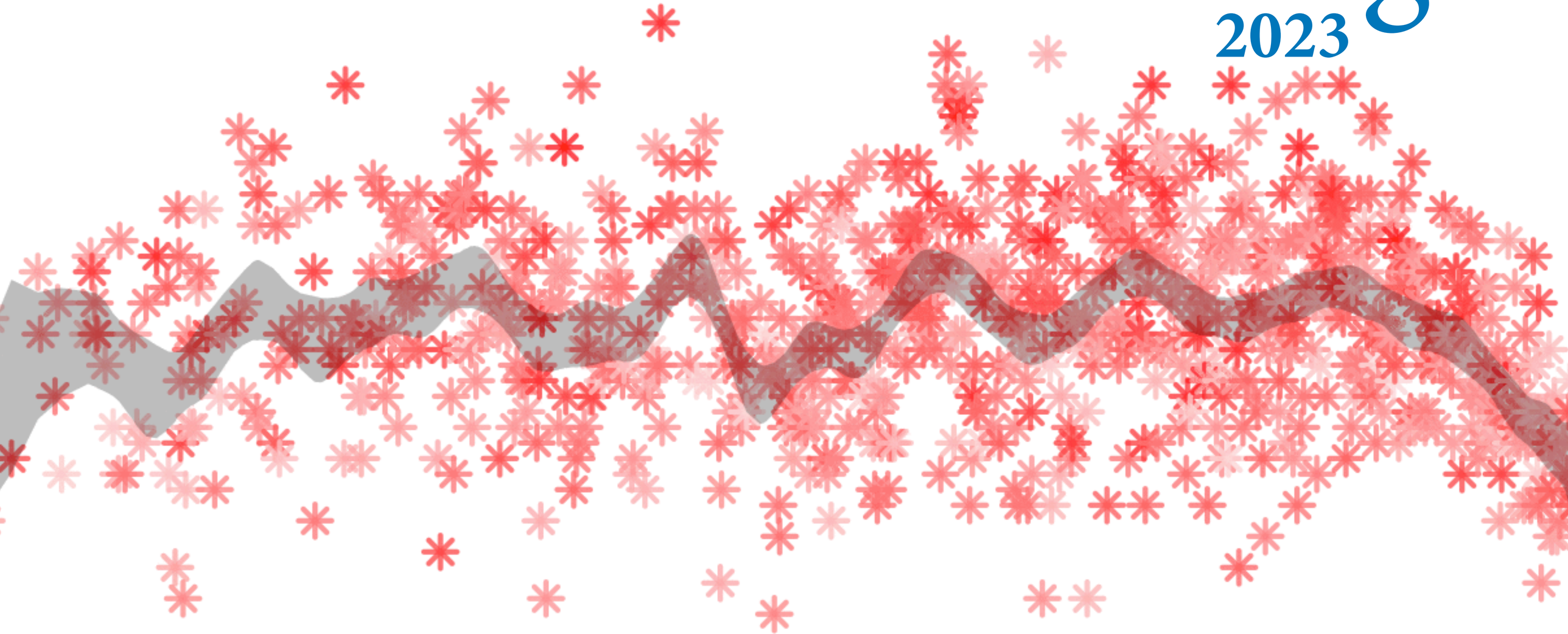
Statistical Rethinking

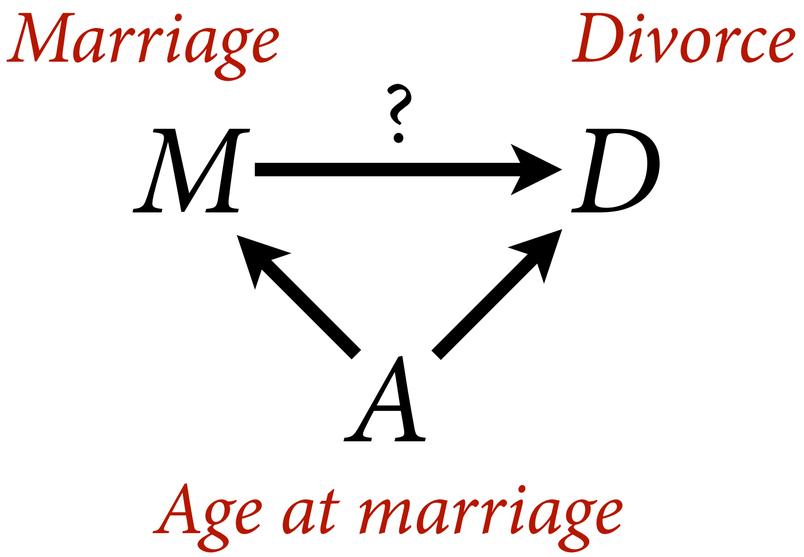
2023

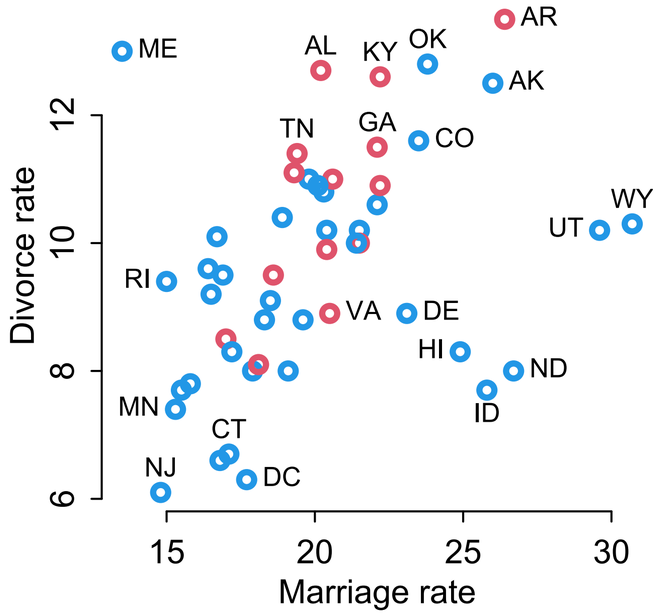


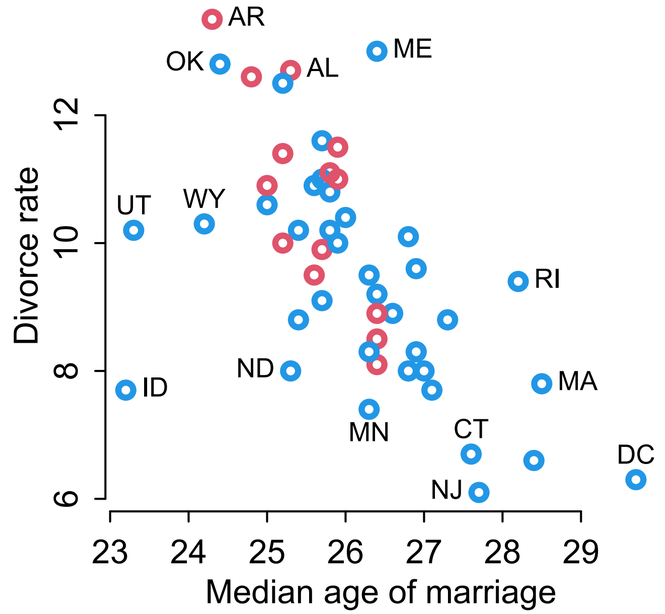
8. Markov chain Monte Carlo

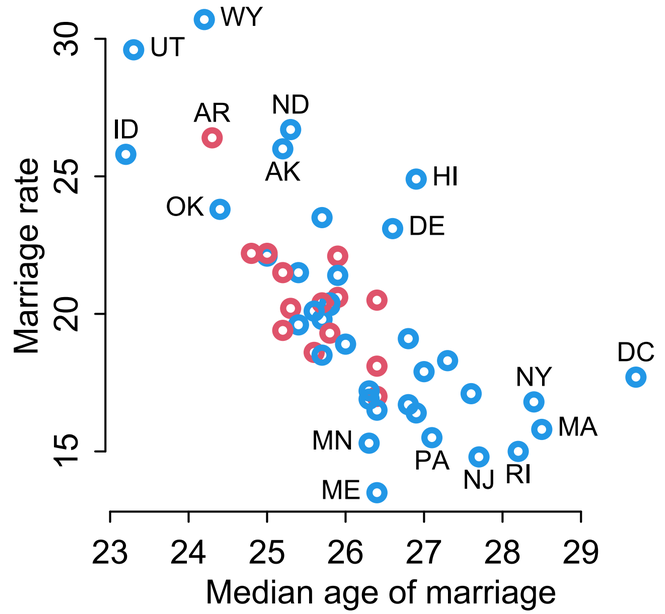
Table of Random Numbers

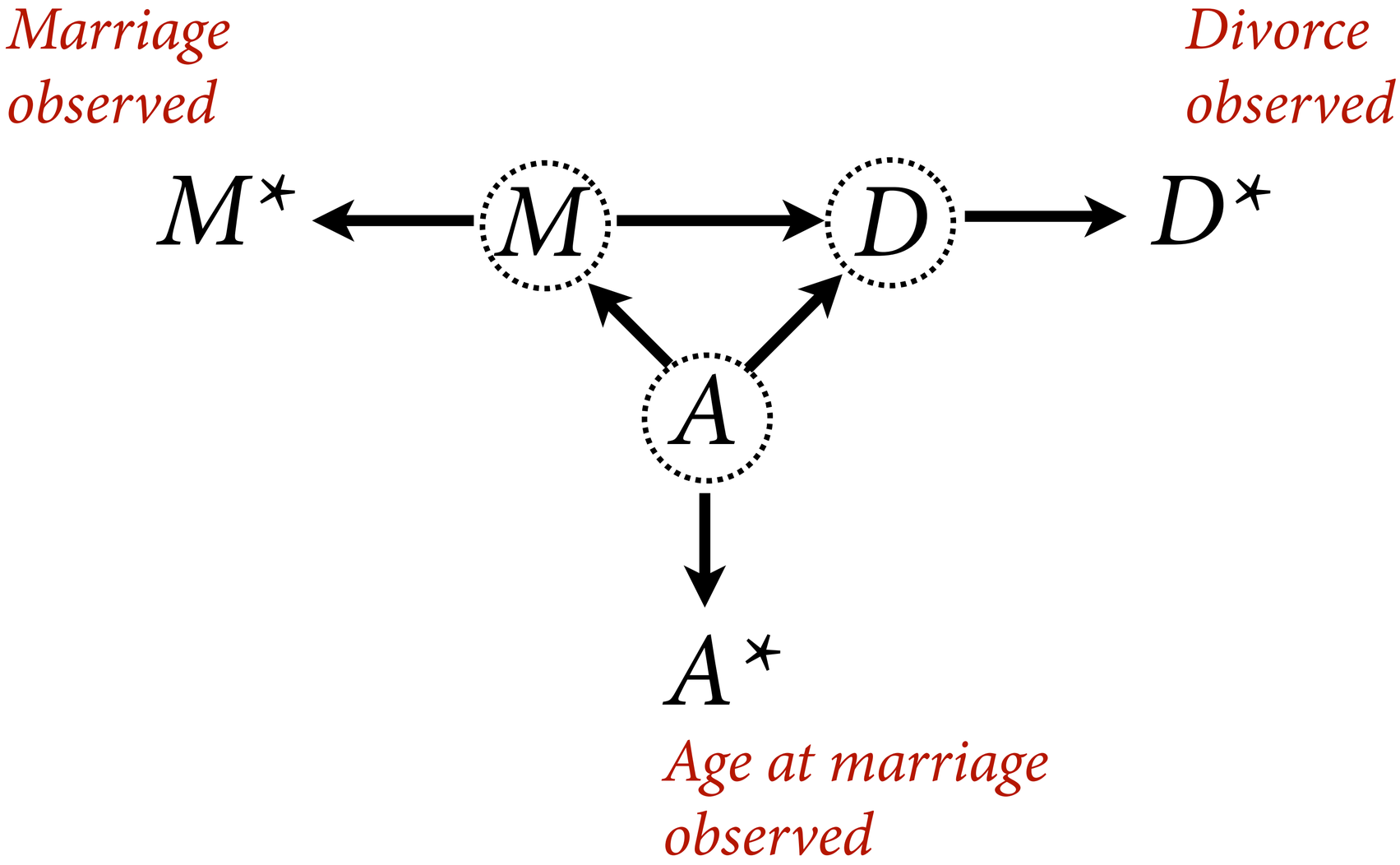
36518 36777 89116 05542 29705 83775 21564 81639 27973 62413 85652 62817 57881
46132 81380 75635 19428 88048 08747 20092 12615 35046 67753 69630 10883 13683
31841 77367 40791 97402 27569 90184 02338 39318 54936 34641 95525 86316 87384
84180 93793 64953 51472 65358 23701 75230 47200 78176 85248 90589 74567 22633
78435 37586 07015 98729 76703 16224 97661 79907 06611 26501 93389 92725 68158
41859 94198 37182 61345 88857 53204 86721 59613 67494 17292 94457 89520 77771
13019 07274 51068 93129 40386 51731 44254 66685 72835 01270 42523 45323 63481
82448 72430 29041 59208 95266 33978 70958 60017 39723 00606 17956 19024 15819
25432 96593 83112 96997 55340 80312 78839 09815 16887 22228 06206 54272 83516
69226 38655 03811 08342 47863 02743 11547 38250 58140 98470 24364 99797 73498
25837 68821 66426 20496 84843 18360 91252 99134 48931 99538 21160 09411 44659
38914 82707 24769 72026 56813 49336 71767 04474 32909 74162 50404 68562 14088
04070 60681 64290 26905 65617 76039 91657 71362 32246 49595 50663 47459 57072
01674 14751 28637 86980 11951 10479 41454 48527 53868 37846 85912 15156 00865
70294 35450 39982 79503 34382 43186 69890 63222 30110 56004 04879 05138 57476
73903 98066 52136 89925 50000 96334 30773 80571 31178 52799 41050 76298 43995
87789 56408 77107 88452 80975 03406 36114 64549 79244 82044 00202 45727 35709
92320 95929 58545 70699 07679 23296 03002 63885 54677 55745 52540 62154 33314
46391 60276 92061 43591 42118 73094 53608 58949 42927 90993 46795 05947 01934
67090 45063 84584 66022 48268 74971 94861 61749 61085 81758 89640 39437 90044
11666 99916 35165 29420 73213 15275 62532 47319 39842 62273 94980 23415 64668
40910 59068 04594 94576 51187 54796 17411 56123 66545 82163 61868 22752 40101
41169 37965 47578 92180 05257 19143 77486 02457 00985 31960 39033 44374 28352
76418

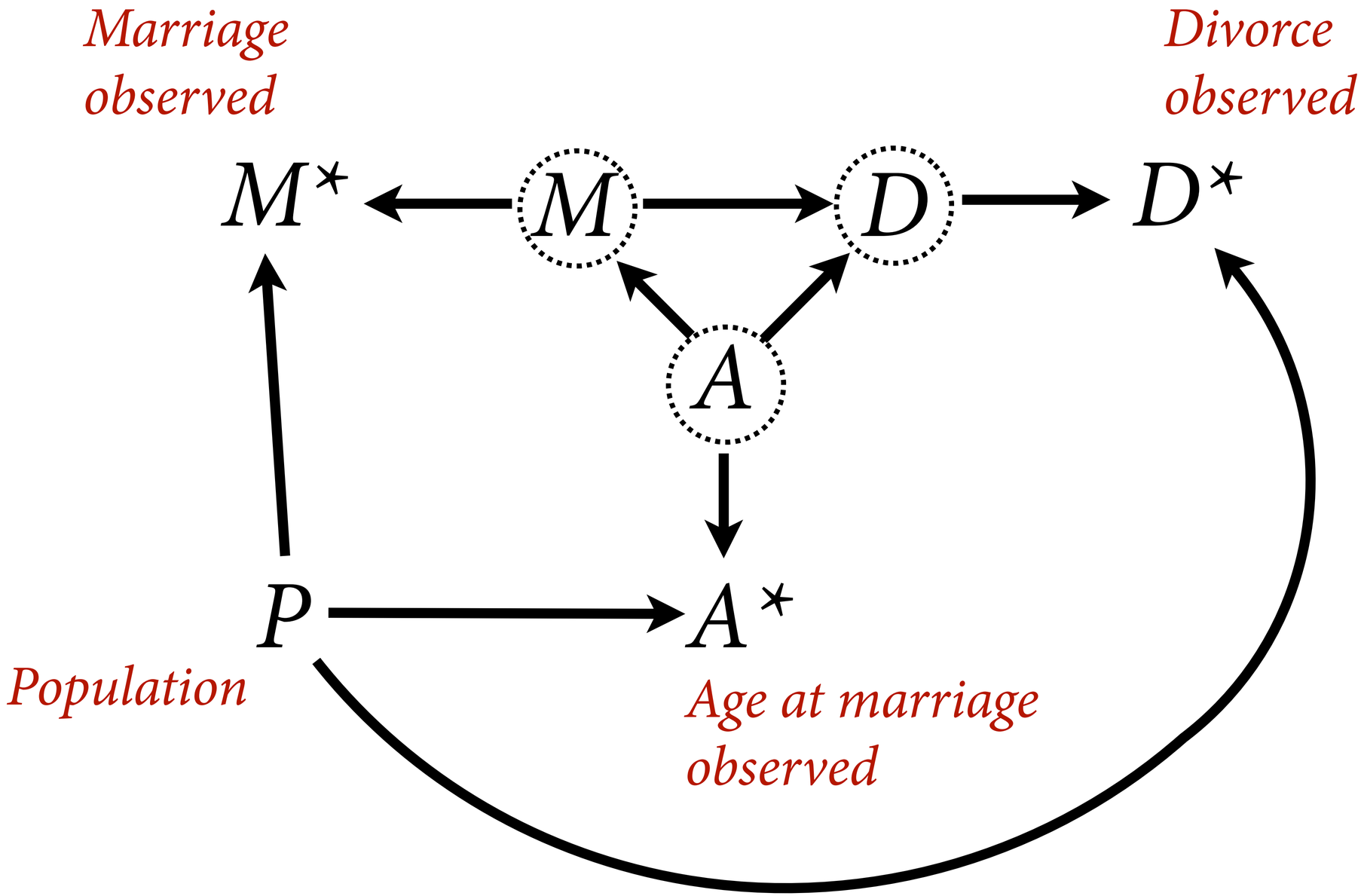

Marriage
Divorce
M
?
D
A
Age at marriage


Divorce rate
Marriage rate
ME
AL
KY
OK
AR
AK
TN
GA
CO
WY
UT
RI
VA
DE
HI
ND
ID
MN
CT
NJ
DC


Divorce rate
Median age of marriage
AR
OK
AL
ME
UT
WY
RI
ND
ID
MA
MN
CT
DC
NJ


Marriage rate
Median age of marriage
WY
UT
ID
AR
ND
AK
HI
OK
DE
DC
NY
MA
MN
PA
NJ
RI
ME


Marriage observed
Divorce observed
M^*
M
D
D^*
A
A^*
Age at marriage observed

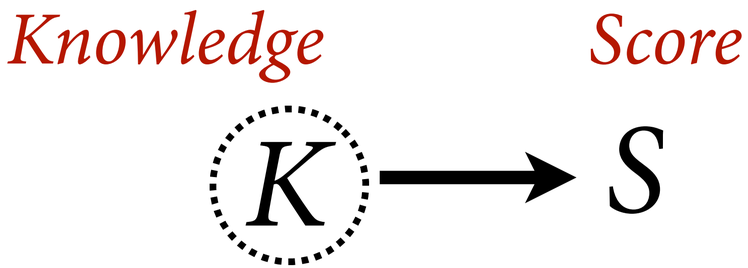

Marriage observed
Divorce observed
M^*
M
D
D^*
A
P
A^*
Population
Age at marriage observed

