Theory of Monte-Carlo Analysis

EES 4891-06/5891-01

Bayesian Statistical Methods

Jonathan Magnolia Gilligan

Class #11: Wednesday February 18, 2026

Continuous Interactions

A Winter Flower

- Tulips grown in greenhouses
  - Soil
  - Light
  - Water

```
data(tulips)
d <- tulips
head(tulips)
```

```
##   bed water shade blooms
## 1   a     1     1   0.00
## 2   a     1     2   0.00
## 3   a     1     3 111.04
## 4   a     2     1 183.47
## 5   a     2     2  59.16
## 6   a     2     3  76.75
```

```
levels(tulips$bed)
```

```
## [1] "a" "b" "c"
```

```
d <- d |> mutate(
  blooms_std = blooms / max(blooms),
  water_cent = water - mean(water),
  shade_cent = shade - mean(shade)
  )
```

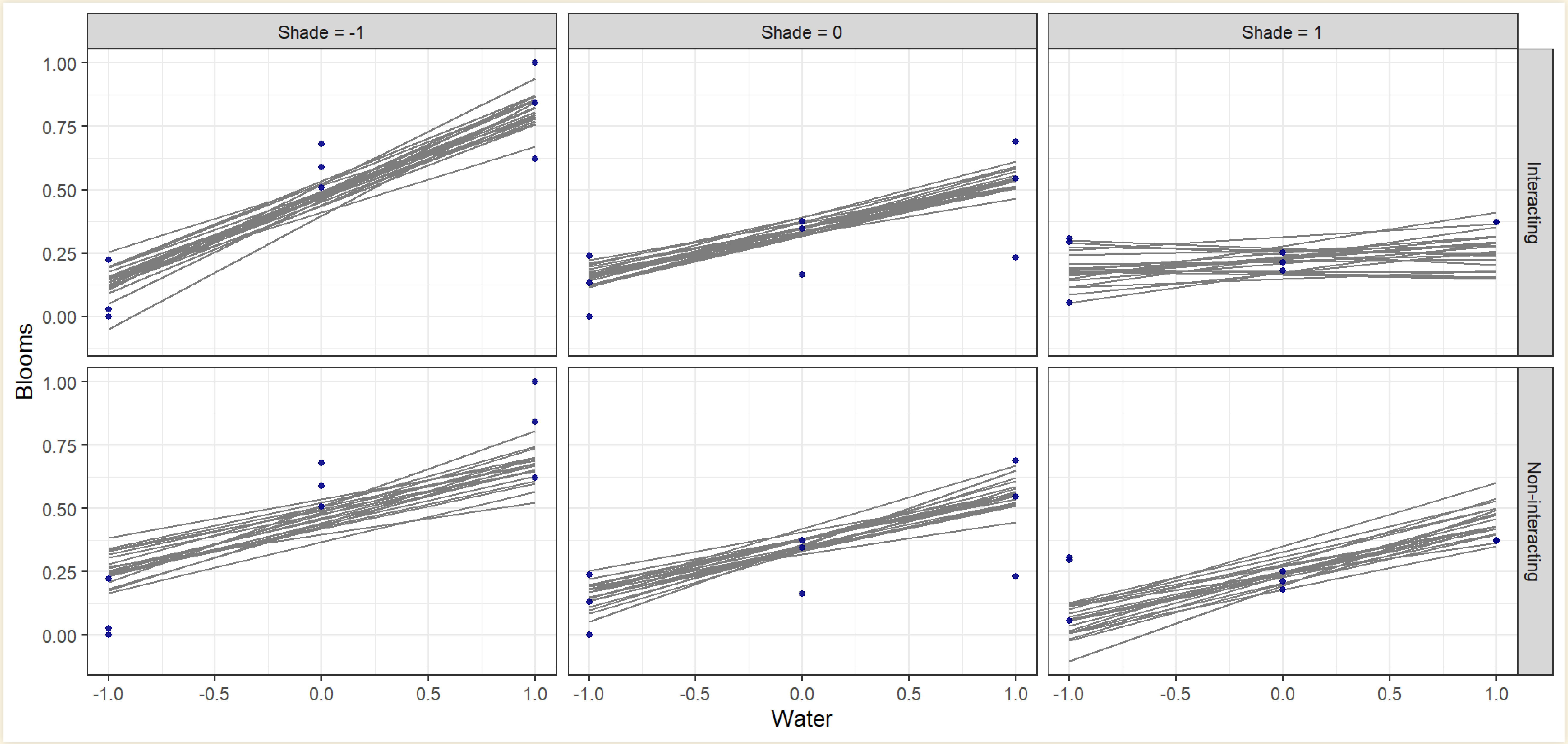
- Predict `blooms` from `water`, and `shade`.
- Two models:
  - **Non-interacting:**

```
mdl_tulip_non <- quap(
  alist(
    blooms_std ~ dnorm(mu, sigma),
    mu <- a + bw * water_cent + bs * shade_cent,
    a ~ dnorm(0.5, 0.25),
    bw ~ dnorm(0, 0.25),
    bs ~ dnorm(0, 0.25),
    sigma ~ dexp(1)
  ), data=d)
```

  - **Interacting:**

```
mdl_tulip_inter <- quap(
  alist(
    blooms_std ~ dnorm(mu, sigma) ,
    mu <- a + bw * water_cent + bs * shade_cent +
          bws * water_cent * shade_cent,
    a ~ dnorm(0.5, 0.25),
    bw ~ dnorm(0, 0.25),
    bs ~ dnorm(0, 0.25),
    bws ~ dnorm(0, 0.25),
    sigma ~ dexp(1)
  ), data=d)
```

