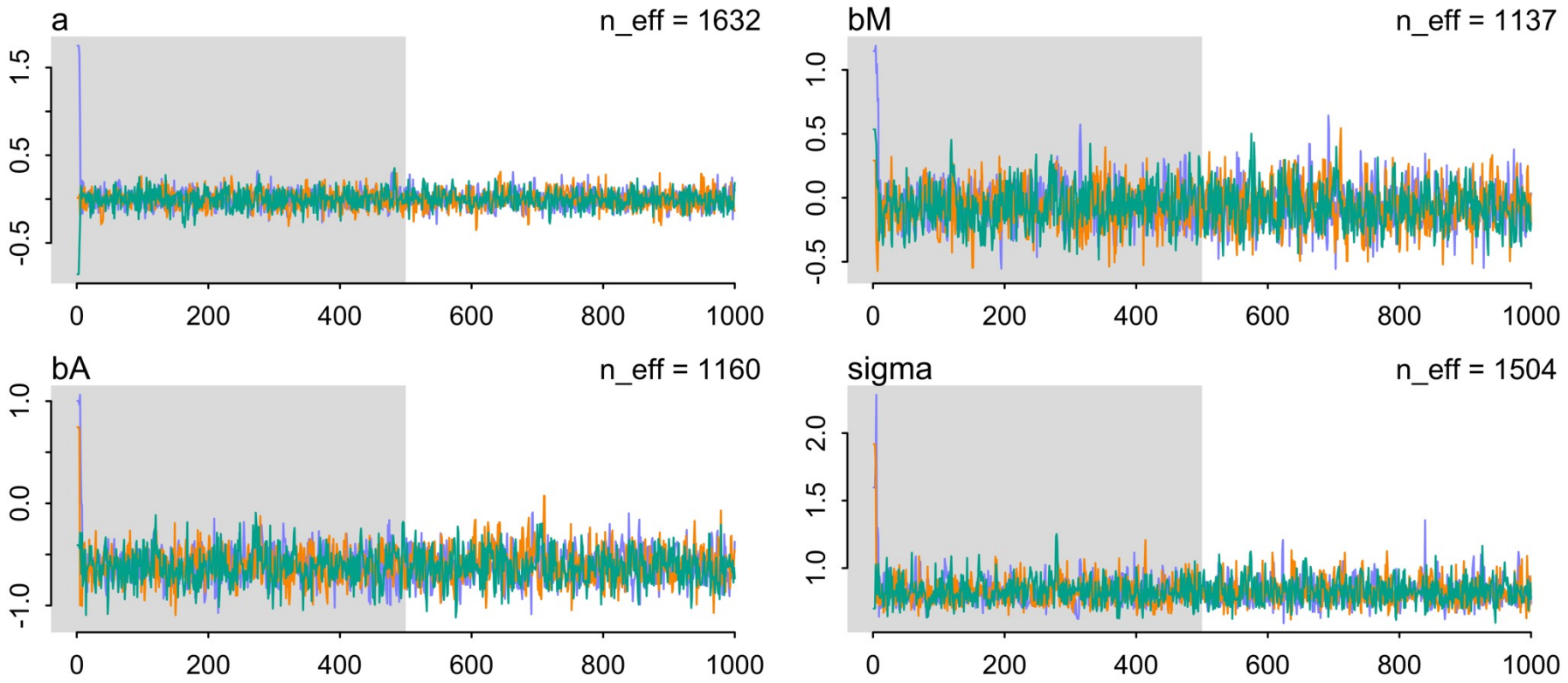
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



