

# Markov Chain Monte Carlo (MCMC)

---

Tim Frasier

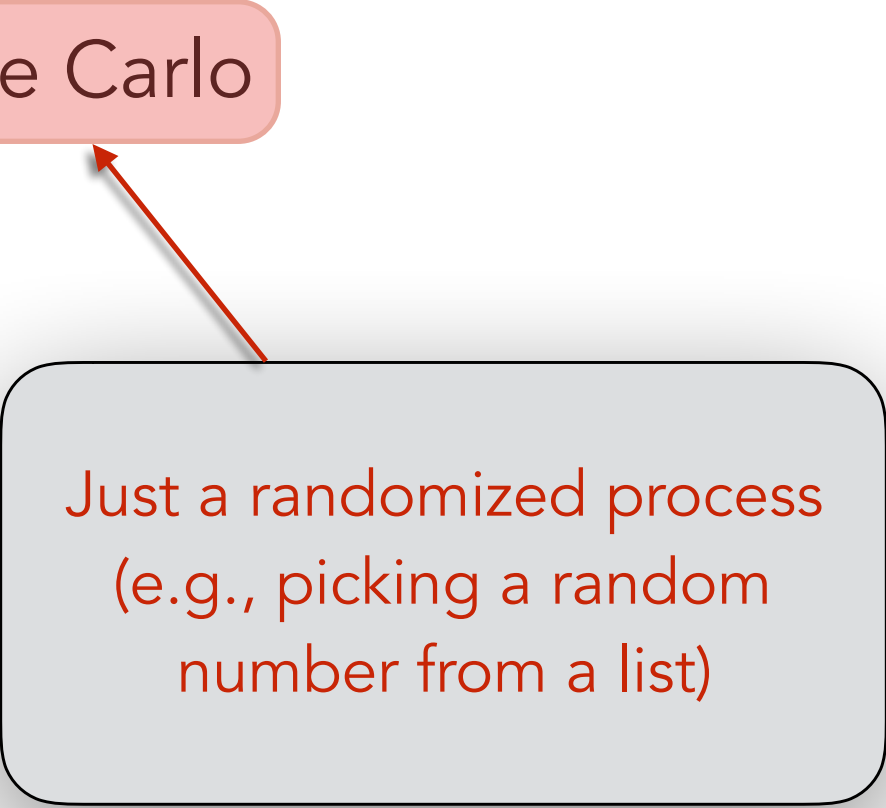
**What is MCMC?**

# What Is MCMC?

- Markov Chain Monte Carlo

# What Is MCMC?

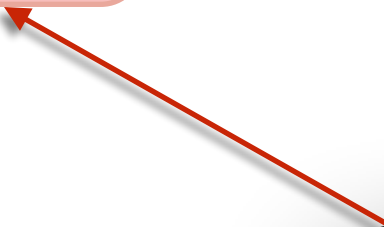
- Markov Chain Monte Carlo



Just a randomized process  
(e.g., picking a random  
number from a list)

# What Is MCMC?

- Markov Chain Monte Carlo



A process where the next step only depends on the current location (this will make more sense in a minute)

# What Is MCMC?

## Think of a robot walking through a field

- Step 1: Pick a random point along an edge to start at
- Step 2: Randomly select what direction it turns
- Step 3: If step will keep robot in field, take one step in that direction

If steps 2 & 3 repeated enough times, will eventually visit every position in the field even though it is a random process

# What Is MCMC?

## Think of a robot walking through a field

- Step 1: Pick a random point along an edge to start at
- Step 2: Randomly select what direction it turns
- Step 3: If step will keep robot in field, take one step in that direction

If steps 2 & 3 repeated enough times, will eventually visit every point in the field, even though it is a

The Monte Carlo part

# What Is MCMC?

## Think of a robot walking through a field

- Step 1: Pick a random point along an edge to start at
- Step 2: Randomly select what direction it turns
- Step 3: If step will keep robot in field, take one step in that direction

If steps 2 & 3 repeated enough times, will eventually visit every position in the field even though it is a

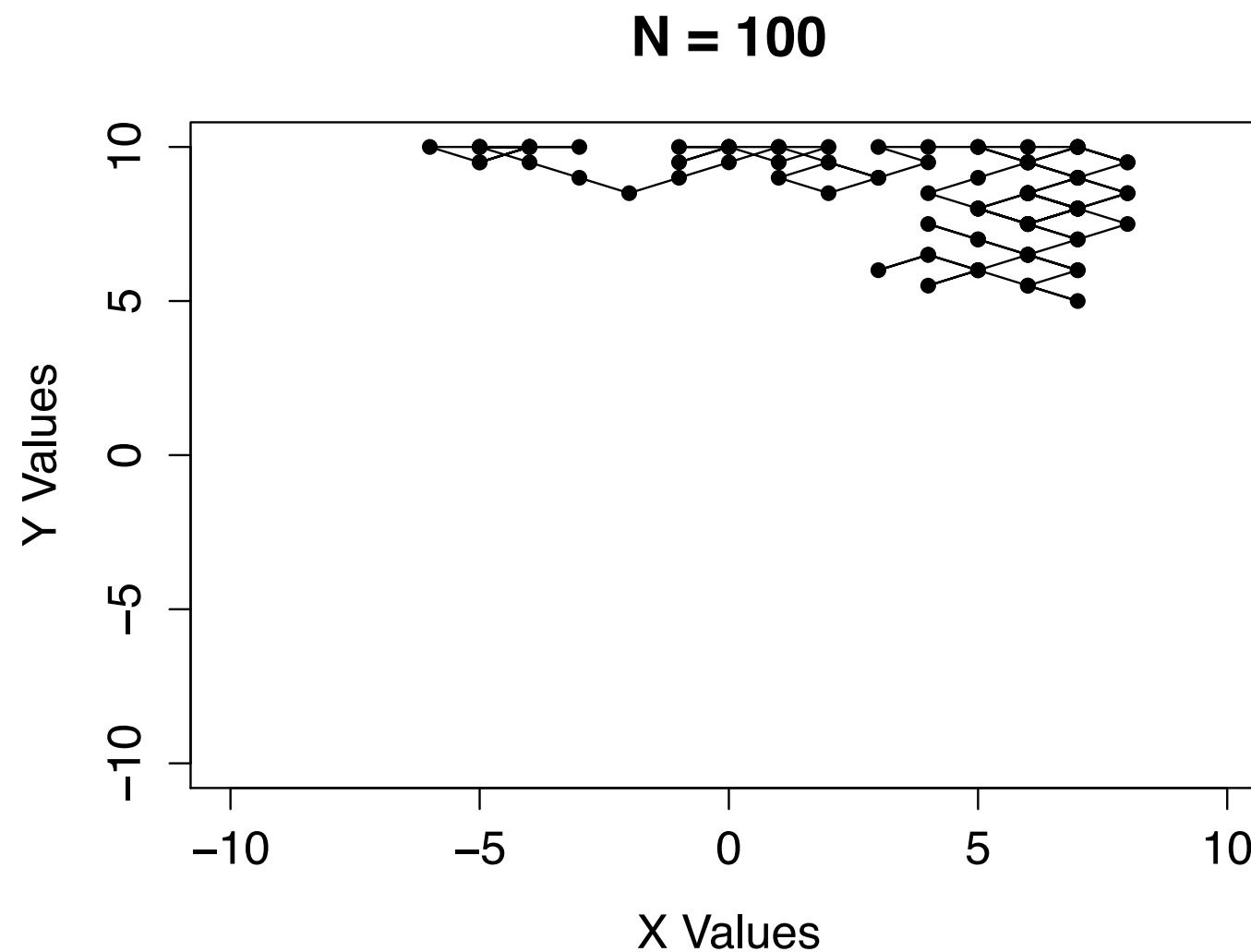
The Markov Chain part  
(where it goes next depends  
on where it is now)