Interpreting the models



Monte-Carlo Analysis

The Problem

- Bayes's Theorem

$$P(\beta|Y,X) = \frac{P(Y|\beta,X)P(\beta)}{P(Y|X)}$$
$$= \frac{P(Y|\beta,X)P(\beta)}{\int P(Y|\beta,X)P(\beta)d\beta}$$

  - β = the set of parameters for model (a vector)
  - Y = all the observed values of the outcome variable (a vector)
  - X = all the observed values of the predictor variables (an array)
  - $P(Y|\beta,X)$ = *likelihood* for Y (e.g., $Y \sim \text{Normal}(\beta x, \sigma)$)
  - $P(\beta)$ = *prior* for β
  - $P(Y|X)$ = *evidence*
- All of these terms are super-easy to calculate except the *evidence* term:

$$\int P(Y|\beta,X)P(\beta)d\beta$$

Calculating the *Evidence*

- For a few simple cases, you can solve the integral analytically
- For most cases, there is not a simple solution
- Numerical integration:
  - Approximate the integral
  - Grid sampling (Chapters 2–3):
    - You need many points on the grid (typically a few hundred to several thousand) to make sure you don’t miss important features of the prior & likelihood.
    - Works for 1 and 2 dimensional problems, if the prior and likelihood are sufficiently smooth.
    - For many dimensions (many parameters), it becomes computationally crazy.
      - 10 parameters, grid with 100 points along each dimension: $100^{10} = 10^{20}$ points.
        - If it takes one microsecond to calculate each grid point, then it would take more than 3 million years to calculate the whole grid.

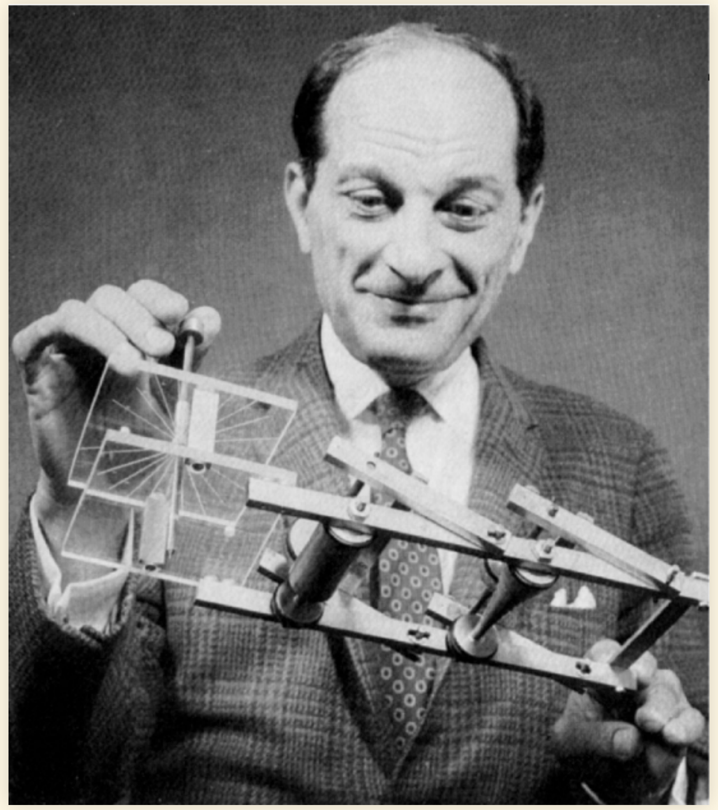
Monte Carlo Sampling

- Nuclear bomb research, 1940s:
  - The problem: Calculating diffusion of neutrons in the core of a thermonuclear bomb.
- Integrals were too hard for the best mathematicians
- Grid sampling was unfeasible
  - No electronic computers
  - “Computers” referfed to dozens of women using slide-rules and mechanical calculators to solve complex mathematical problems
- Stanislaw Ulam: Instead of a regular grid, pick a bunch of random numbers.
  - John von Neumann and Nicholas Metropolis made important contributions.

Computers (1943)

Source: NASA Ames Research Center, Photo ARC-1943-AAL-4961. Public Domain

Stanislaw Ulam



Source: Wikipedia, Public Domain

Origin of Monte-Carlo Integration

The first were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? Whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays.

This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations.