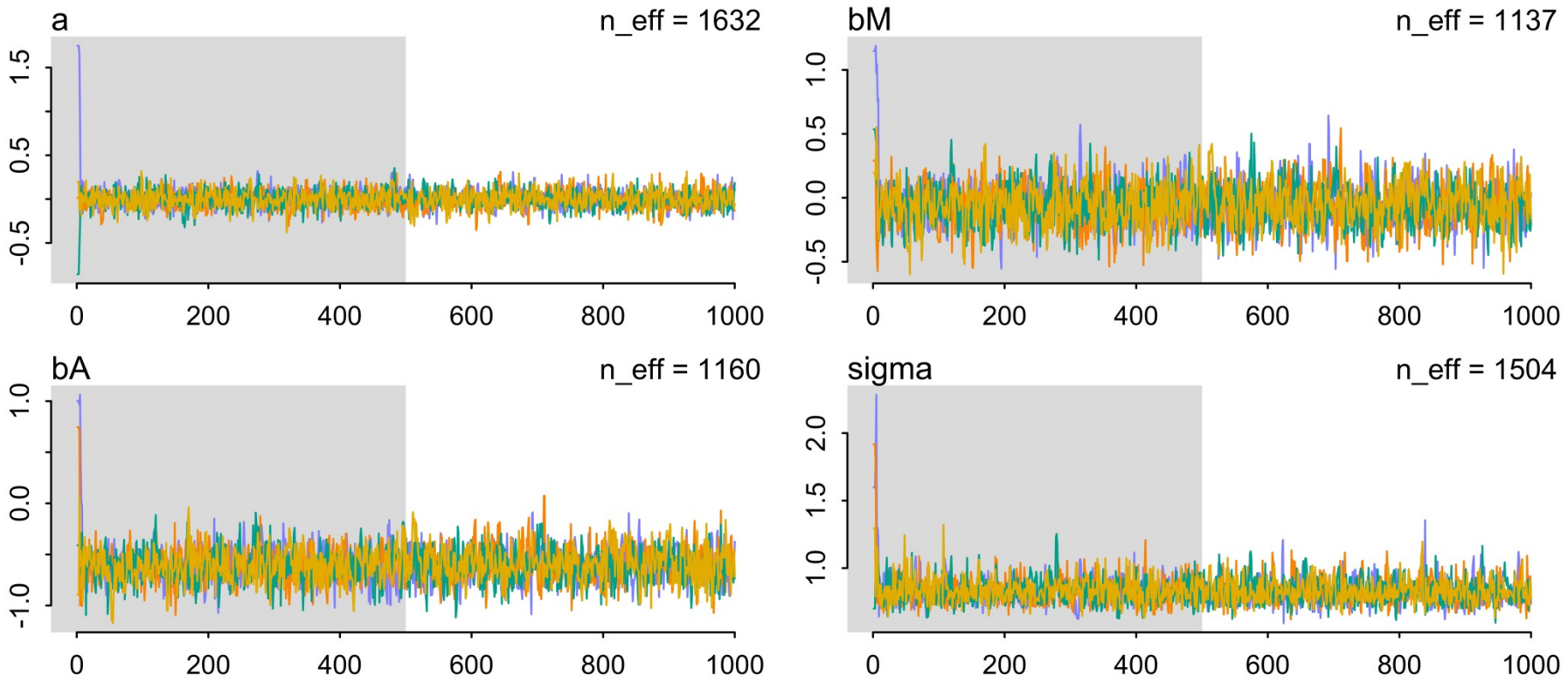
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



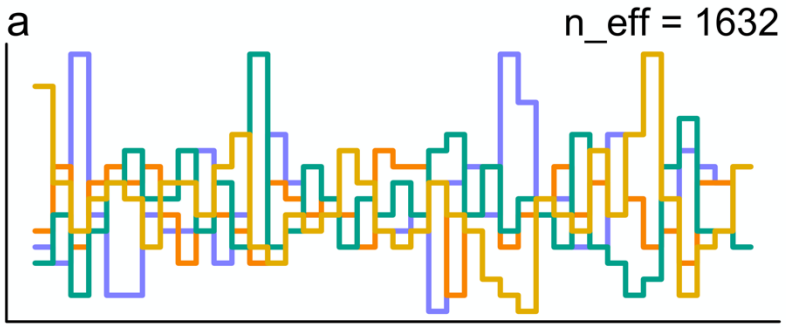
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace plots