# What Is MCMC?

Think of a robot walking through a field

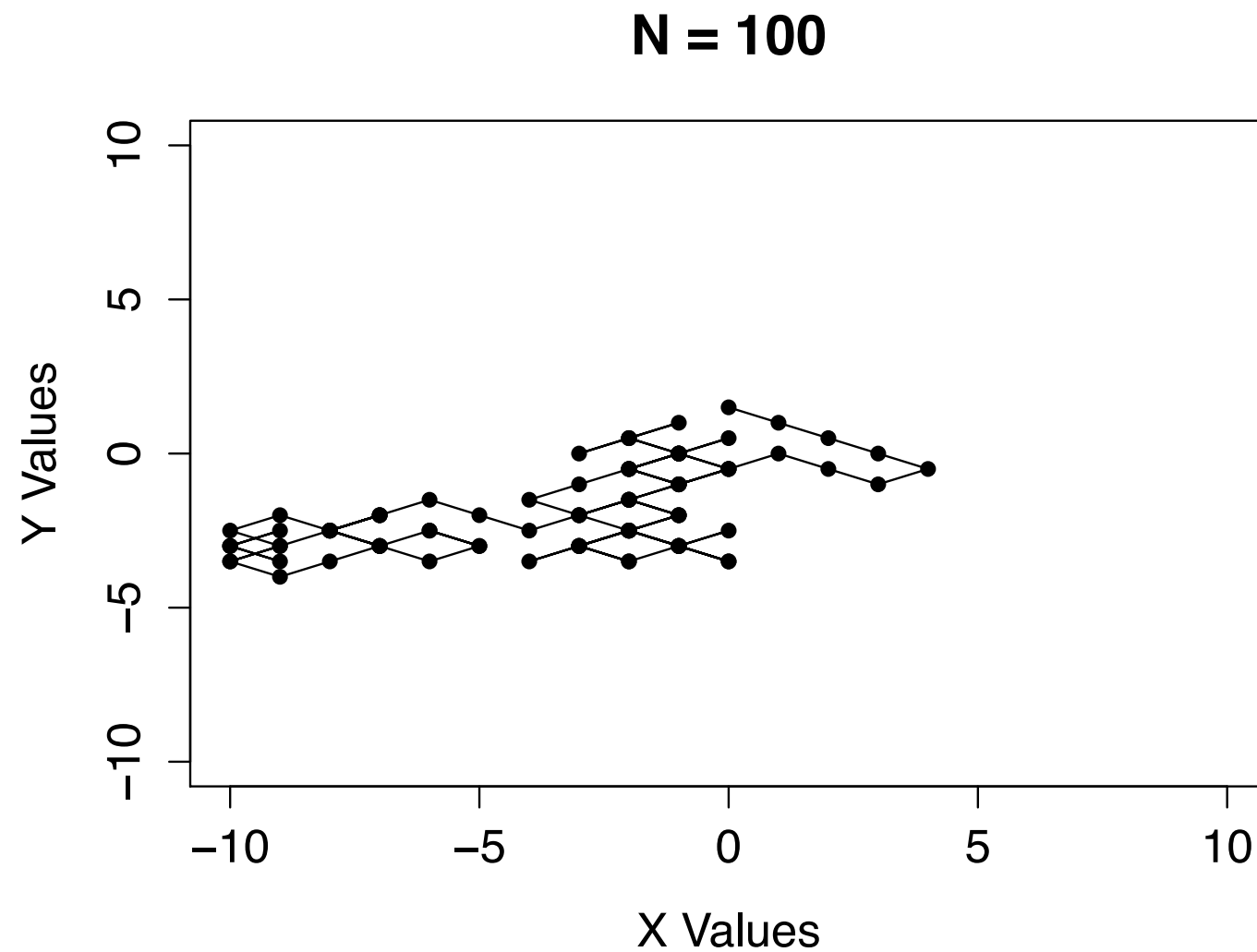
- 100 steps



# What Is MCMC?

Think of a robot walking through a field

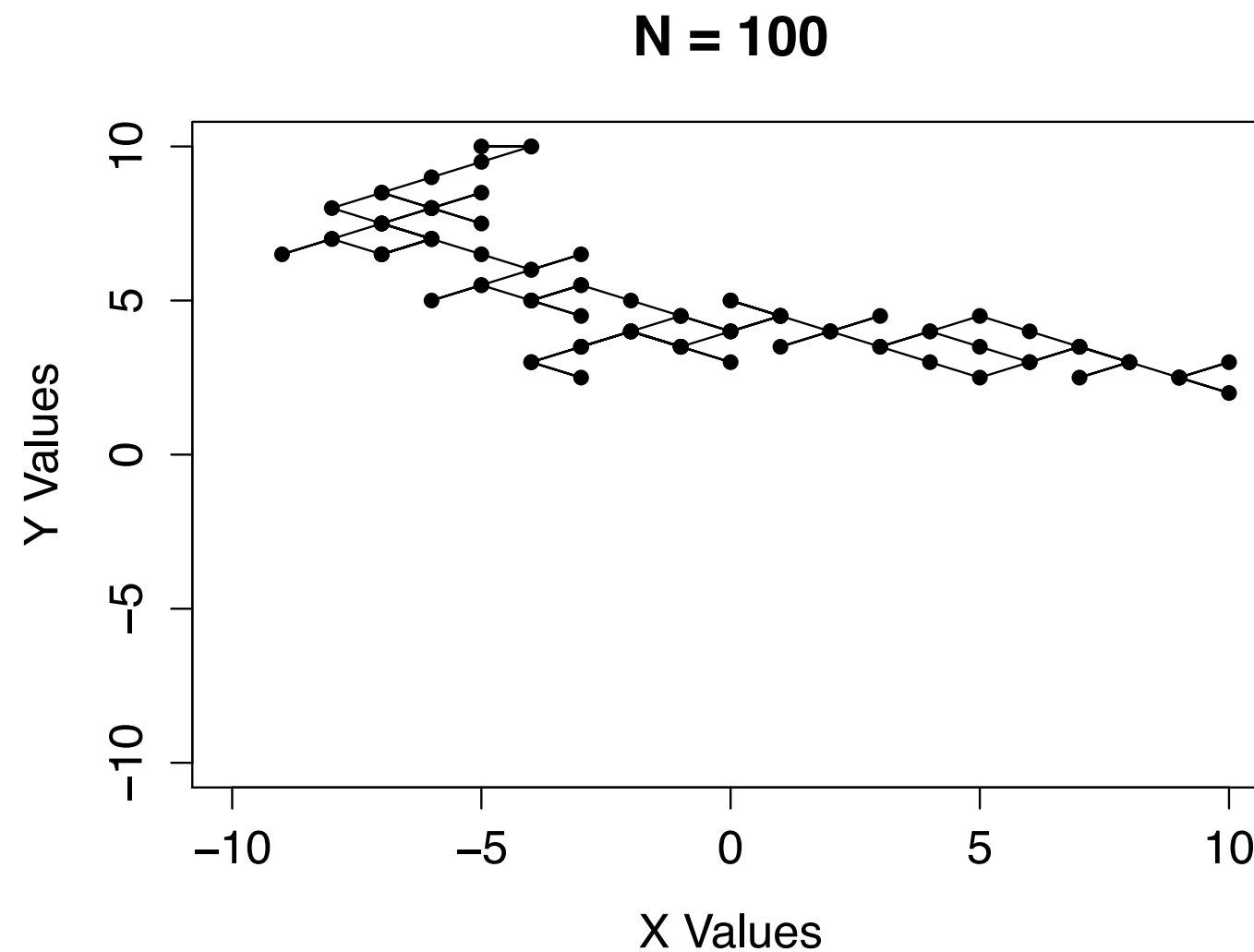
- 100 steps (again)



# What Is MCMC?

Think of a robot walking through a field

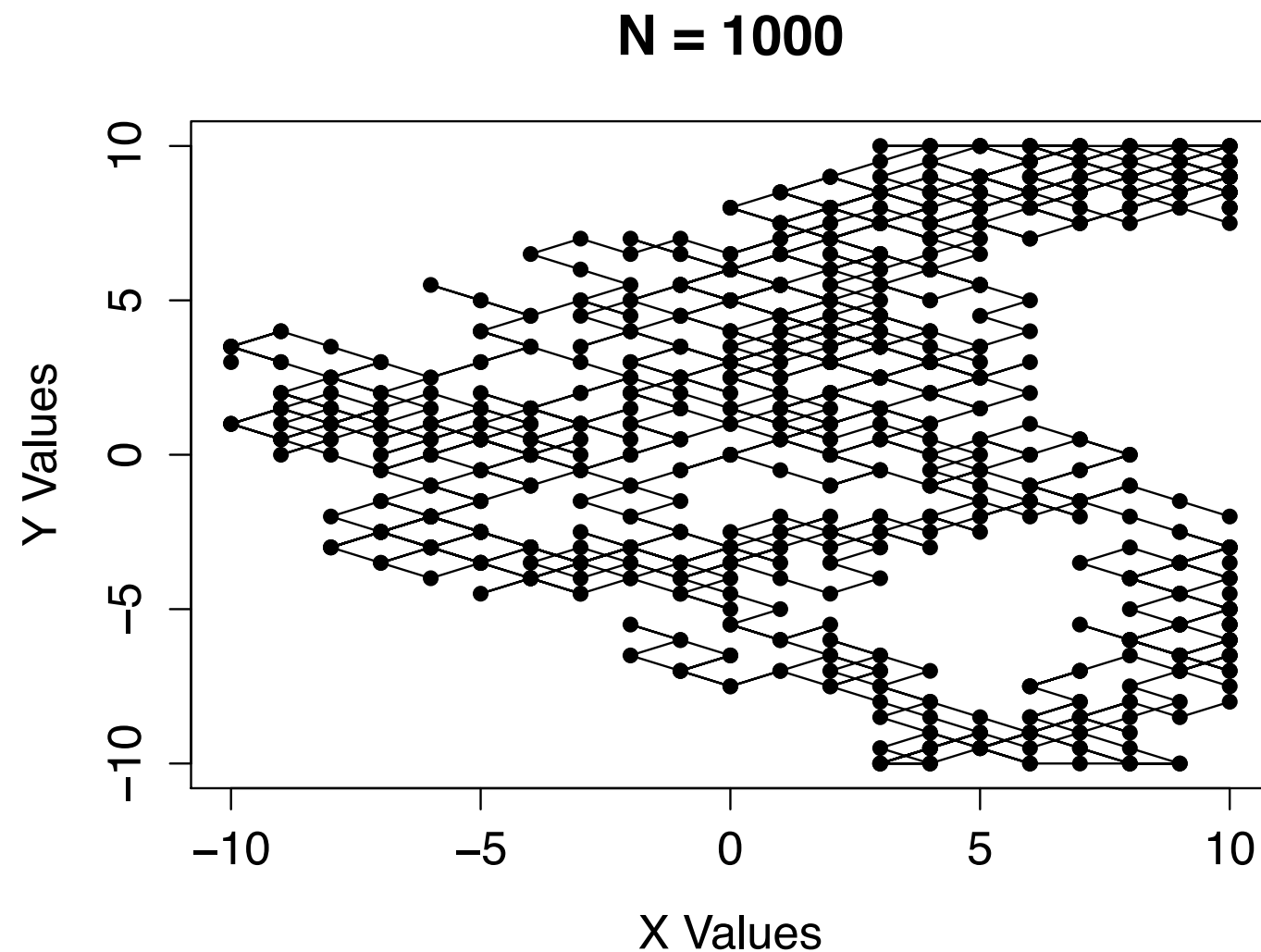
- 100 steps (and again)