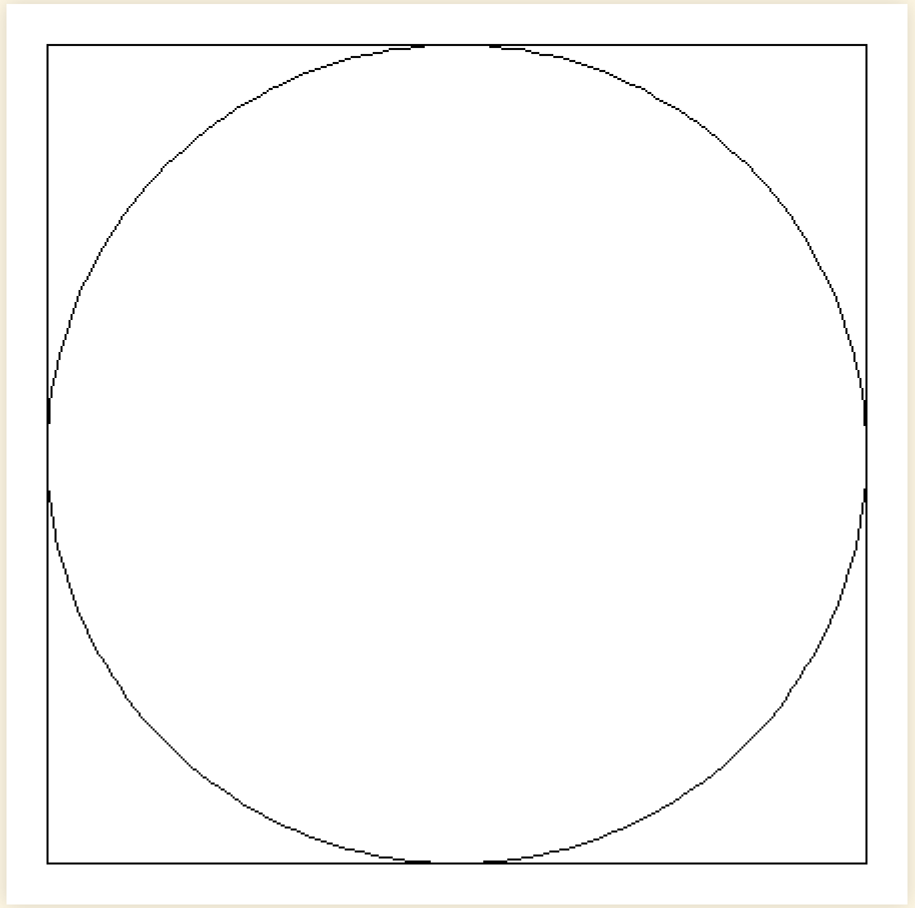
— Stanislaw Ulam

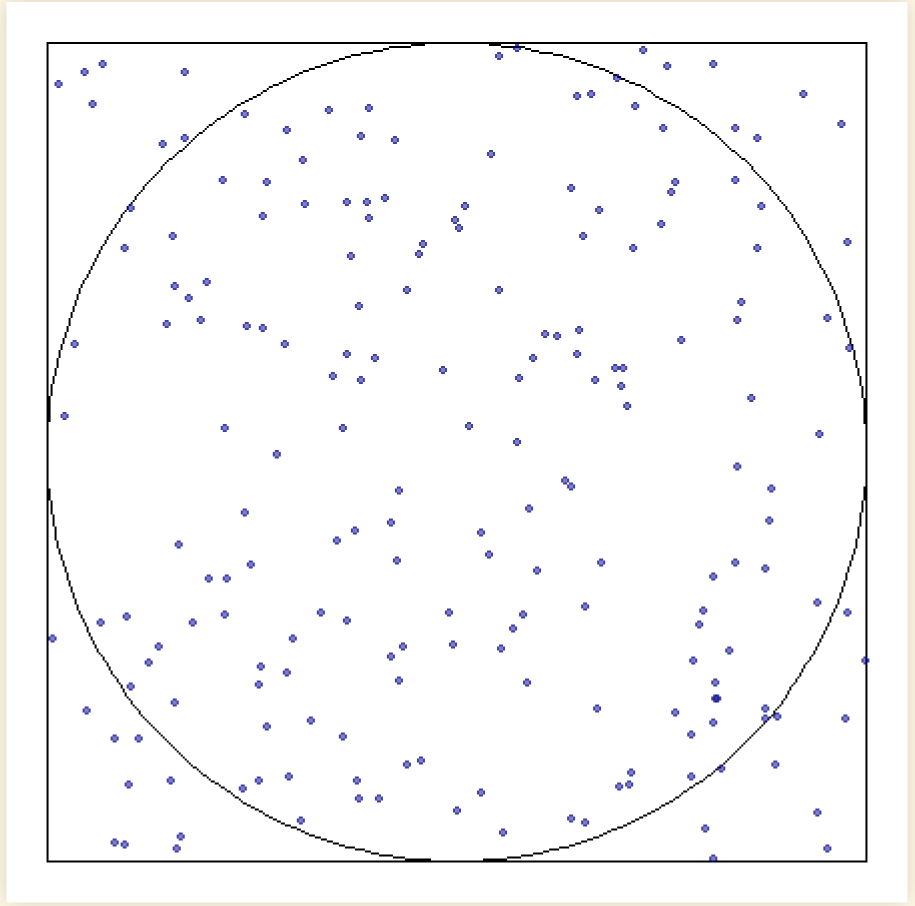
Simple Illustration

- Estimate the area of a circle
  - Draw a square 1 inch by 1 inch
  - Inscribe a circle 1 inch in diameter
  - Throw a dart at the paper many times
  - Count the number of times the dart lands inside the circle and the number it lands anywhere inside the square

$$\frac{\#\text{in circle}}{\#\text{in square}} = \frac{\text{area of circle}}{\text{area of square}}$$

  - 200 throws land in the square. 165 land inside the circle.
    - The ratio is 0.825.
    - The exact area of the circle is 0.785 in^2.
    - The difference is 5.0%
  - The method works just as well for the area inside any complicated shape





Monte-Carlo for Bayesian Analysis

- Monte-Carlo:
  - If you have *n parameters* (β_1, β_2, β_3, ..., β_n):
    1. Draw *n* random numbers $\beta_{i,1}$, $\beta_{i,2}$, ..., $\beta_{i,n}$:
       - each is a potential value for a *parameter*
       - Together, they represent one point in the *n*-dimensional *parameter space* $\beta_i = (\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,n})$
    2. Calculate *prior* $P(\beta_i)$ and *likelihood* $P(Y|\beta_i, X)$
    3. Repeat many times for $i = 1$ to $i = N_{\text{samples}}$
       - Generate β_1, β_2, ..., $\beta_{N_{\text{samples}}}$
       - Typically N_{samples} is several thousand
- For any point β in parameter space, the posterior for β is

$$P(\beta|Y, X) = \frac{P(Y|\beta, X)P(\beta)}{\sum_{i=1}^{N_{\text{samples}}} P(Y|\beta_i, X)P(\beta_i)}$$

- The bigger *n* is, the more samples you need.
  - For high *n*, this can become very large, computationally hard.
- Can we find a smarter way to pick random numbers?
  - What does it mean to be smart about randomness?
- Markov-Chain Monte Carlo:
  - The probability distribution for the *next* random number depends on the *last* one.