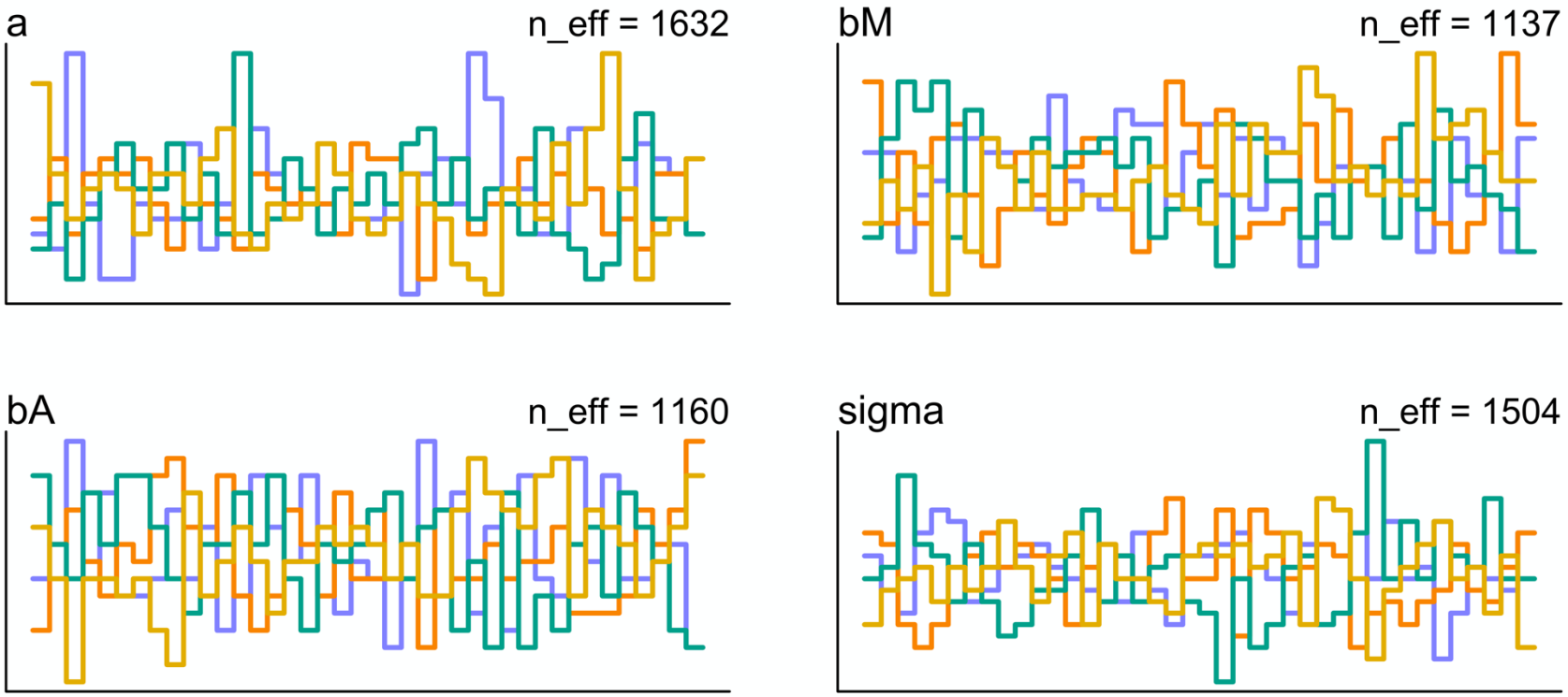
Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace rank (Trank) plots



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Trace rank (Trank) plots



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

R-hat

When chains converge:

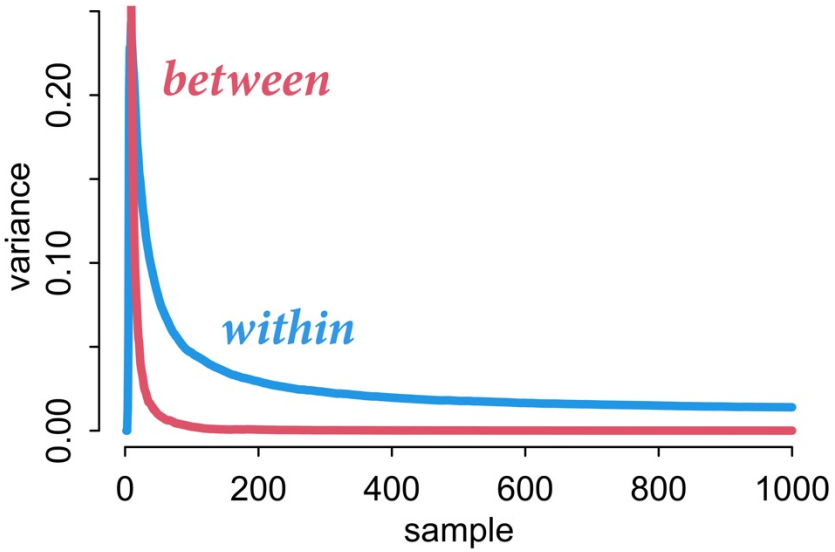
(1) Start and end of each chain explores same region

(2) Independent chains explore same region

R-hat is a ratio of variances:
As total variance shrinks to average variance within chains, R-hat approaches 1

NO GUARANTEES; NOT A TEST

```
> precis(mHMC)
       mean   sd  5.5% 94.5% n_eff Rhat4
a      0.00 0.10 -0.16  0.16  1632     1
bM    -0.06 0.17 -0.32  0.21  1137     1
bA    -0.61 0.17 -0.86 -0.34  1160     1
sigma  0.83 0.09  0.70  0.99  1504     1
>
```



Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022

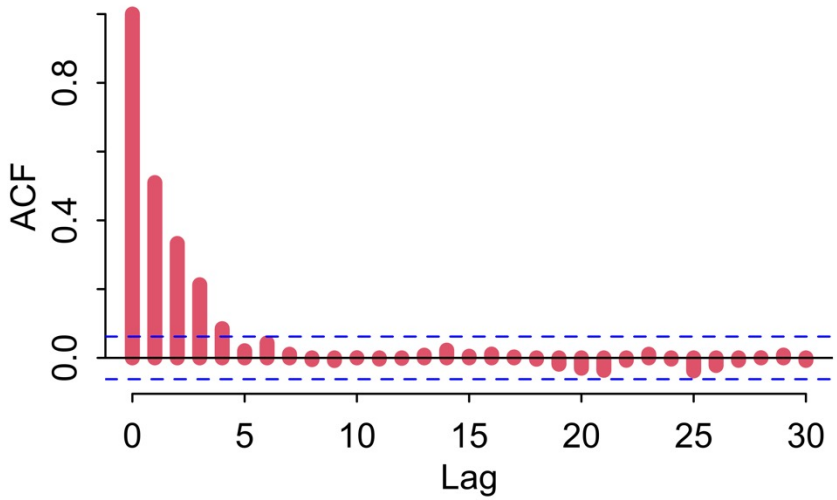
n_eff

Estimate of number of effective samples

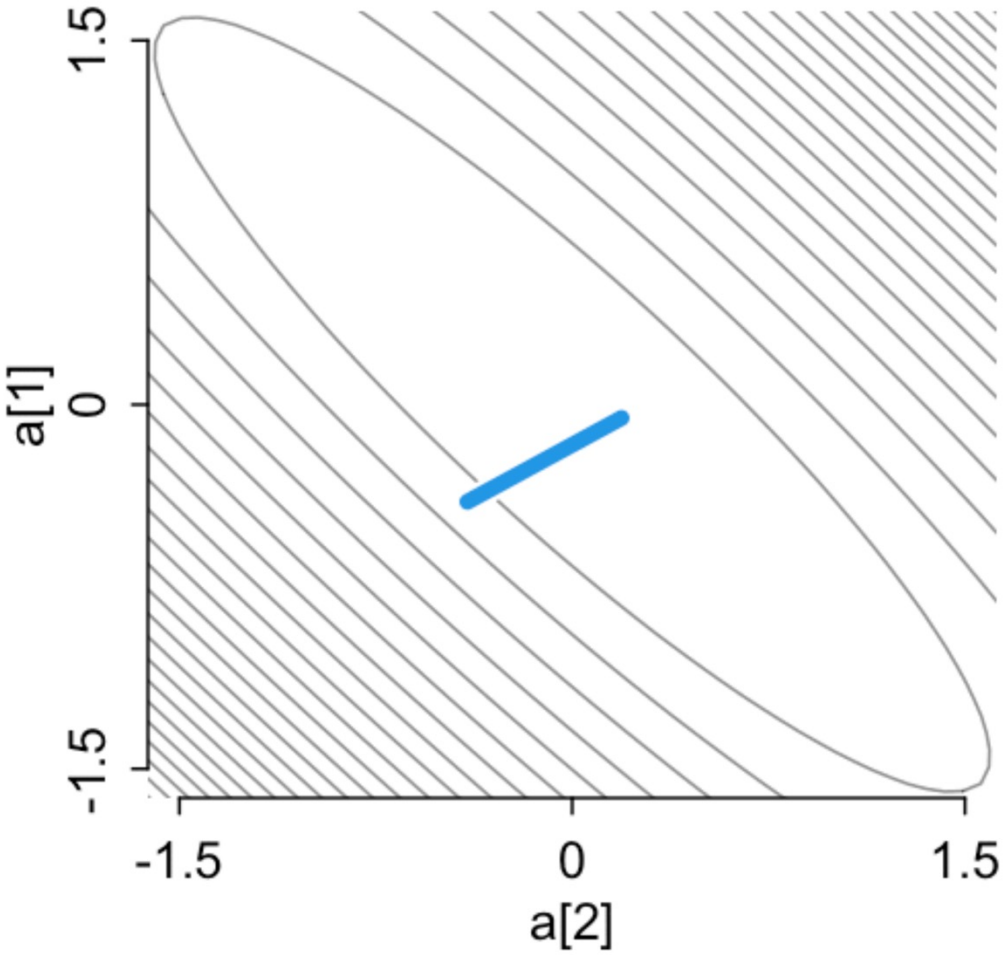
"How long would the chain be, if each sample was independent of the one before it?"