# What Is MCMC?

Think of a robot walking through a field

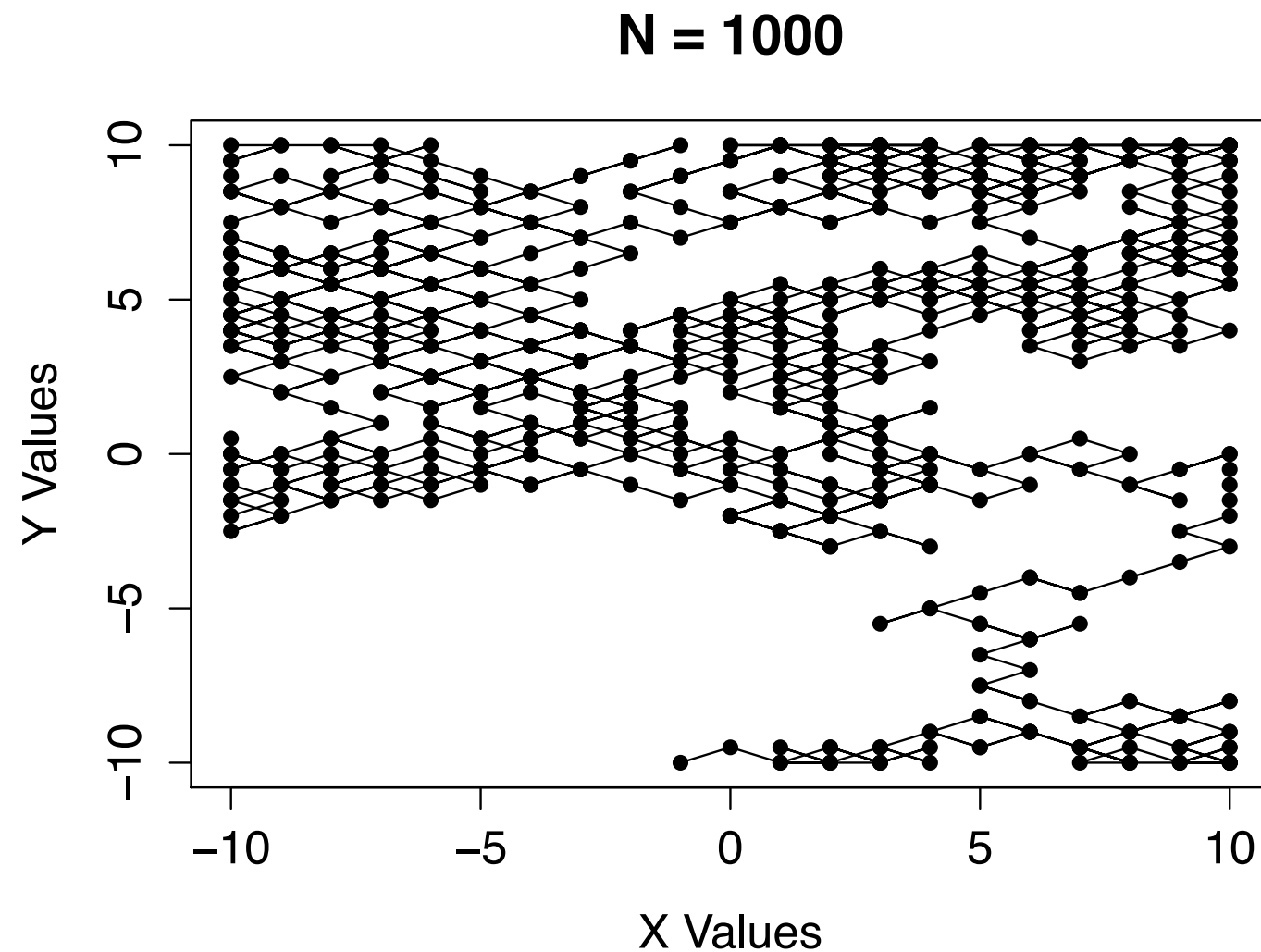
- 1,000 steps



# What Is MCMC?

Think of a robot walking through a field

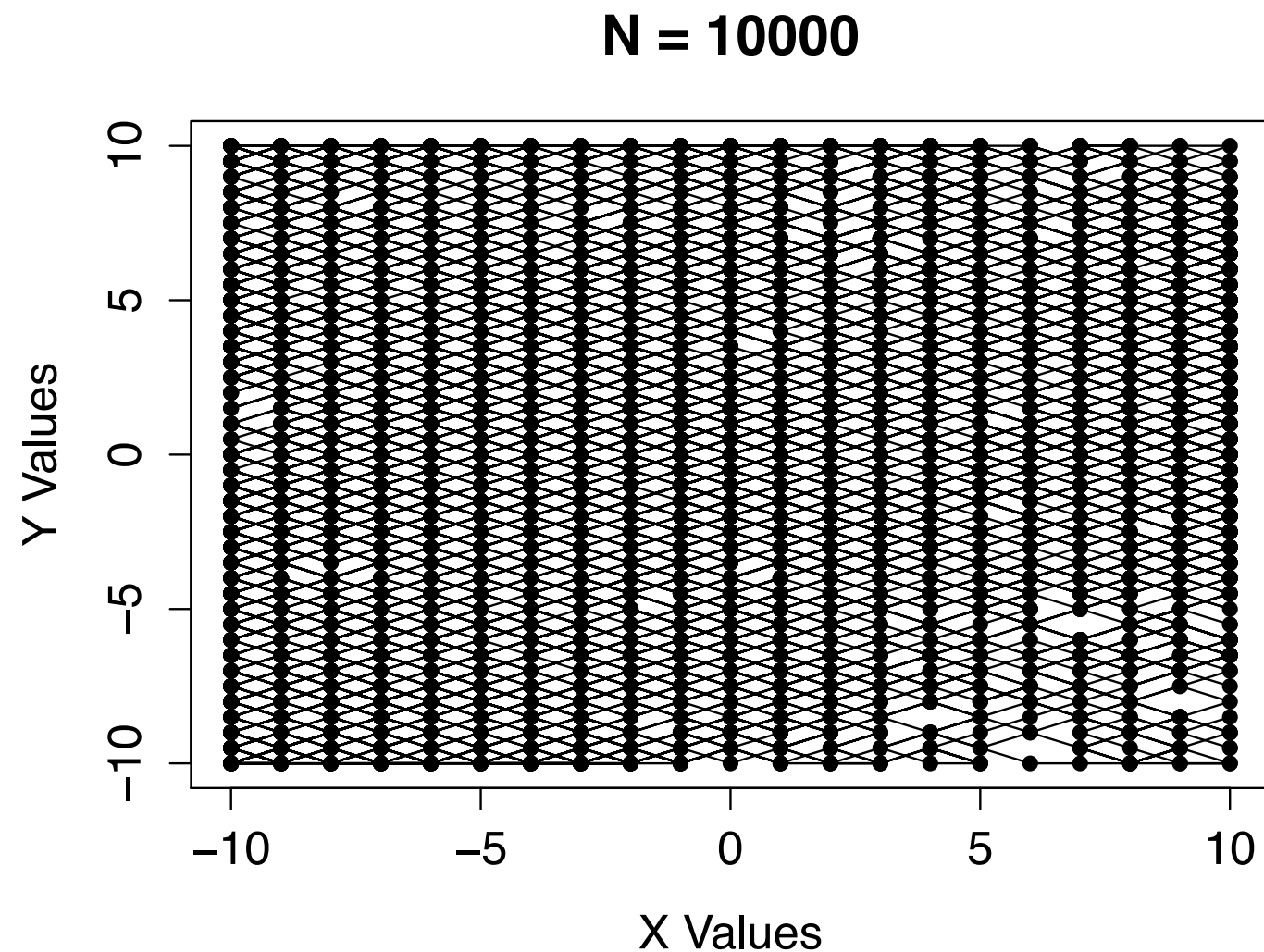
- 1,000 steps (again)