Markov-Chain Monte Carlo

Metropolis Algorithm

- Start with a random value for the parameter β_1
- To pick the next value β_{i+1}, from a current value β_i:

1. Toss a coin.
    - **Heads:** your proposed new value β' is a random number greater than β_i.
    - **Tails:** your proposed new value β' is a random number less than β_i.
2. Calculate

$$\begin{aligned} q &= \text{likelihood}(Y|\beta_i, X) \times \text{prior}(\beta_i) \\ q' &= \text{likelihood}(Y|\beta', X) \times \text{prior}(\beta') \end{aligned}$$

3. If $q' > q$, set $\beta_{i+1} = \beta_{\text{test}}$
4. Otherwise, pick a random number r between 0 and 1.
    - If $r \leq q'/q$ then set $\beta_{i+1} = \beta'$
    - Otherwise, reject the proposal and set $\beta_{i+1} = \beta_i$

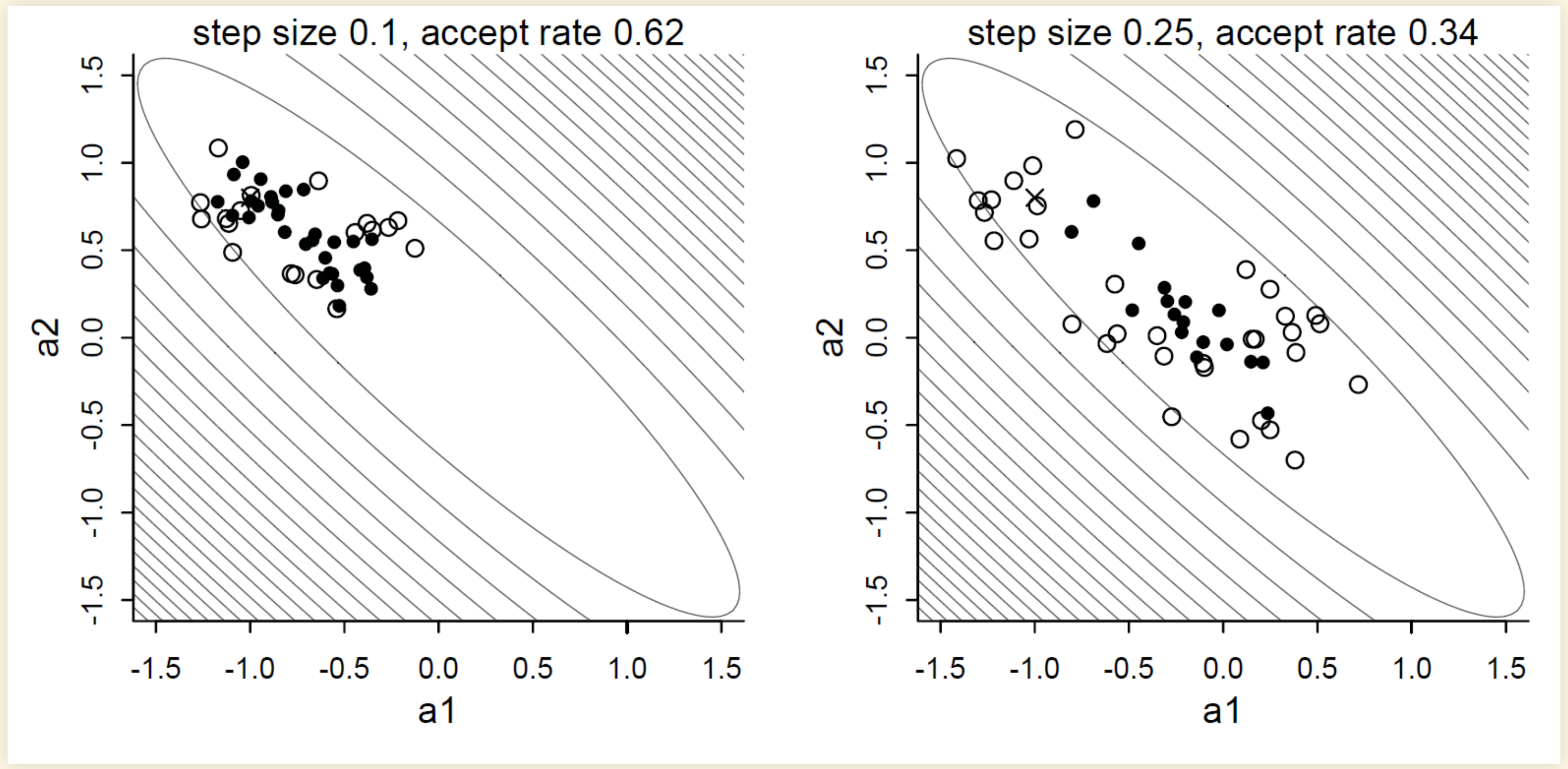
- This guarantees that you visit points where the posterior is greater more often than points where the posterior is small
    - The frequency of visiting a point is proportional to the posterior at that point
- If the posterior is relatively smooth, and only has one maximum, and is very small over most of *parameter space*, then you can estimate the integral by only looking at a small part of *parameter space*, where the posterior is significantly greater than zero.