Real, latent modeling problems

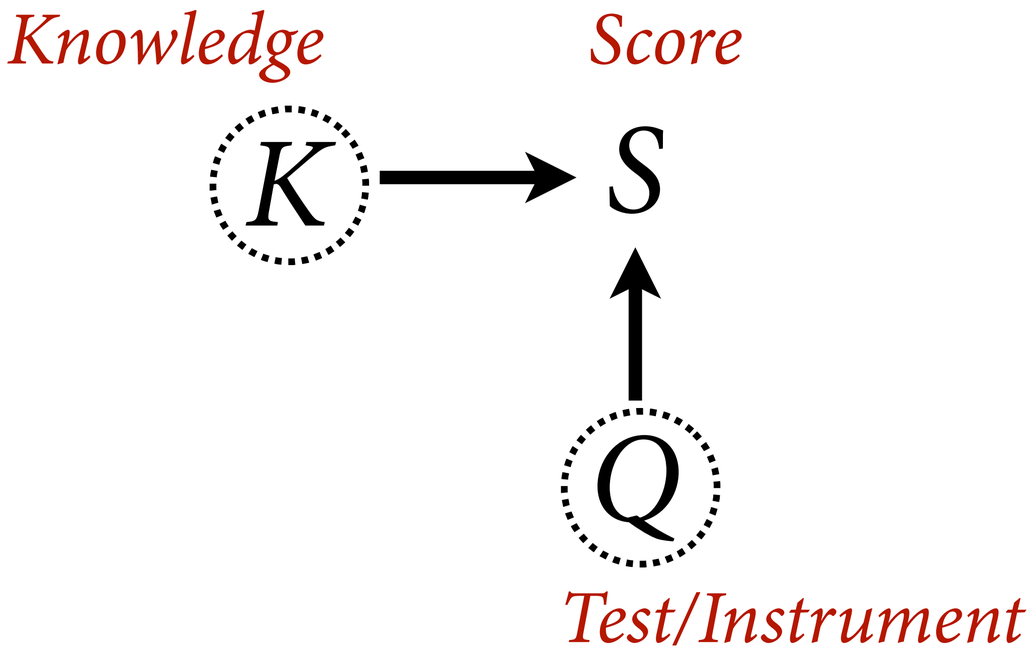
Knowledge

$$K$$

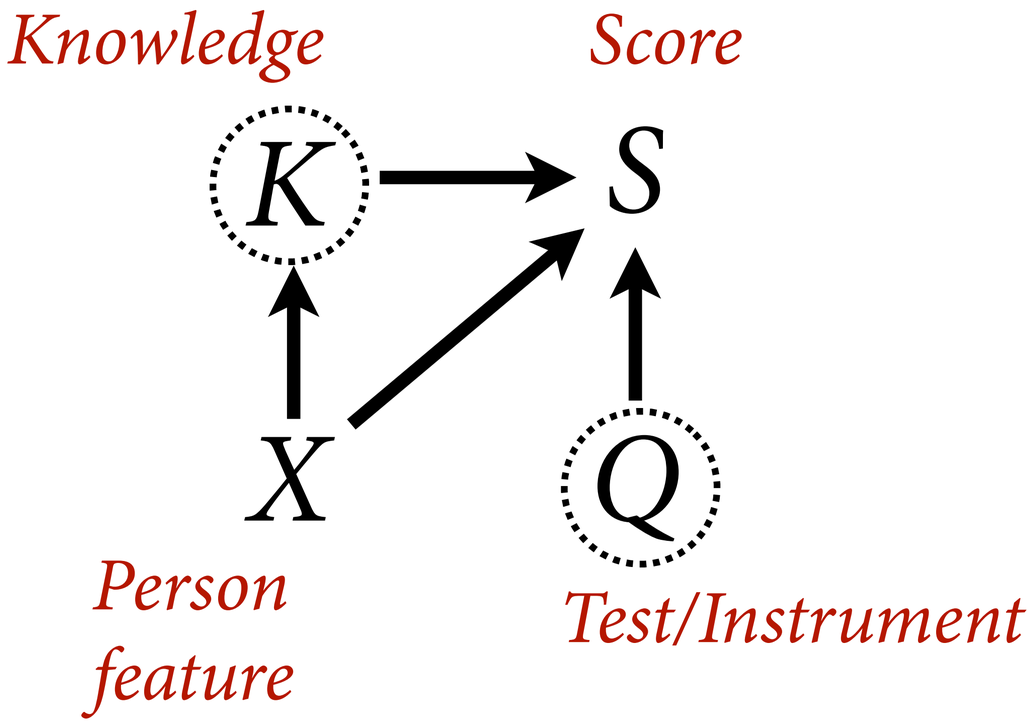
Real, latent modeling problems



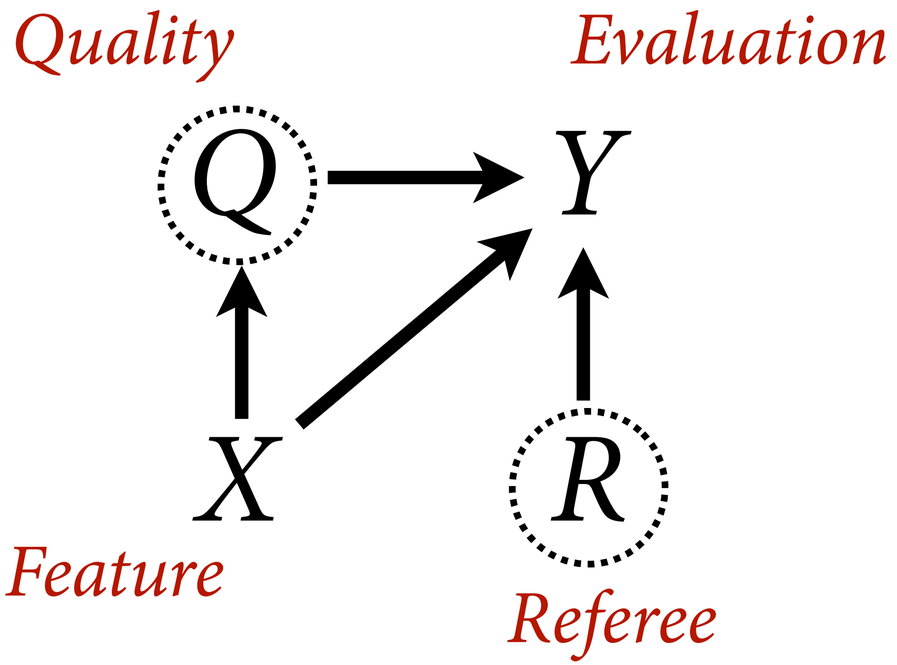
Real, latent modeling problems



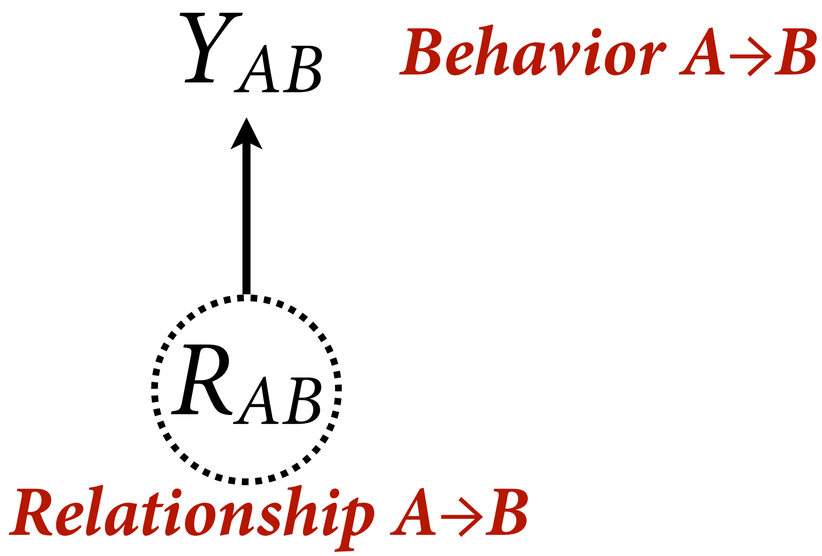
Real, latent modeling problems



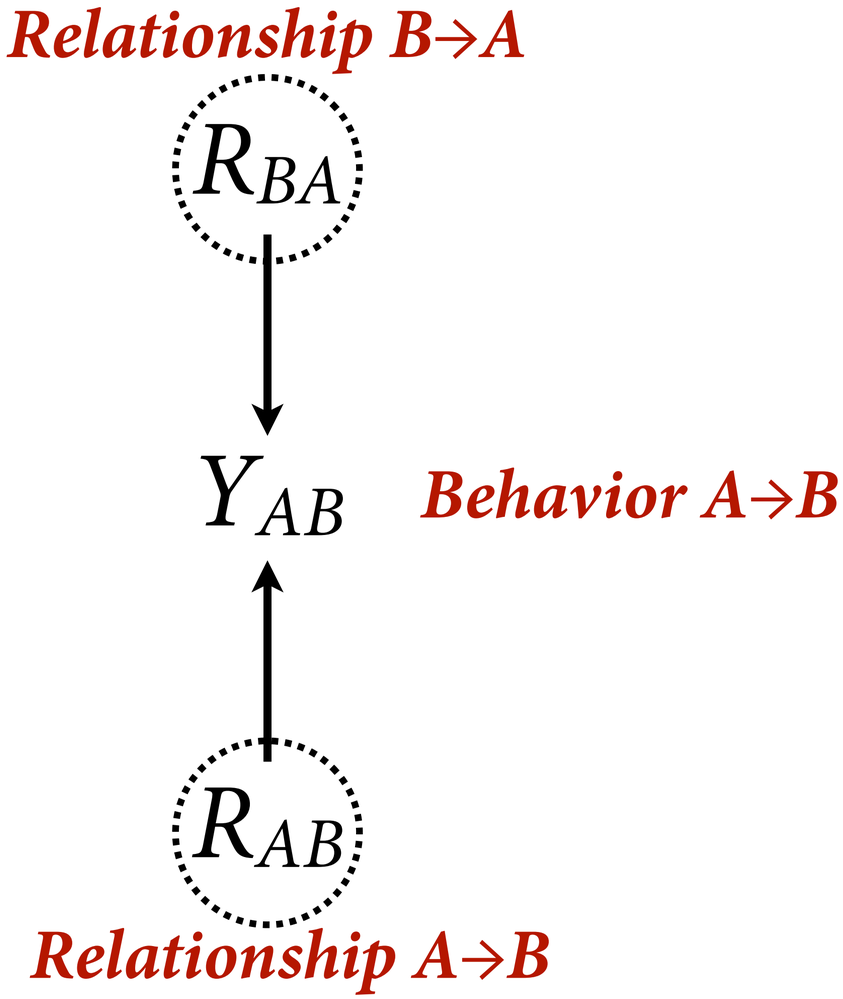
Real, latent modeling problems



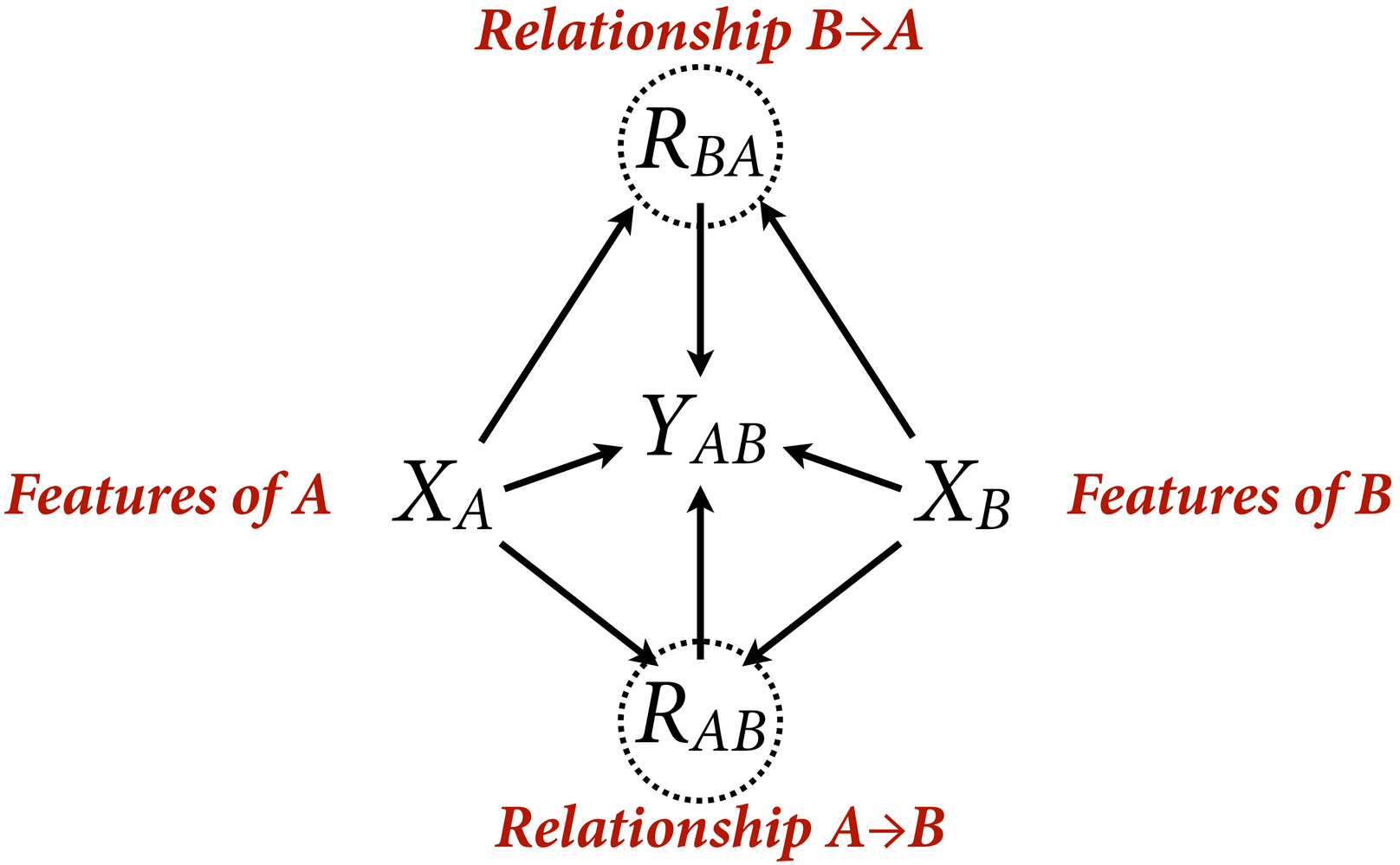
Social networks



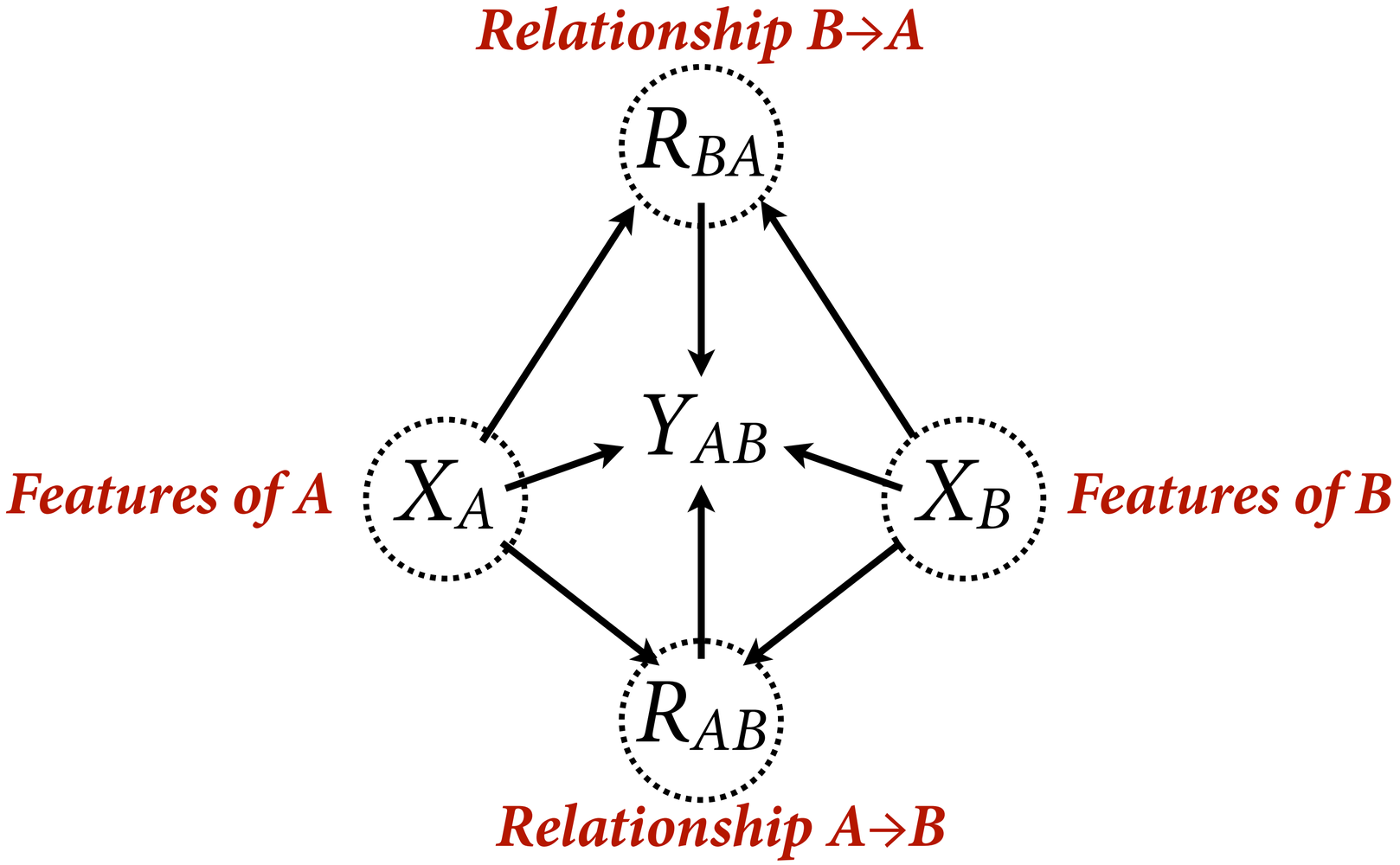
Social networks



Social networks



Social networks



Problems & solutions

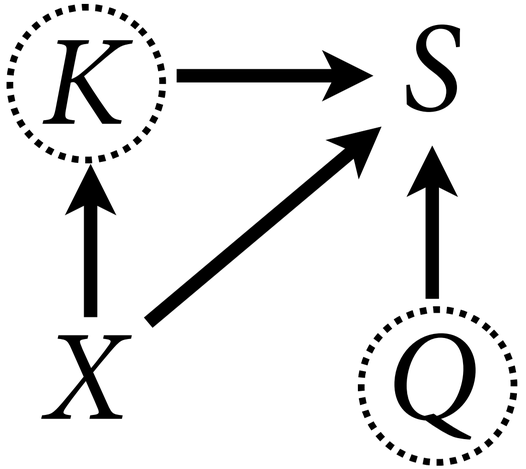
Real research problems:

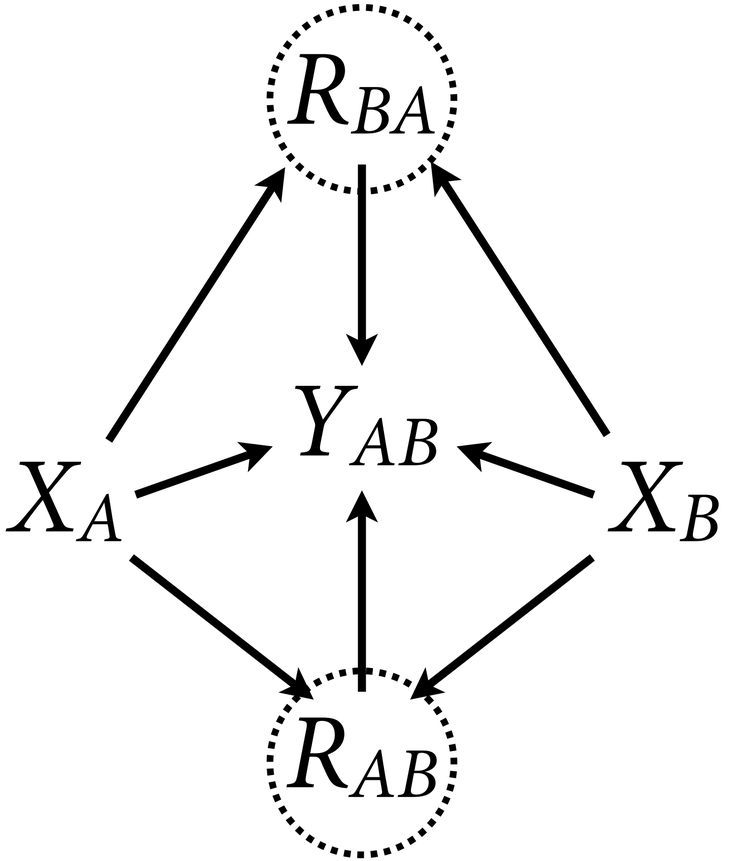
Many unknowns

Nested relationships

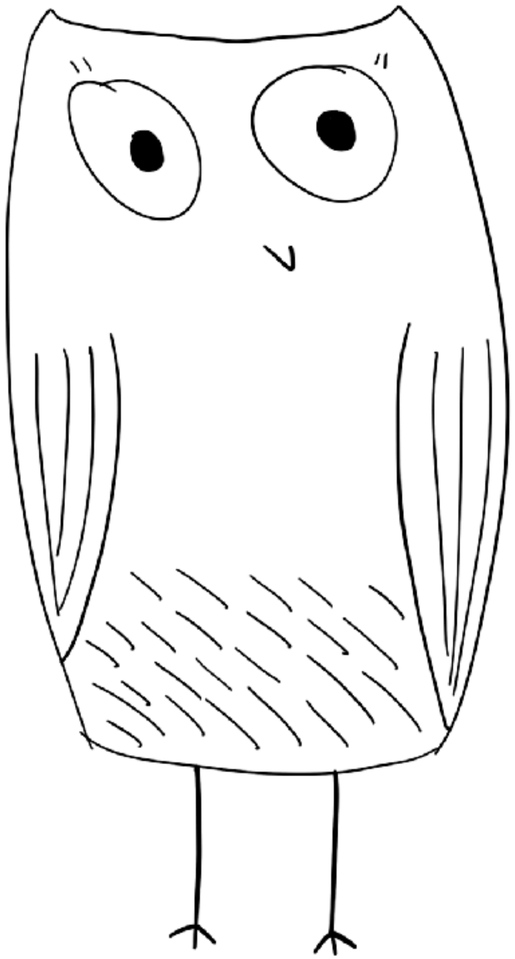
Bounded outcomes

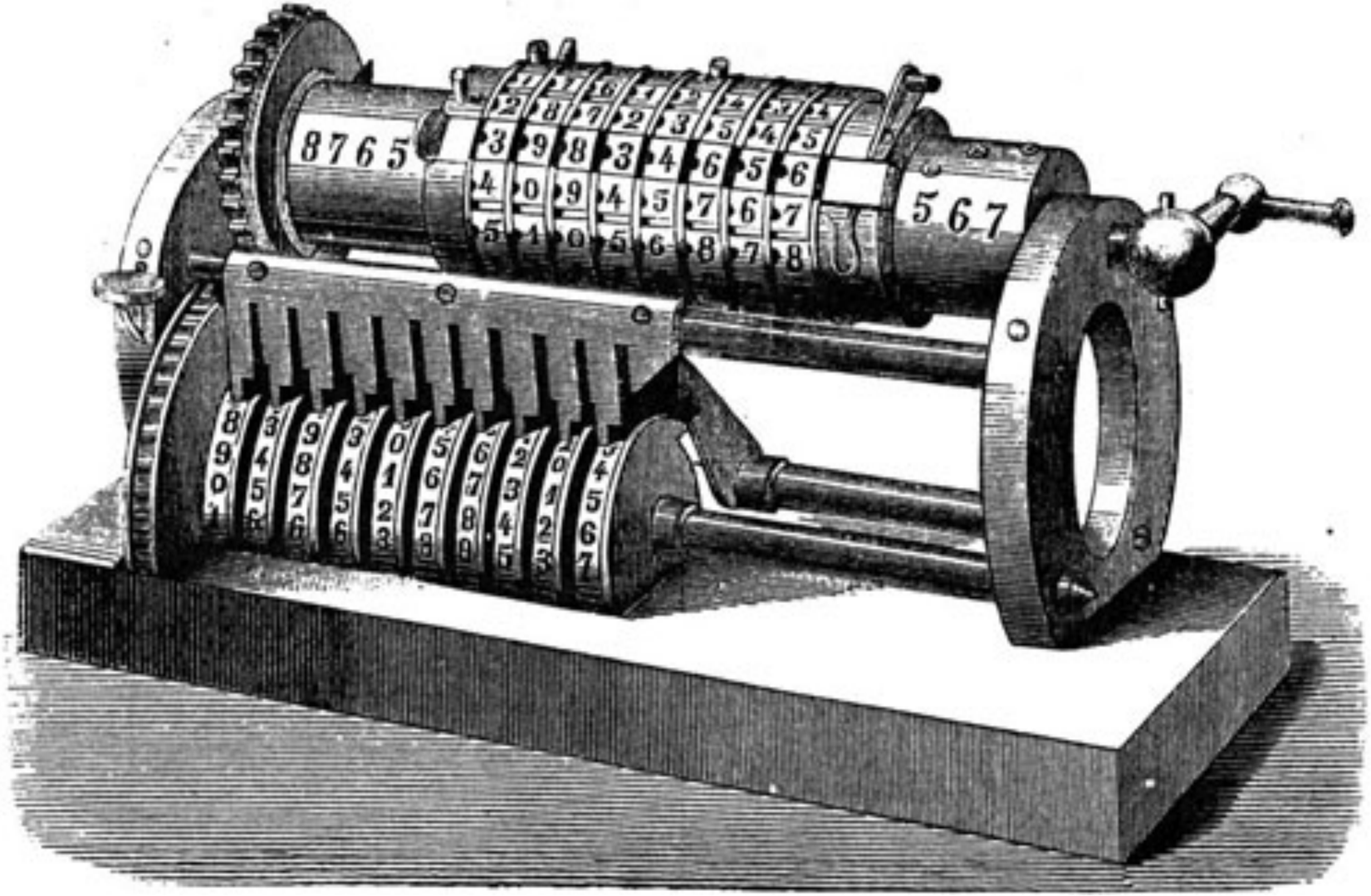
Difficult calculations





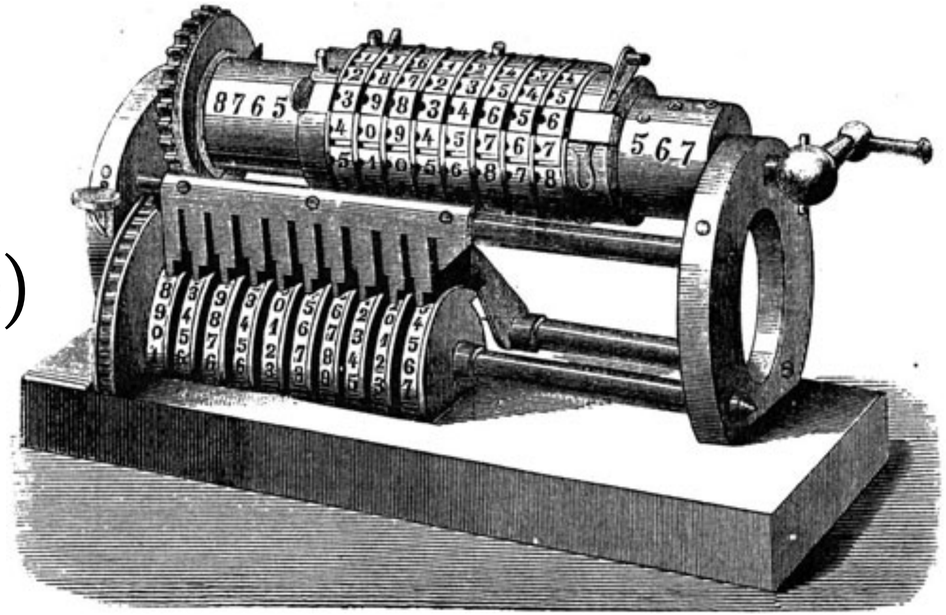
aNAlYzE tHe DatA



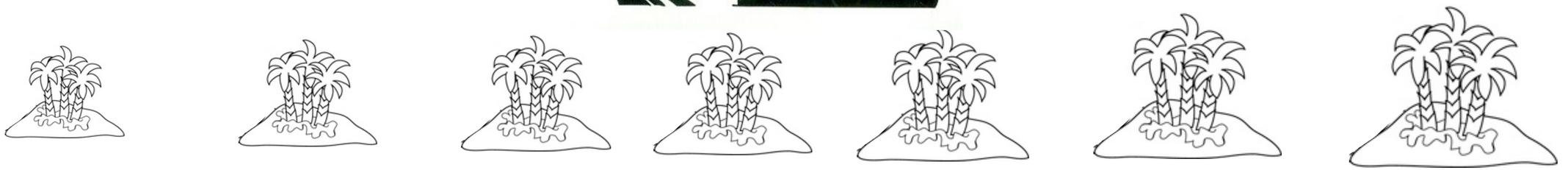


Computing the posterior

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



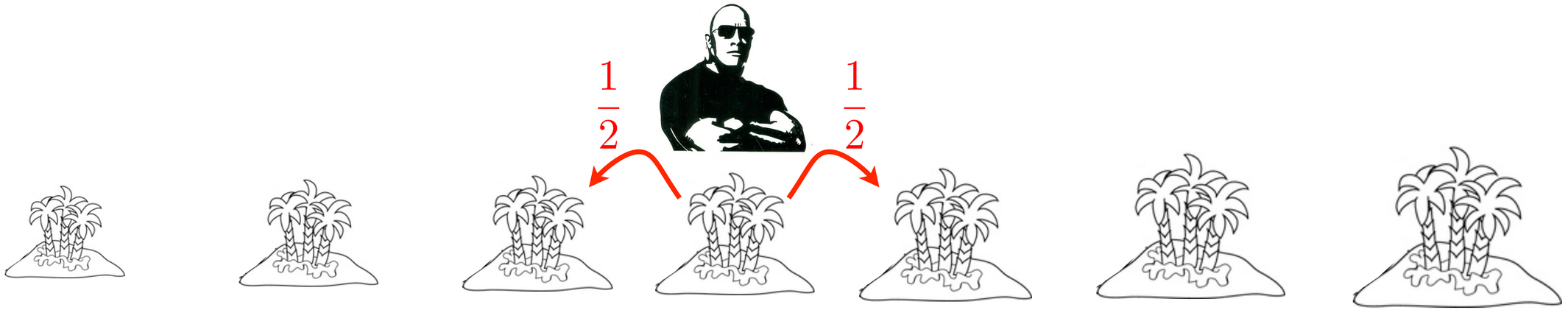
King Markov



The Metropolis Archipelago

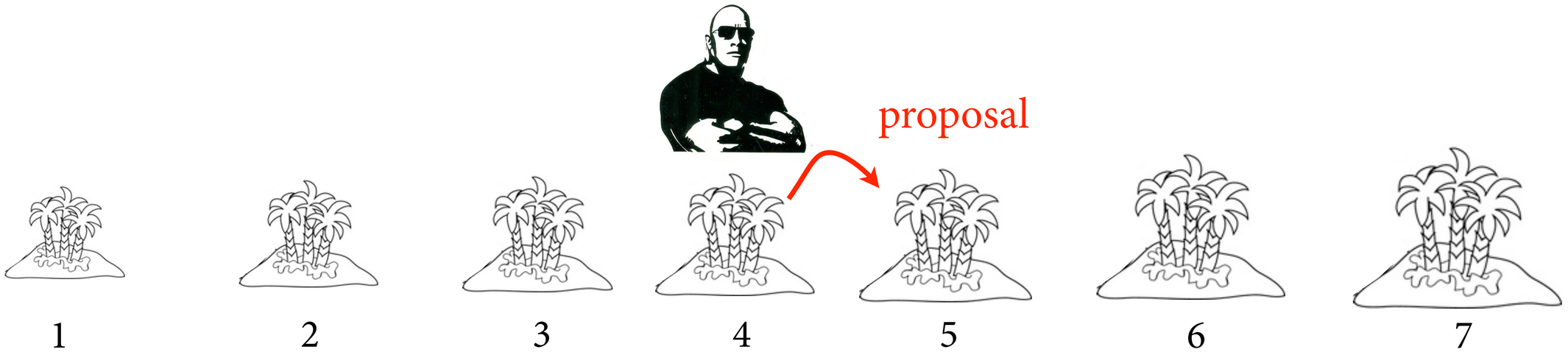
Contract: King Markov must visit each island in proportion to its population size

Here's how he does it...



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

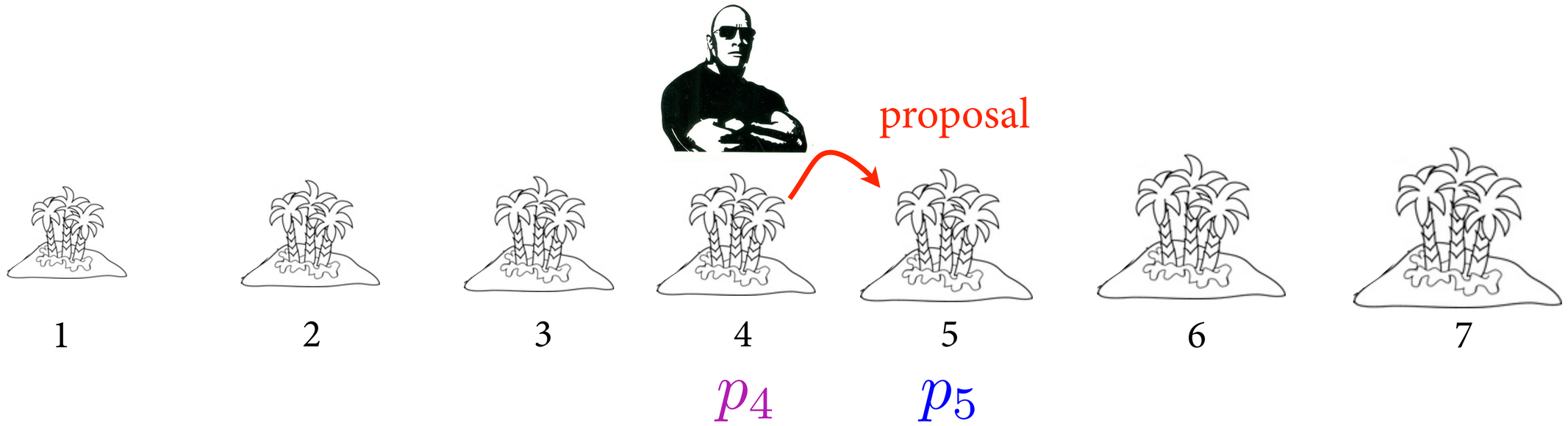
(1) Flip a coin to choose island on left or right.
Call it the "proposal" island.



(2) Find population of proposal island.

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

(2) Find population of proposal island.

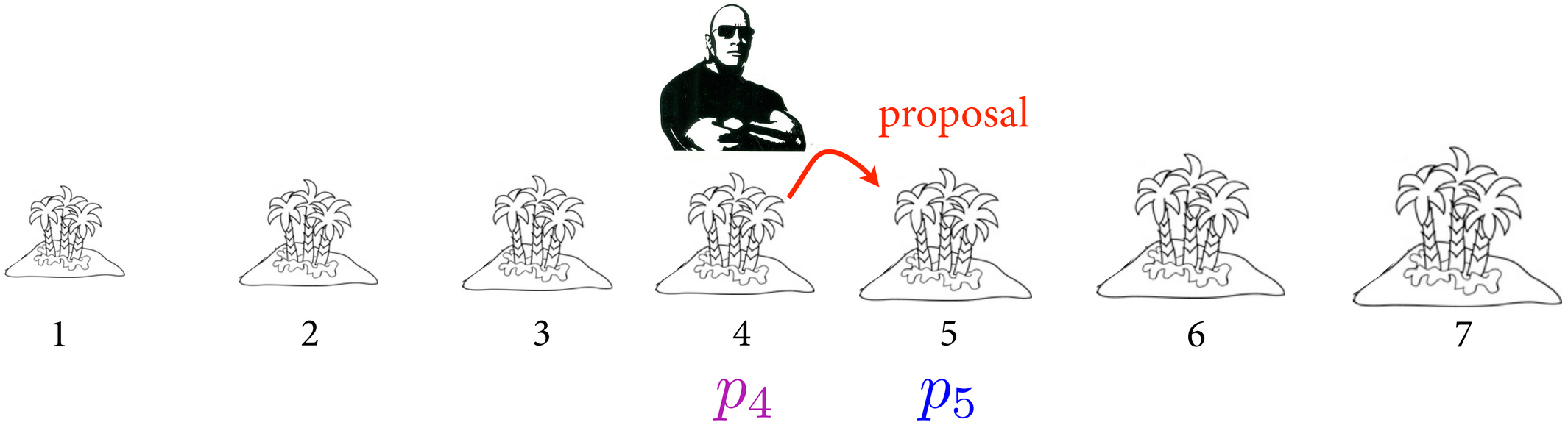


(3) Find population of current island.

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

(2) Find population of proposal island.

(3) Find population of current island.



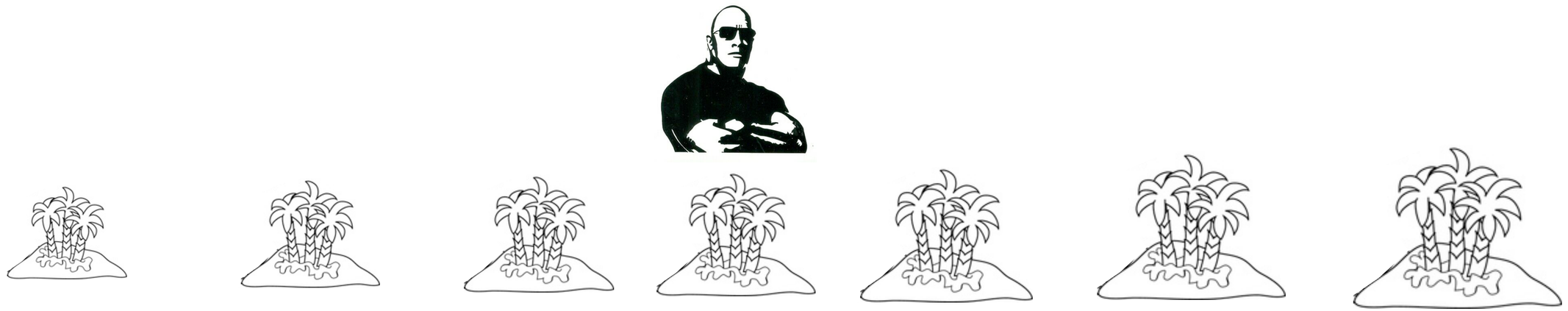
(4) Move to proposal, with probability = $\frac{p_5}{p_4}$

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

(2) Find population of proposal island.

(3) Find population of current island.

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



(5) Repeat from (1)

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

(2) Find population of proposal island.

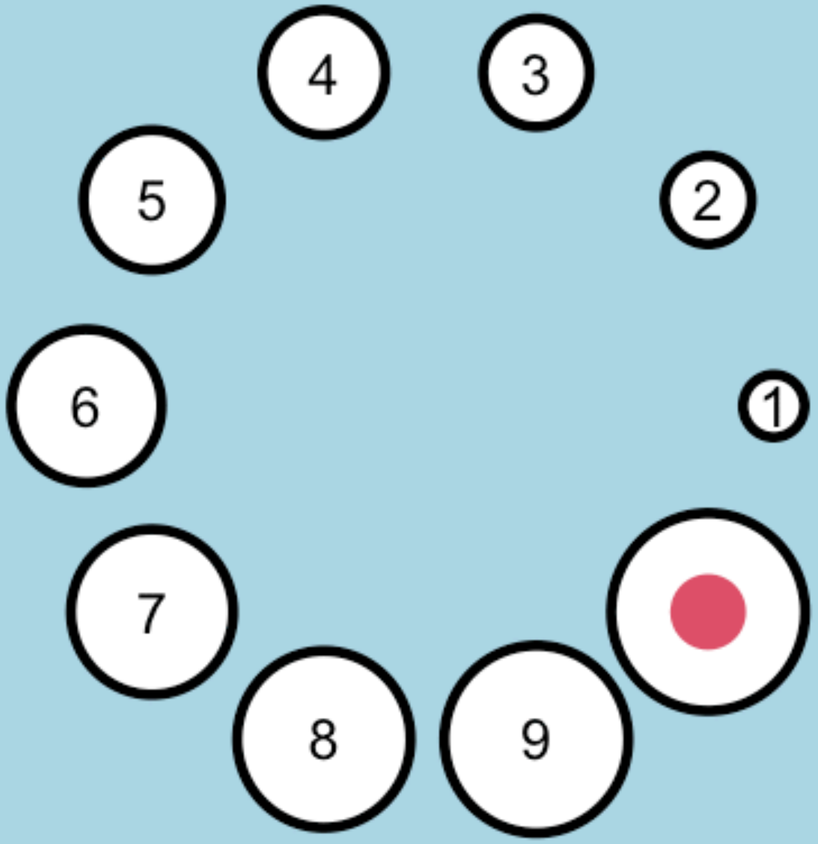
(3) Find population of current island.