# What Is MCMC?

Think of a robot walking through a field

- 10,000 steps



**Your  
Turn!**

# MCMC Exercise 1

1. Put `basicMCMC.R` file in R's working directory
2. Load it into R

```
source("basicMCMC.R")
```



# MCMC Exercise 1

3. Explore how different starting points influence chain. For example, repeat the command below a number of times. (remember the  $\uparrow$  functionality). Try other chain lengths too!

```
basicMCMC(nSteps = 100)
```

# MCMC Exercise 1

4. Explore how different chain lengths influence chain. Some examples are below, but play around yourself!

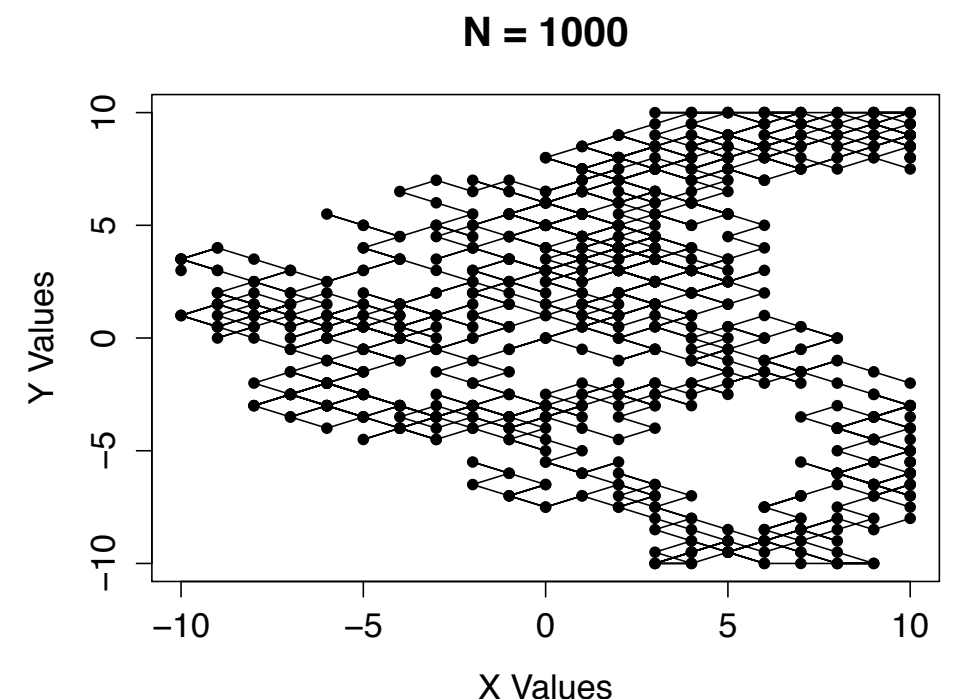
```
basicMCMC(nSteps = 100)  
basicMCMC(nSteps = 500)  
basicMCMC(nSteps = 1000)  
basicMCMC(nSteps = 5000)
```

**How Does It Find  
Values With Highest  
Probabilities?**

# How Does It Find High Probabilities?

## Clarification

- Instead of a robot walking through a field, is really a **model** moving through **parameter space**



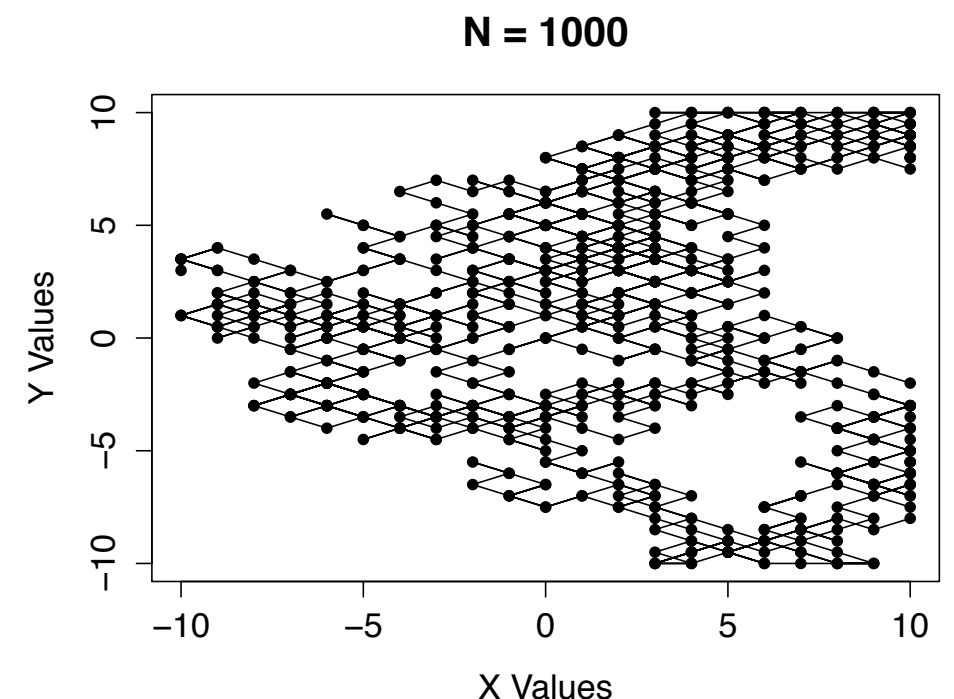
# How Does It Find High Probabilities?

## Clarification

- Instead of a robot walking through a field, is really a **model** moving through **parameter space**

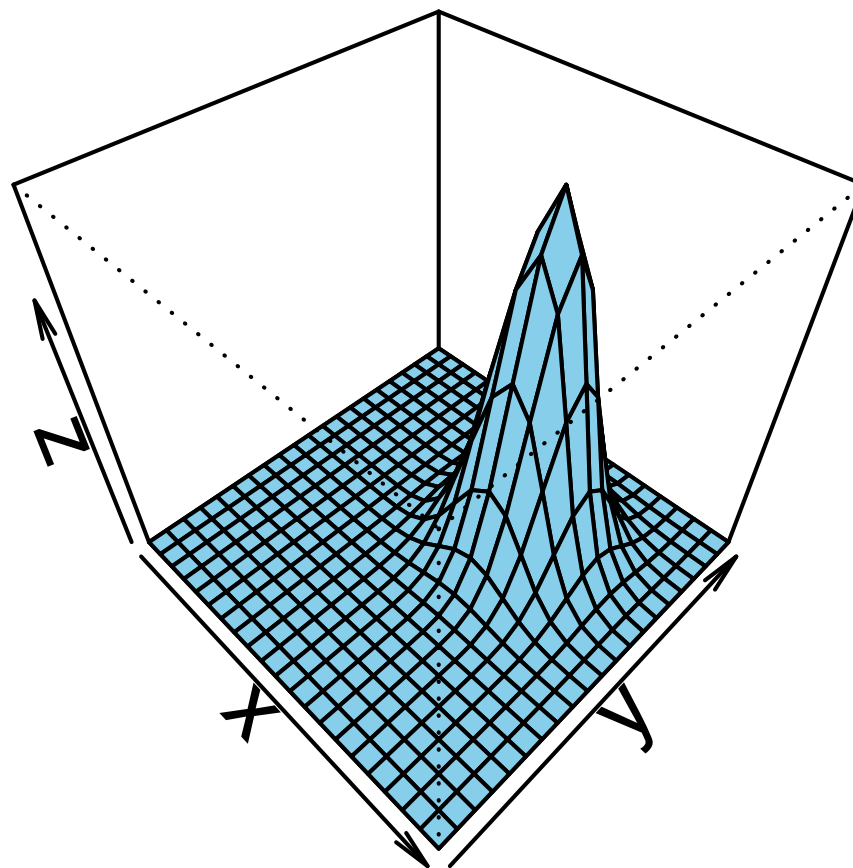
- Parameter space:  
possible x and y values
- Model (simple):  
If proposed value is in  
possible values (-10 to 10 in this case),  
take step

Current parameter space is  
**flat** (no values have higher  
likelihood than others)



# How Does It Find High Probabilities?

- In real problems, different values will have different likelihoods
- Suppose a simple parameter space with a single peak representing a normal distribution with a mean of 3 and a standard deviation of 2 (in both the x- and y-directions)



# How Does It Find High Probabilities?

- The height of the peak for a given value represents the likelihood of that value
- For all probability distributions, this value is calculable (often ugly, but still calculable)
- For the normal distribution, this equation is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$x$  = observed value

$\mu$  = mean

$\sigma$  = s.d.