Gibbs Sampling

- Metropolis sampling:
  - **Symmetric:** The probability of choosing a proposal to go from β_1 to β_2 is the same a proposal to go from β_2 to β_1.
  - **Random:** You don't use any information about what you know about the posterior to choose the next point.
- Gibbs Sampling:
  - **Asymmetric** and **adaptive:** Makes smarter proposals that sample the posterior much more efficiently.
  - Cost: You can only use certain kinds of priors, called *conjugate pairs*.
  - From 1989 to 2012, Gibbs sampling was the state of the art for most Bayesian analysis.
    - R package `rjags`

Limitations of Metropolis & Gibbs

- Metropolis and Gibbs sampling work very well for small numbers of parameters (<100 or so)
- For hundreds or thousands of parameters, they break badly.
- Even for small numbers of parameters, they break badly when parameters are highly correlated
  - (think about height and length of legs, or exercise 6M2)



- Samples don't uniformly fill high-probability region
  - × = starting point (β_1)
  - ● = accepted proposals
  - ○ = rejected proposals

- The problem is that these algorithms only look along a few directions at a time, and with many dimensions, they may not be looking in the most interesting direction, so they keep poking around in the dark, not looking at where the interesting stuff is.

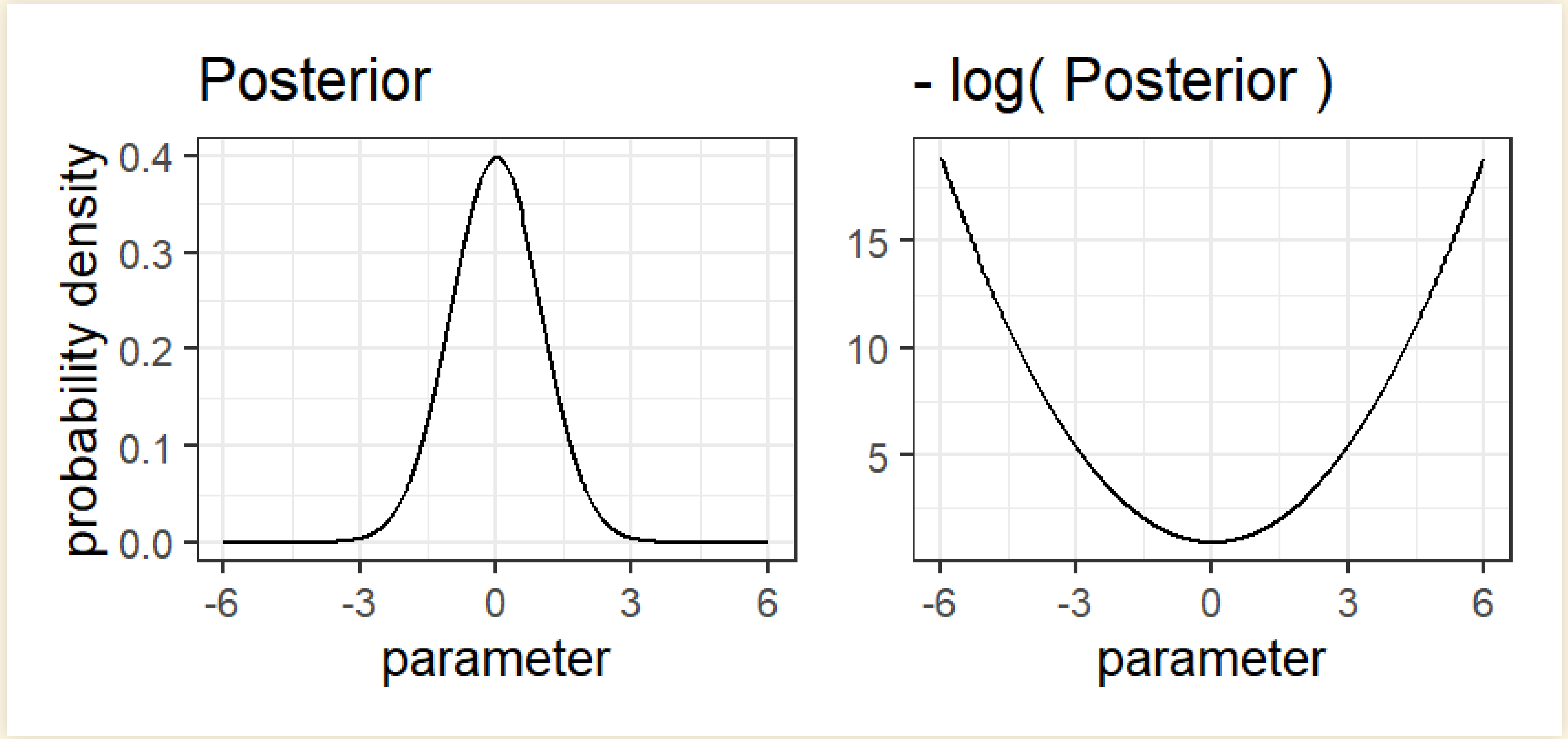
Hamiltonian Monte Carlo

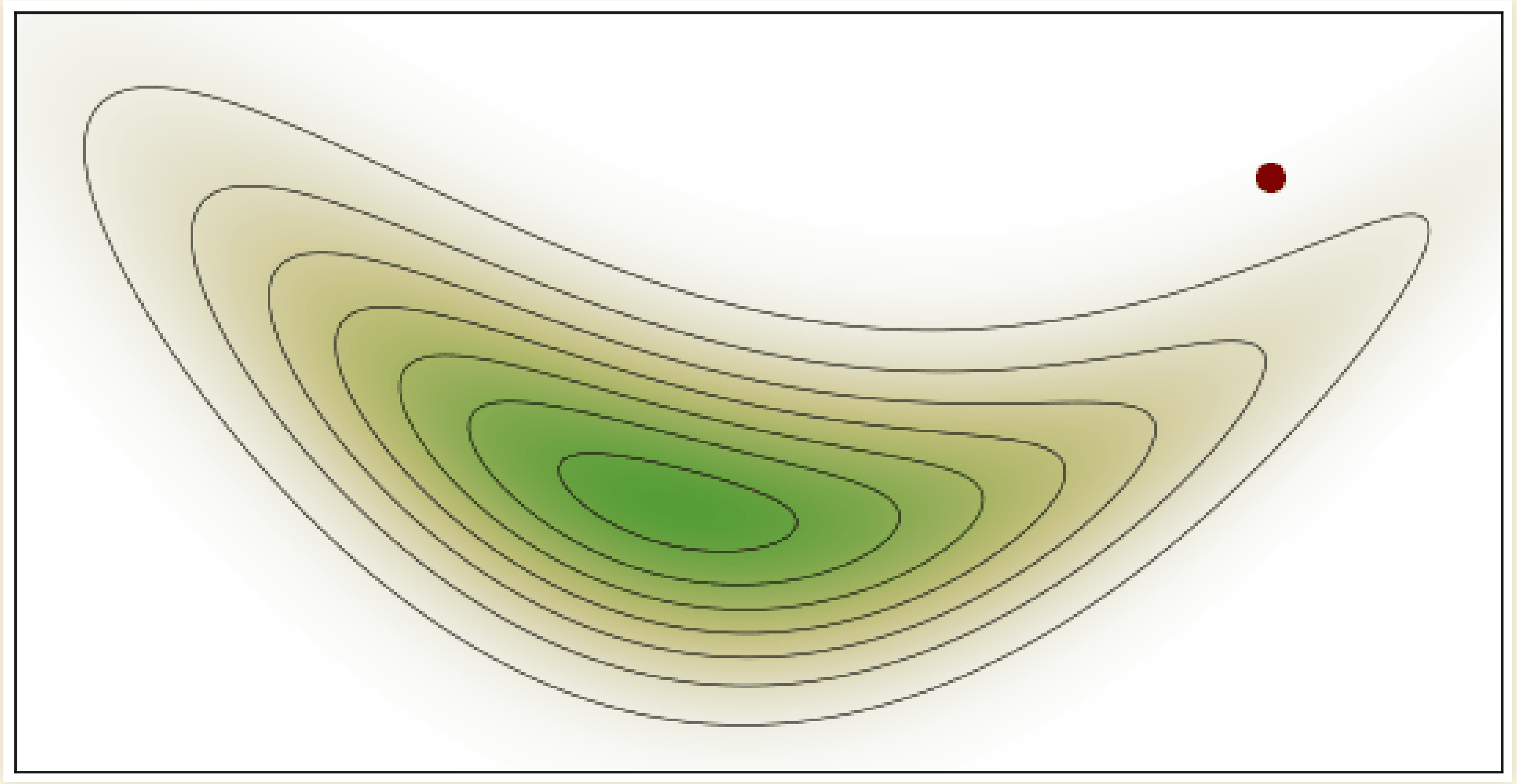
Simple vs. Complex Monte Carlo

- Metropolis and Gibbs make simple proposals.
  - The computational cost of making a proposal is small
  - The quality of the proposals is also small
  - For simple problems, you don’t need high-quality proposals.
  - For complex problems, low-quality proposals waste time and the algorithm spins its wheels.
- Hamiltonian Monte Carlo (HMC)
  - Proposals are costly
  - But their quality is much greater
  - For complex problems, HMC finds solutions much faster.

Hamiltonian Monte Carlo Carlo

- Hamiltonian Monte Carlo uses a physics simulation to do statistical calculations
  - Think of probability distributions as hills and valleys
    - The elevation is the negative logarithm of the posterior probability density
  - Consider your Monte Carlo sampling point like a ball rolling on the landscape
    1. Put the ball at a random starting point.
    2. Flick the ball in a random direction with a random velocity
    3. Allow the ball to roll over the landscape for some amount of time
    4. After the time is up, wherever the ball is, that's your next proposal.
    5. Repeat steps 2–4