When samples are autocorrelated, you have fewer *effective* samples

```
> precis(mHMC)
       mean   sd  5.5% 94.5% n_eff Rhat4
a      0.00 0.10 -0.16  0.16  1632     1
bM    -0.06 0.17 -0.32  0.21  1137     1
bA    -0.61 0.17 -0.86 -0.34  1160     1
sigma  0.83 0.09  0.70  0.99  1504     1
>
```



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

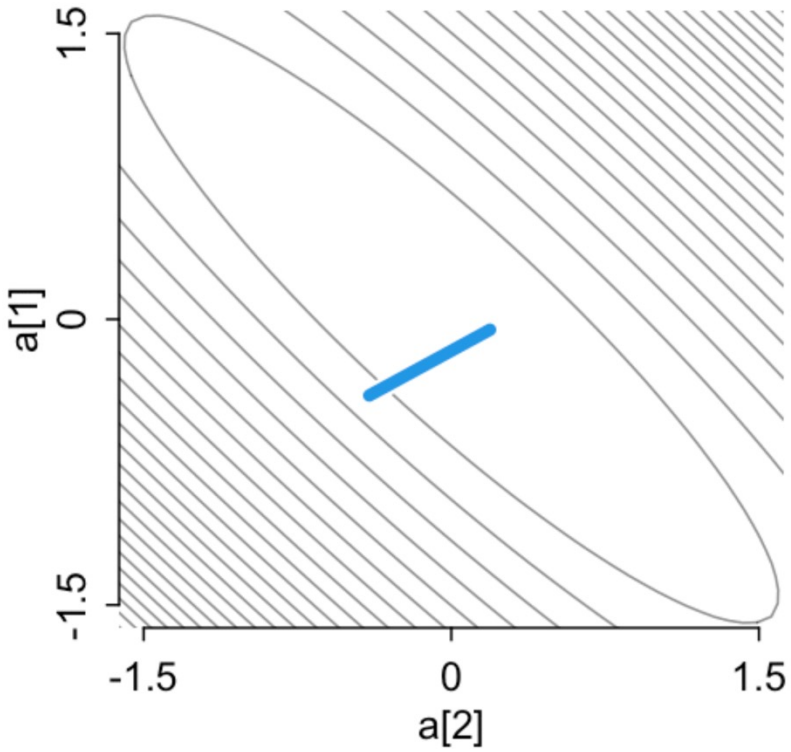
Divergent transitions

Divergent transition: A kind of rejected proposal

Simulation *diverges* from true path

Many DTs: poor exploration & possible bias

Will discuss again in later lecture



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Bad chains

```
y <- c(-1,1)
set.seed(11)
m9.2 <- ulam(
    alist(
        y ~ dnorm( mu , sigma ),
        mu <- alpha,
        alpha ~ dnorm( 0 , 1000 ),
        sigma ~ dexp( 0.0001 )
    ) , data=list(y=y) , chains=3 , cores=3 )
```

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Bad chains

```
y <- c(-1,1)
set.seed(11)
m9.2 <- ulam(
    alist(
        y ~ dnorm( mu , sigma ),
        mu <- alpha,
        alpha ~ dnorm( 0 , 1000 ),
        sigma ~ dexp( 0.0001 )
    ) , data=list(y=y) , chains=3 , cores=3 )
```

```
> precis(m9.2)
          mean     sd    5.5%   94.5% n_eff Rhat4
alpha -27.98 280.33 -485.11  291.74   121  1.01
sigma 390.82 993.07    1.83 1870.86   169  1.03
>
```