# How Does It Find High Probabilities?

- Need to set up an **algorithm** that will cause the model to find peaks (if they exist)

## Algorithm

**A series of instructions for a computer program to follow (like a recipe, but for a computer)**



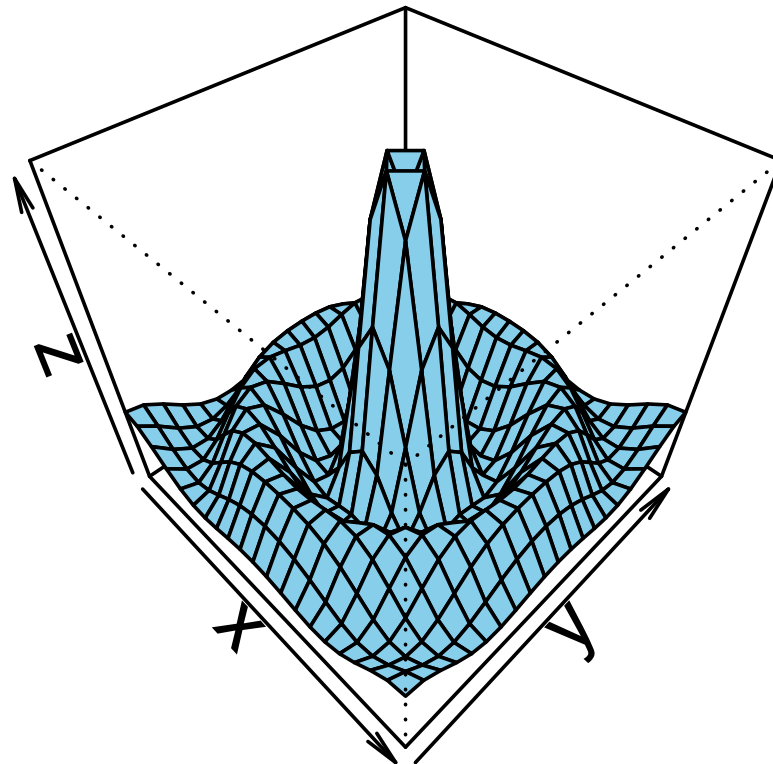
# How Does It Find High Probabilities?

- Need to set up an **algorithm** that will cause the model to find peaks (if they exist)
- Example (**algorithm 1**):
  - Step 1: Propose new value (next step)
  - Step 2: Calculate the probability at proposed position
  - Step 3: (a) If proposed probability  $>$  probability at current position, take the step  
(b) If not, stay in current position

Repeat steps 1-3 many times

# How Does It Find High Probabilities?

- **Algorithm 1** will “work”, in that it will climb the first peak it finds and stay there
- However
  - Will get stuck on local peaks (no way to move “downhill”)
  - Won’t explore parameter space very well



# How Does It Find High Probabilities?

- Need an algorithm that **preferentially** takes steps to values with higher likelihoods, but has **some potential** to also move to values with lower likelihoods (to walk through valleys)
- **Metropolis-Hastings Algorithm**

# How Does It Find High Probabilities?

- **Metropolis-Hastings Algorithm**

- Step 1: Propose new value (next step)
- Step 2: Calculate the probability at proposed position
- Step 3: (a) If proposed probability  $>$  probability at current position, take the step  
(b) If not,
  - (i) Calculate the ratio of the probabilities at the proposed and current position
  - (ii) Draw a random number between 0 and 1 (inclusive)
  - (iii) If random number is  $\geq$  this ratio, take step
  - (iv) If not, don't take step

# How Does It Find High Probabilities?

- **Metropolis-Hastings Algorithm**

- Step 1: Propose new value (next
- Step 2: Calculate the probability at proposed position
- Step 3: (a) If proposed probability  $>$  probability at current position, take the step  
(b) If not,

More readily takes small steps down than big steps, but big steps not impossible

- (i) Calculate the ratio of the probabilities at the proposed and current position
- (ii) Draw a random number between 0 and 1 (inclusive)
- (iii) If random number is  $\geq$  this ratio, take step
- (iv) If not, don't take step

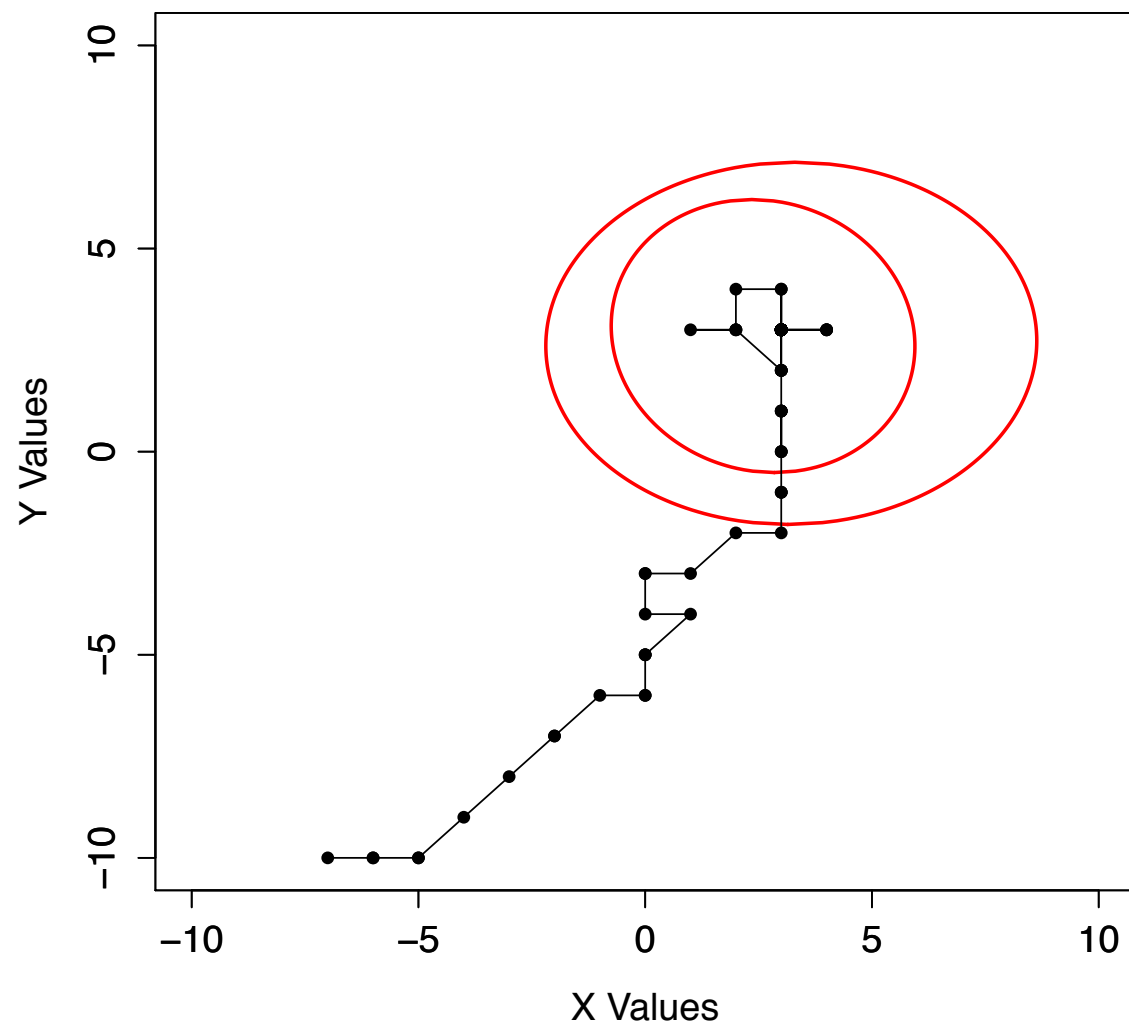
# How Does It Find High Probabilities?