Credit: Wikipedia

Interlude

Poods bowl session w/ 187 & Triple 8 team, Encinitas CA

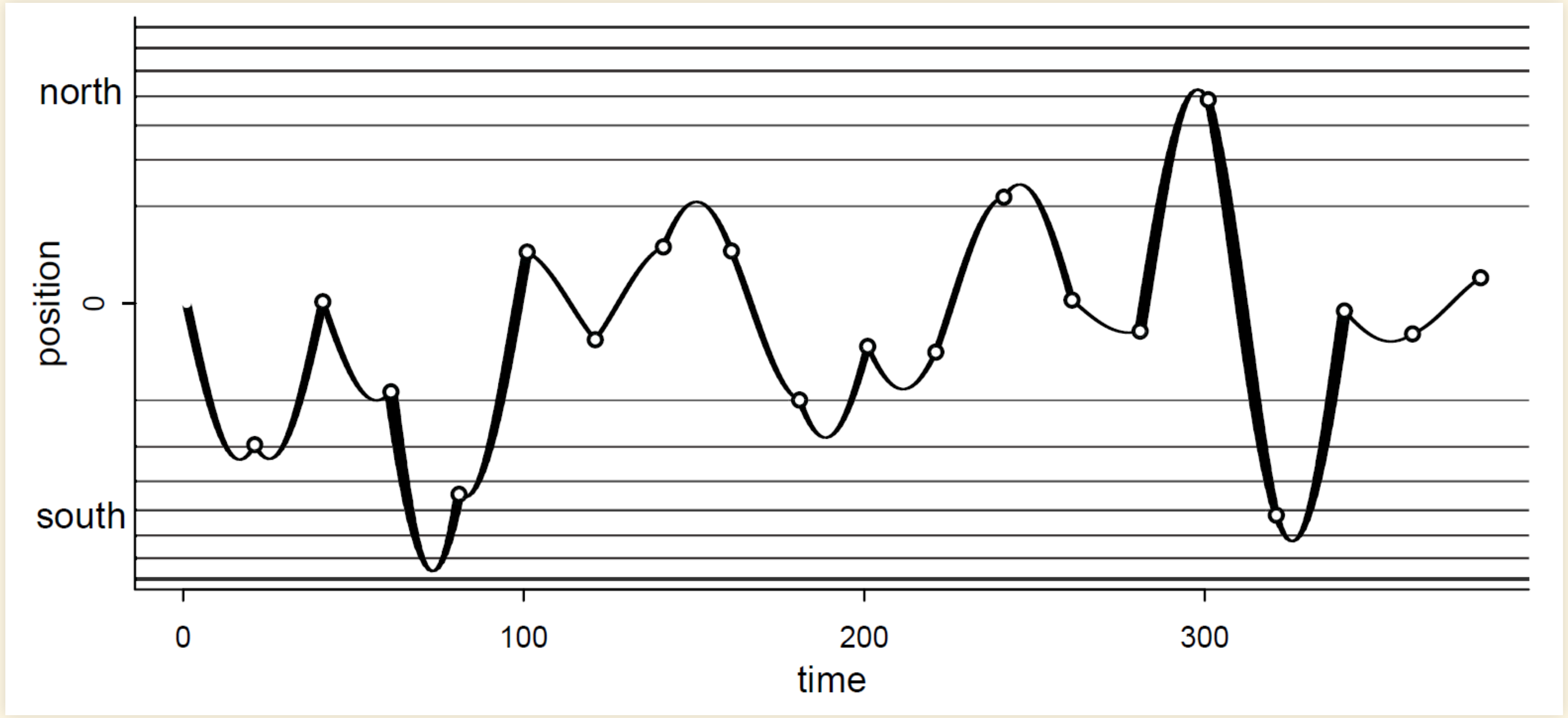
Stan

- For many years, HMC wasn't very useful:
  - You had to write custom code for the simulation
  - There are many options to control the performance of the simulation
    - No one knew how to adjust the options effectively.

- In 2012, the `stan` program changed everything:
  - The developers figured out how to adjust the HMC options effectively and automatically.
  - `Stan` lets you write your model in a simple programming language and translates it to efficient `c++`, which is compiled and runs very fast.
  - With R packages like `rethinking` and `brms`:
    - You specify the model in `R`
    - The package automatically translates your model into the `stan` language
    - `stan` then creates a `c++` program for an HMC simulator of your model

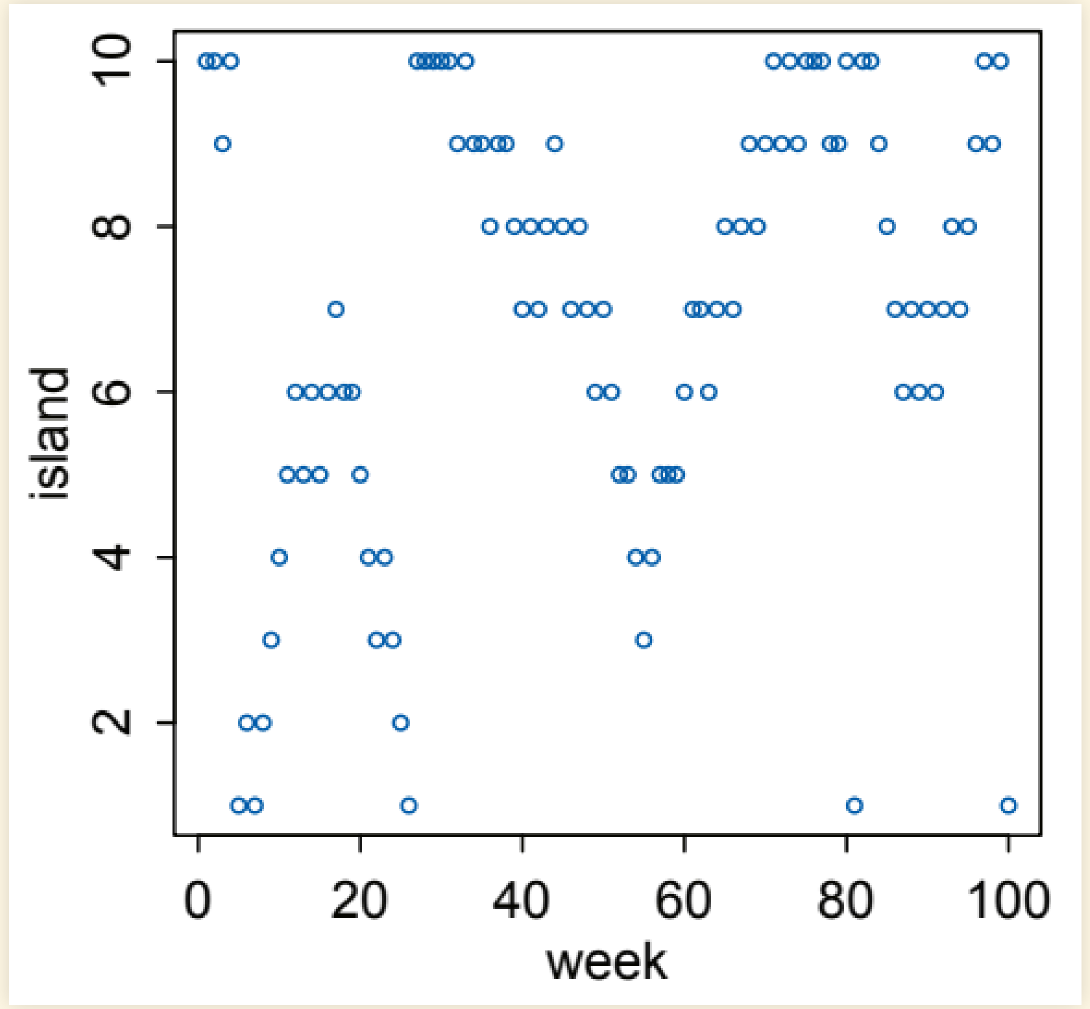
Illustrations of Hamiltonian Monte Carlo