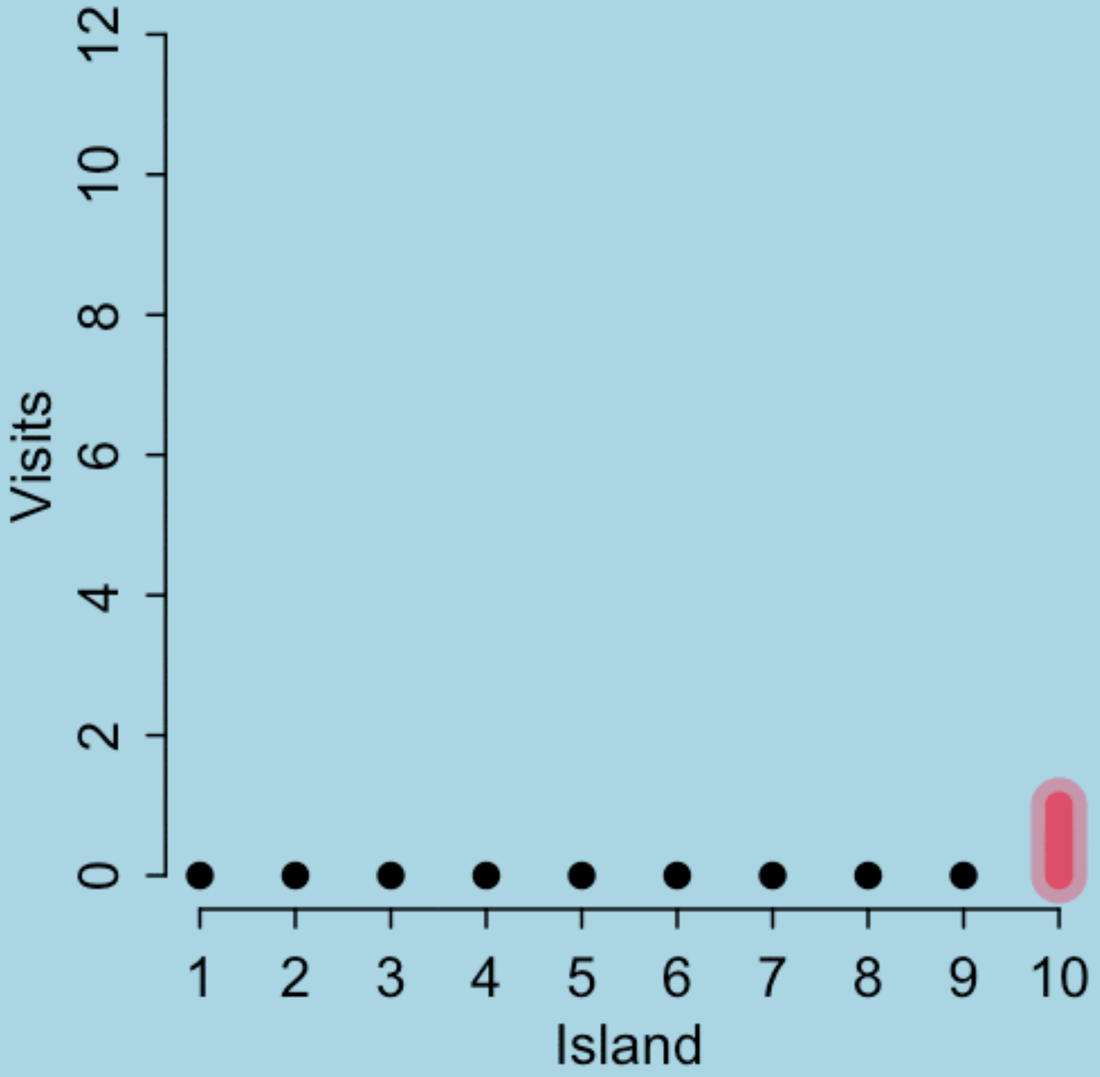
(4) Move to proposal, with probability = $\frac{p_5}{p_4}$

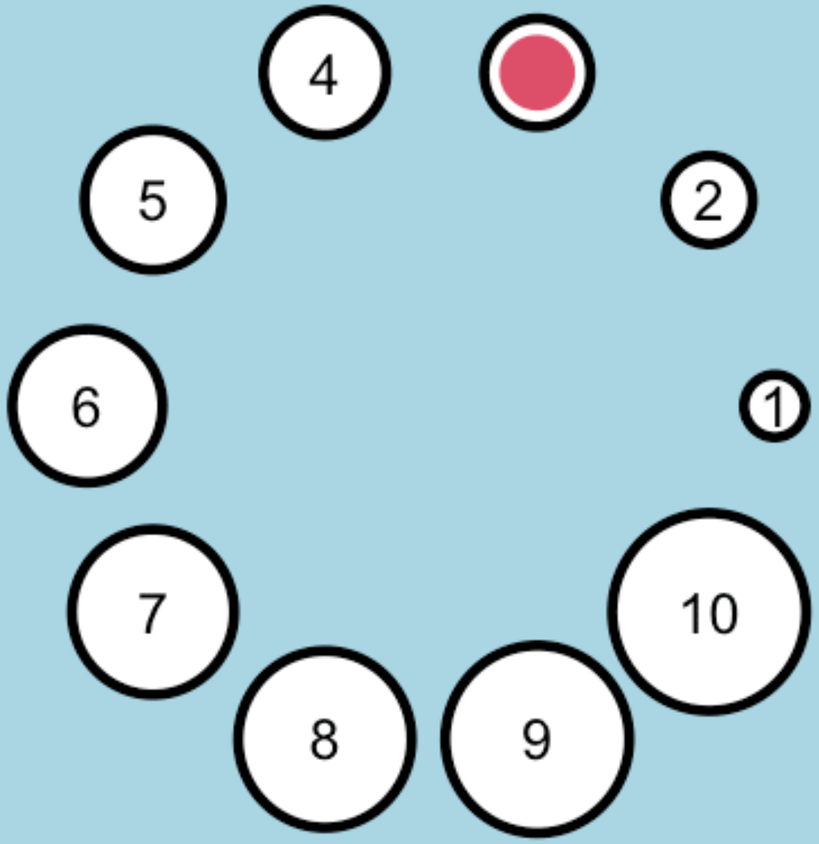
(5) Repeat from (1)

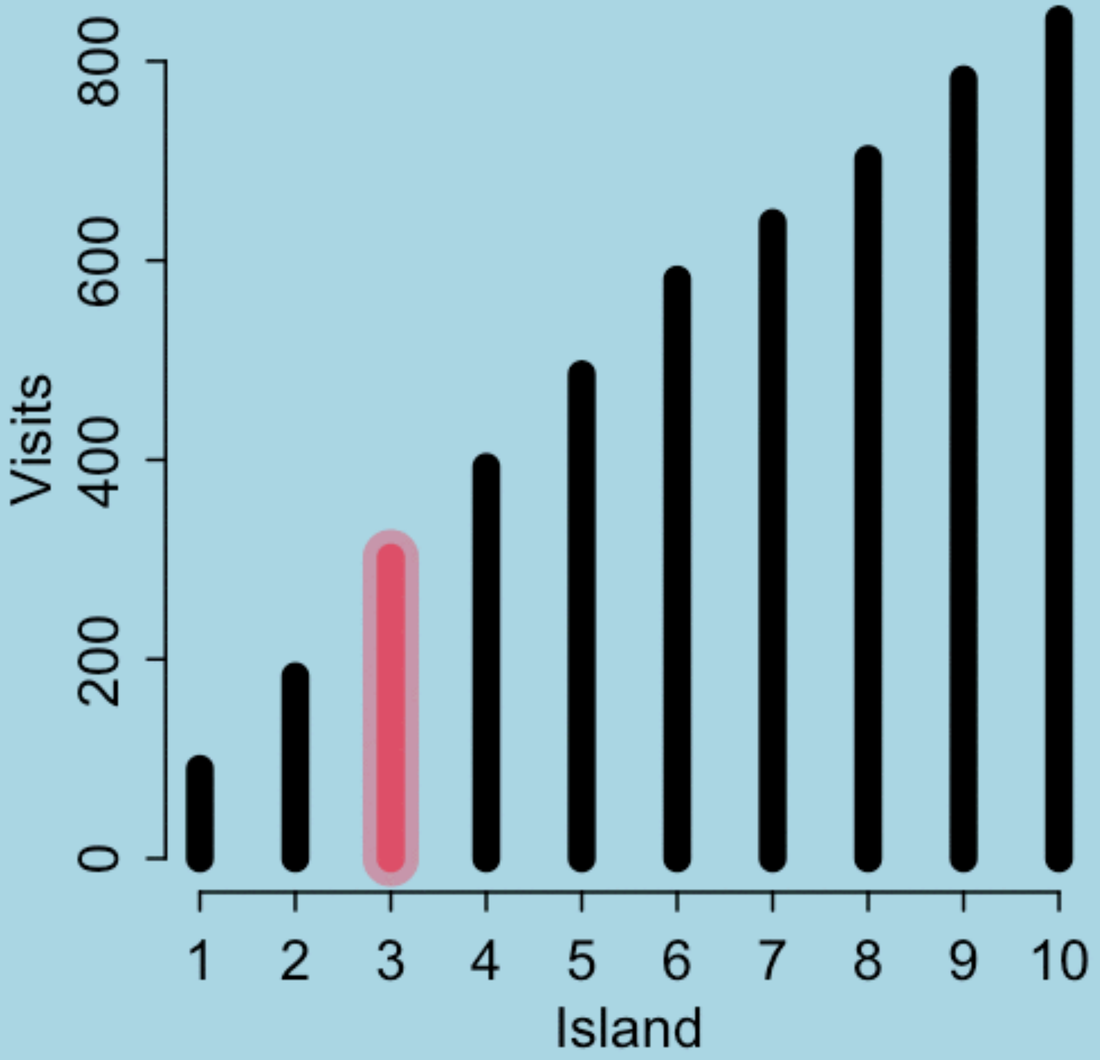
1 2 3 4 5 6 7

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


4
3
5
2
6
1
7
8
9

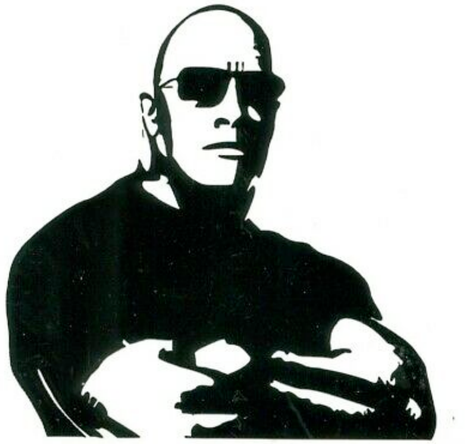

Visits
0
2
4
6
8
10
12
1
2
3
4
5
6
7
8
9
10
Island


4
5
2
6
1
7
10
8
9


Visits
0
200
400
600
800
1
2
3
4
5
6
7
8
9
10
Island

Markov chain Monte Carlo

Usual use: Draw samples from a posterior distribution



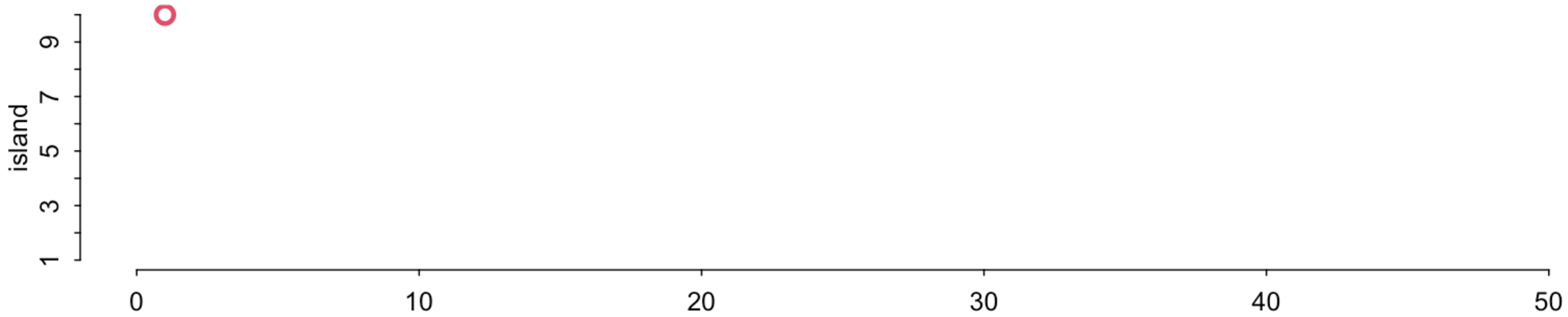
“Islands”: parameter values

“Population size”: posterior probability

Visit each parameter value in proportion to its posterior probability

Any number of dimensions (parameters)

“Markov chain Monte Carlo”

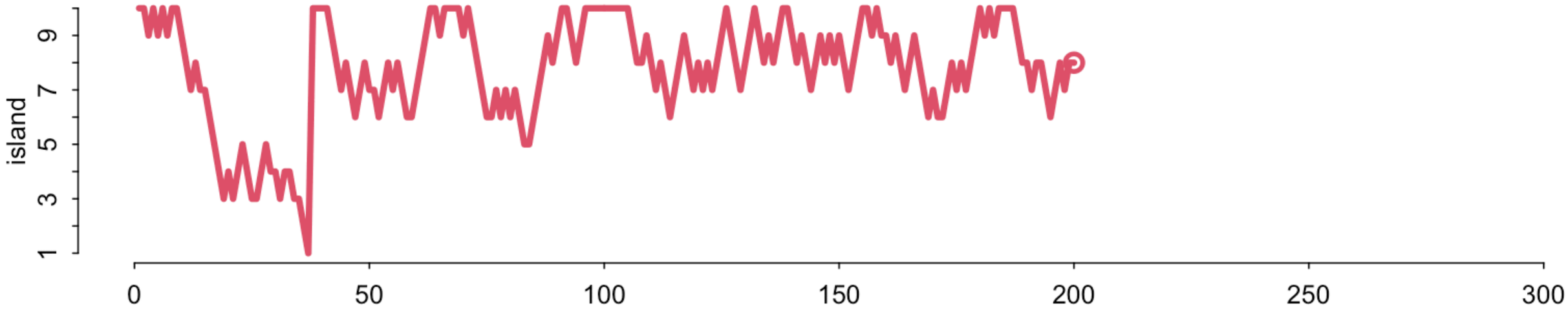


Chain: Sequence of draws from distribution

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

Monte Carlo: Random simulation

“Markov chain Monte Carlo”



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

Easy to write, very general, often inefficient

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.

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

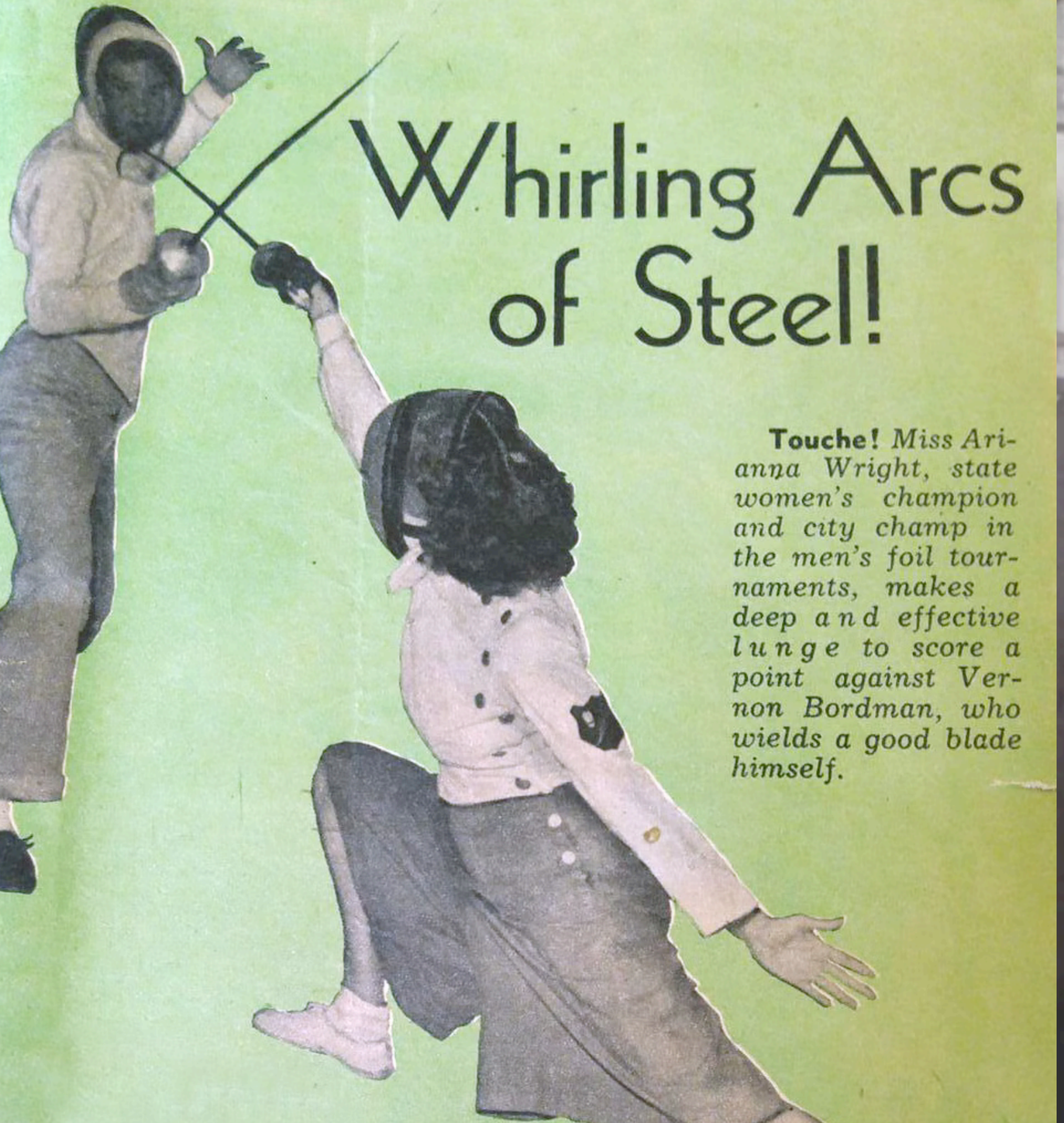
Equa

NICHOLAS METR

ER,

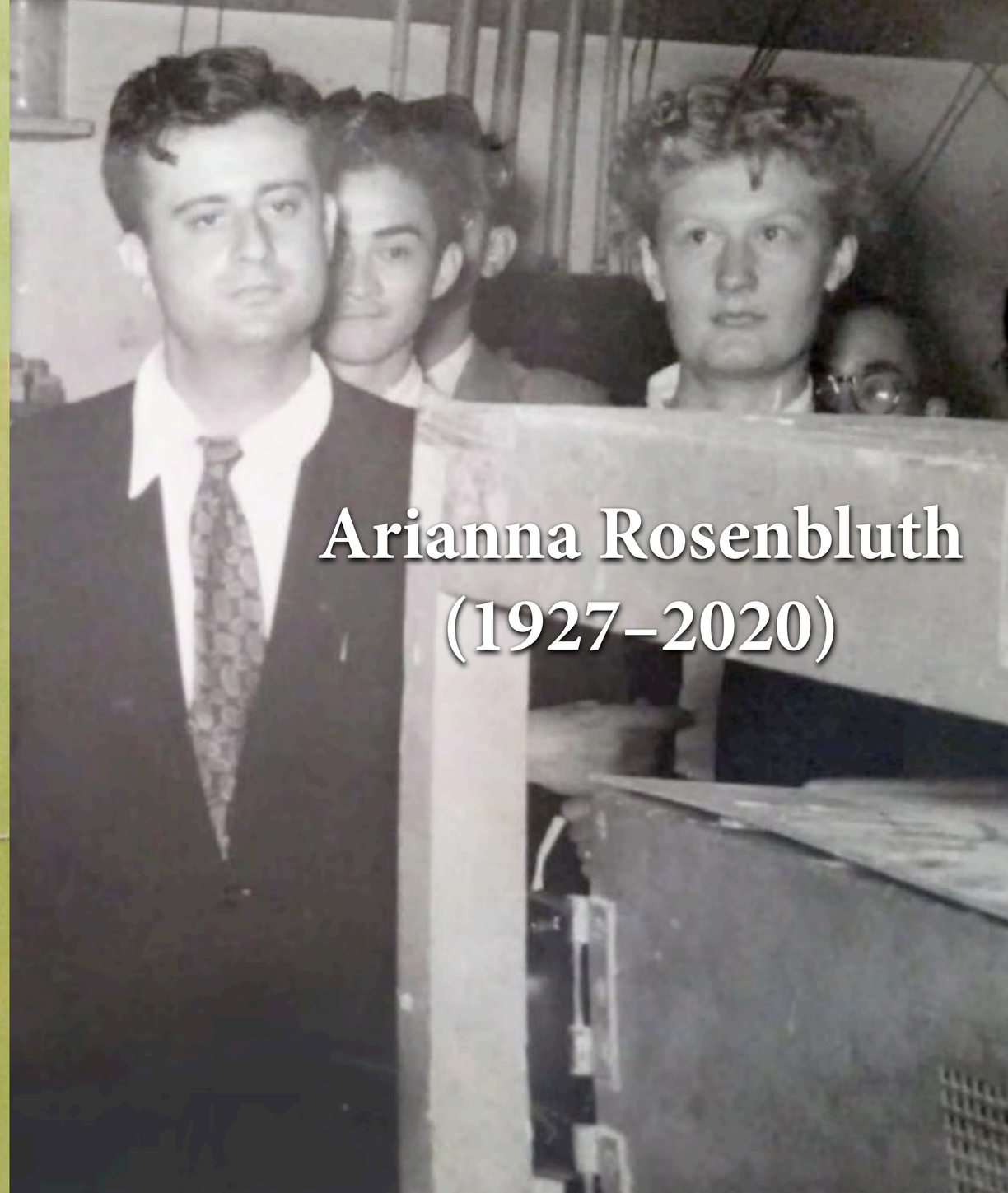
EL

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.

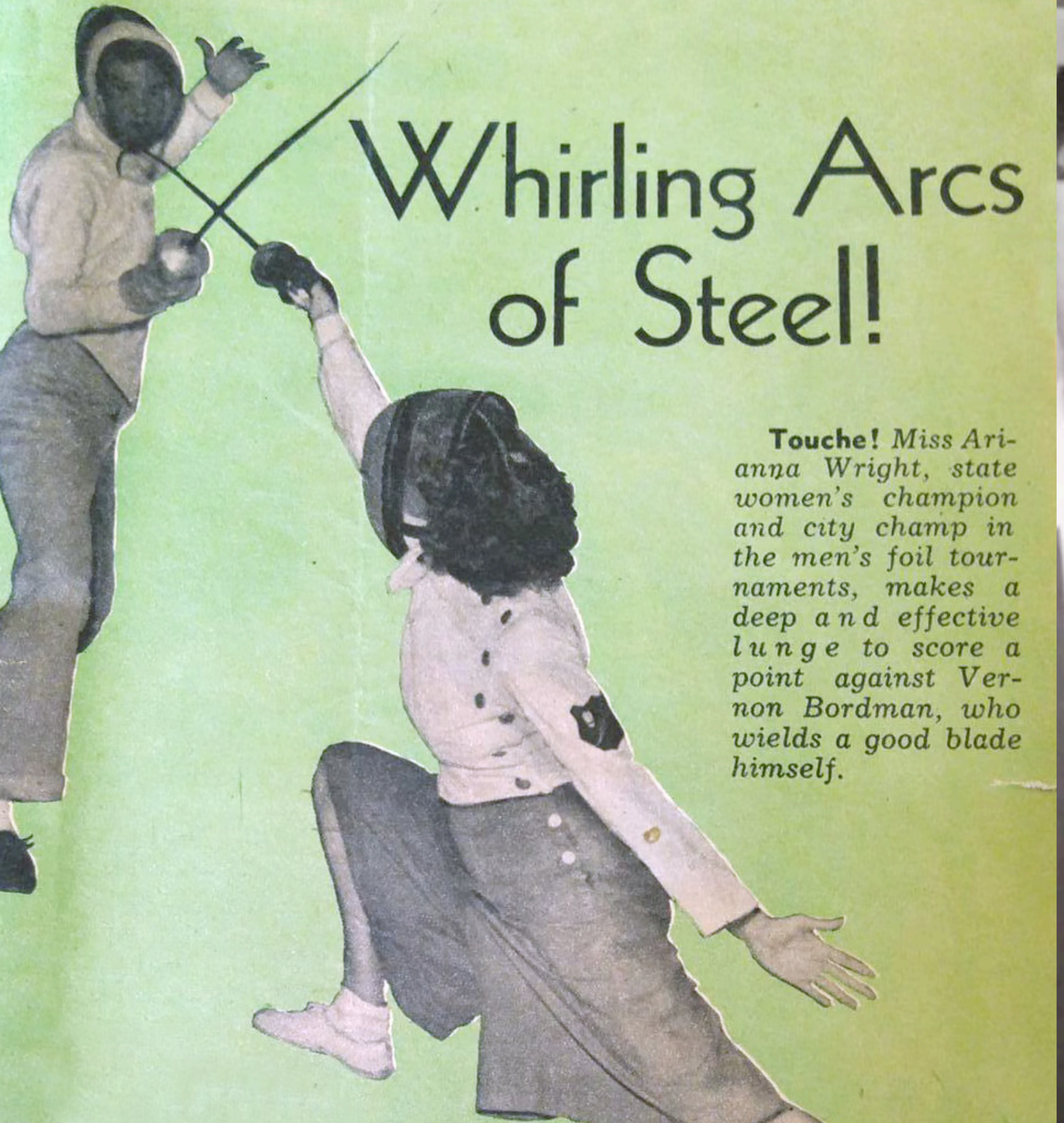


Whirling Arcs of Steel!

Touche! *Miss Arianna Wright, state women's champion and city champ in the men's foil tournaments, makes a deep and effective lunge to score a point against Vernon Bordman, who wields a good blade himself.*



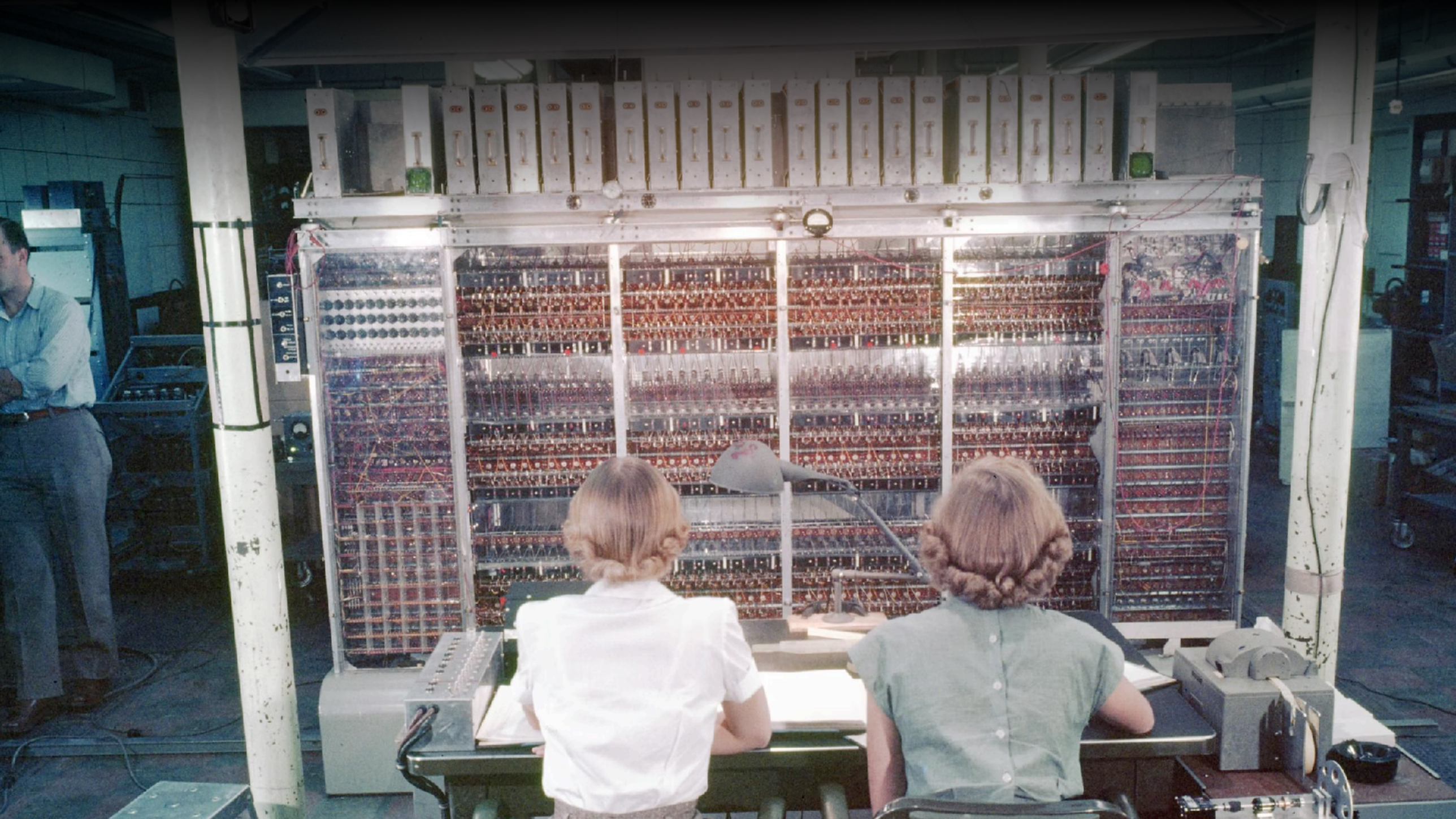
Arianna Rosenbluth (1927–2020)

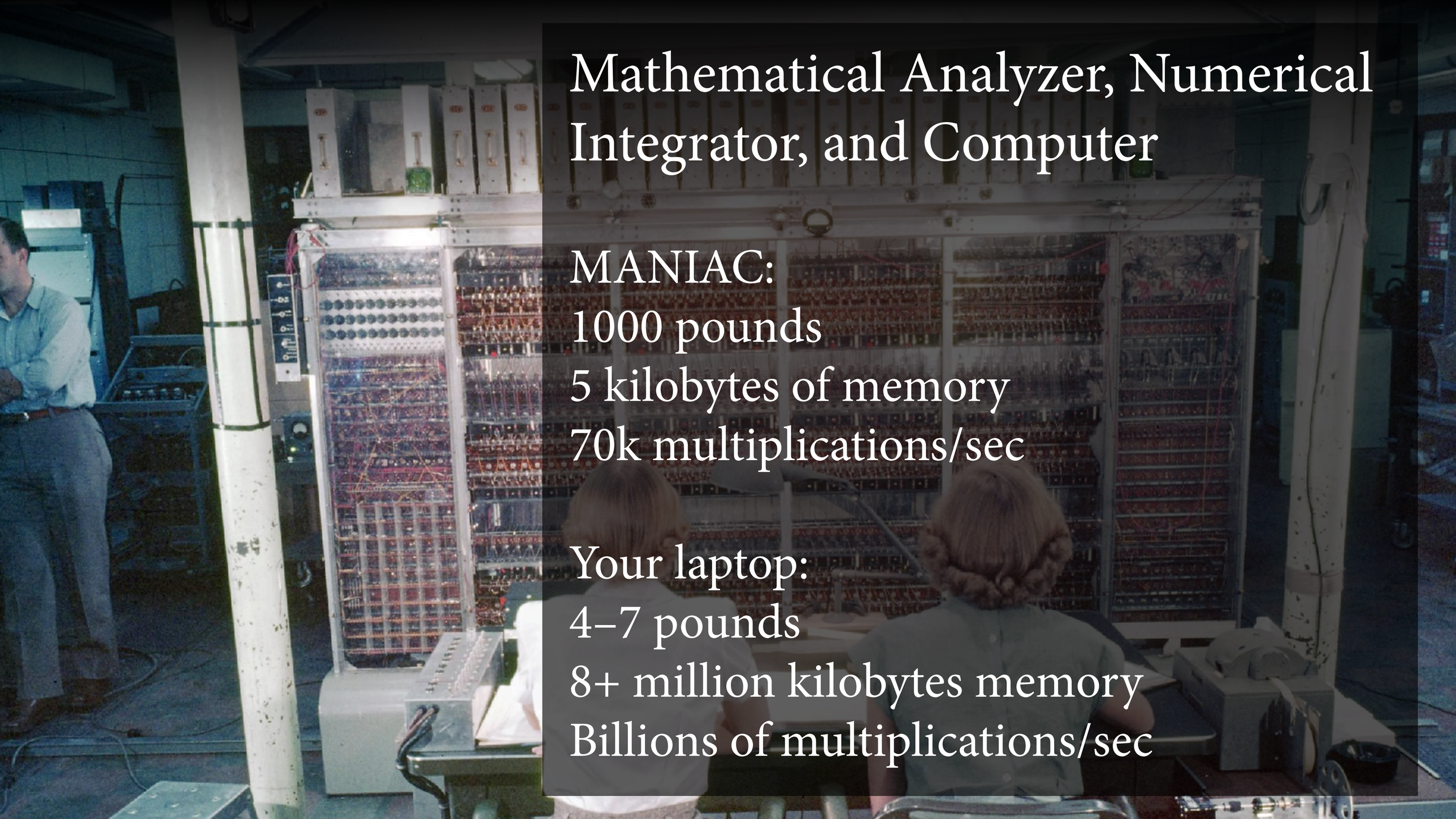


Whirling Arcs of Steel!