- Again, the model doesn't have to "know" where peaks are
- If we follow these rules, it should walk around parameter space until it finds a peak, then it should climb the peak and stay there (with some potential to cross valleys)

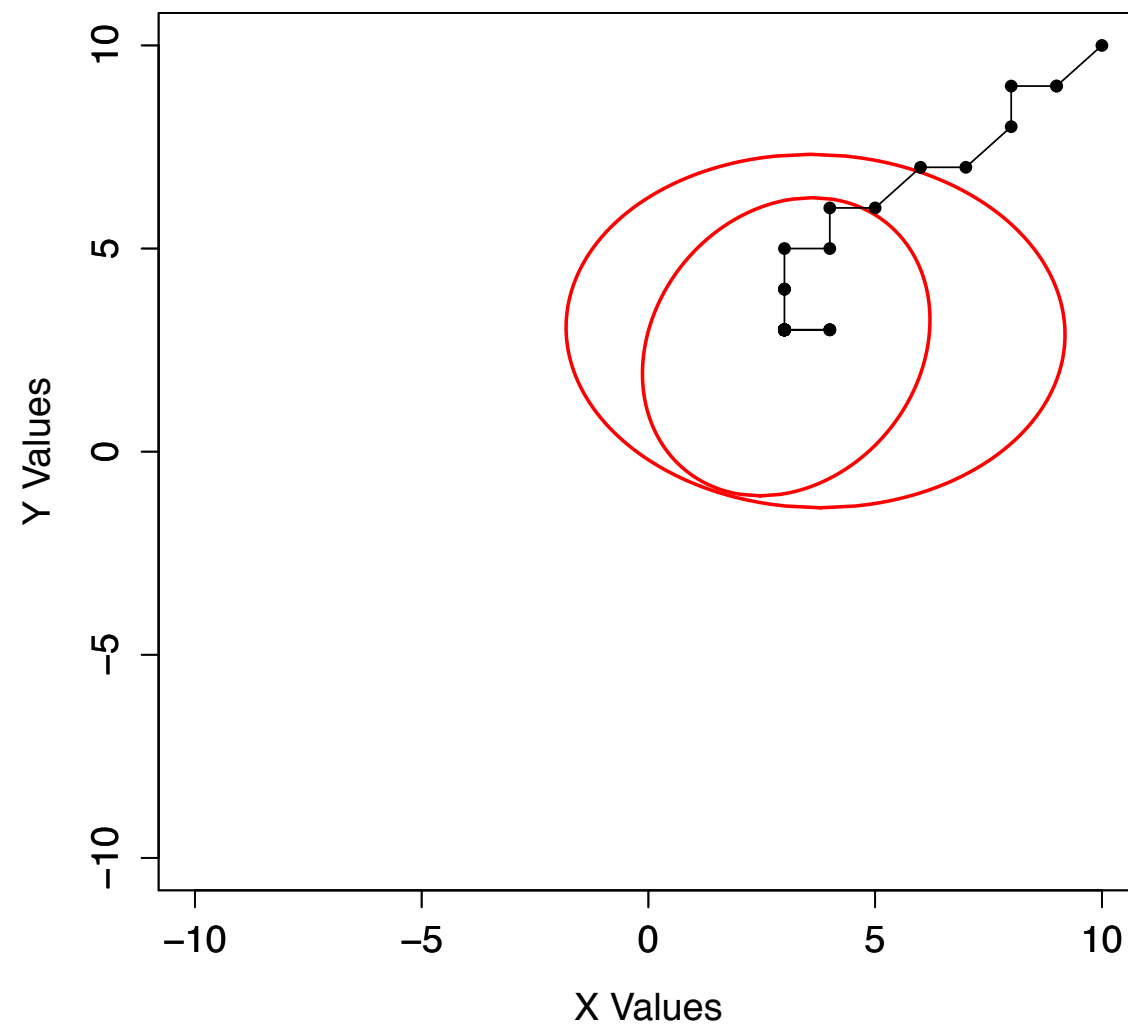
# How Does It Find High Probabilities?

## Some examples

N = 100



N = 100



# Video

<https://chi-feng.github.io/mcmc-demo/>



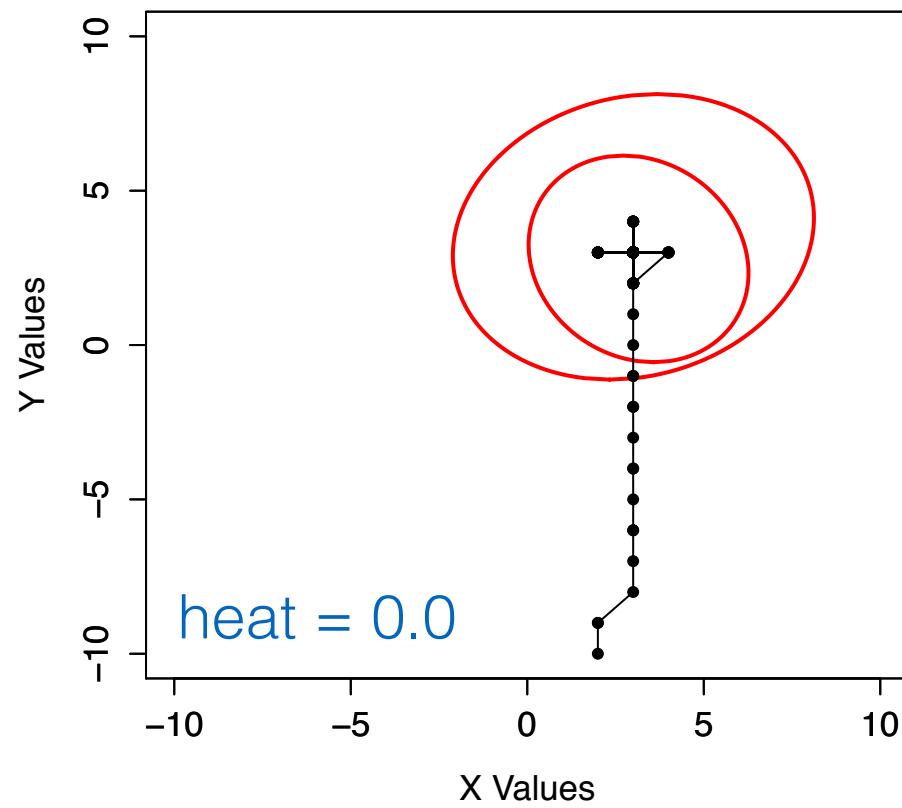
# How Does It Find High Probabilities?

## Heating chains

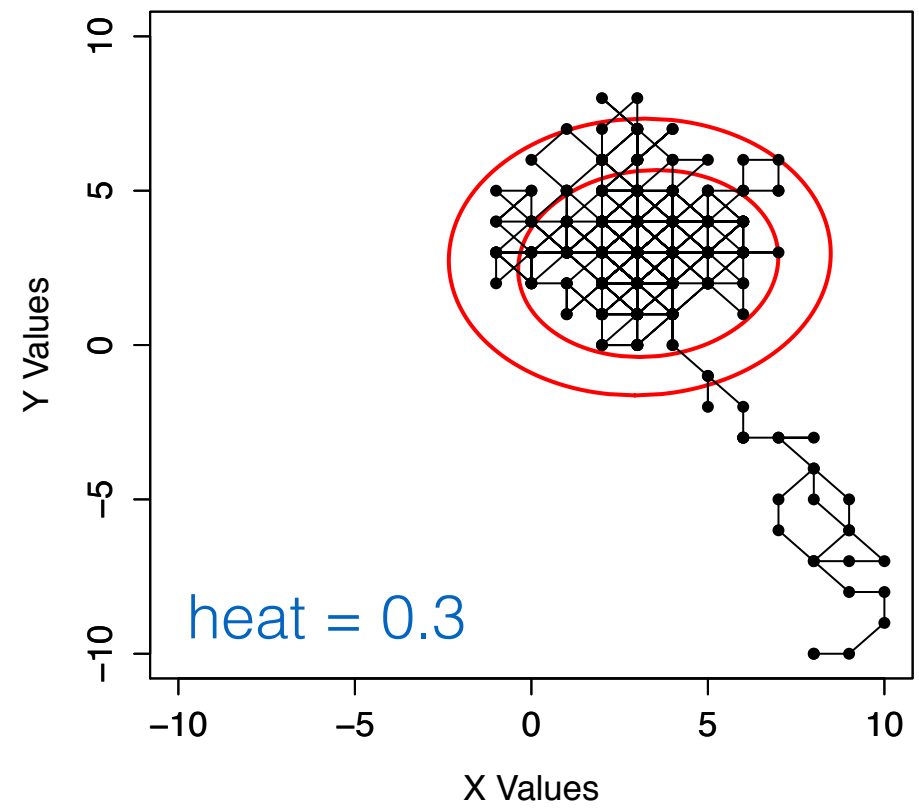
- Can tweak how stringent conditions are for taking a step to a lower likelihood (downhill)
- Chains that more readily take such steps are called “heated”
  - The more heated a chain is, the more readily it will explore parameter space - good
  - The more heated a chain is, the less time it will stay on top of peaks - bad