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

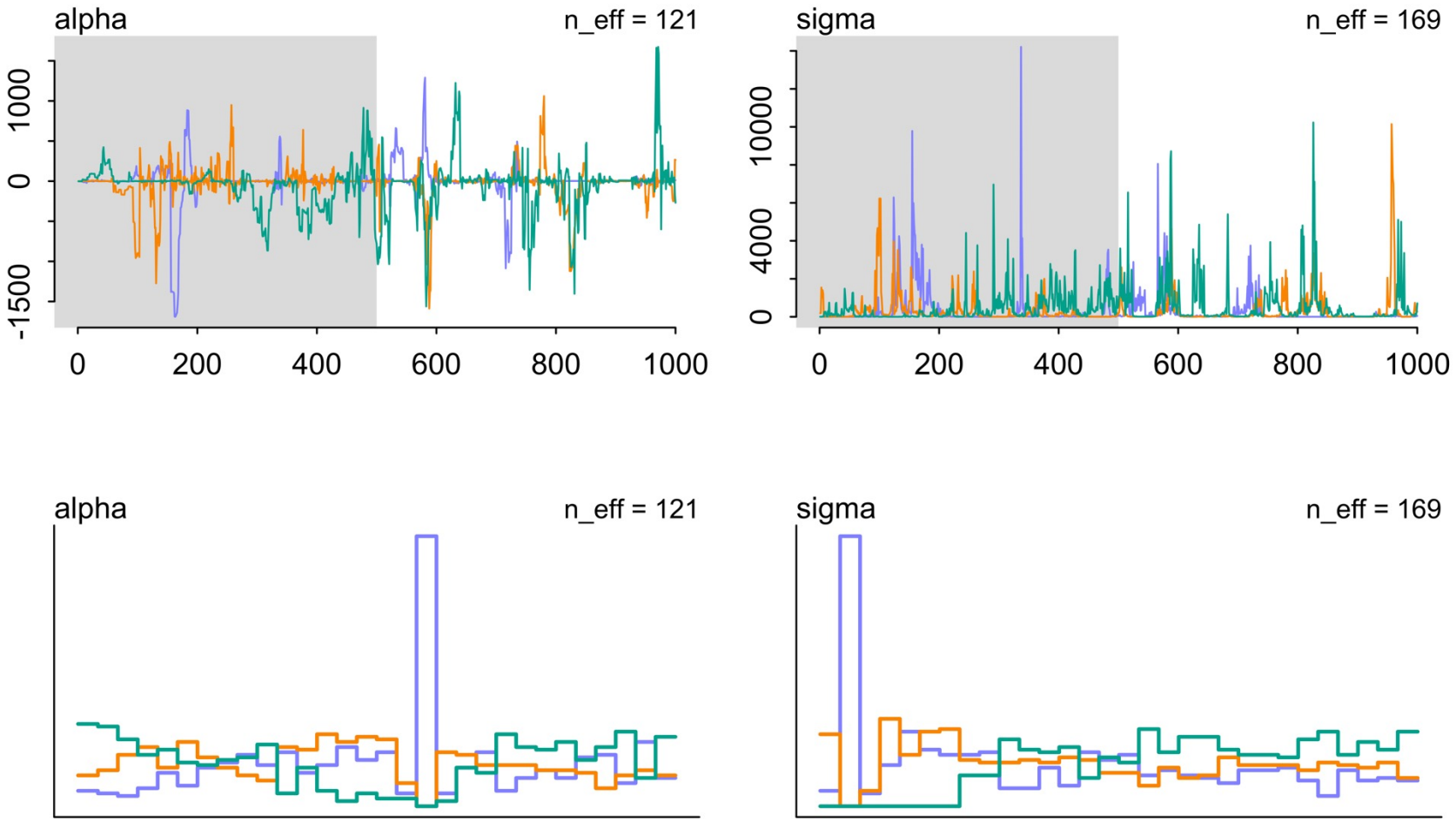
Bad chains

```
y <- c(-1,1)
set.seed(11)
m9.2 <- ulam(
    alist(
        y ~ dnorm( mu , sigma ),
        mu <- alpha,
        alpha ~ dnorm( 0 , 1000 ),
        sigma ~ dexp( 0.0001 )
    ) , data=list(y=y) , chains=3 , cores=3 )
```