Touche! *Miss Arianna Wright, state women's champion and city champ in the men's foil tournaments, makes a deep and effective lunge to score a point against Vernon Bordman, who wields a good blade himself.*

Arianna Rosenbluth (1927–2020)





Mathematical Analyzer, Numerical Integrator, and Computer

MANIAC:
1000 pounds
5 kilobytes of memory
70k multiplications/sec

Your laptop:
4–7 pounds
8+ million kilobytes memory
Billions of multiplications/sec

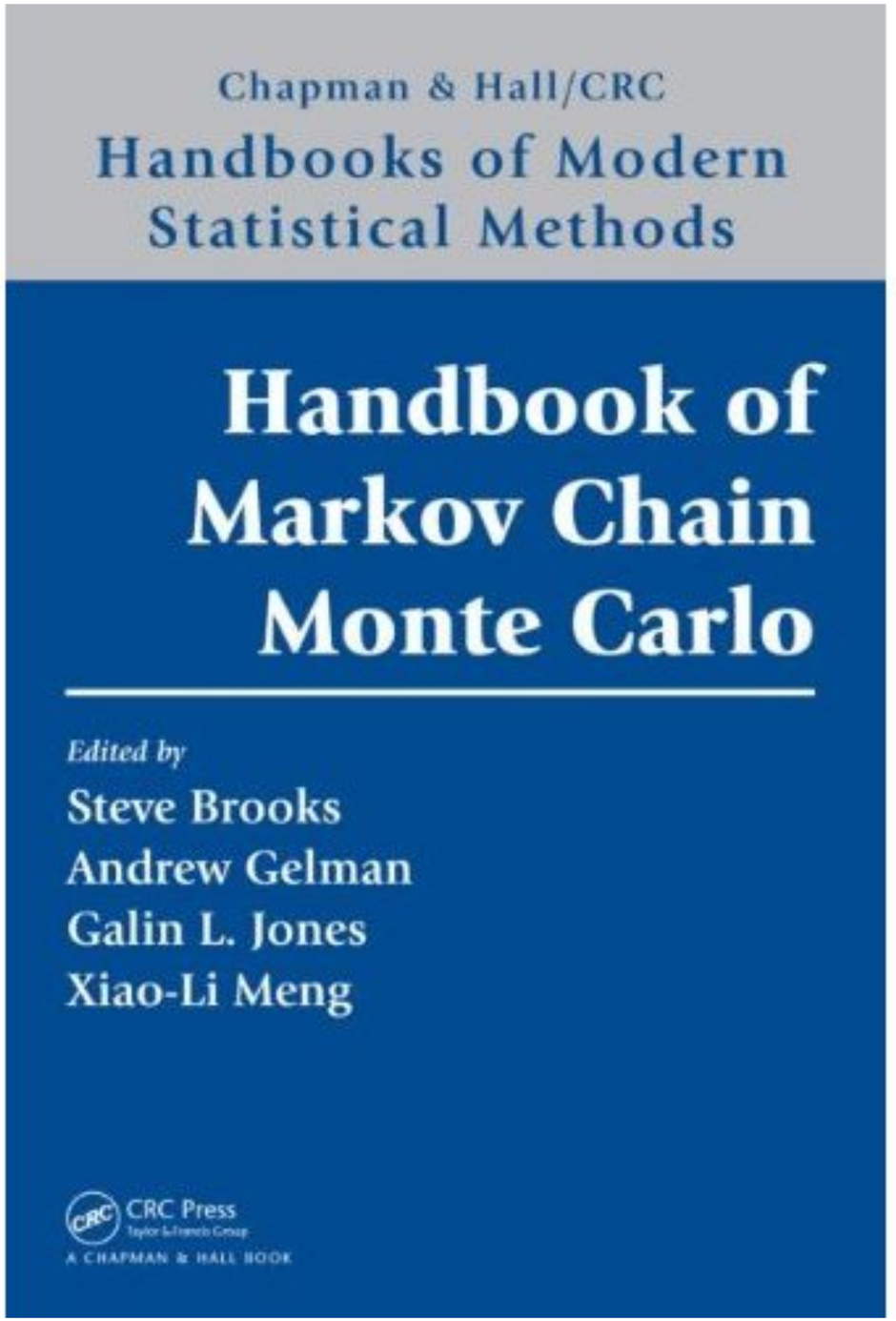
MCMC is diverse

Metropolis has yielded to newer, more efficient algorithms

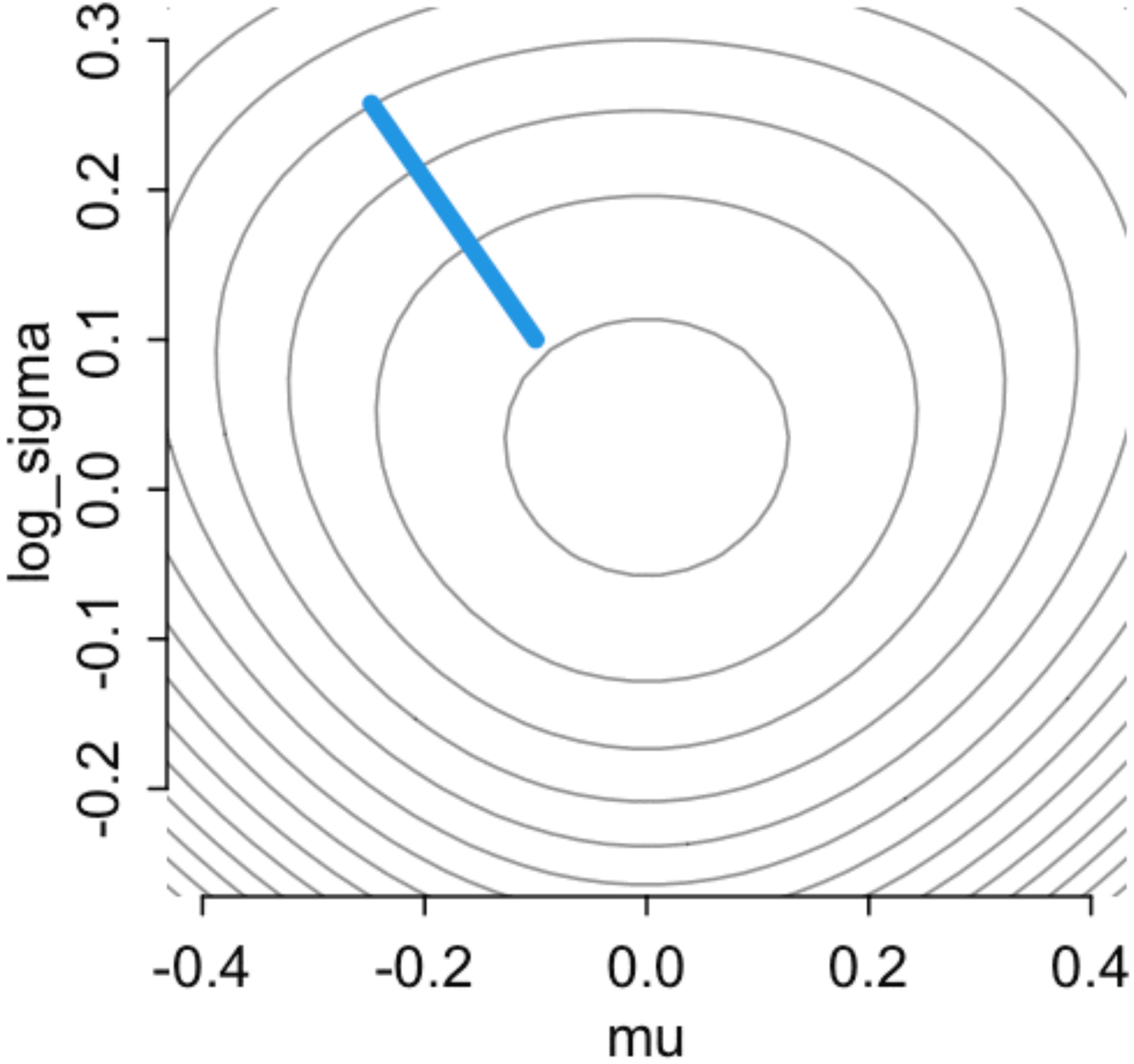
Many innovations in the last decades

Best methods use ***gradients***

We'll use Hamiltonian Monte Carlo

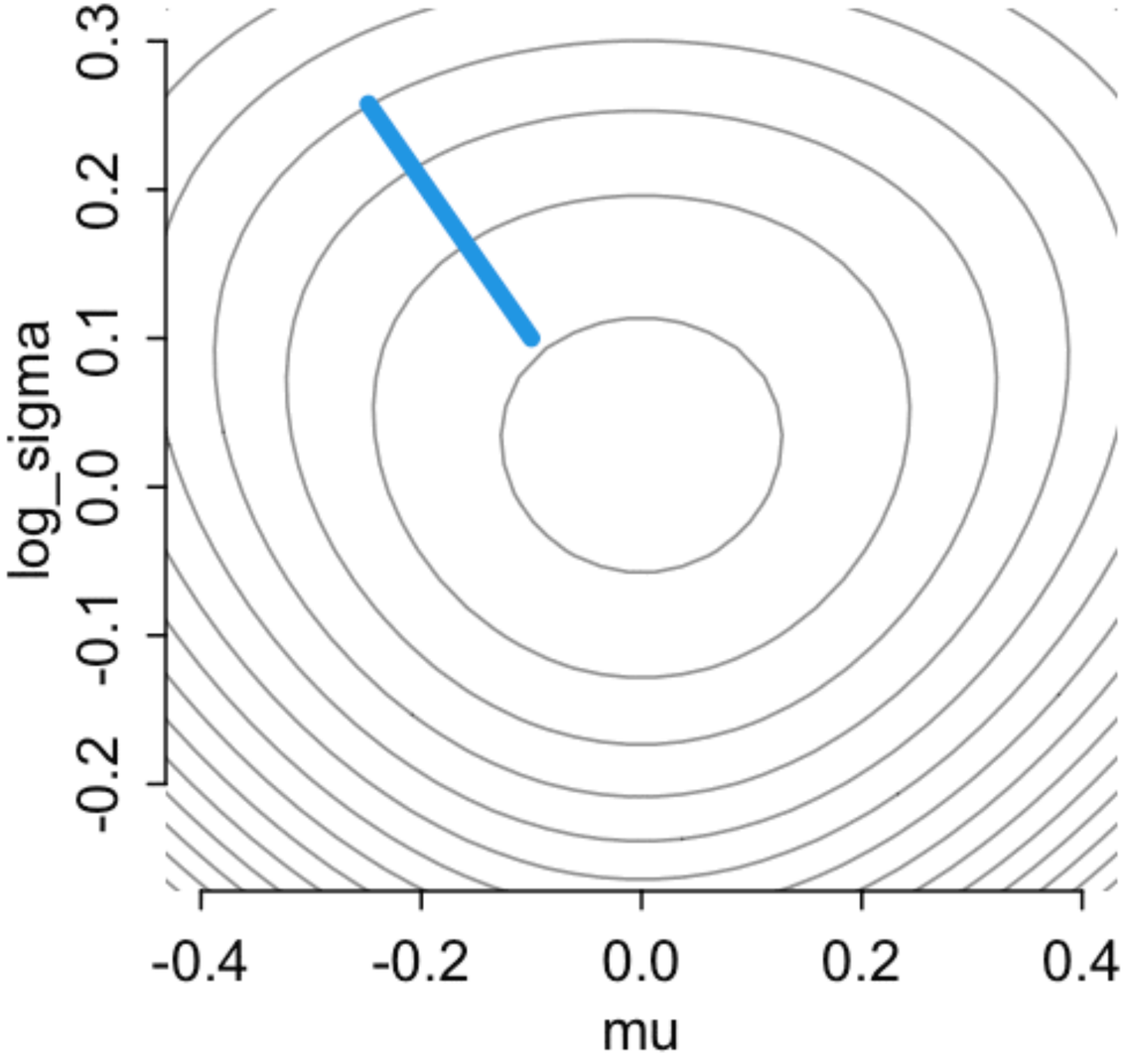


Basic Rosenbluth (aka Metropolis) algorithm

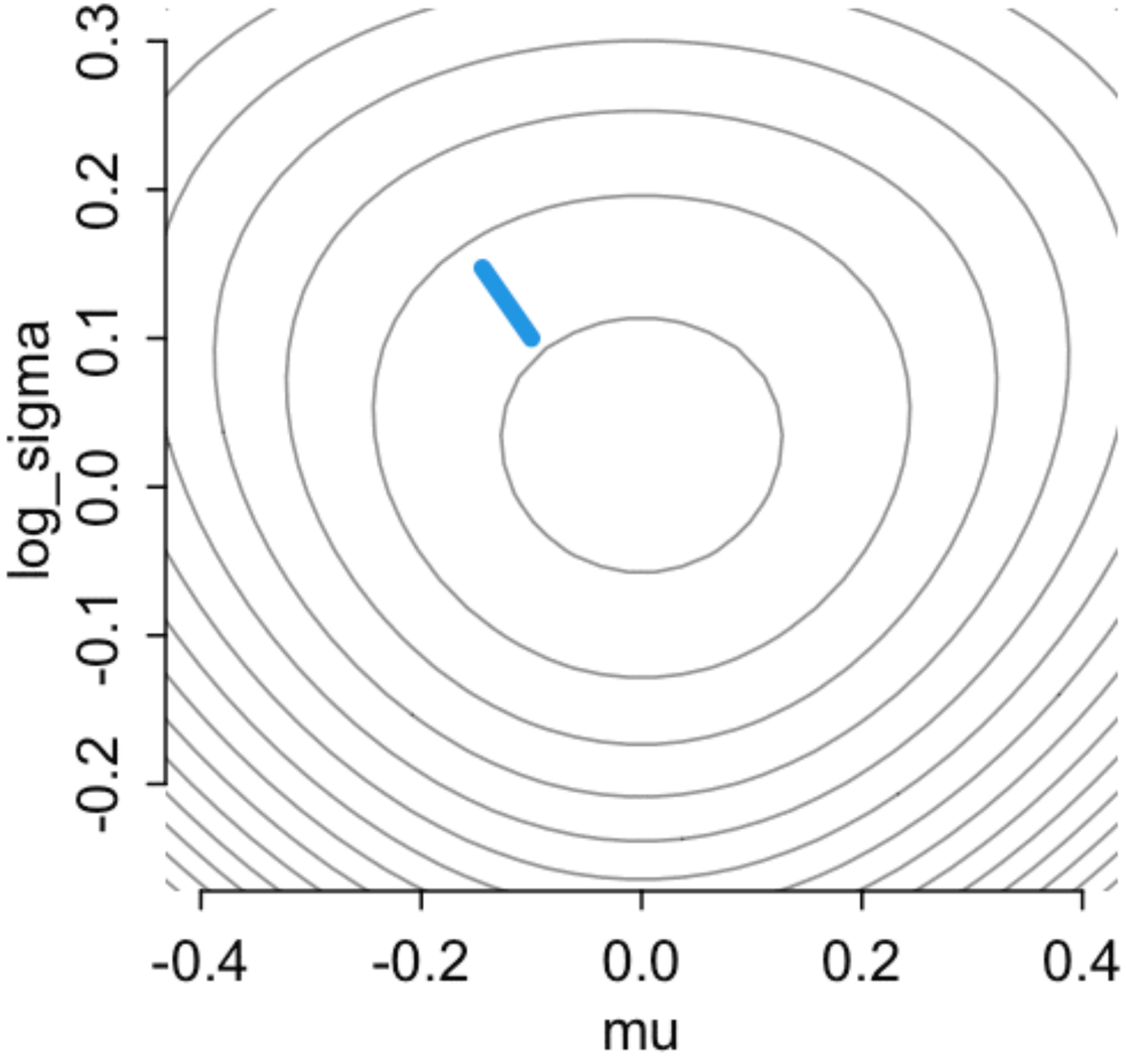


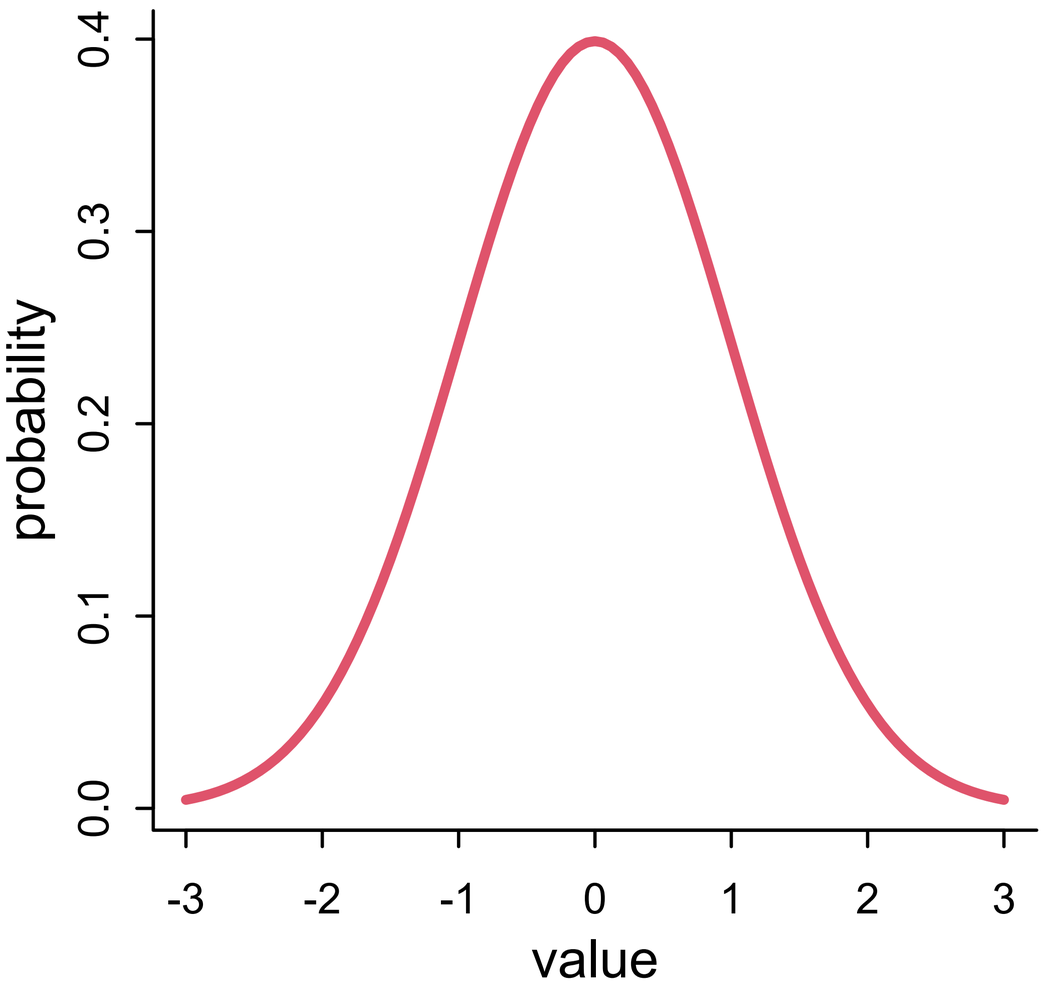
Basic Rosenbluth (aka Metropolis) algorithm