Need a balance

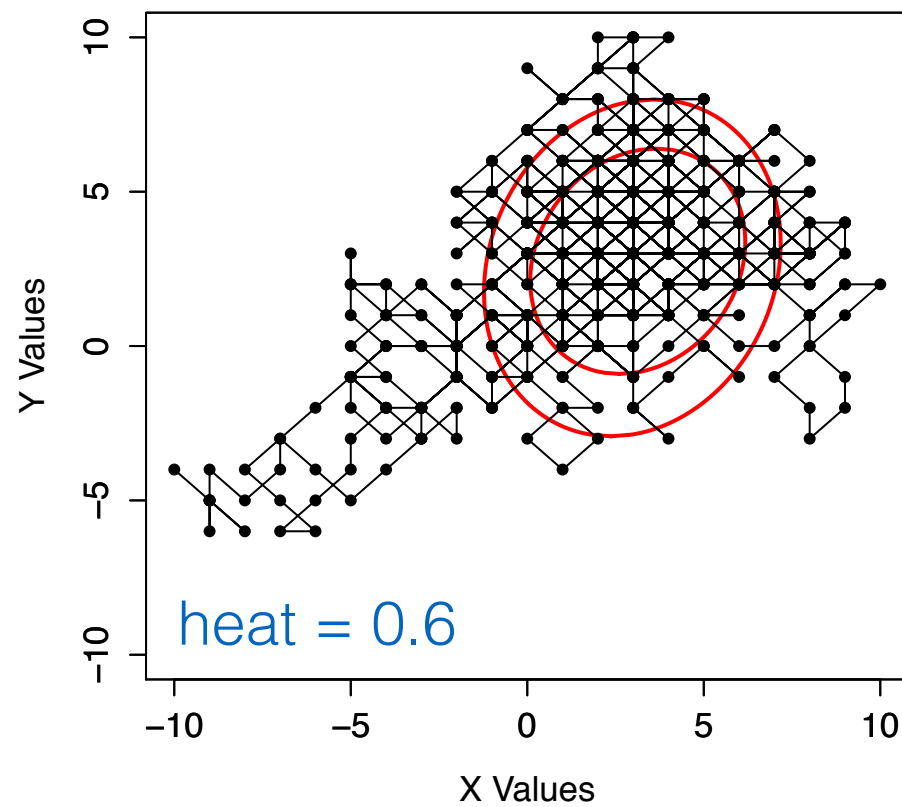
**N = 1000**



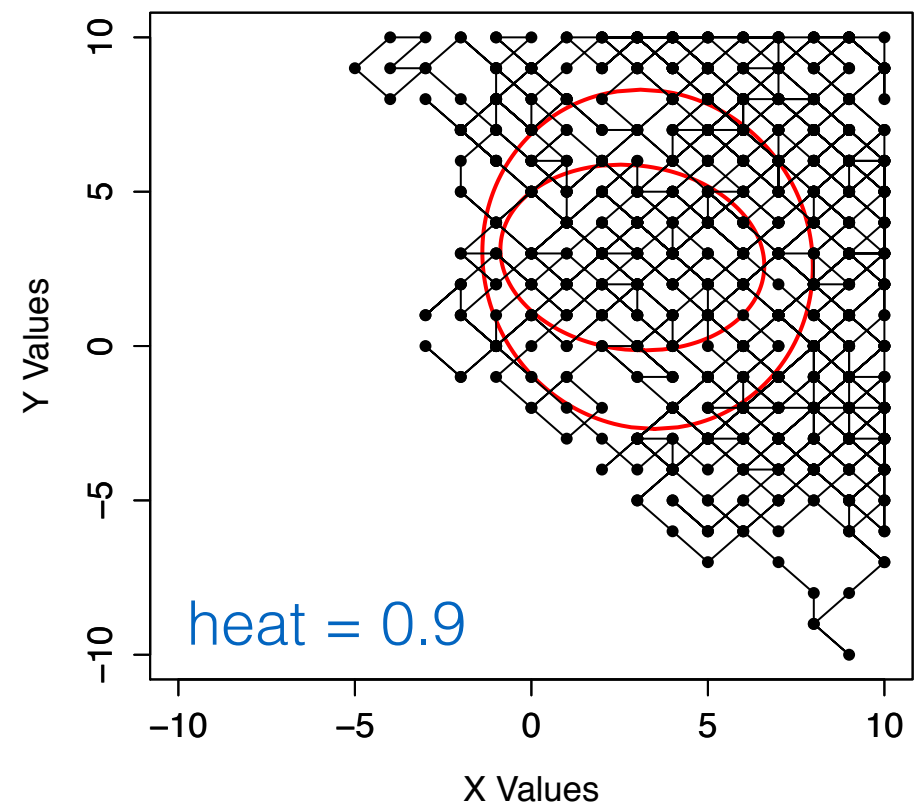
**N = 1000**



**N = 1000**



**N = 1000**



**Your  
Turn!**

# Peak-Finding Exercise 2

1. Need to install the following libraries
  - a. MASS
  - b. cluster
2. Put `onePeakMCMC.R` file in R's working directory
3. Load it into R

```
source( "onePeakMCMC.R" )
```

# Peak-Finding Exercise 2

3. Explore how different starting points influence chain's ability to find peaks. For example, repeat the command below a number of times. (remember the  $\uparrow$  functionality). Try other chain lengths too!

```
onePeakMCMC(nSteps = 100, heat = 0, burnin = 0)
```

# Peak-Finding Exercise 2

4. Explore how different heating strategies influence how readily the chain steps away from the peak. For example:

```
onePeakMCMC(nSteps = 1000, heat = 0.0, burnin = 0)
onePeakMCMC(nSteps = 1000, heat = 0.2, burnin = 0)
onePeakMCMC(nSteps = 1000, heat = 0.4, burnin = 0)
onePeakMCMC(nSteps = 1000, heat = 0.6, burnin = 0)
onePeakMCMC(nSteps = 1000, heat = 0.8, burnin = 0)
onePeakMCMC(nSteps = 1000, heat = 1.0, burnin = 0)
```

Try some of your own!

**What does this have to  
do with Bayesian analysis?**

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

- Remember Bayes' Rule

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$



# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

- Remember Bayes' Rule

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

Unsolvable for most  
problems

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

- Remember Bayes' Rule

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

- If MCMC model is exploring space well:
  - Proportion of time spent at a value represents the posterior probability of that value

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

1. Propose new value of (H)

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

1. Propose new value of (H)
2. Multiply the prior and likelihood for proposed value

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

1. Propose new value of (H)
2. Multiply the prior and likelihood for proposed value
  - a. If this product is  $\geq$  current value, take step
  - b. If this product is  $\leq$  current value, follow some rules to determine whether or not to take step

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

$$P(H|D) = \frac{P(H) \times P(D|H)}{\int P(H) \times P(D|H)}$$

1. Propose new value of (H)
2. Multiply the prior and likelihood for proposed value
  - a. If this product is  $\geq$  current value, take step
  - b. If this product is  $\leq$  current value, follow some rules to determine whether or not to take step
3. If MCMC model is exploring space well, **proportion** of time spent at a specific value of H will represent it's posterior probability

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

- The benefits of this cannot be overstated
- Allows easy computation of very difficult problems
  - Problems not solvable (or even fathomable) before MCMC & good computing power
- Only downside is that MCMC is computationally “expensive”

# MCMC & Bayesian Analysis

## Like peanut butter & chocolate

- Can easily accommodate problems dealing with 10s or even 100s of parameters
  - As long as likelihood for each can be calculated...
  - ... and model is set up properly