Hamiltonian Monte Carlo



- Horizontal lines are elevation contours
- Bottom of the valley at 0: uphill to North and South.
- Thickness of the line is momentum (speed)
  - Ball slows down as it moves uphill
  - Ball speeds up as it moves downhill

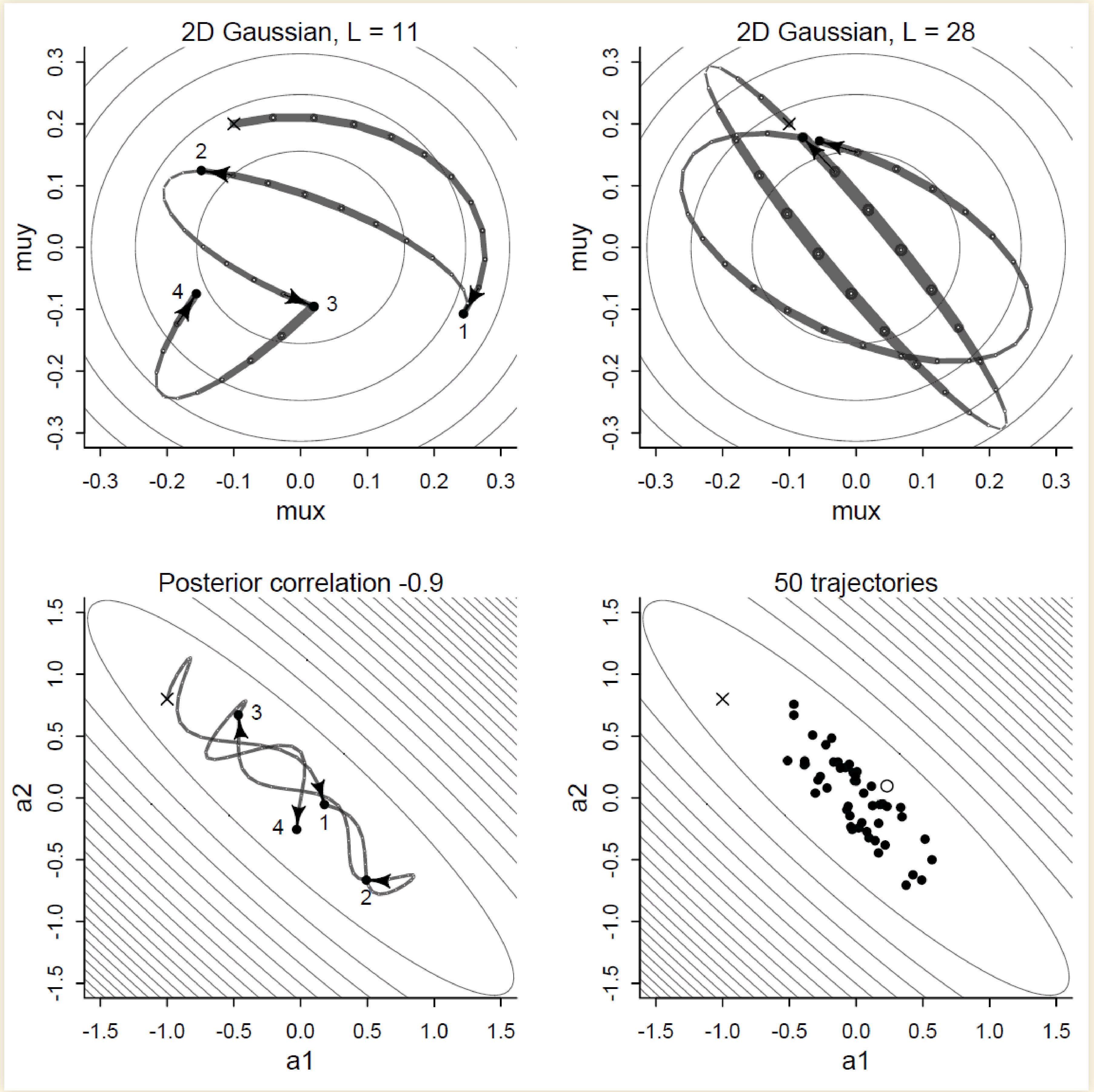
Metropolis Monte Carlo



- Much greater autocorrelation among samples

Illustrations of Hamiltonian Monte Carlo

- × = start, ● = accepted proposal, ○ = rejected proposal
- Top: Uncorrelated parameters
  - Top left: Well-tuned HMC
    - Samples are far apart, uncorrelated
  - Top right: Poorly-tuned HMC
    - Samples are close together because "rolling ball" made U-turns
    - Stan has a "No U-Turn Sampler" (NUTS) to avoid this.
- Bottom: Highly correlated parameters
  - Lower left: HMC samples the space effectively, with little autocorrelation
  - Lower right: Only one proposal was rejected.
    - Effectively explores the space and quickly finds the maximum.



Limitations of HMC

- HMC can only work with priors that have continuous parameters and are continuously differentiable.
- Metropolis and Gibbs can solve models with discrete parameters (integers, categories, etc.)
- You can usually find workarounds for models with discrete parameters, but this requires clever thinking
- HMC can fail and it can be very confusing to figure out why.
- But when it works well, HMC is enormously effective and much faster than Gibbs or Metropolis sampling