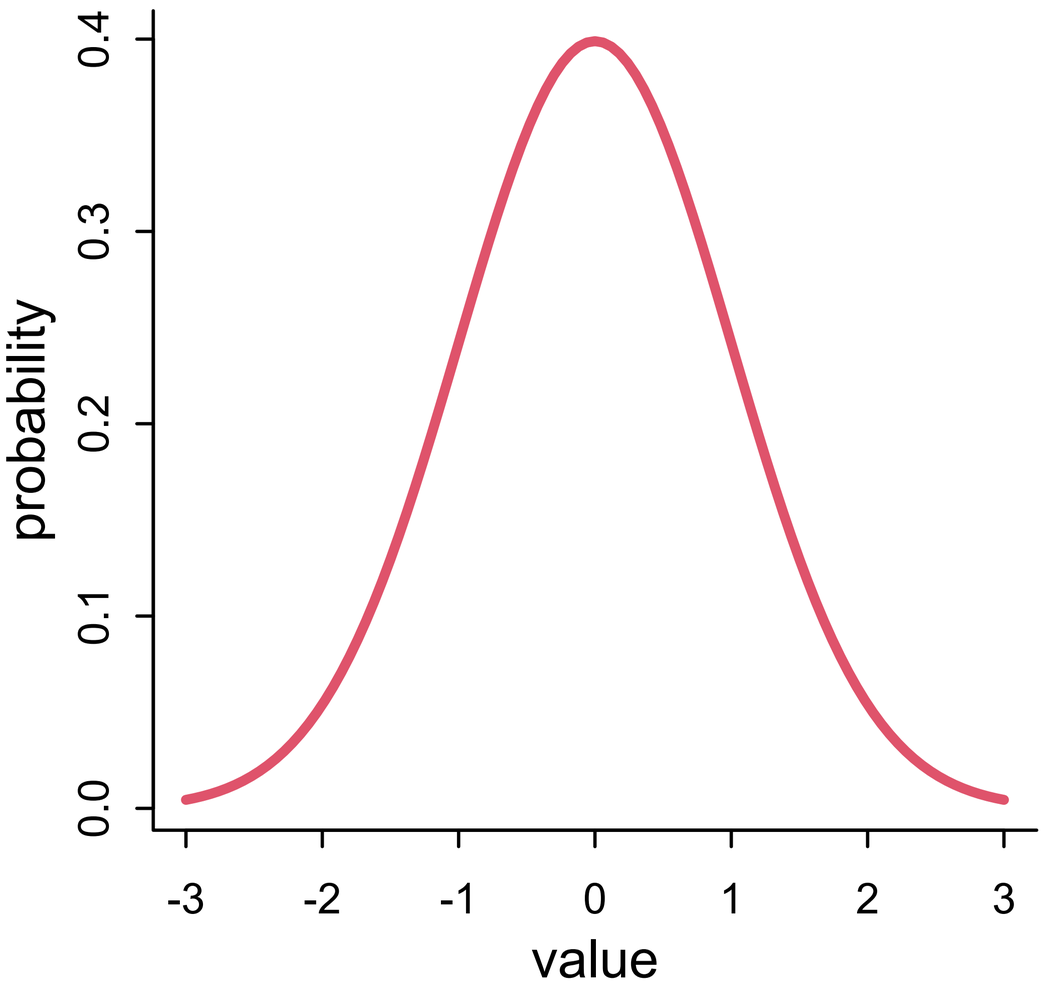
large step size

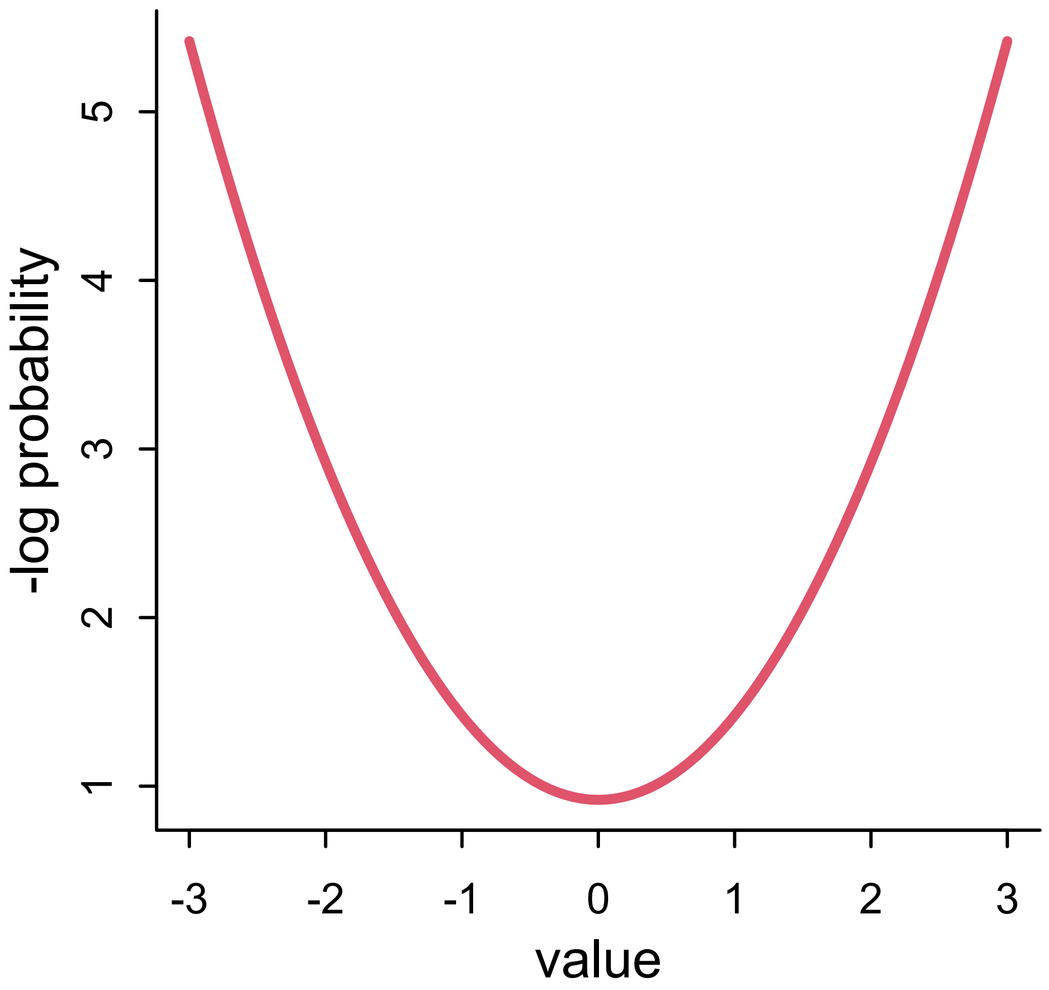


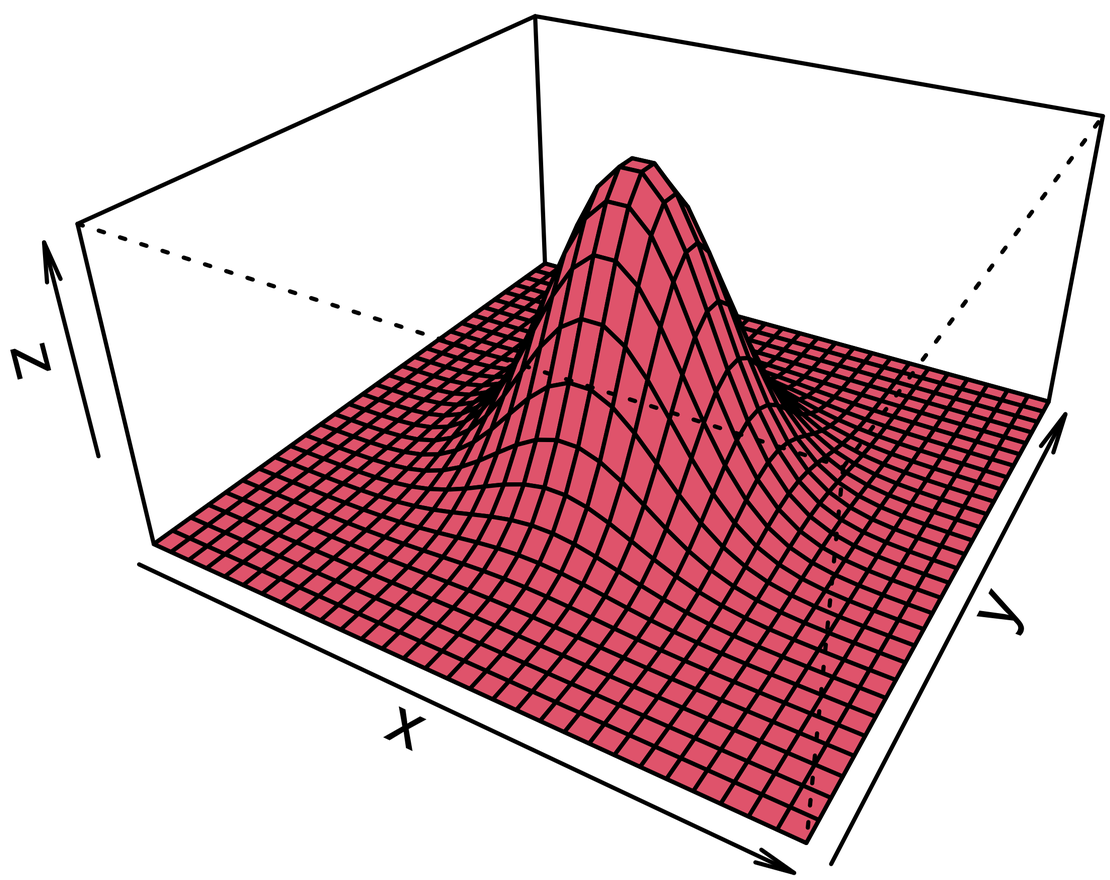
small step size

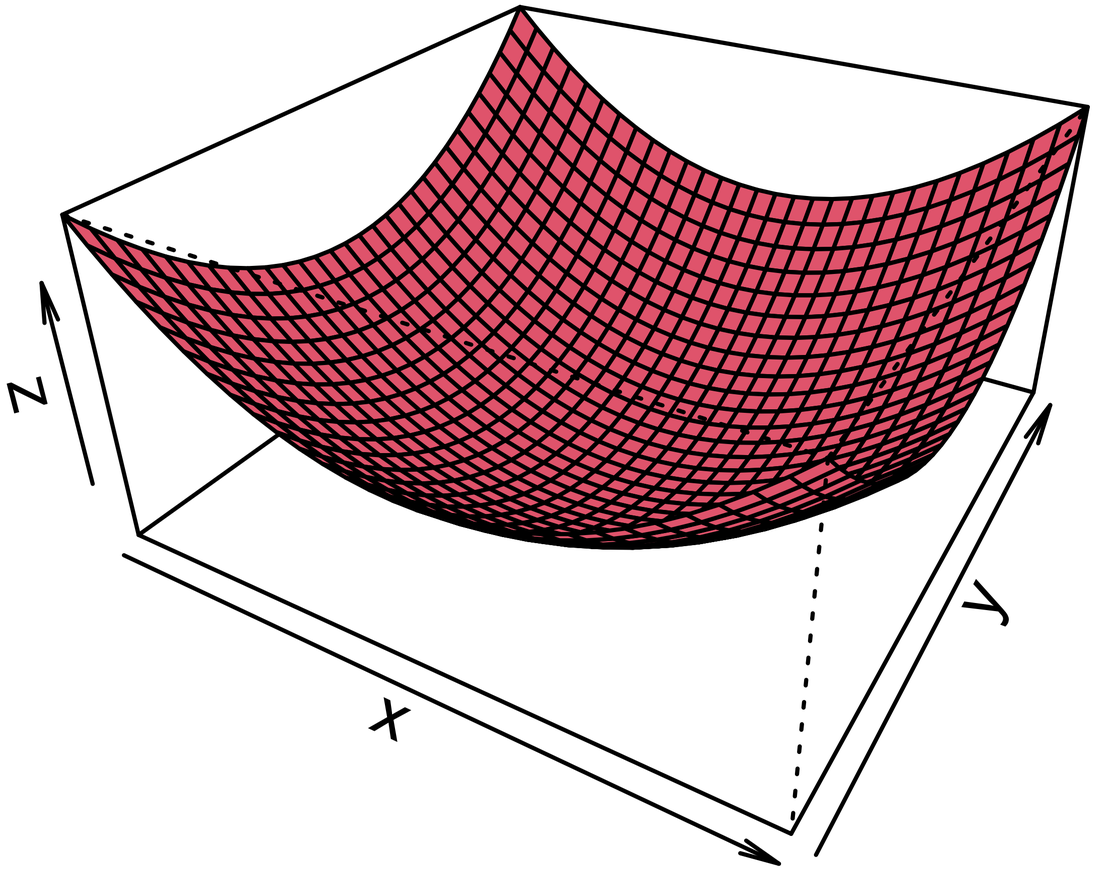



probability
0.0
0.1
0.2
0.3
0.4
-3
-2
-1
0
1
2
3
value


probability
0.0
0.1
0.2
0.3
0.4
-3
-2
-1
0
1
2
3
value

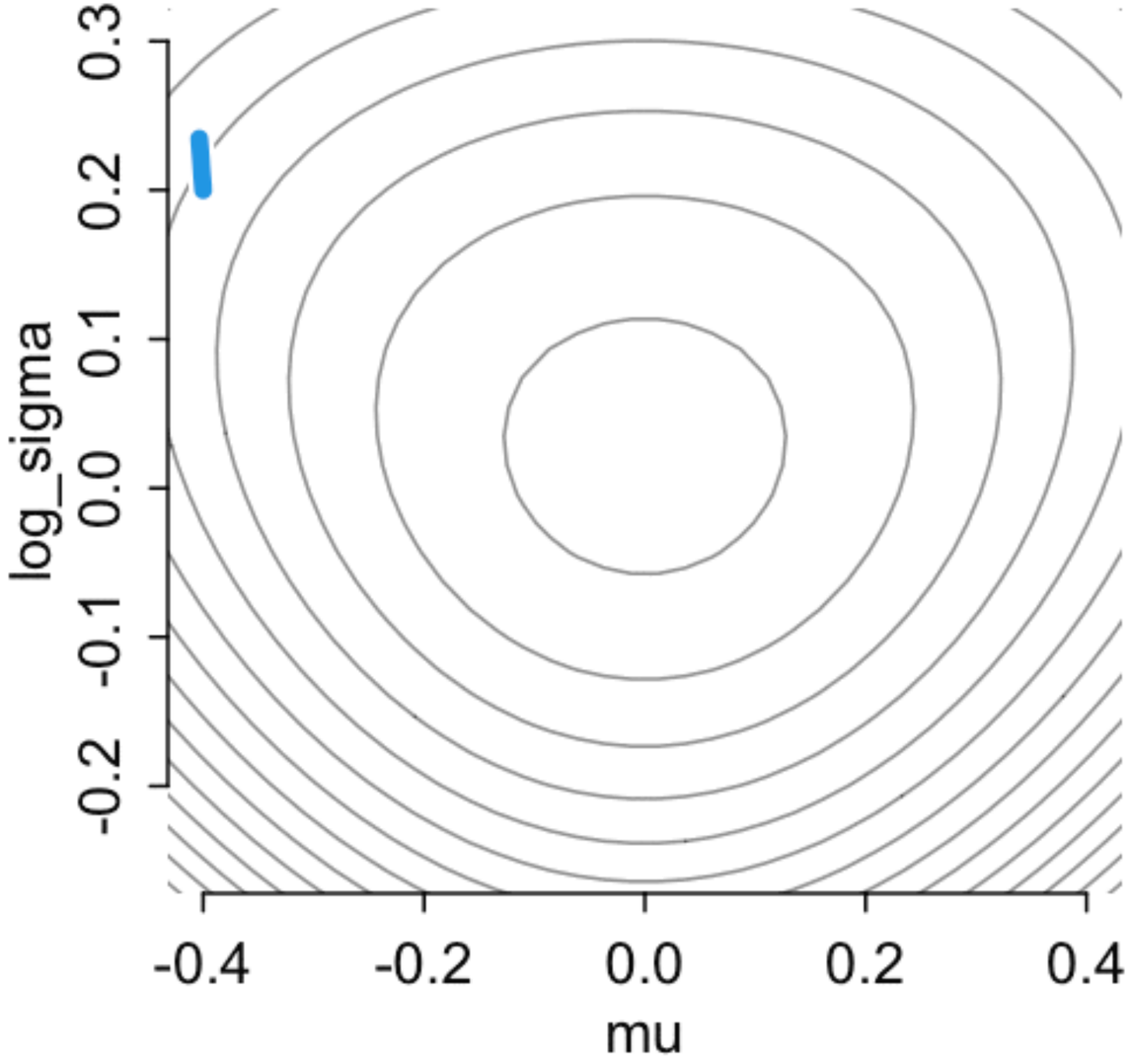

-log probability
1
2
3
4
5
-3
-2
-1
0
1
2
3
value


z
x
y

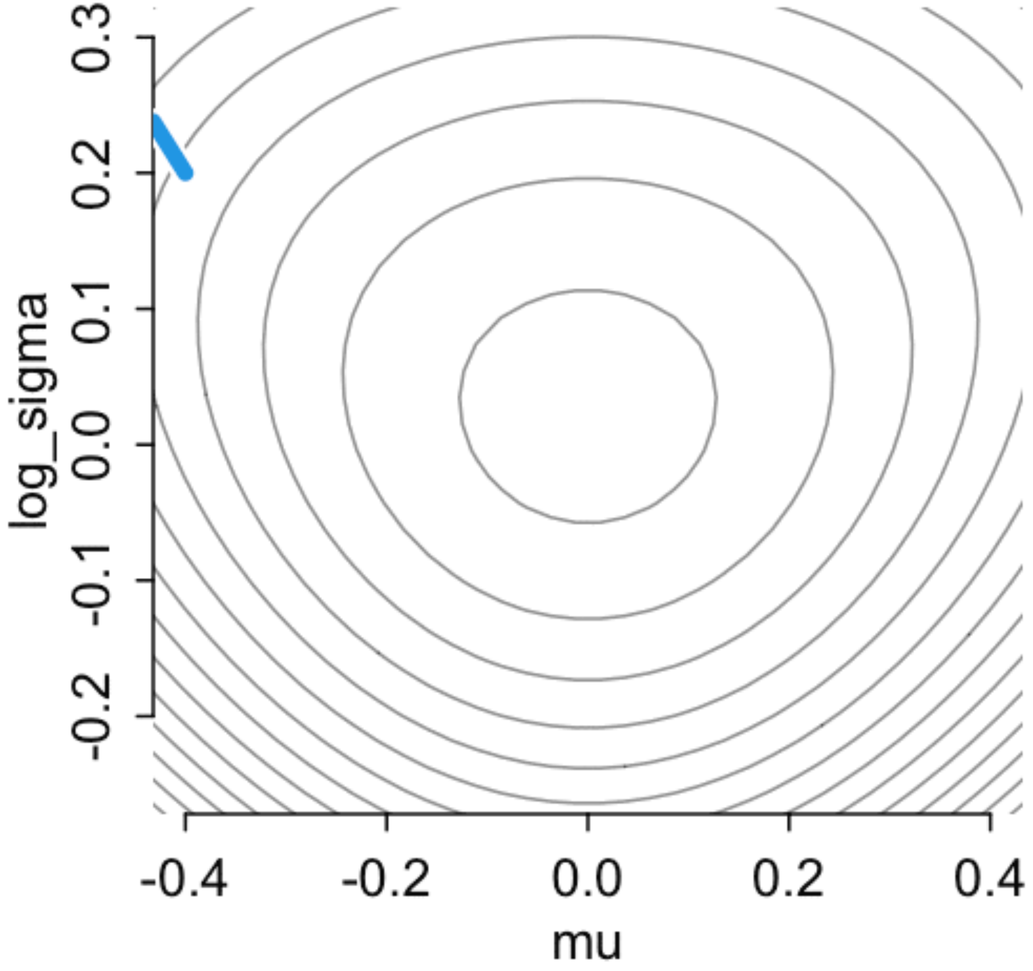

z
x
y

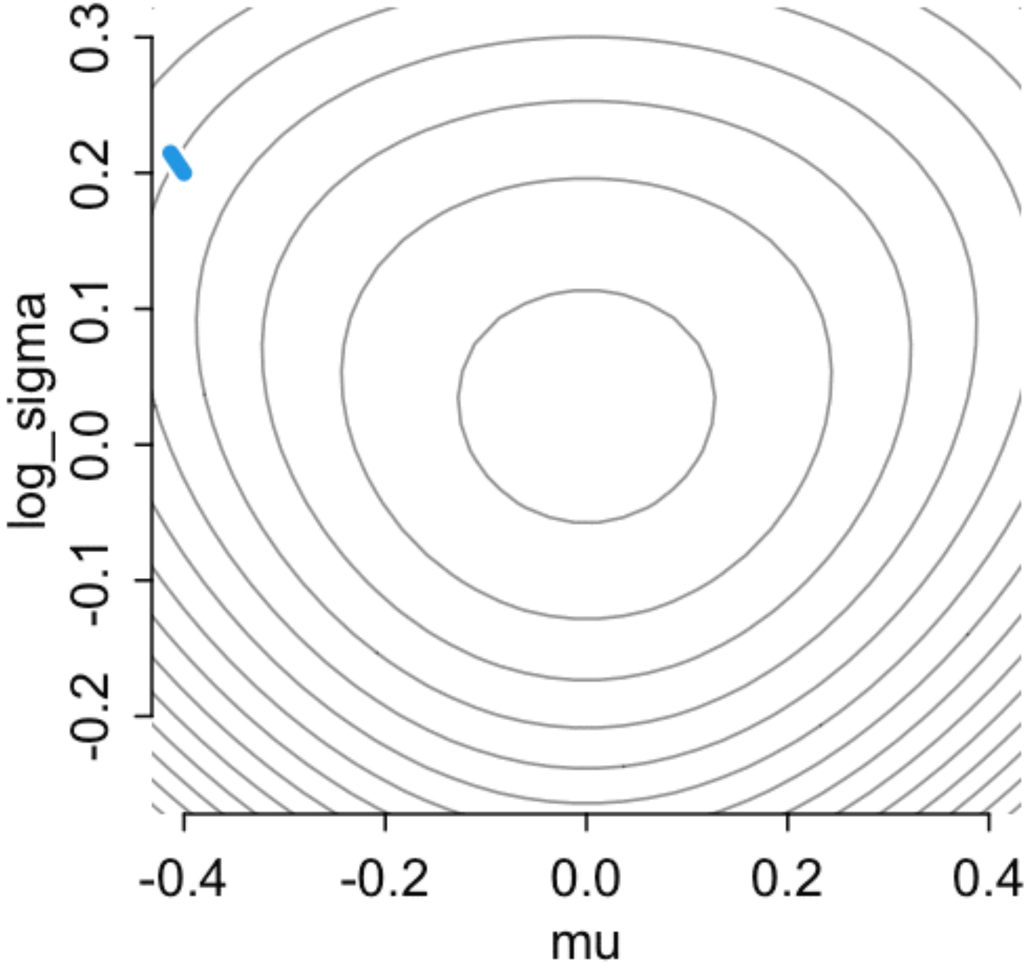

Low probability
High probability

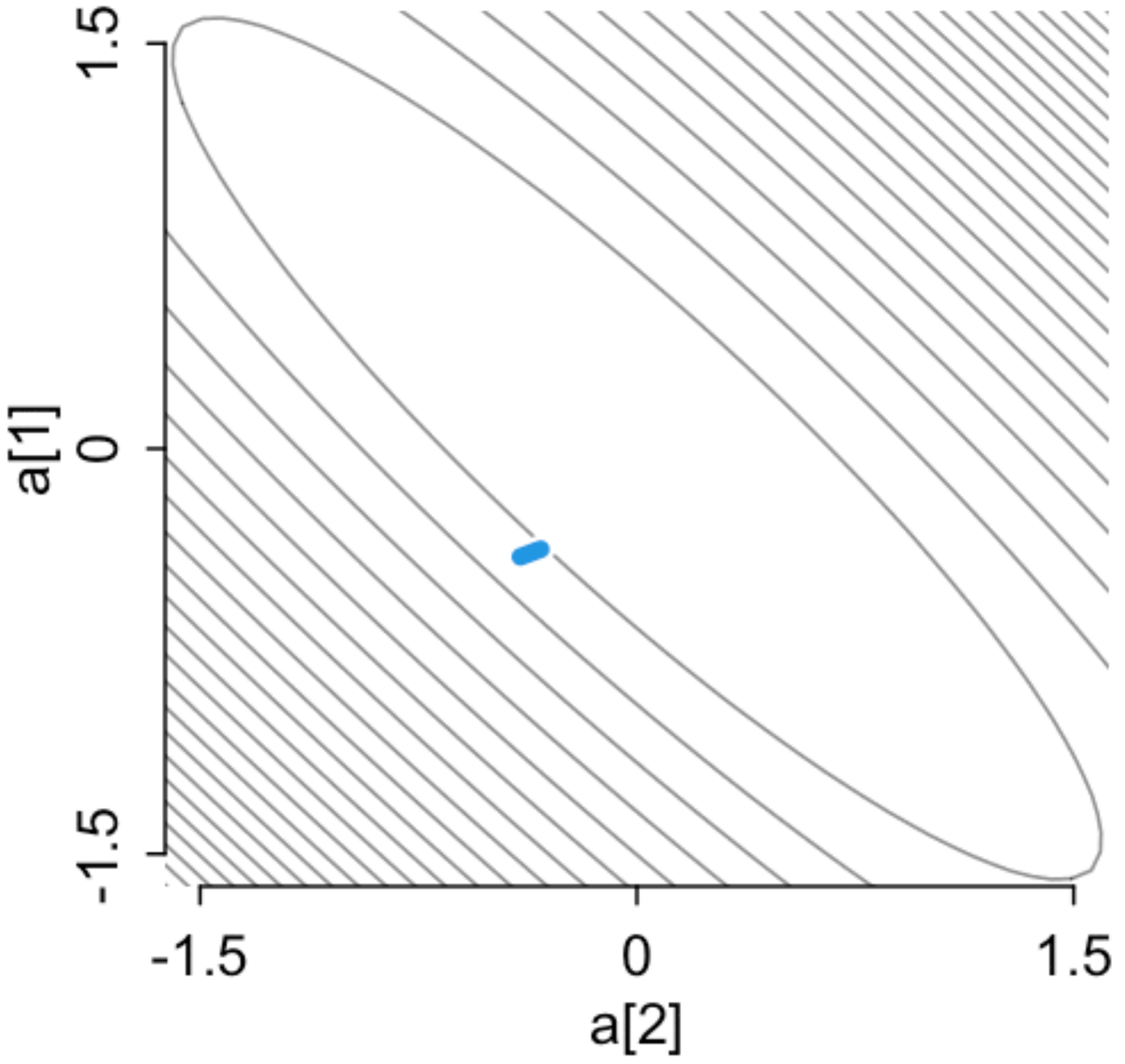
Hamiltonian Monte Carlo



Hamiltonian Monte Carlo






1.5
0
-1.5
a[1]
-1.5
0
1.5
a[2]

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:

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

Calculus is a superpower

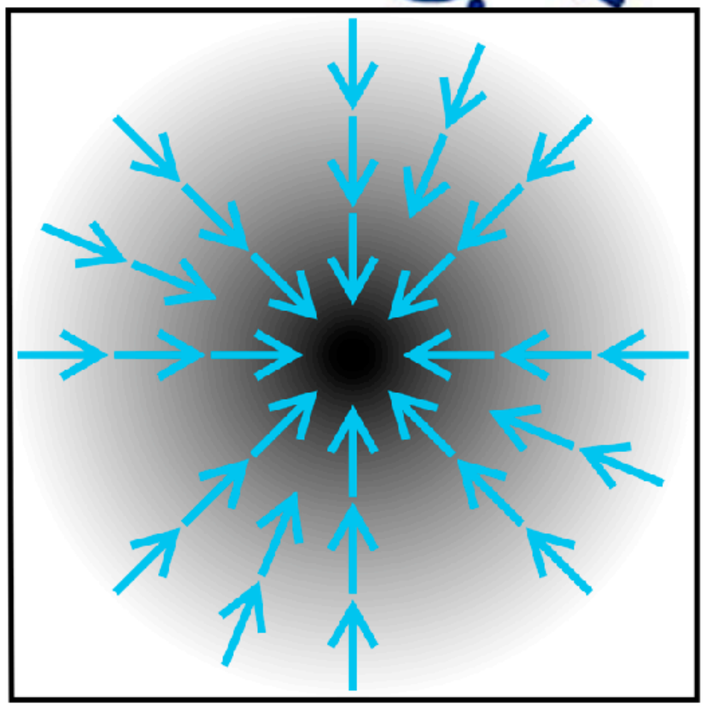
Hamiltonian Monte Carlo needs gradients

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

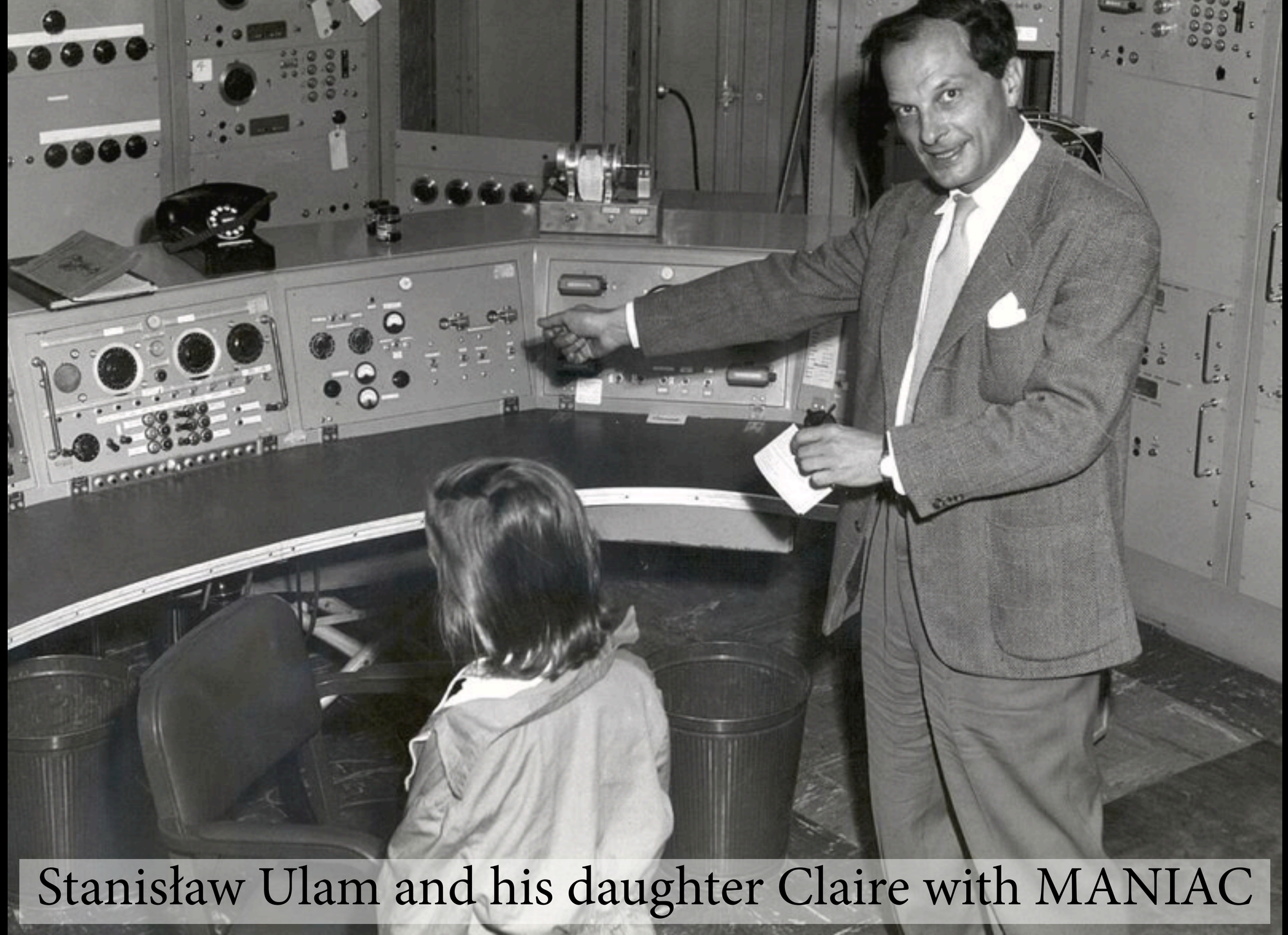


mc-stan.org

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.



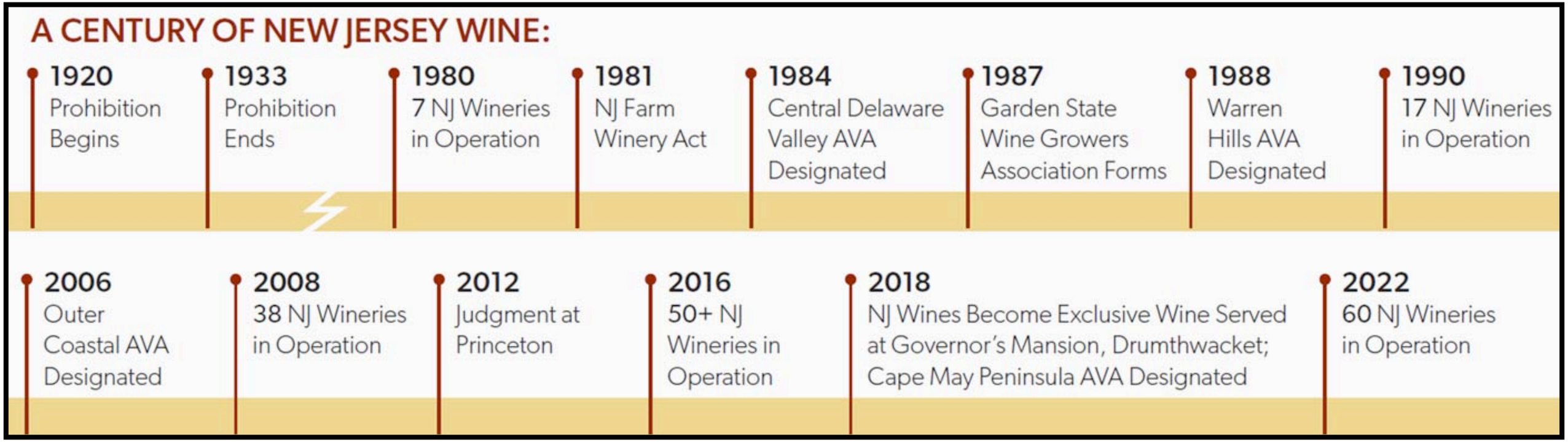
Stanisław Ulam and his daughter Claire with MANIAC

PAUSE

Example: The Judgment at Princeton

Example: The Judgment at Princeton

```
library(rethinking)
data(Wines2012)
```



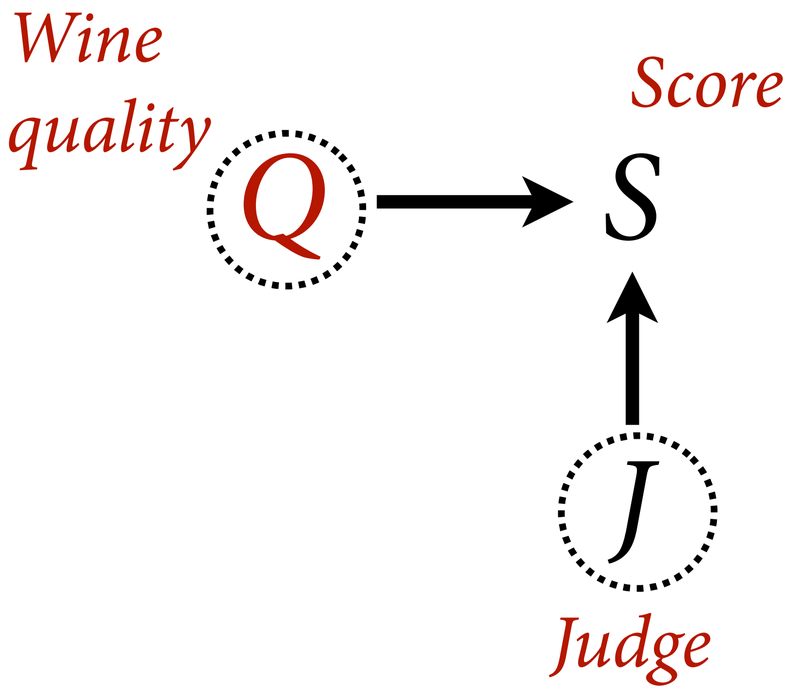
Example: The Judgment at Princeton

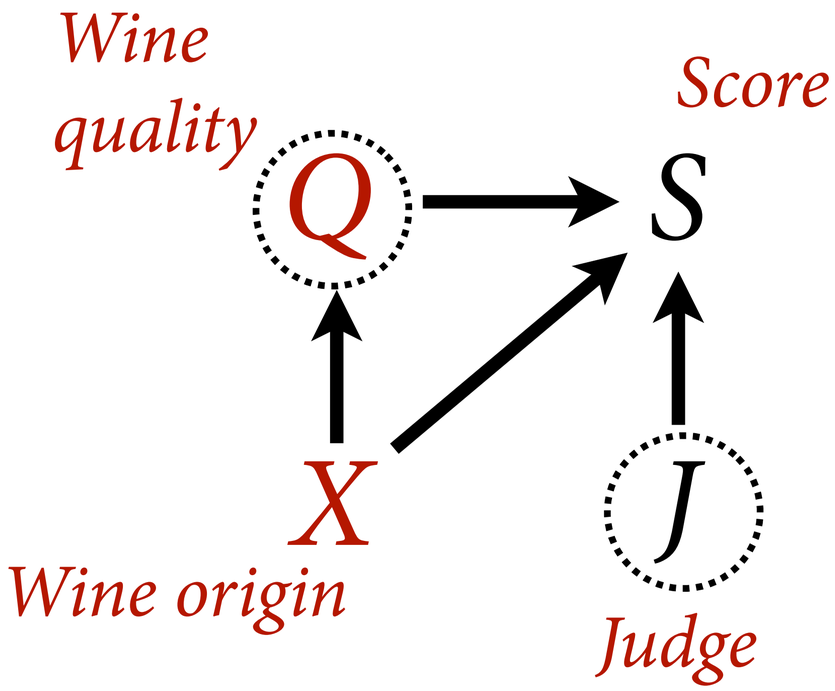
```
library(rethinking)
data(Wines2012)
```

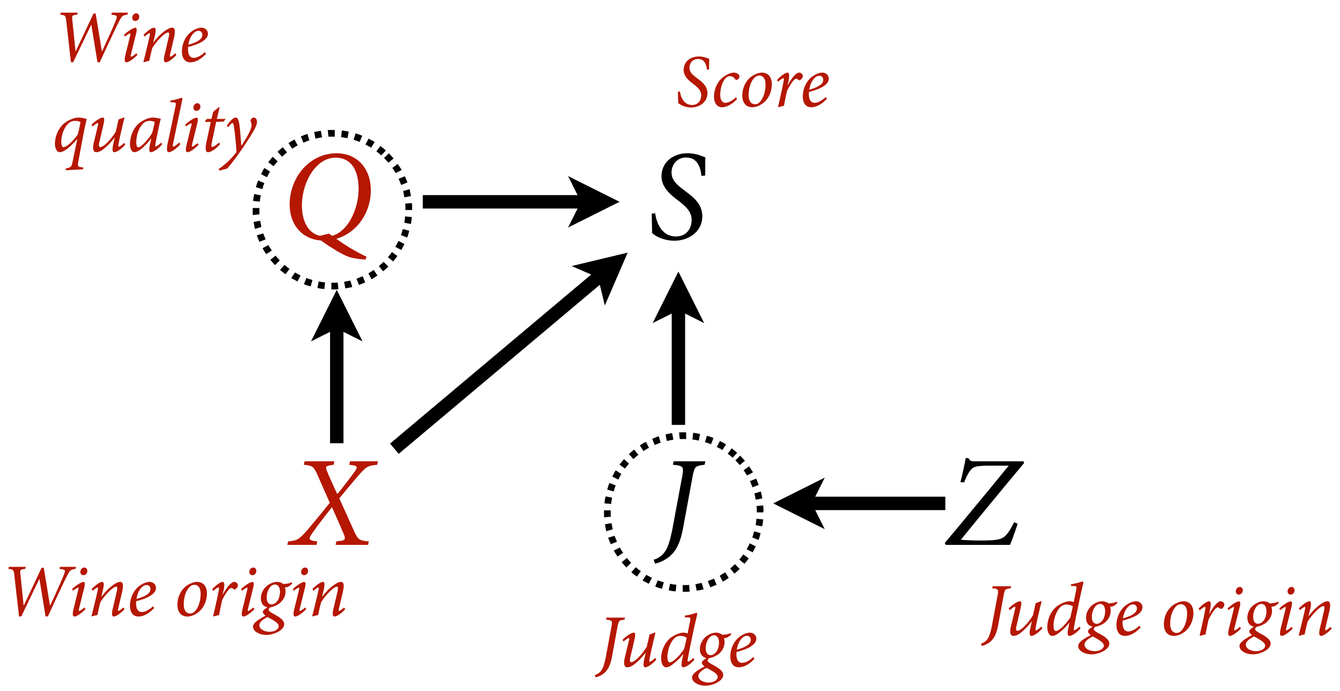
20 wines (10 French, 10 NJ)

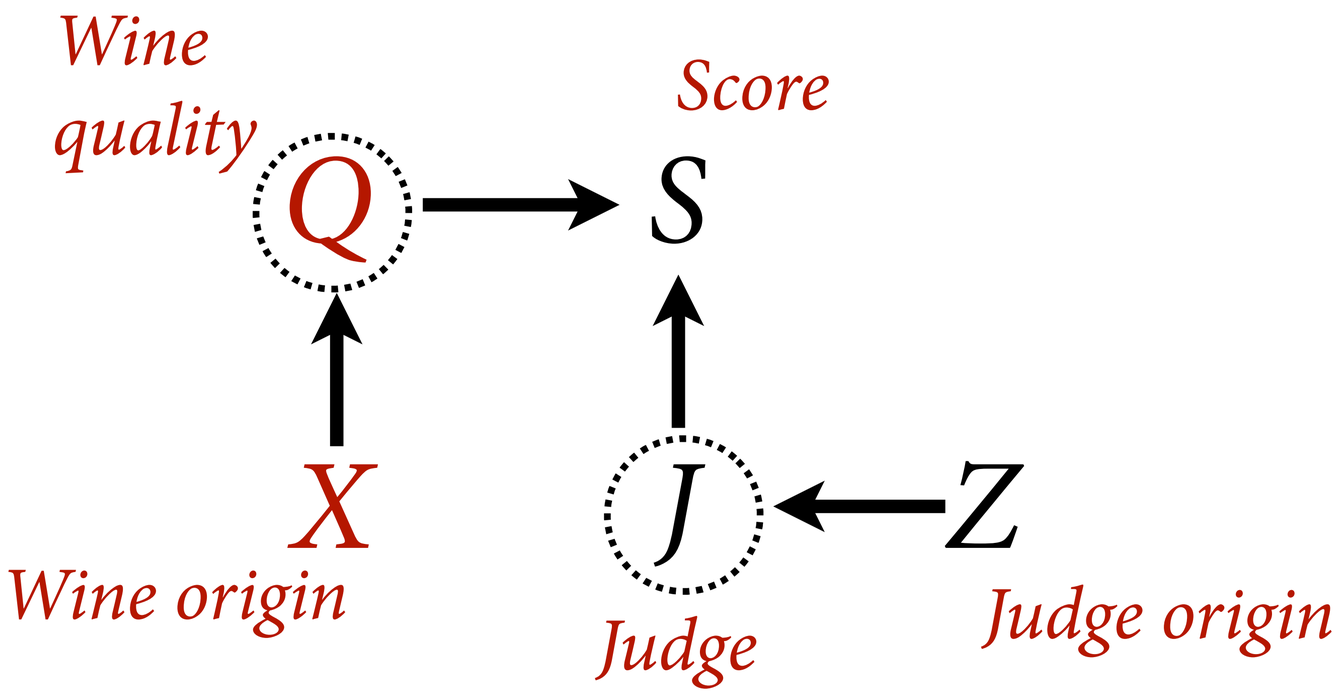
9 French & American judges

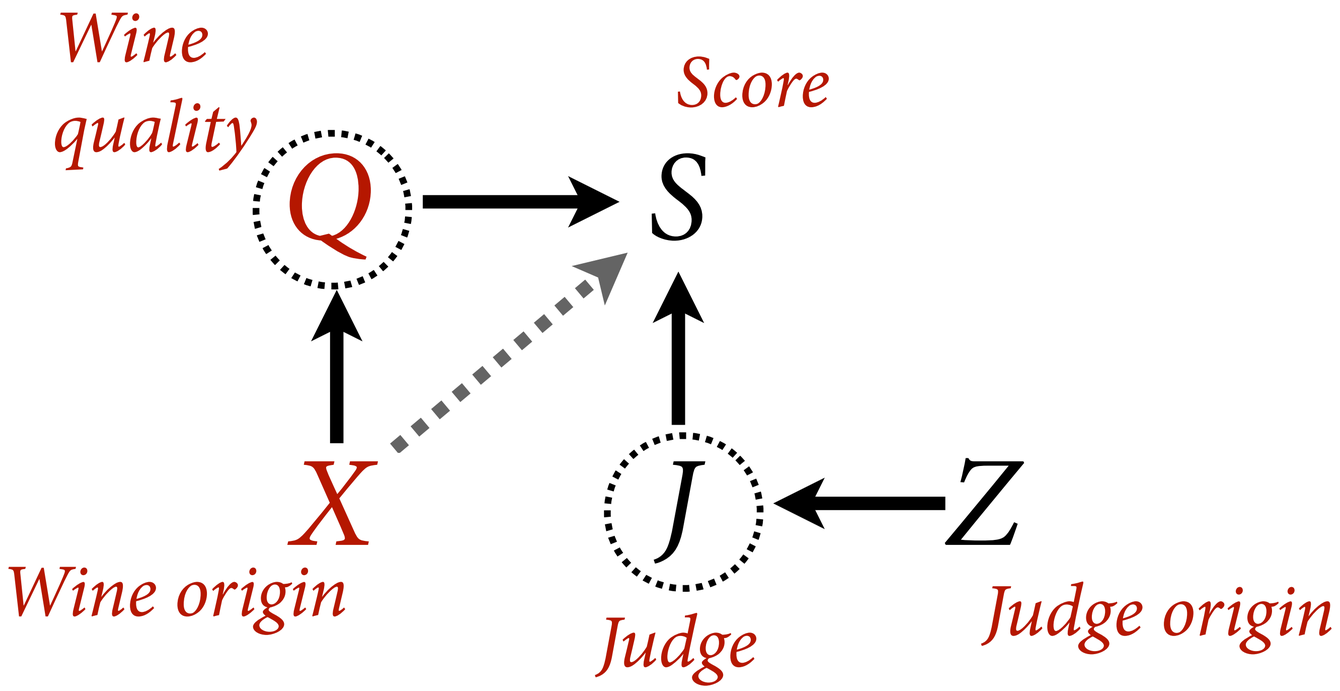
180 scores

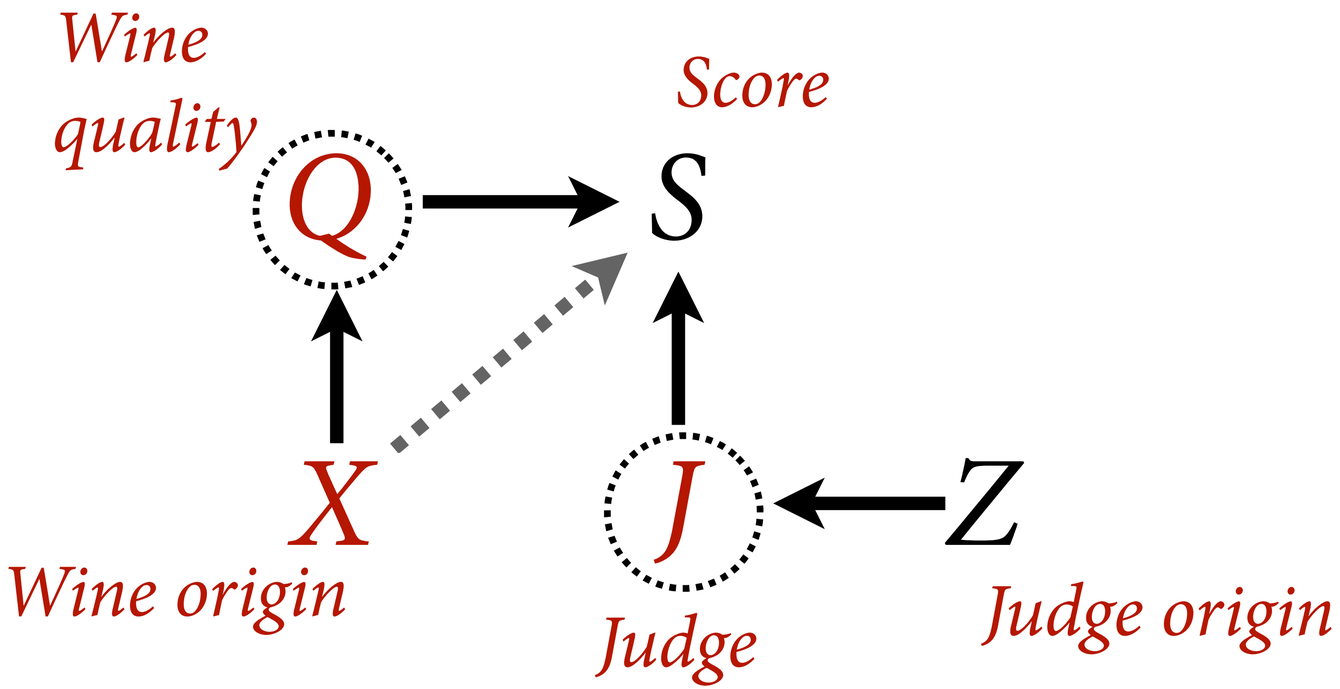

Wine
quality
Q
Score
S
J
Judge


Wine
quality
Q
Score
S
X
Wine origin
J
Judge


Wine
quality
Score
Q
S
X
J
Z
Wine origin
Judge
Judge origin

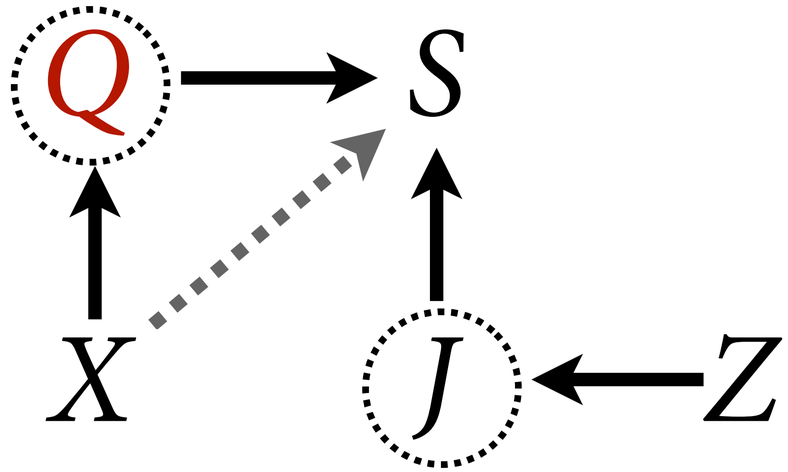

Wine
quality
Score
Q
S
X
J
Z
Wine origin
Judge
Judge origin


Wine
quality
Score
Q
S
X
J
Z
Wine origin
Judge
Judge origin



Estimand: Association between wine quality and wine origin. Stratify by judge for efficiency.

```r
library(rethinking)
data(Wines2012)
d <- Wines2012

dat <- list(
    S=standardize(d$score),
    J=as.numeric(d$judge),
    W=as.numeric(d$wine),
    X=ifelse(d$wine.amer==1,1,2),
    Z=ifelse(d$judge.amer==1,1,2)
)

mQ <- ulam(
    alist(
        S ~ dnorm(mu,sigma),
        mu <- Q[W],
        Q[W] ~ dnorm(0,1),
        sigma ~ dexp(1)
    ) , data=dat , chains=4 , cores=4 )

precis(mQ,2)
```