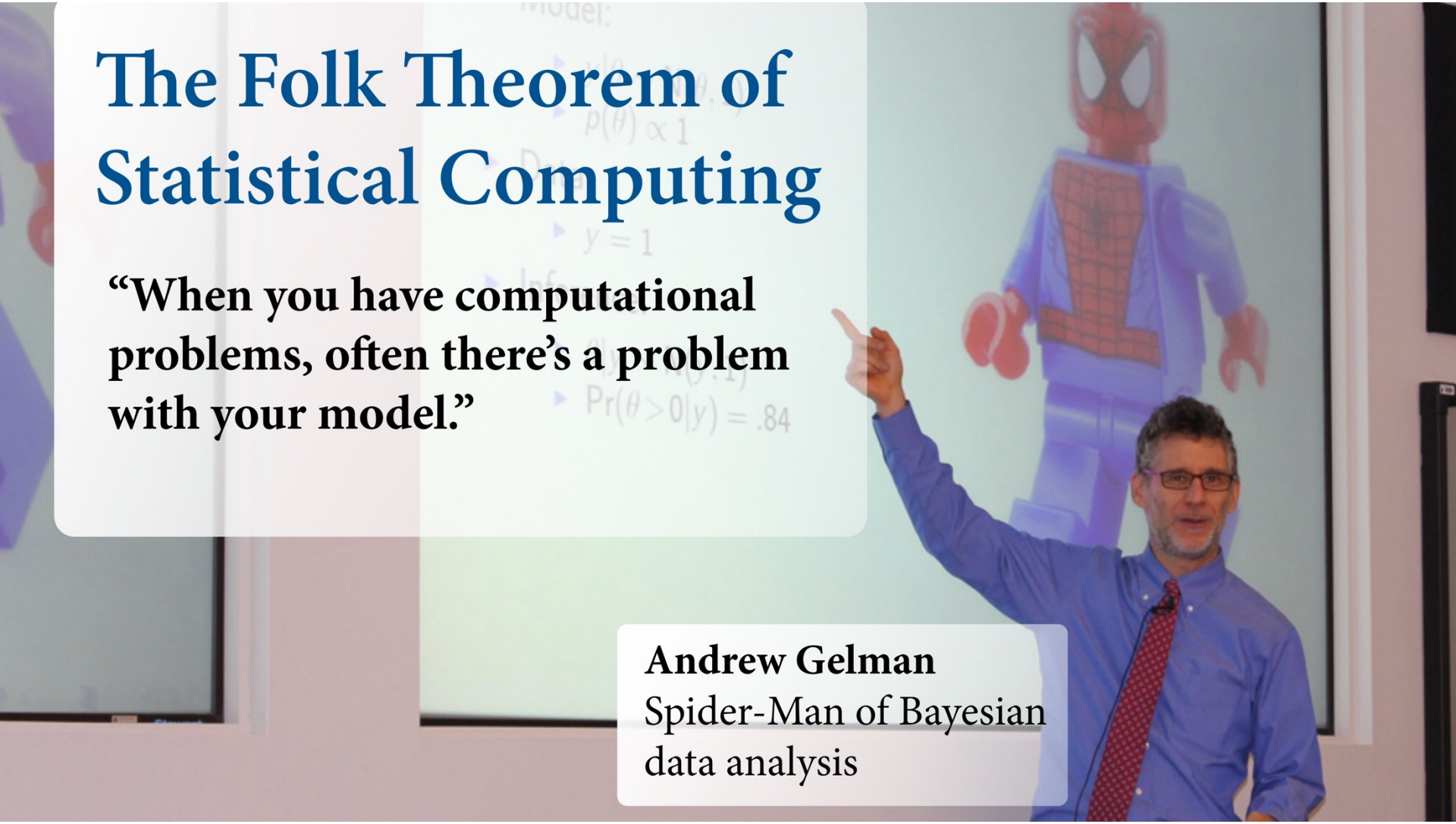
```
> precis(m9.2)
          mean      sd    5.5%   94.5% n_eff Rhat4
alpha  -27.98  280.33 -485.11  291.74   121  1.01
sigma  390.82  993.07    1.83 1870.86   169  1.03
>
```

```
Warning: 108 of 1500 (7.0%) transitions ended with a divergence.
This may indicate insufficient exploration of the posterior distribution.
Possible remedies include:
  * Increasing adapt_delta closer to 1 (default is 0.8)
  * Reparameterizing the model (e.g. using a non-centered parameterization)
  * Using informative or weakly informative prior distributions
```

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022



Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

Bad chains

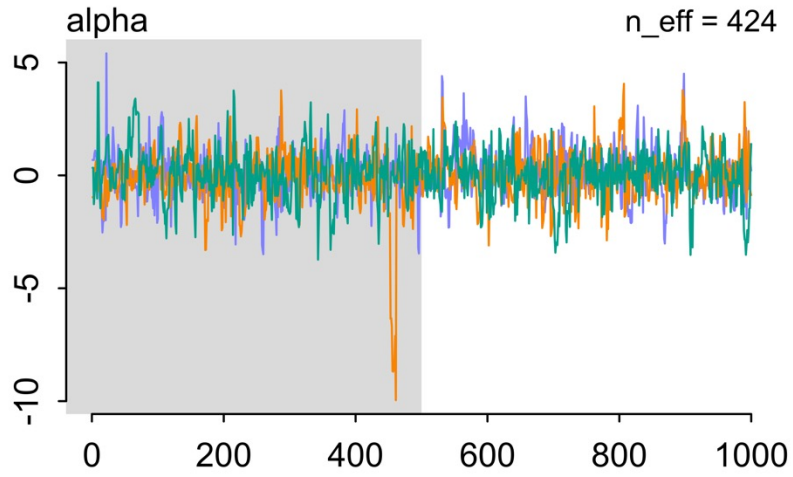
```
y <- c(-1,1)
set.seed(11)
m9.2 <- ulam(
    alist(
        y ~ dnorm( mu , sigma ),
        mu <- alpha,
        alpha ~ dnorm( 0 , 1000 ),
        sigma ~ dexp( 0.0001 )
    ) , data=list(y=y) , chains=3 ,
cores=3 )
```