$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

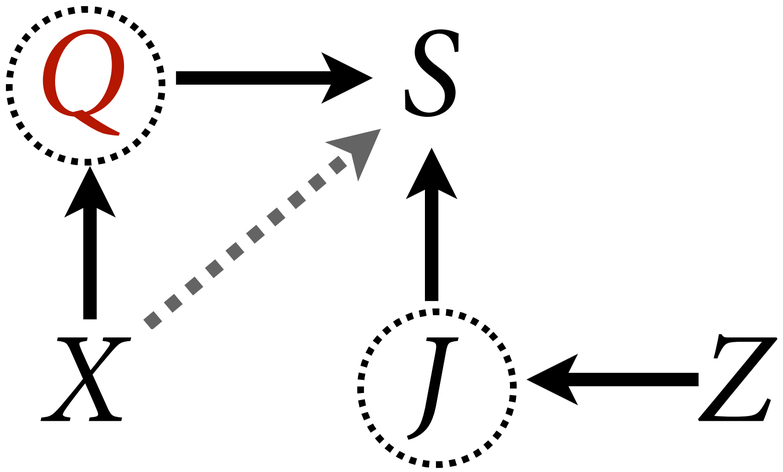
$$\mu_i = Q_{W[i]}$$

$$Q_j \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Exponential}(1)$$

```
library(rethinking)
data(Wines2
d <- Wines2

dat <- list
    S=stanc
    J=as.nu
    W=as.nu
    X=ifels
    Z=ifels
)

mQ <- ulam(
    alist(
        S ~
        mu
        Q[W
        sig
    ) , dat

precis(mQ,2
```

	mean	sd	5.5%	94.5%	n_eff	Rhat4
Q[1]	0.14	0.30	-0.34	0.64	2962	1
Q[2]	0.11	0.32	-0.40	0.61	3033	1
Q[3]	0.27	0.31	-0.21	0.75	2608	1
Q[4]	0.55	0.33	0.04	1.09	2598	1
Q[5]	-0.13	0.32	-0.63	0.37	2726	1
Q[6]	-0.37	0.32	-0.88	0.12	2710	1
Q[7]	0.29	0.32	-0.22	0.81	2679	1
Q[8]	0.27	0.32	-0.24	0.79	3248	1
Q[9]	0.08	0.30	-0.40	0.56	3410	1
Q[10]	0.12	0.32	-0.40	0.63	2697	1
Q[11]	-0.02	0.31	-0.52	0.50	2740	1
Q[12]	-0.03	0.32	-0.53	0.48	2809	1
Q[13]	-0.11	0.32	-0.62	0.39	3225	1
Q[14]	0.01	0.33	-0.51	0.52	2690	1
Q[15]	-0.22	0.31	-0.71	0.28	2754	1
Q[16]	-0.21	0.32	-0.72	0.30	2120	1
Q[17]	-0.14	0.31	-0.64	0.37	3831	1
Q[18]	-0.86	0.32	-1.37	-0.35	3093	1
Q[19]	-0.16	0.30	-0.65	0.31	2783	1
Q[20]	0.38	0.32	-0.12	0.88	2873	1
sigma	1.00	0.06	0.91	1.09	2759	1



$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = Q_{W[i]}$$

$$Q_j \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Exponential}(1)$$

Drawing the Markov Owl

Complex machinery, but lots of diagnostics

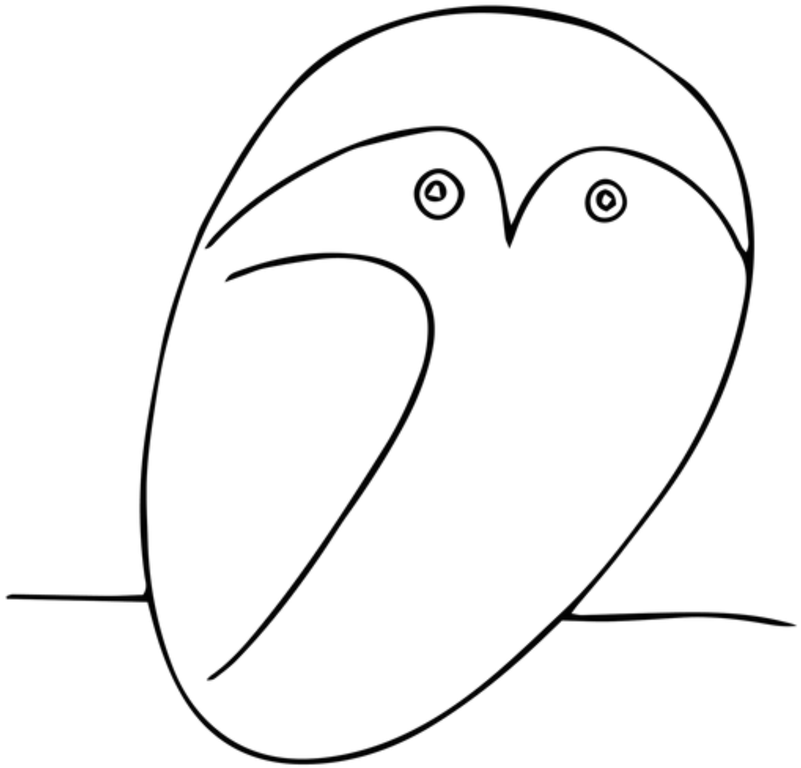
(1) Trace plots

(2) Trace rank plots

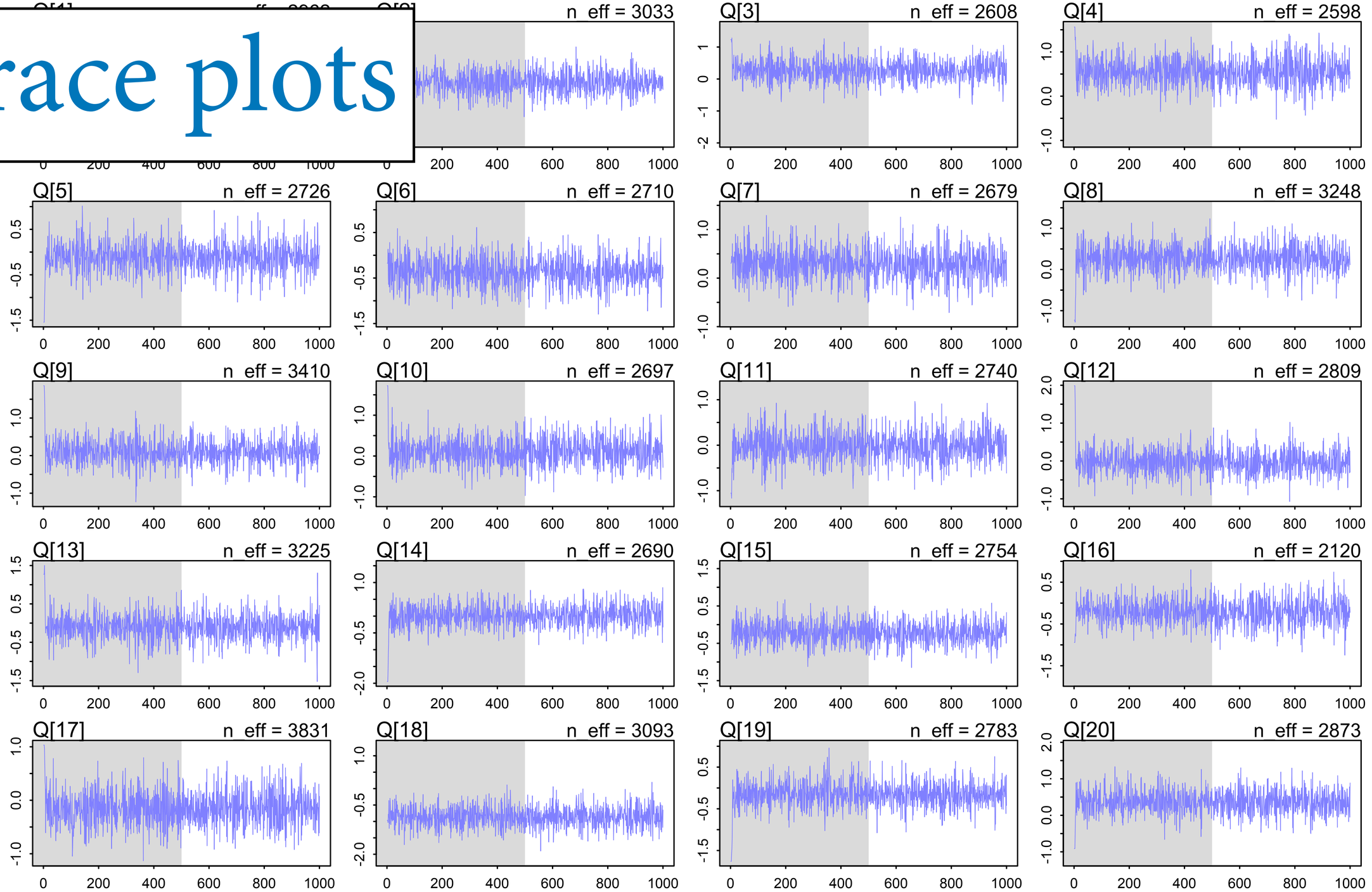
(3) R-hat convergence measure

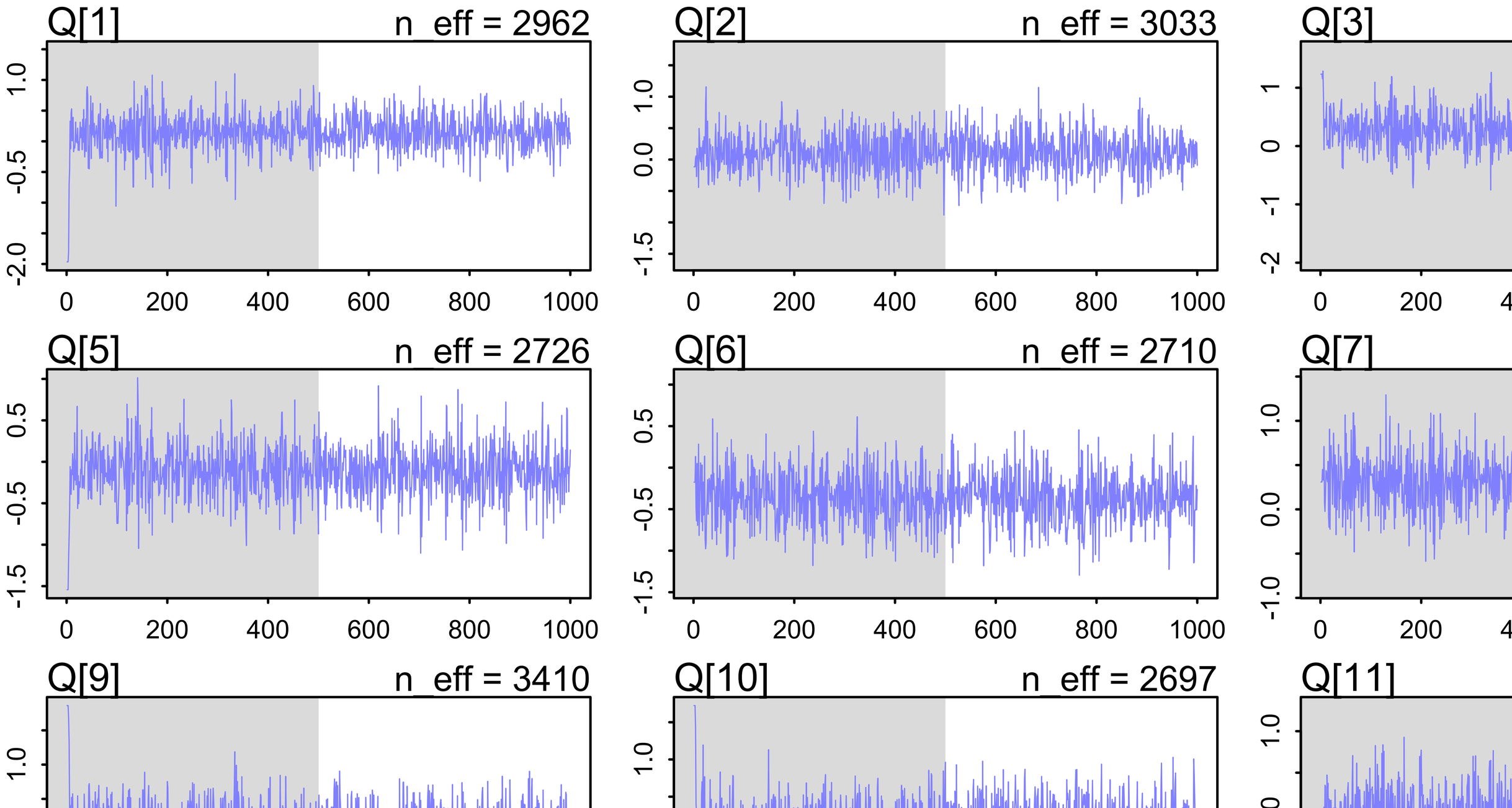
(4) Number of effective samples

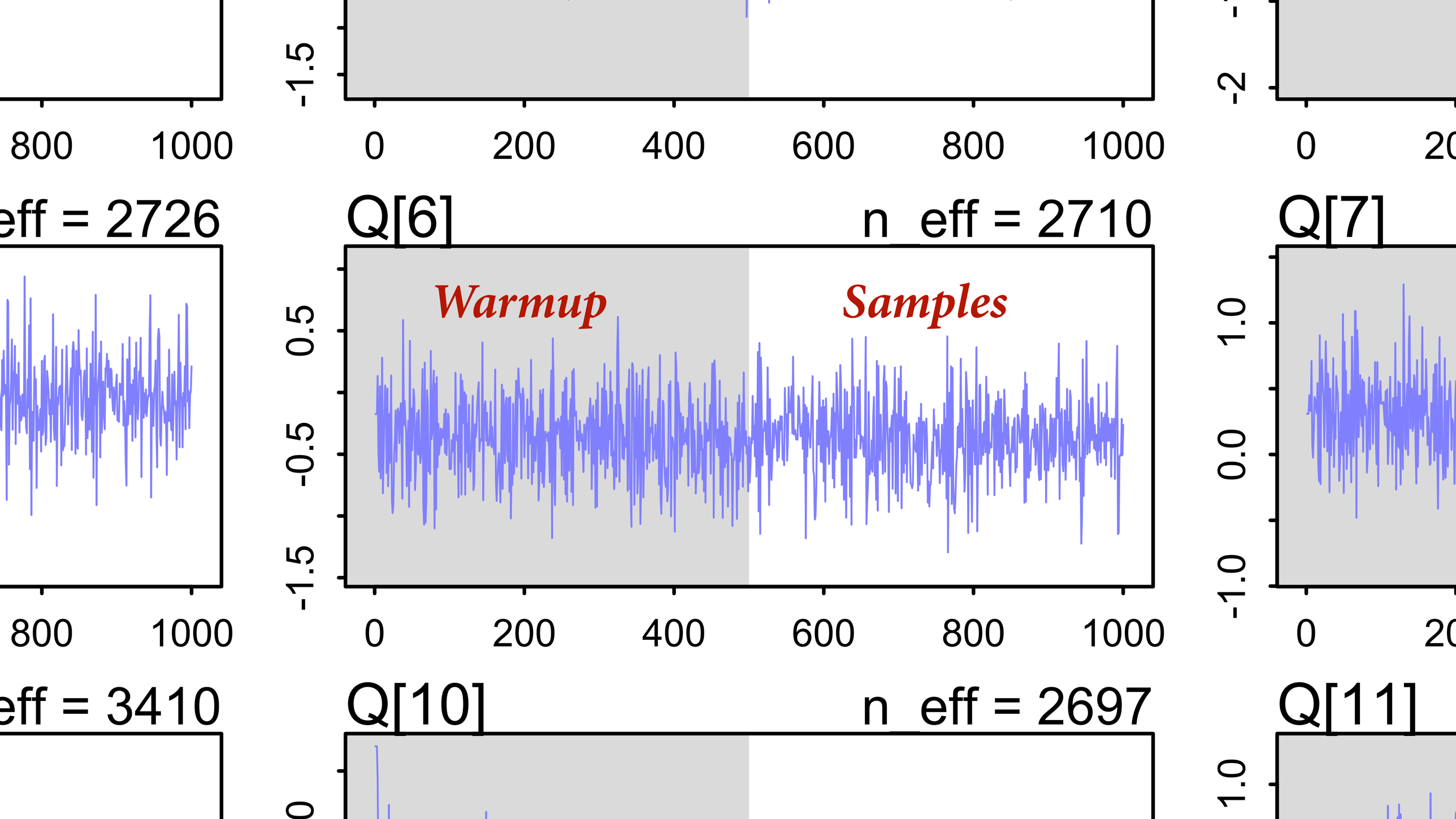
(5) Divergent transitions



Trace plots


Q[2] n_eff = 3033
Q[3] n_eff = 2608
Q[4] n_eff = 2598
Q[5] n_eff = 2726
Q[6] n_eff = 2710
Q[7] n_eff = 2679
Q[8] n_eff = 3248
Q[9] n_eff = 3410
Q[10] n_eff = 2697
Q[11] n_eff = 2740
Q[12] n_eff = 2809
Q[13] n_eff = 3225
Q[14] n_eff = 2690
Q[15] n_eff = 2754
Q[16] n_eff = 2120
Q[17] n_eff = 3831
Q[18] n_eff = 3093
Q[19] n_eff = 2783
Q[20] n_eff = 2873

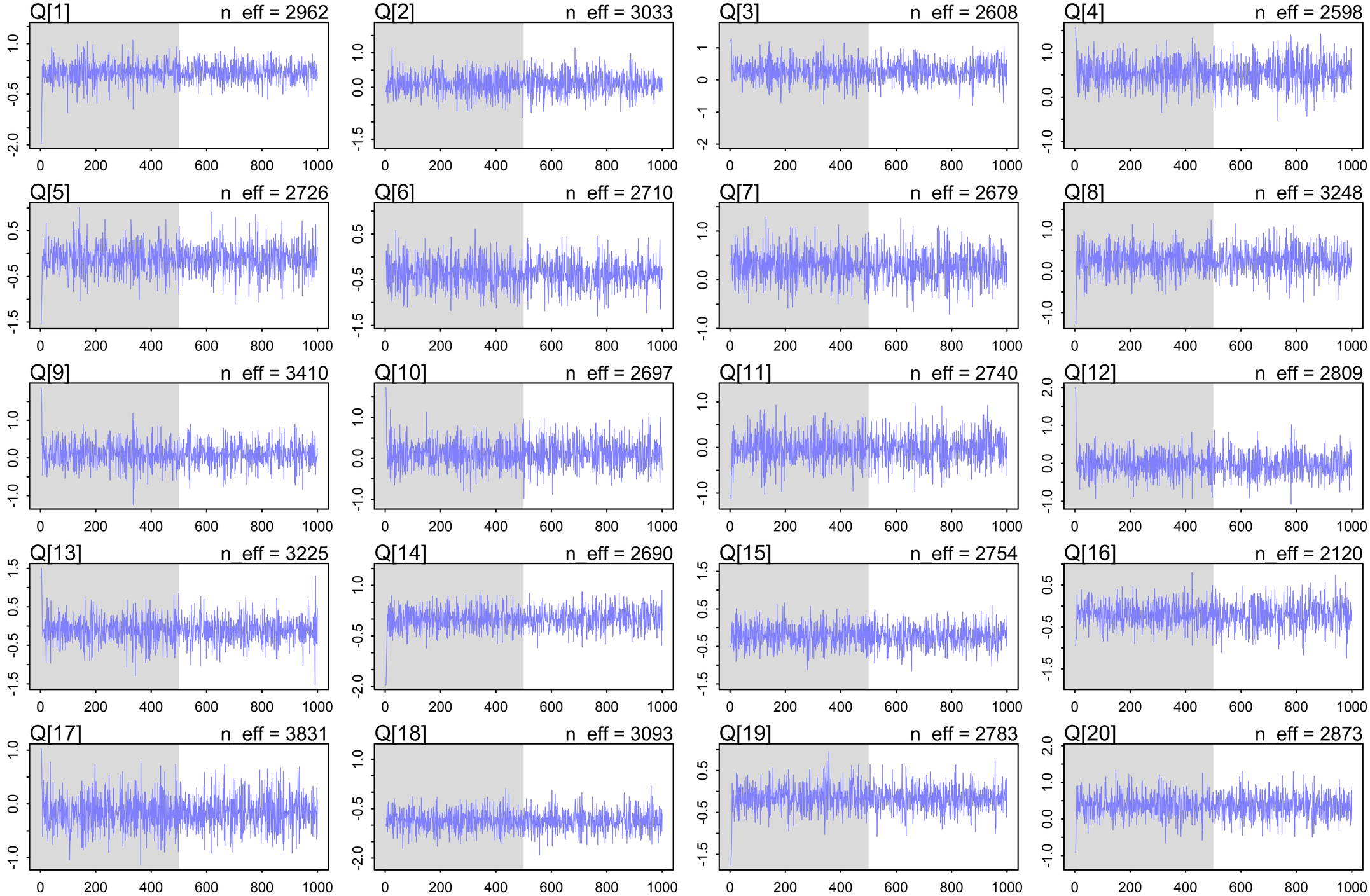

Q[1]
n_eff = 2962
Q[2]
n_eff = 3033
Q[3]
Q[5]
n_eff = 2726
Q[6]
n_eff = 2710
Q[7]
Q[9]
n_eff = 3410
Q[10]
n_eff = 2697
Q[11]

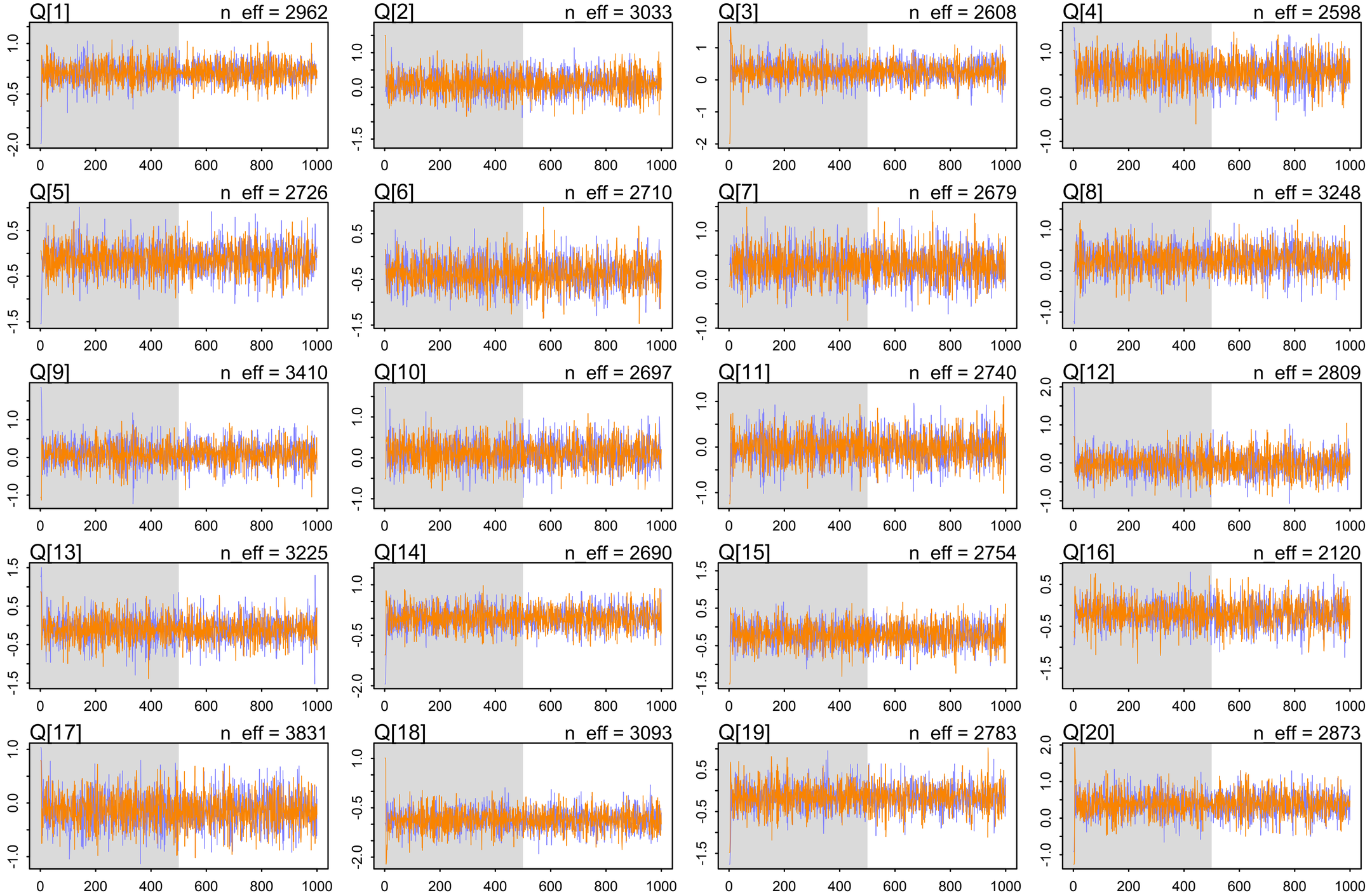

eff = 2726
Q[6]
n_eff = 2710
Q[7]
Warmup
Samples
-1.5
-0.5
0.5
-1.0
0.0
1.0
0
200
400
600
800
1000
eff = 3410
Q[10]
n_eff = 2697
Q[11]

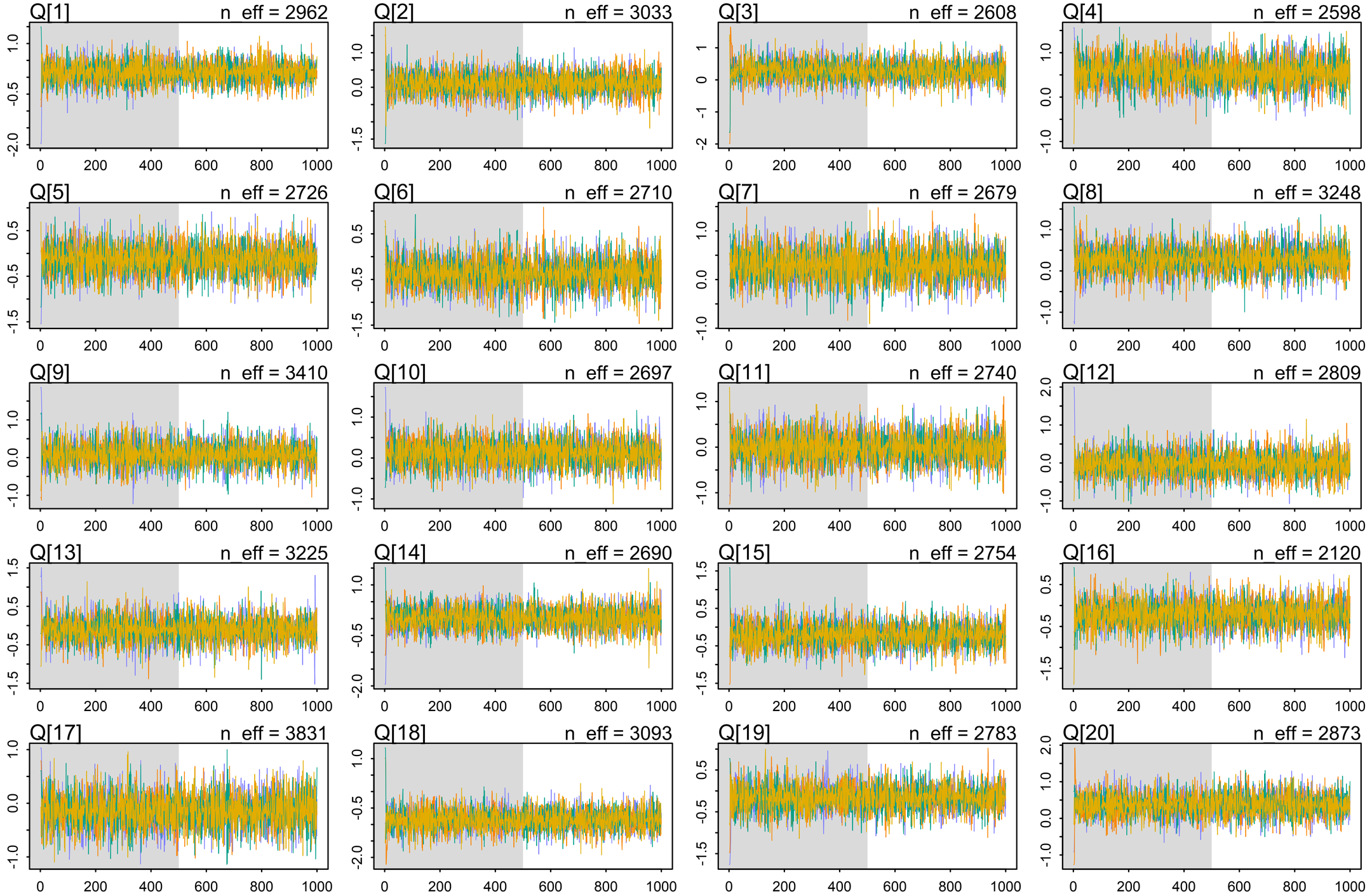
```
library(rethinking)
data(Wines2012)
d <- Wines2012

dat <- list(
    S=standardize(d$score),
    J=as.numeric(d$judge),
    W=as.numeric(d$wine),
    X=ifelse(d$wine.amer==1,1,2),
    Z=ifelse(d$judge.amer==1,1,2)
)

mQ <- ulam(
    alist(
        S ~ dnorm(mu,sigma),
        mu <- Q[W],
        Q[W] ~ dnorm(0,1),
        sigma ~ dexp(1)
    ) , data=dat , chains=4 , cores=4 )

precis(mQ,2)
```

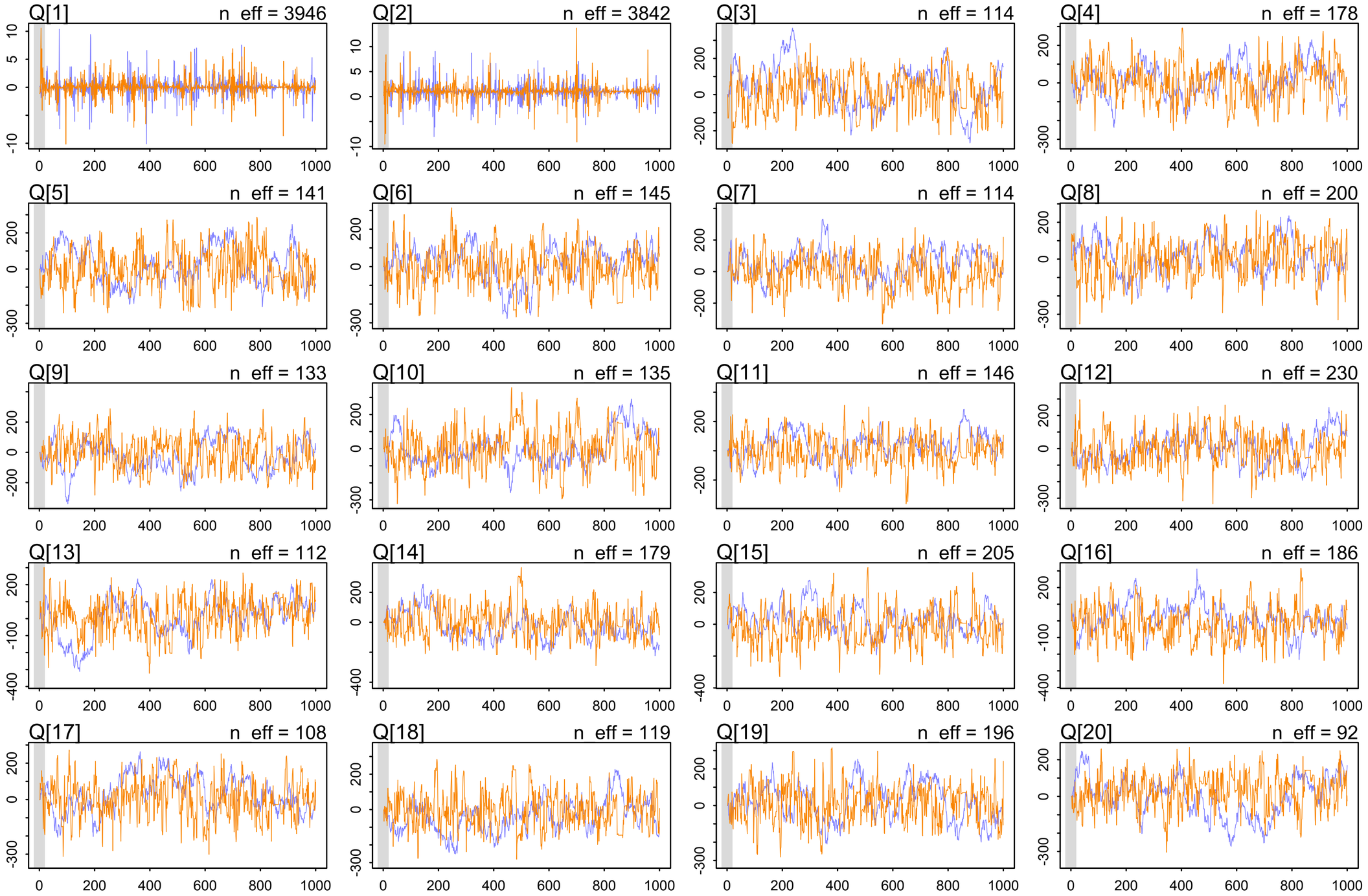
Need more than 1 chain to check convergence

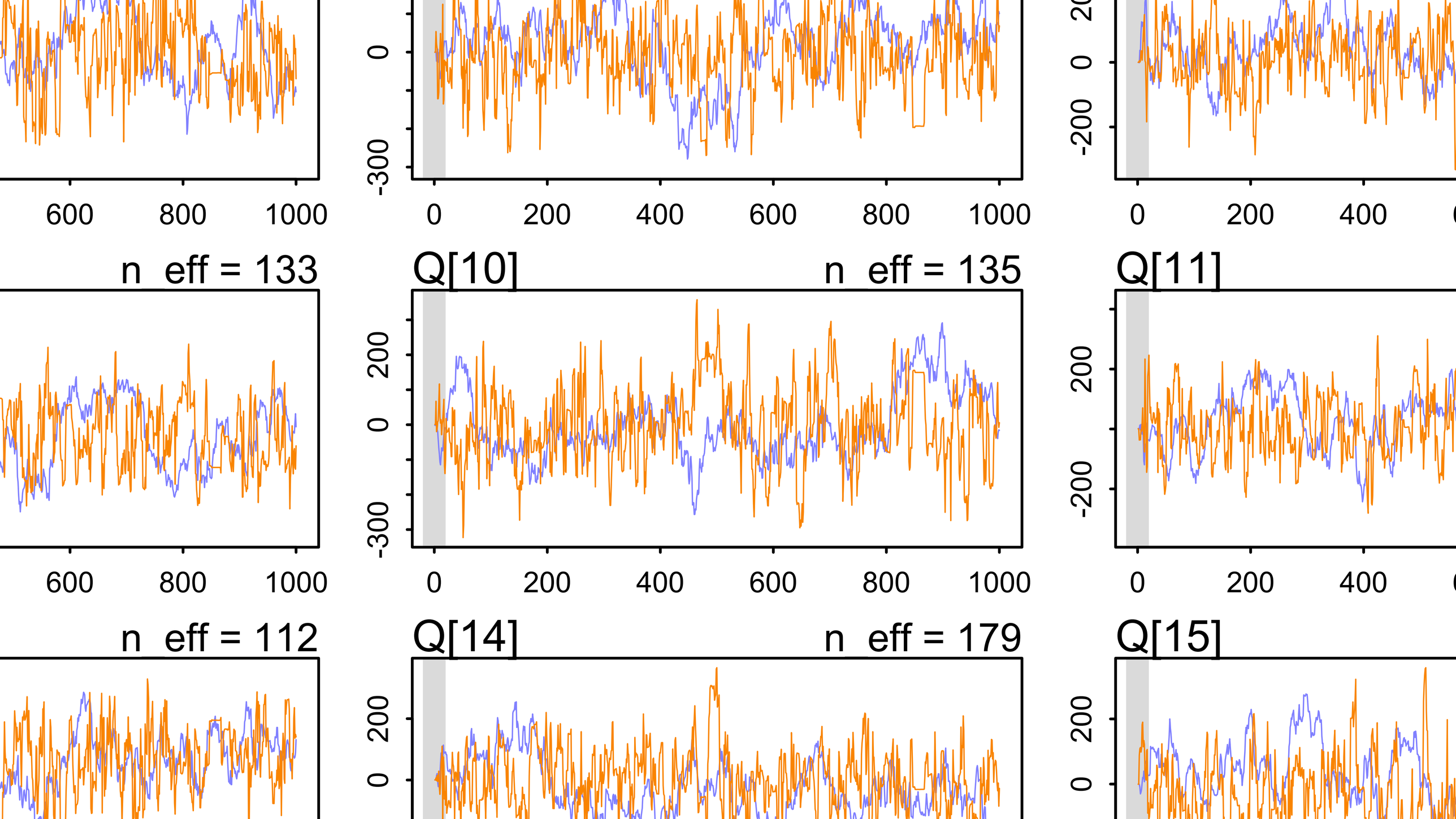
Convergence: Each chain explores the right distribution and every chain explores the same distribution


Q[1] n_eff = 2962
Q[2] n_eff = 3033
Q[3] n_eff = 2608
Q[4] n_eff = 2598
Q[5] n_eff = 2726
Q[6] n_eff = 2710
Q[7] n_eff = 2679
Q[8] n_eff = 3248
Q[9] n_eff = 3410
Q[10] n_eff = 2697
Q[11] n_eff = 2740
Q[12] n_eff = 2809
Q[13] n_eff = 3225
Q[14] n_eff = 2690
Q[15] n_eff = 2754
Q[16] n_eff = 2120
Q[17] n_eff = 3831
Q[18] n_eff = 3093
Q[19] n_eff = 2783
Q[20] n_eff = 2873

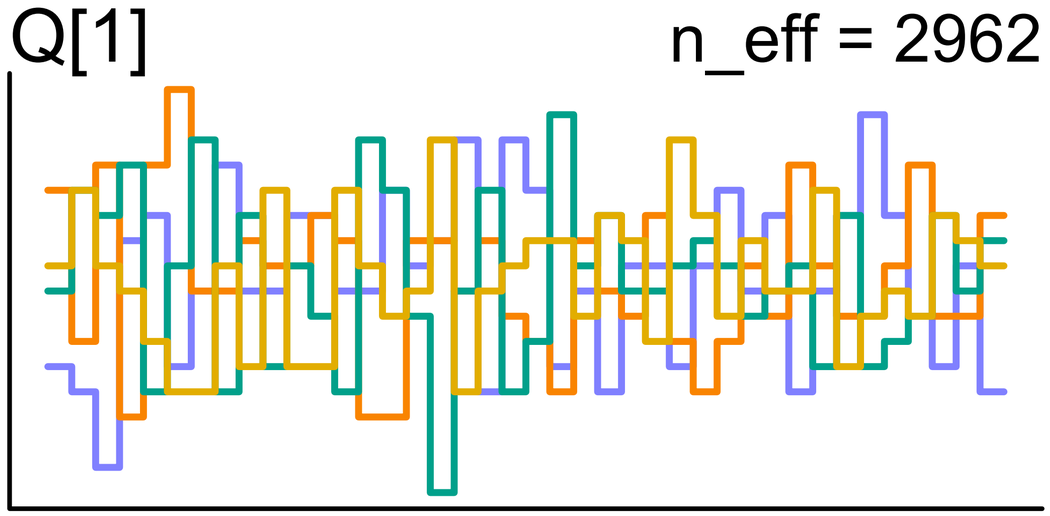

Q[1] n_eff = 2962
Q[2] n_eff = 3033
Q[3] n_eff = 2608
Q[4] n_eff = 2598
Q[5] n_eff = 2726
Q[6] n_eff = 2710
Q[7] n_eff = 2679
Q[8] n_eff = 3248
Q[9] n_eff = 3410
Q[10] n_eff = 2697
Q[11] n_eff = 2740
Q[12] n_eff = 2809
Q[13] n_eff = 3225
Q[14] n_eff = 2690
Q[15] n_eff = 2754
Q[16] n_eff = 2120
Q[17] n_eff = 3831
Q[18] n_eff = 3093
Q[19] n_eff = 2783
Q[20] n_eff = 2873


Q[1] n_eff = 2962
Q[2] n_eff = 3033
Q[3] n_eff = 2608
Q[4] n_eff = 2598
Q[5] n_eff = 2726
Q[6] n_eff = 2710
Q[7] n_eff = 2679
Q[8] n_eff = 3248
Q[9] n_eff = 3410
Q[10] n_eff = 2697
Q[11] n_eff = 2740
Q[12] n_eff = 2809
Q[13] n_eff = 3225
Q[14] n_eff = 2690
Q[15] n_eff = 2754
Q[16] n_eff = 2120
Q[17] n_eff = 3831
Q[18] n_eff = 3093
Q[19] n_eff = 2783
Q[20] n_eff = 2873

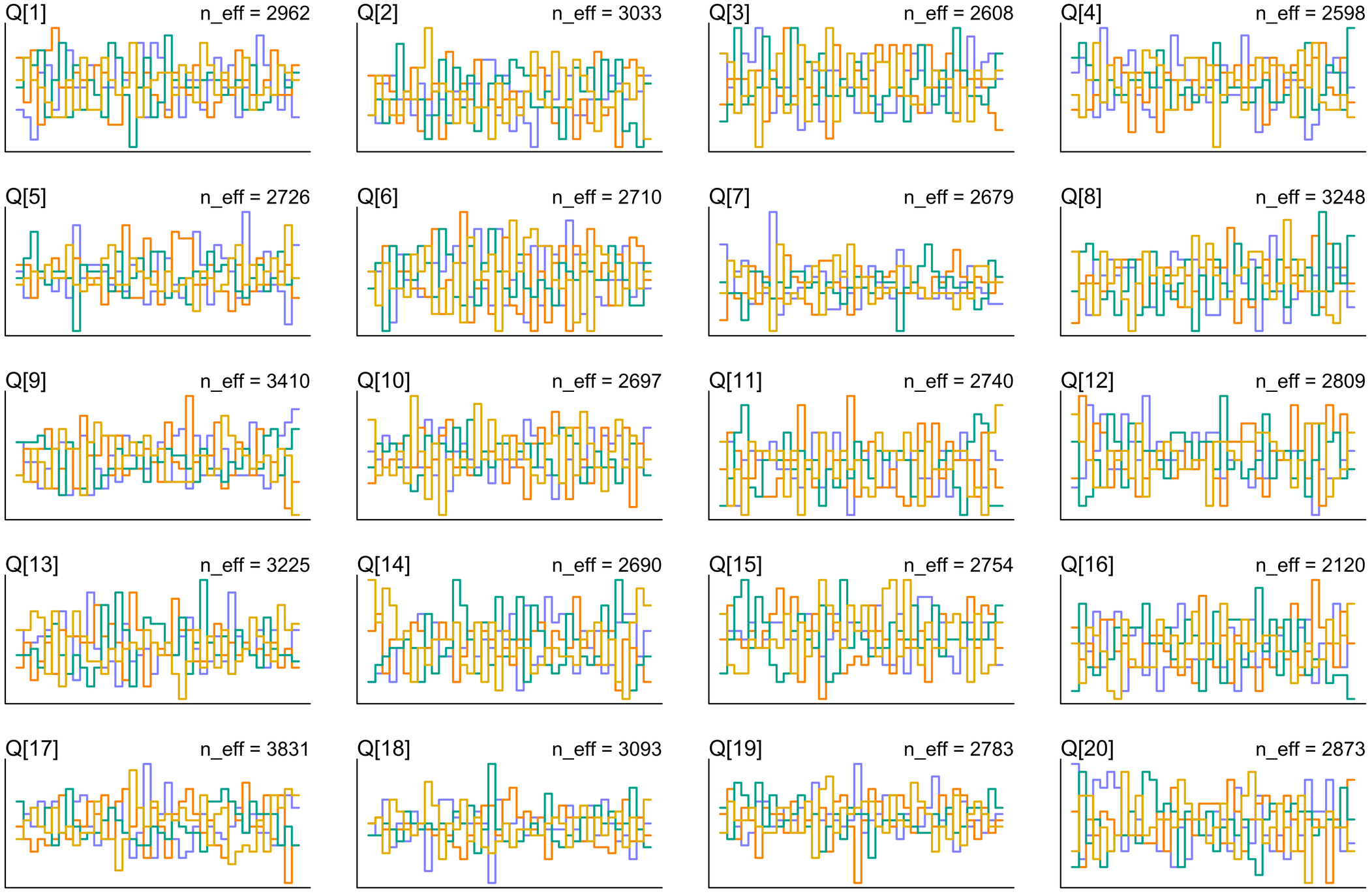

Q[1] n eff = 3946
Q[2] n eff = 3842
Q[3] n eff = 114
Q[4] n eff = 178
Q[5] n eff = 141
Q[6] n eff = 145
Q[7] n eff = 114
Q[8] n eff = 200
Q[9] n eff = 133
Q[10] n eff = 135
Q[11] n eff = 146
Q[12] n eff = 230
Q[13] n eff = 112
Q[14] n eff = 179
Q[15] n eff = 205
Q[16] n eff = 186
Q[17] n eff = 108
Q[18] n eff = 119
Q[19] n eff = 196
Q[20] n eff = 92

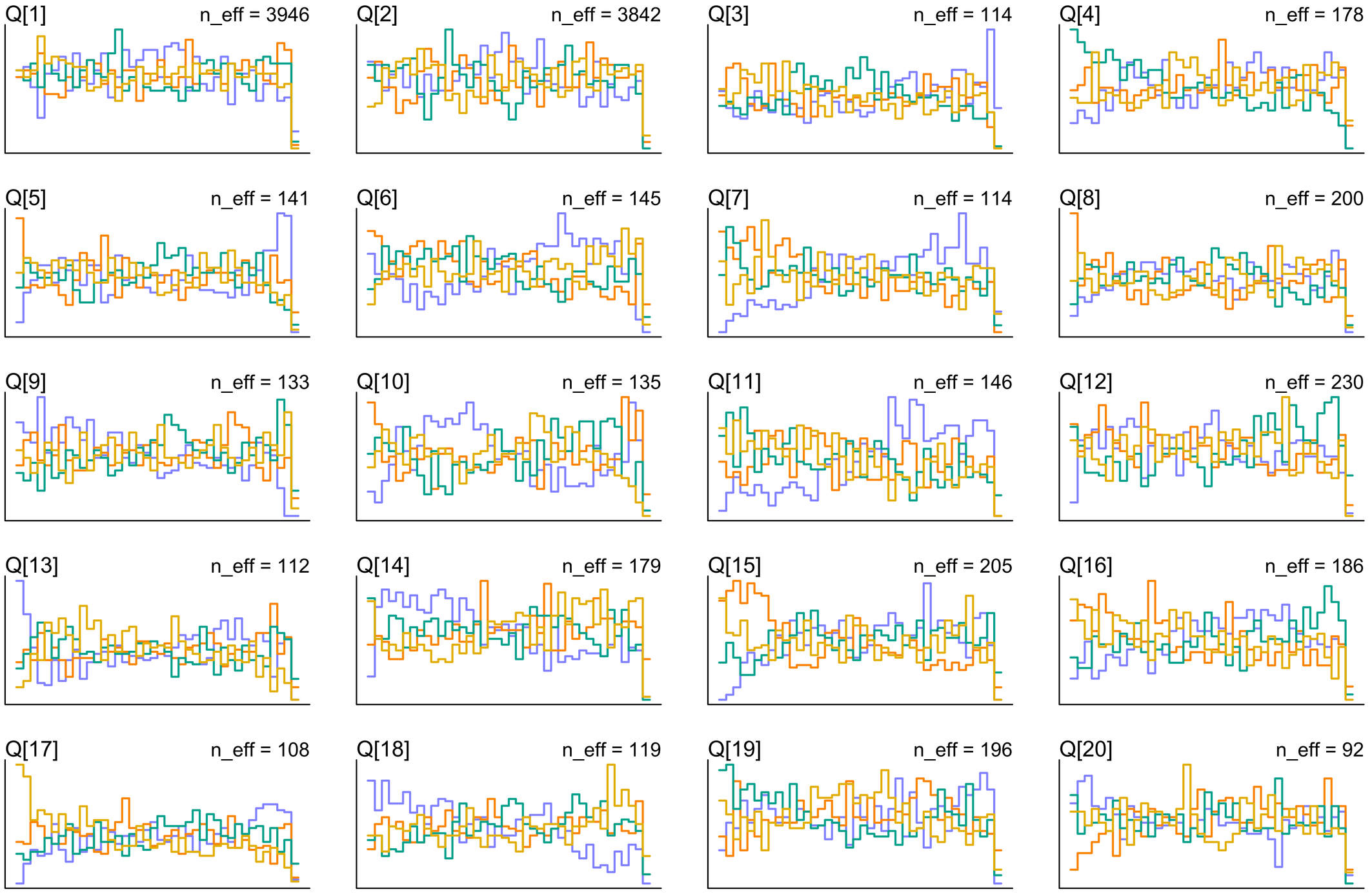

n eff = 133
Q[10]
n eff = 135
Q[11]
n eff = 112
Q[14]
n eff = 179
Q[15]
600
800
1000
0
200
400
600
800
1000
-300
0
200
-200
0
200

Trace rank (Trank) plots

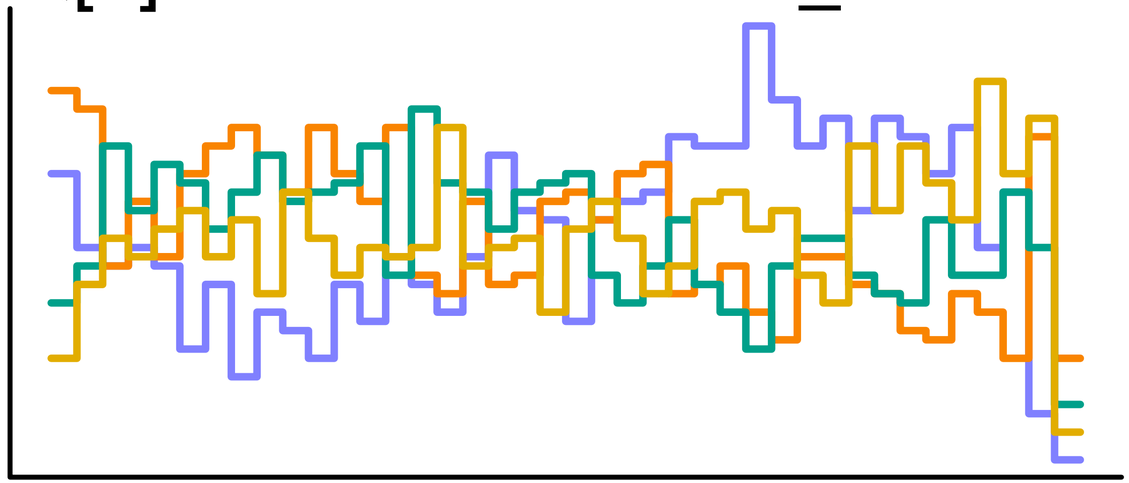


Rank orders of samples. No chain should tend to be below/ above others.

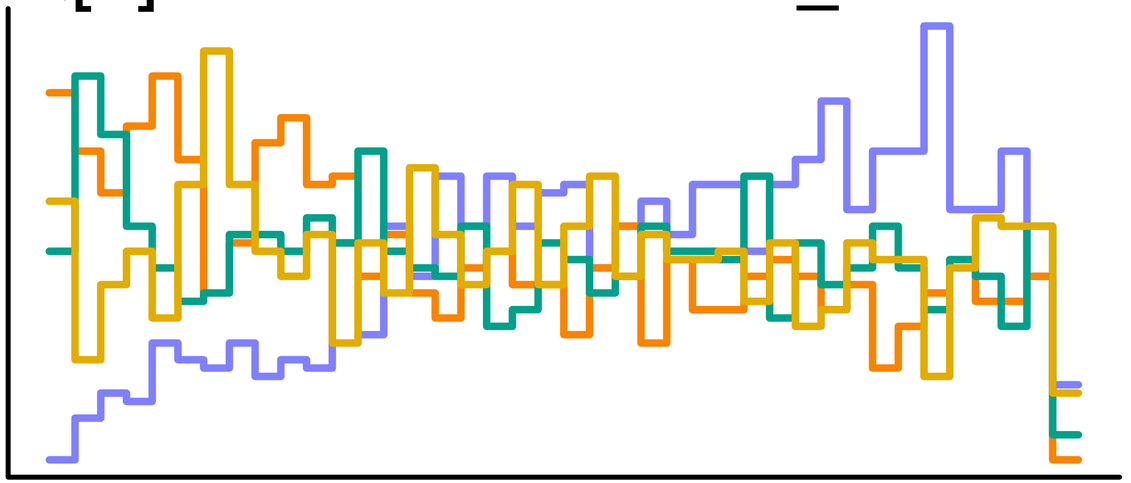

Q[1]
n_eff = 2962
Q[2]
n_eff = 3033
Q[3]
n_eff = 2608
Q[4]
n_eff = 2598
Q[5]
n_eff = 2726
Q[6]
n_eff = 2710
Q[7]
n_eff = 2679
Q[8]
n_eff = 3248
Q[9]
n_eff = 3410
Q[10]
n_eff = 2697
Q[11]
n_eff = 2740
Q[12]
n_eff = 2809
Q[13]
n_eff = 3225
Q[14]
n_eff = 2690
Q[15]
n_eff = 2754
Q[16]
n_eff = 2120
Q[17]
n_eff = 3831
Q[18]
n_eff = 3093
Q[19]
n_eff = 2783
Q[20]
n_eff = 2873


Q[1]
n_eff = 3946
Q[2]
n_eff = 3842
Q[3]
n_eff = 114
Q[4]
n_eff = 178
Q[5]
n_eff = 141
Q[6]
n_eff = 145
Q[7]
n_eff = 114
Q[8]
n_eff = 200
Q[9]
n_eff = 133
Q[10]
n_eff = 135
Q[11]
n_eff = 146
Q[12]
n_eff = 230
Q[13]
n_eff = 112
Q[14]
n_eff = 179
Q[15]
n_eff = 205
Q[16]
n_eff = 186
Q[17]
n_eff = 108
Q[18]
n_eff = 119
Q[19]
n_eff = 196
Q[20]
n_eff = 92

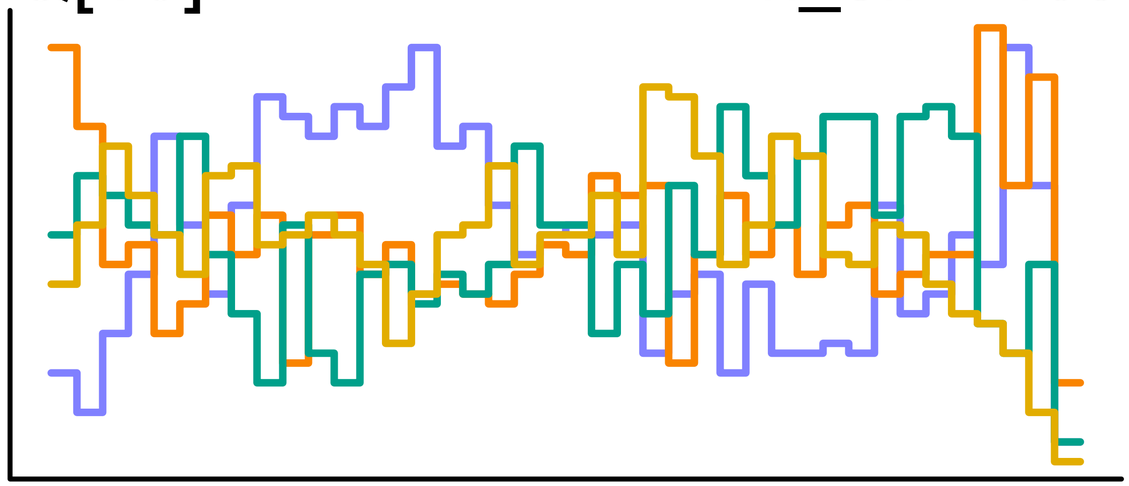
Q[6] n_eff = 145



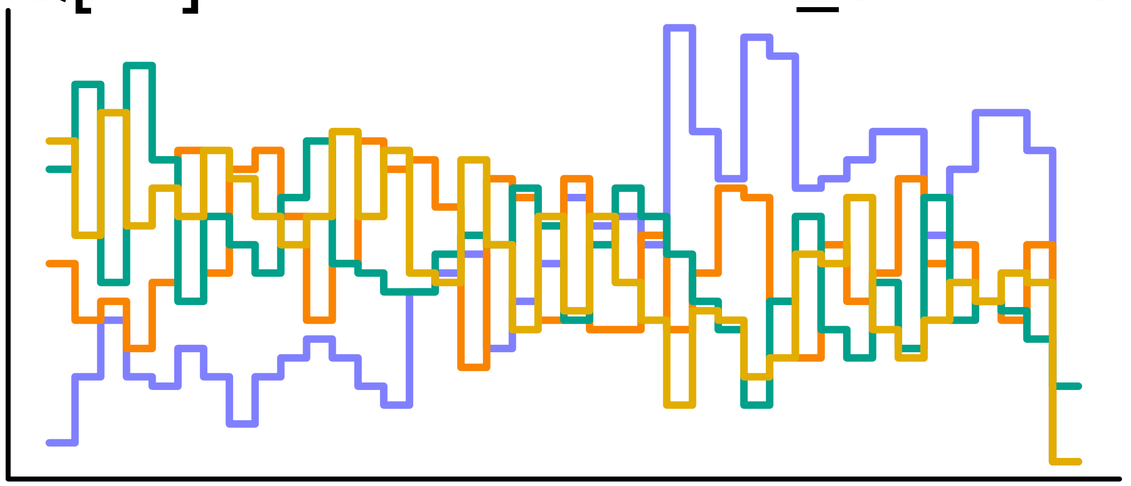
Q[7] n_eff = 114



Q[10] n_eff = 135



Q[11] n_eff = 146



R-hat

```
       mean   sd  5.5% 94.5% n_eff Rhat4
Q[1]   0.14 0.30 -0.34  0.64  2962     1
Q[2]   0.11 0.32 -0.40  0.61  3033     1
Q[3]   0.27 0.31 -0.21  0.75  2608     1
```

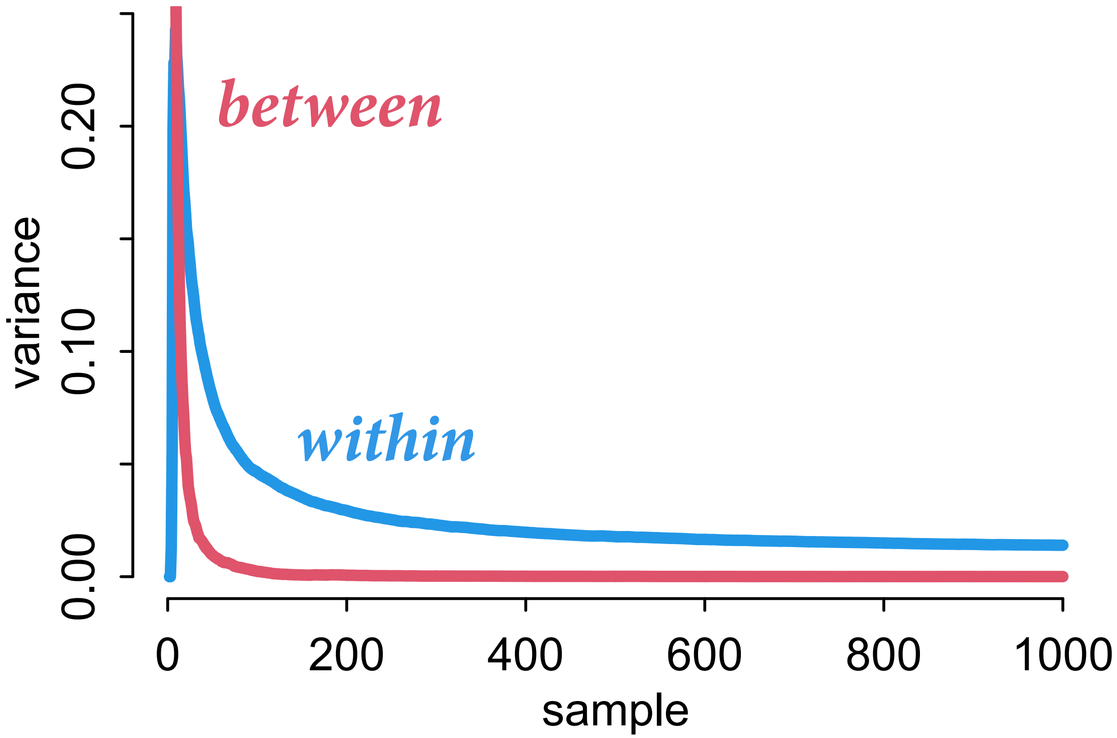
When chains converge:

(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



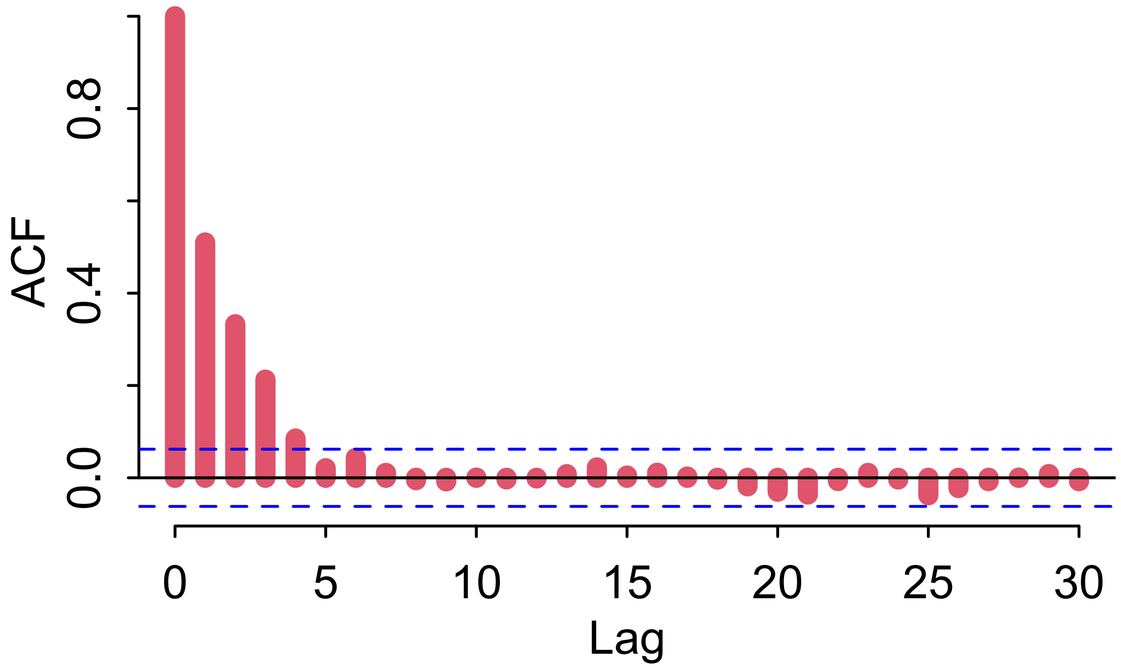
n_eff

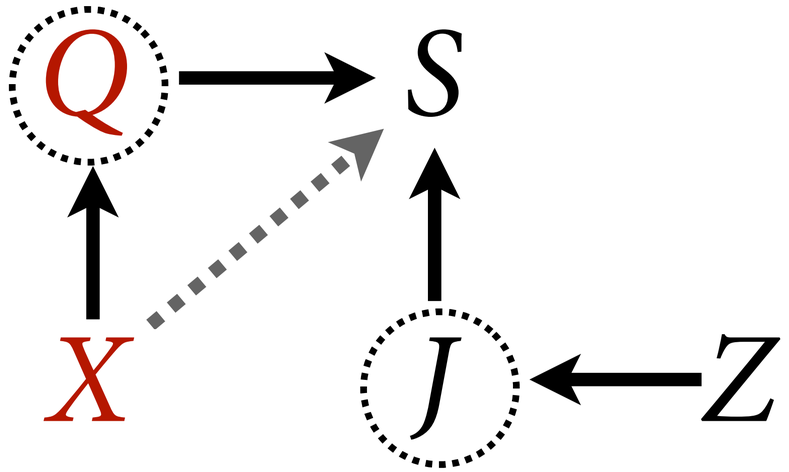
Estimate of number of effective samples

"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

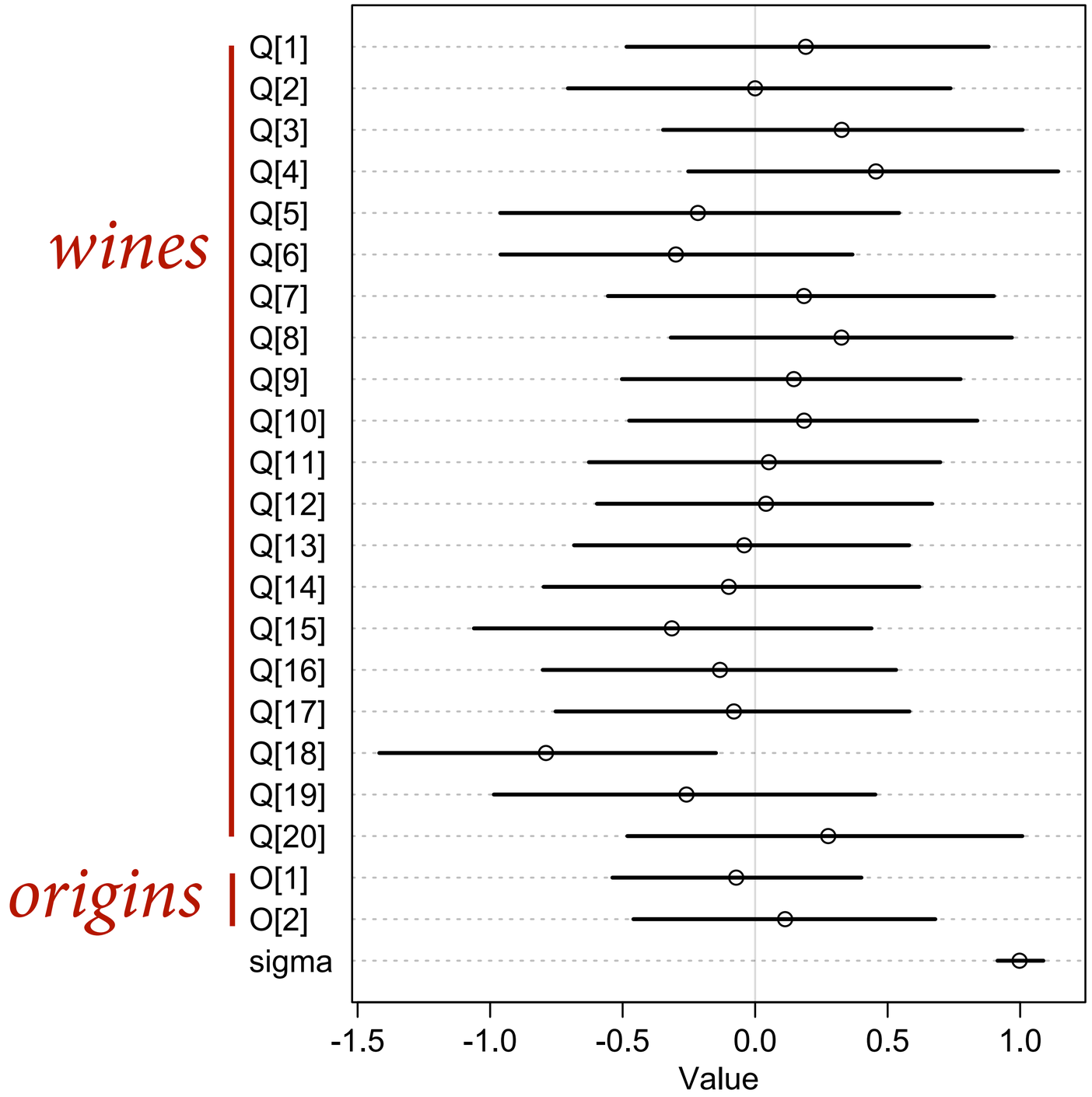
```
       mean   sd  5.5% 94.5% n_eff Rhat4
Q[1]   0.14 0.30 -0.34  0.64  2962     1
Q[2]   0.11 0.32 -0.40  0.61  3033     1
Q[3]   0.27 0.31 -0.21  0.75  2608     1
```

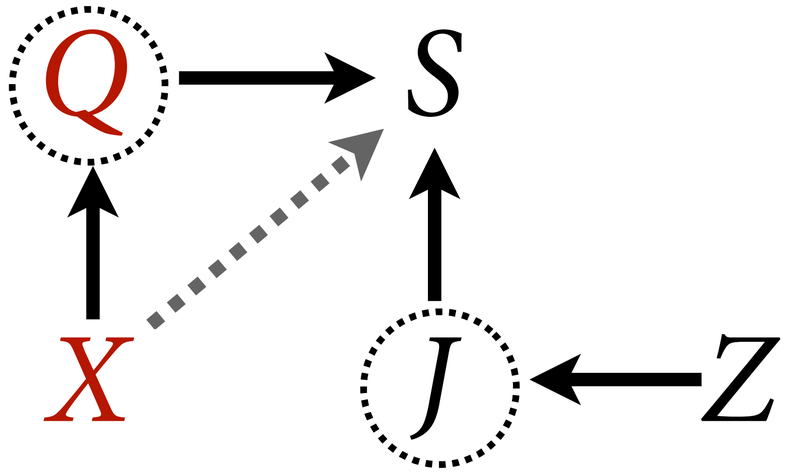





```
mQO <- ulam(
    alist(
        S ~ dnorm(mu,sigma),
        mu <- Q[W] + O[X],
        Q[W] ~ dnorm(0,1),
        O[X] ~ dnorm(0,1),
        sigma ~ dexp(1)
    ) , data=dat , chains=4 , cores=4 )

plot(precis(mQO,2))
```

$$
\begin{aligned}
S_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= Q_{W[i]} + O_{X[i]} \\
Q_j &\sim \text{Normal}(0, 1) \\
O_j &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$





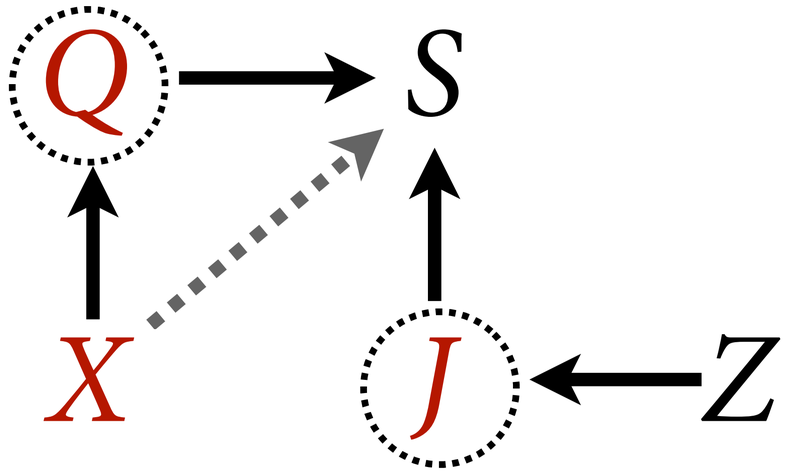
$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = Q_{W[i]} + O_{X[i]}$$

$$Q_j \sim \text{Normal}(0, 1)$$

$$O_j \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Exponential}(1)$$




```
mQOJ <- ulam(
    alist(
        S ~ dnorm(mu,sigma),
        mu <- (Q[W] + O[X] - H[J])*D[J],
        Q[W] ~ dnorm(0,1),
        O[X] ~ dnorm(0,1),
        H[J] ~ dnorm(0,1),
        D[J] ~ dexp(1),
        sigma ~ dexp(1)
    ) , data=dat , chains=4 )

plot(precis(mQOJ,2))
```

$$
\begin{aligned}
S_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]} \\
Q_j &\sim \text{Normal}(0, 1) \\
O_j &\sim \text{Normal}(0, 1) \\
H_j &\sim \text{Normal}(0, 1) \\
D_j &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]}$$

expected score

$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]}$$

expected score

wine quality

$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]}$$

expected score

wine quality

origin

$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]}$$

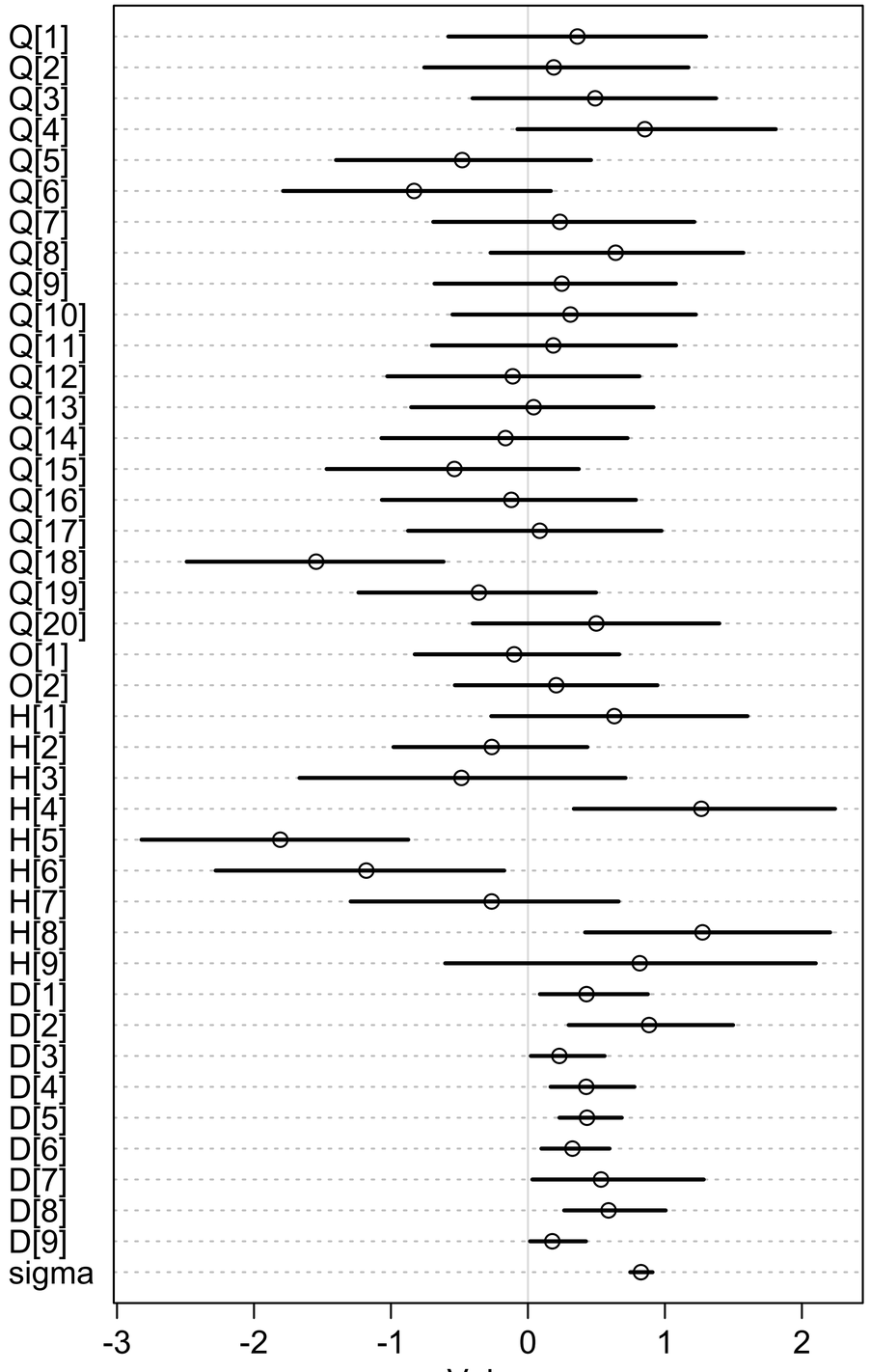
expected score

wine quality

origin

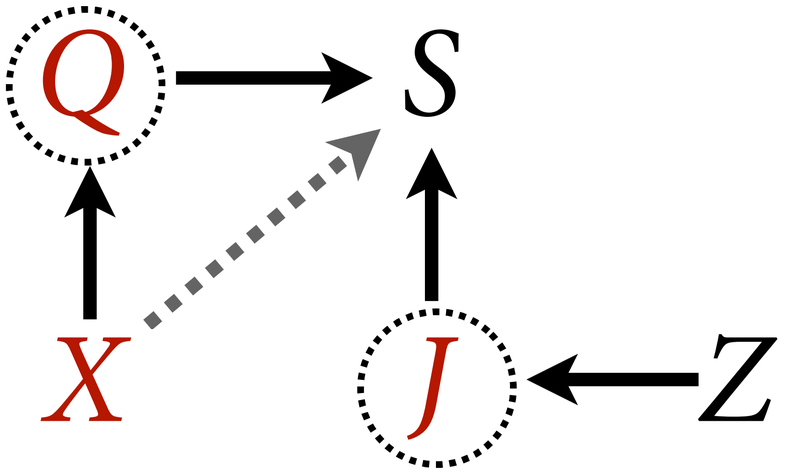
judge harshness

judge discrimination



harshness

discrimination



$$S_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]}$$

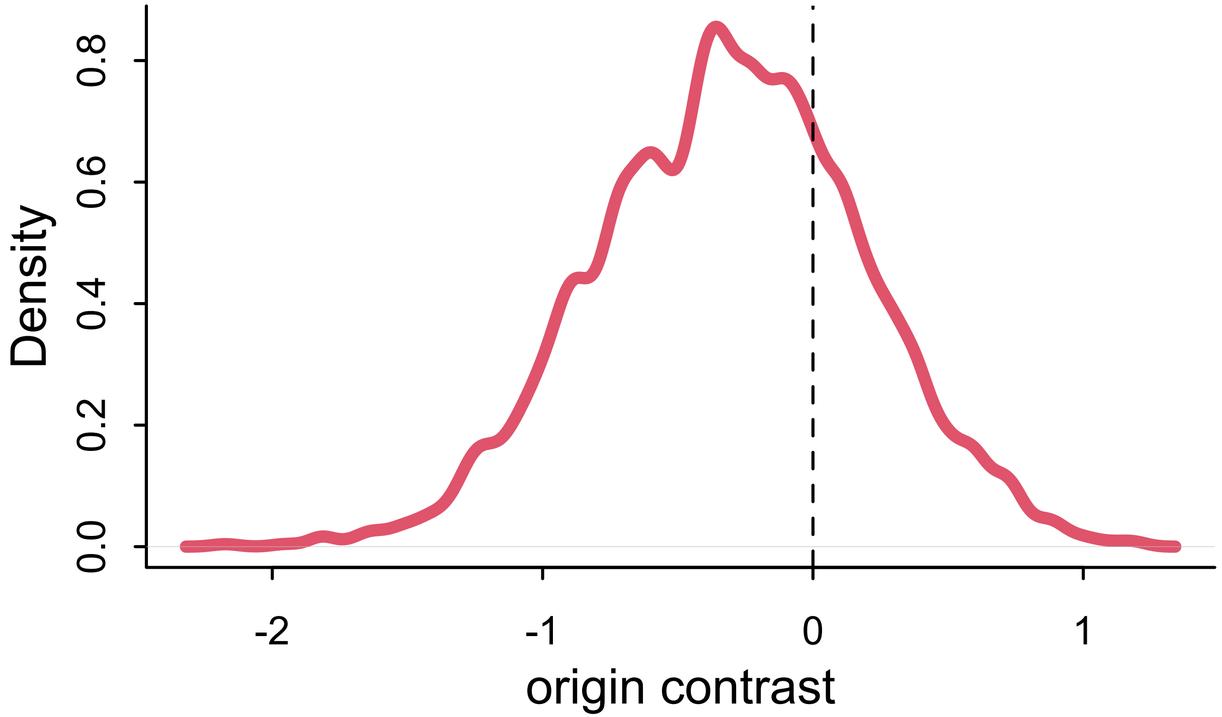
$$Q_j \sim \text{Normal}(0, 1)$$

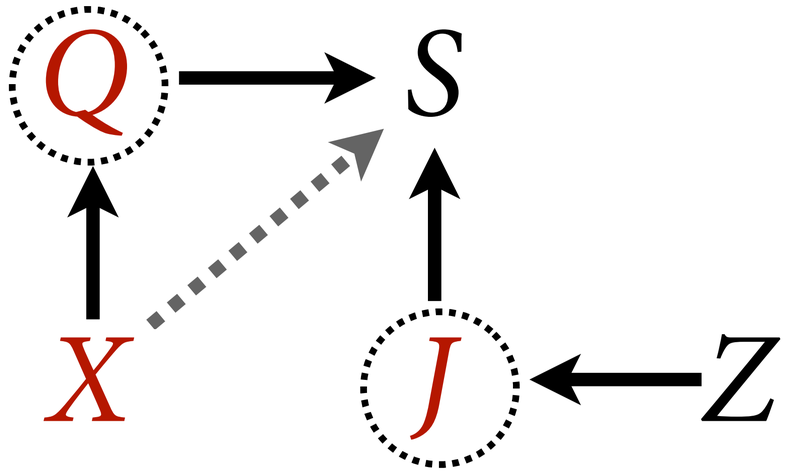
$$O_j \sim \text{Normal}(0, 1)$$

$$H_j \sim \text{Normal}(0, 1)$$

$$D_j \sim \text{Normal}(0, 1)$$

$$\sigma \sim \text{Exponential}(1)$$

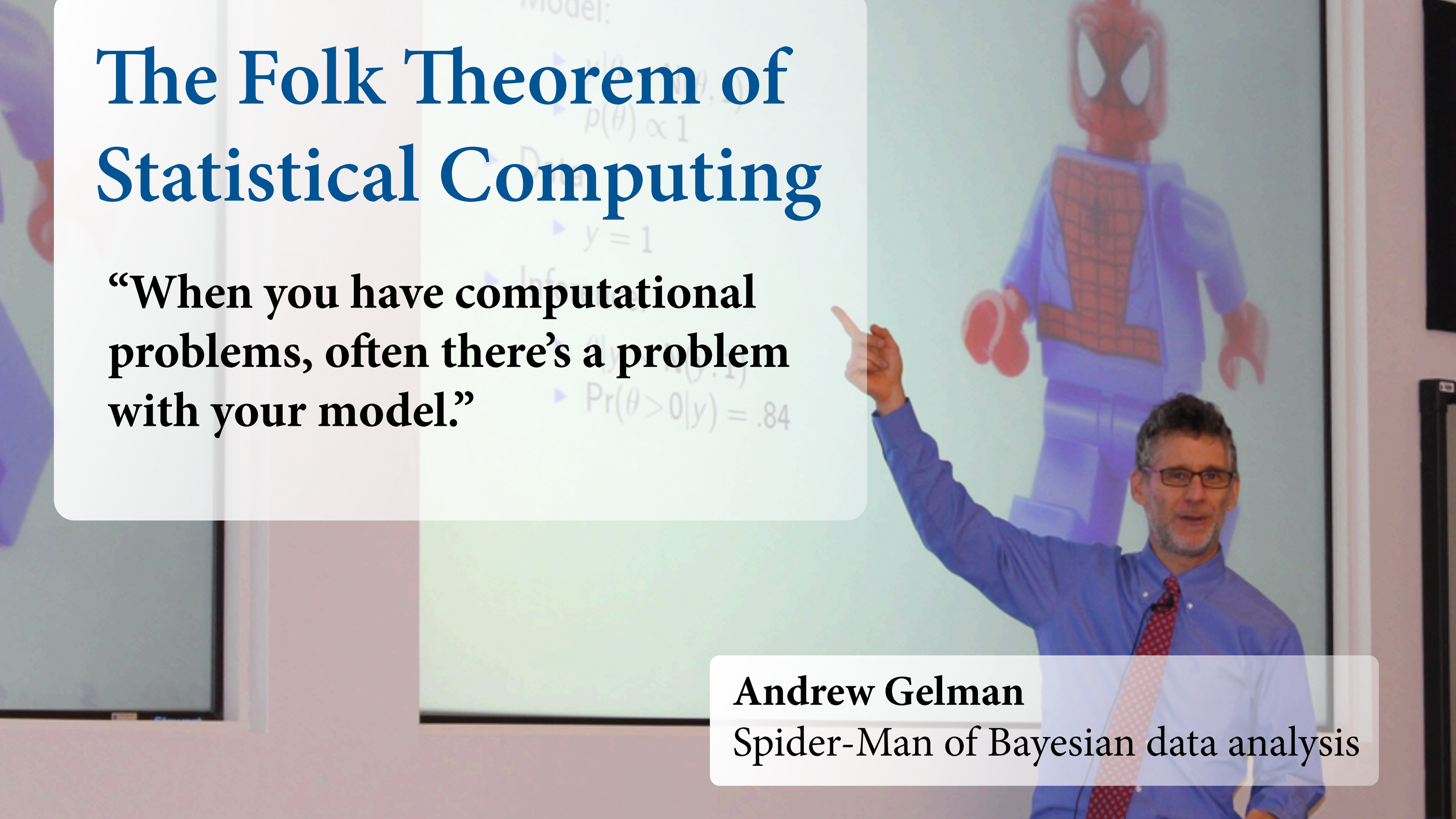




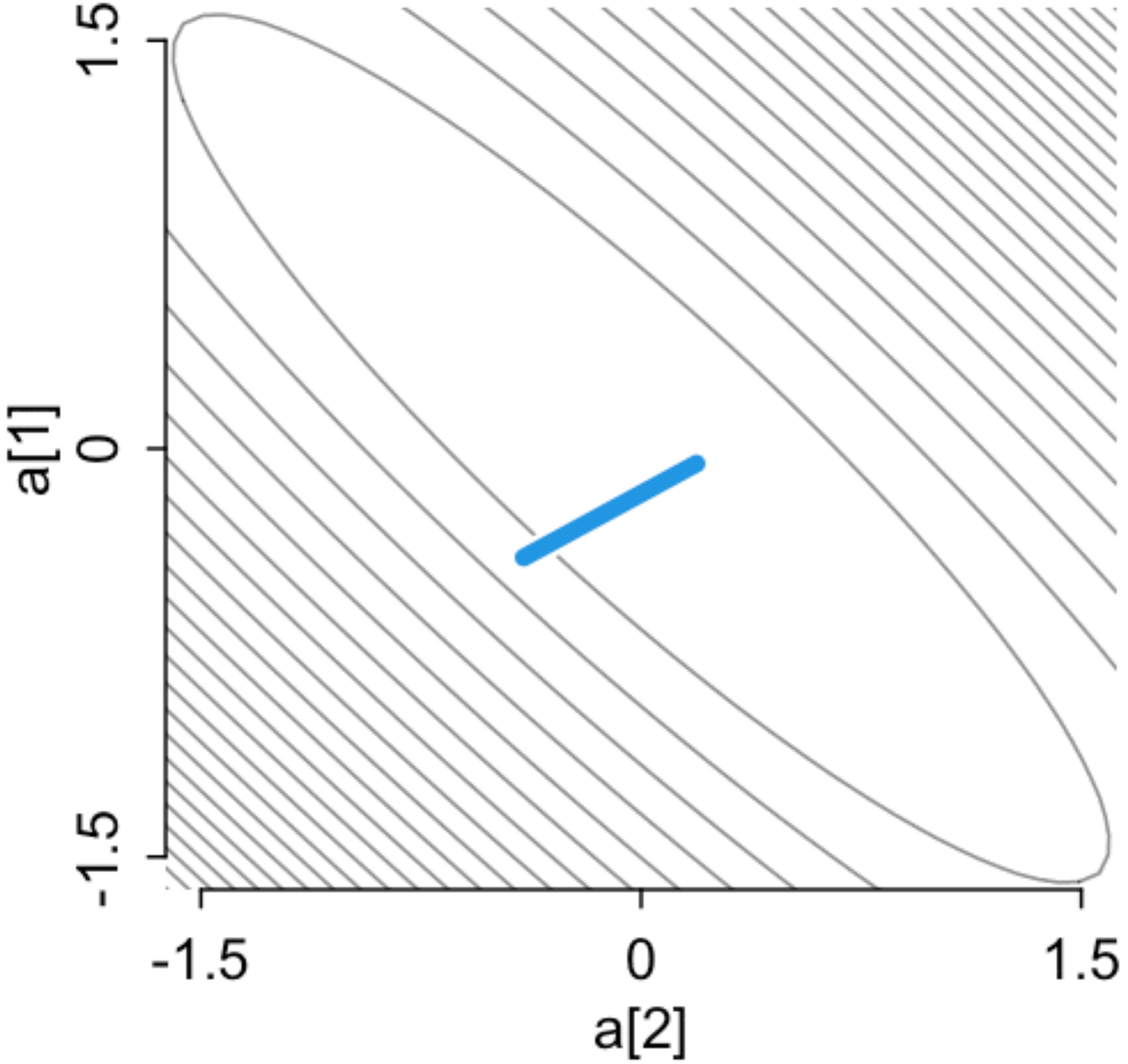
$$
\begin{aligned}
S_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= (Q_{W[i]} + O_{X[i]} - H_{J[i]})D_{J[i]} \\
Q_j &\sim \text{Normal}(0, 1) \\
O_j &\sim \text{Normal}(0, 1) \\
H_j &\sim \text{Normal}(0, 1) \\
D_j &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

The Folk Theorem of Statistical Computing

“When you have computational problems, often there’s a problem with your model.”



Andrew Gelman
Spider-Man of Bayesian data analysis


1.5
0
-1.5
a[1]
-1.5
0
1.5
a[2]

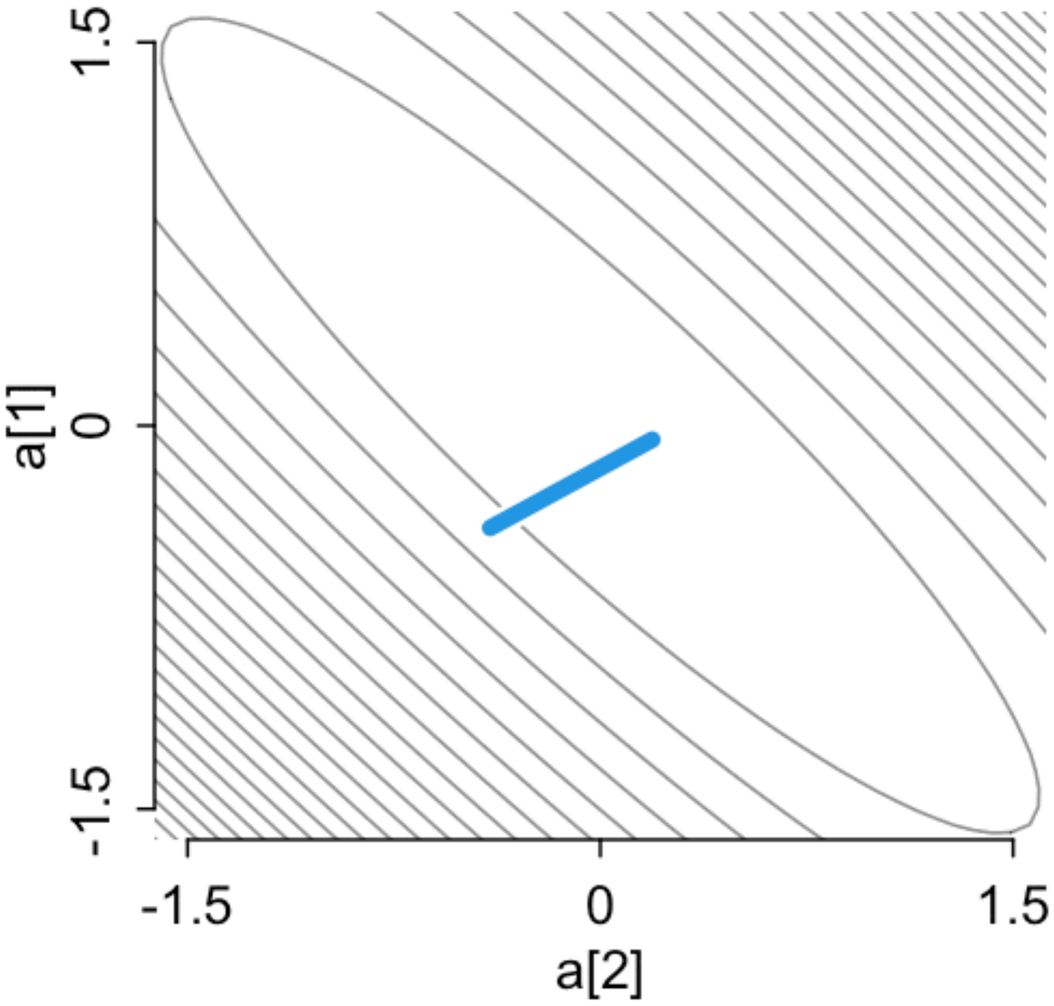
Divergent transitions

Divergent transition: A kind of rejected proposal

Simulation *diverges* from true path

Many DTs: poor exploration & possible bias

Will discuss again in later lecture



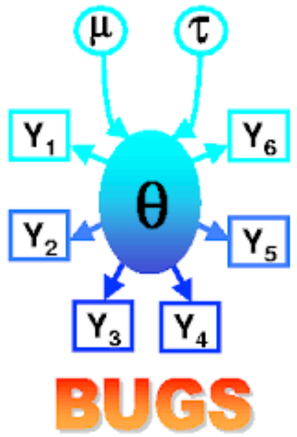
El Pueblo Unido

Desktop MCMC has been a revolution in scientific computing

Custom scientific modeling

High-dimension

Propagate measurement error



Sir David Spiegelhalter

Course Schedule

Week 1	Bayesian inference	Chapters 1, 2, 3
Week 2	Linear models & Causal Inference	Chapter 4
Week 3	Causes, Confounds & Colliders	Chapters 5 & 6
Week 4	Overfitting / MCMC	Chapters 7, 8, 9
Week 5	Generalized Linear Models	Chapters 10, 11
Week 6	Integers & Other Monsters	Chapters 11 & 12
Week 7	Multilevel models I	Chapter 13
Week 8	Multilevel models II	Chapter 14
Week 9	Measurement & Missingness	Chapter 15
Week 10	Generalized Linear Madness	Chapter 16

https://github.com/rmcelreath/stat_rethinking_2023