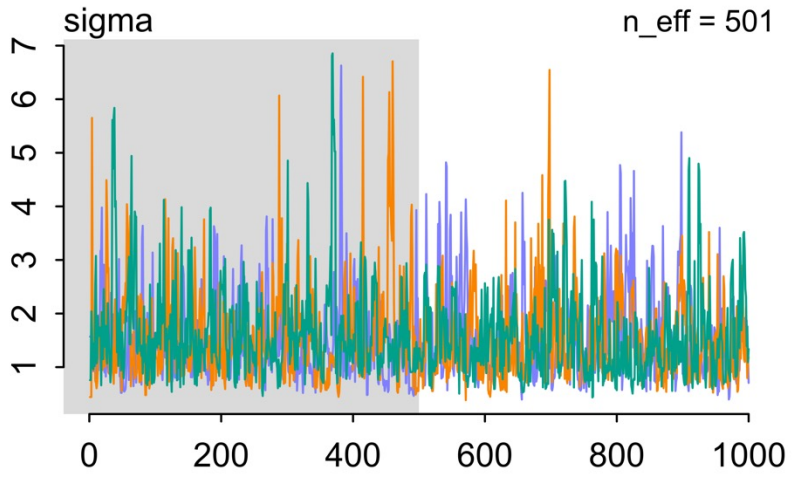
```
> precis(m9.2)
         mean      sd    5.5%   94.5% n_eff Rhat4
alpha -27.98 280.33 -485.11  291.74   121  1.01
sigma 390.82 993.07    1.83 1870.86   169  1.03
>
```

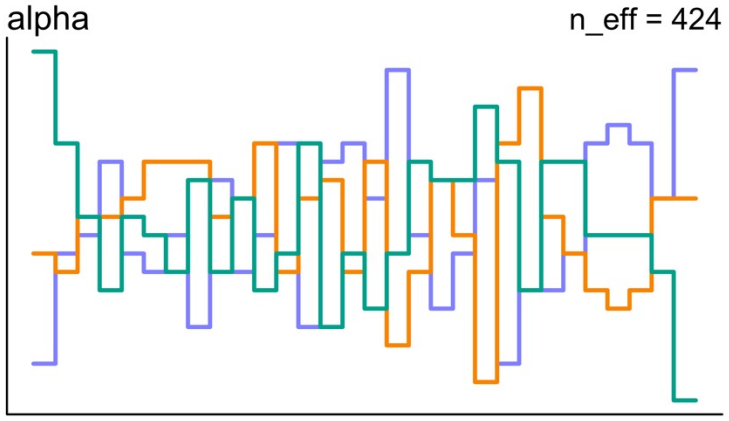
```
m9.3 <- ulam(
    alist(
        y ~ dnorm( mu , sigma ),
        mu <- alpha,
        alpha ~ dnorm( 1 , 10 ),
        sigma ~ dexp( 1 )
    ) , data=list(y=y) , chains=3 , cores=3 )
```

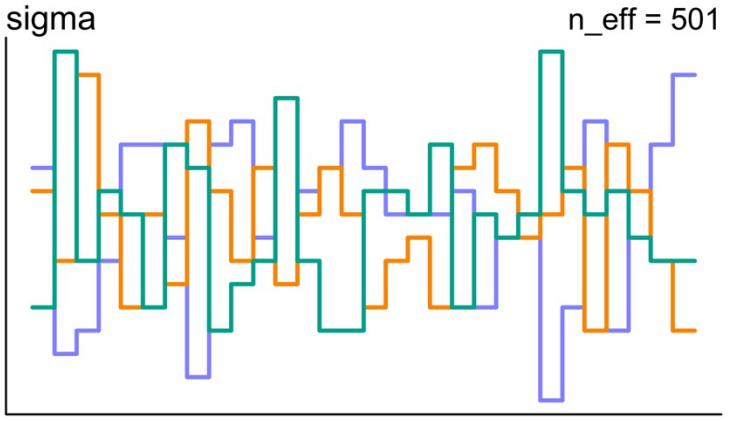
```
> precis(m9.3)
      mean   sd  5.5% 94.5% n_eff Rhat4
alpha 0.06 1.07 -1.62  1.80   424  1.01
sigma 1.51 0.80  0.65  3.02   501  1.00
>
```

Source: Richard McElreath, https://github.com/rmcelreath/stat_rethinking_2022









Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022

El Pueblo Unido

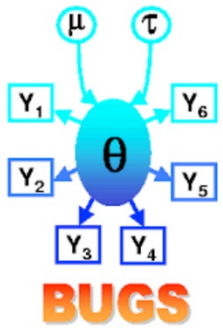
Desktop MCMC has been a revolution in scientific computing

Custom scientific modeling

High-dimension

Propagate measurement error

Do not “pipette by mouth”



Sir David Spiegelhalter

Source: Richard McElraeth, https://github.com/rmcelreath/stat_rethinking_2022