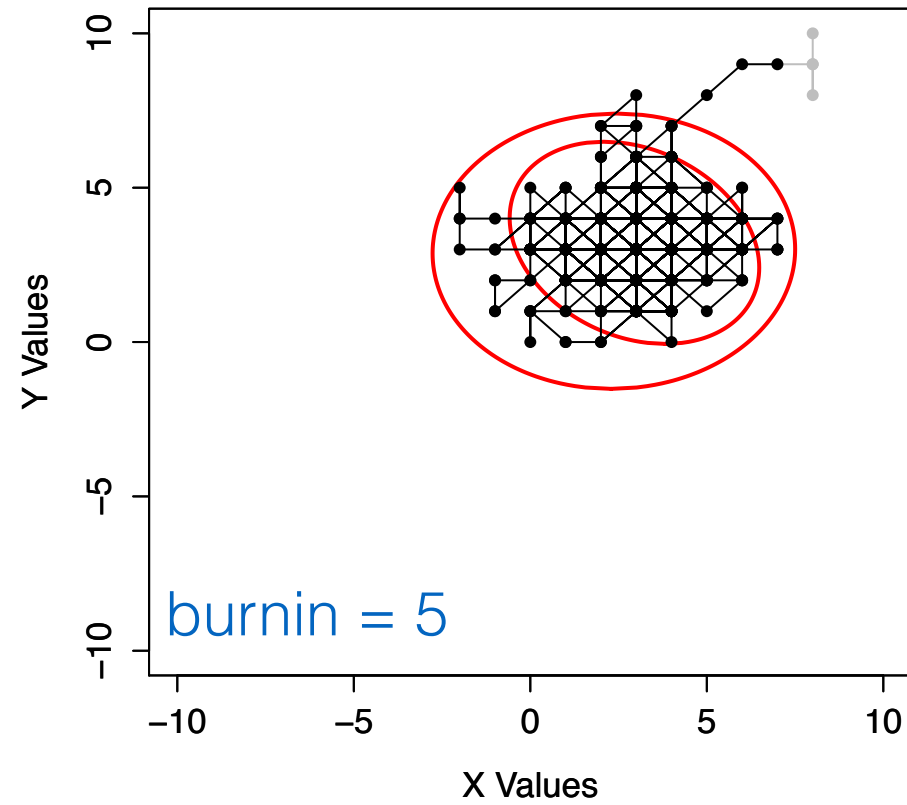


**Burn-in**

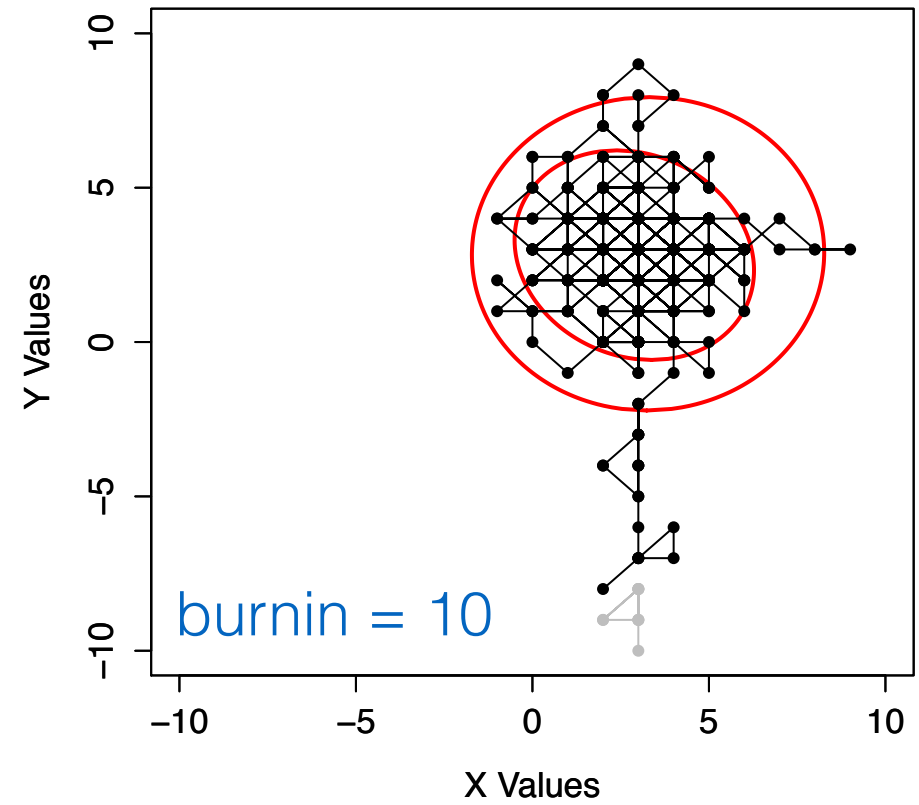
# Burn-in

- To be useful in Bayesian analyses, proportion of time spent at a value should be reflective of that value's probability
- But, remember that the model wanders aimlessly, and is heavily biased by where it starts, until it actually finds a peak
- Need to ignore these steps when making estimation
  - Number of steps ignored is called the "burn-in"
  - No way to know ahead of time, but better to ignore too many than too few

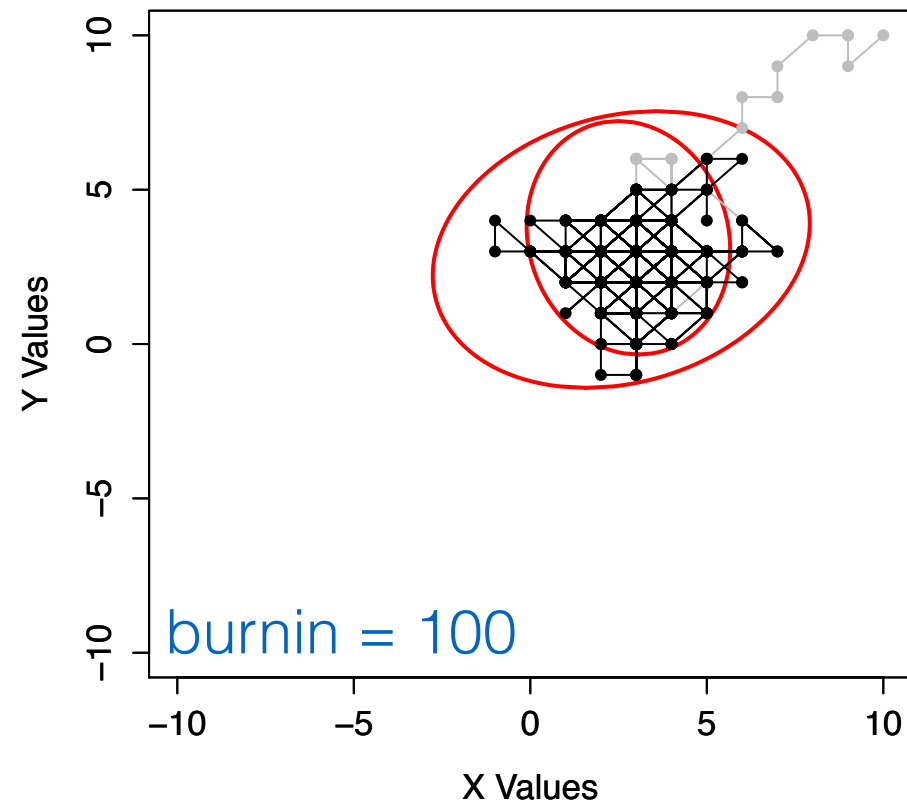
**N = 1000**



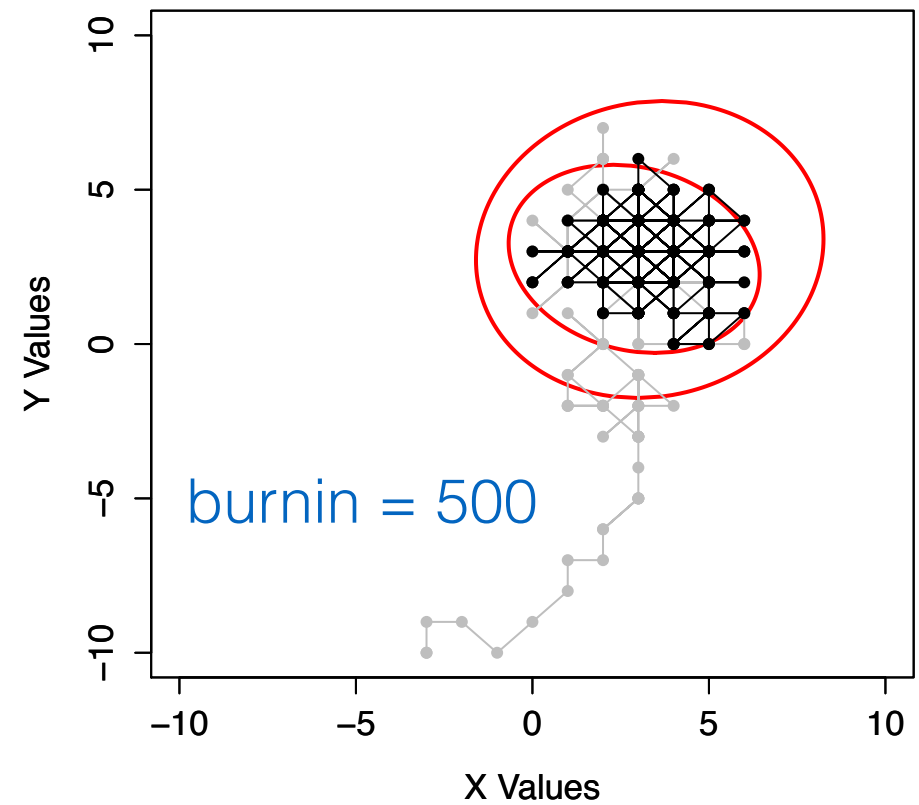
**N = 1000**



**N = 1000**



**N = 1000**



**Your  
Turn!**

# Burn-in Exercise

1. Explore how different burn-in lengths fit with different chain lengths and heating values.

```
onePeakMCMC(nSteps = 1000, heat = 0.5, burnin = 10)  
onePeakMCMC(nSteps = 1000, heat = 0.5, burnin = 50)  
onePeakMCMC(nSteps = 1000, heat = 0.5, burnin = 100)  
etc.
```

Try some of your own!

# Burn-in

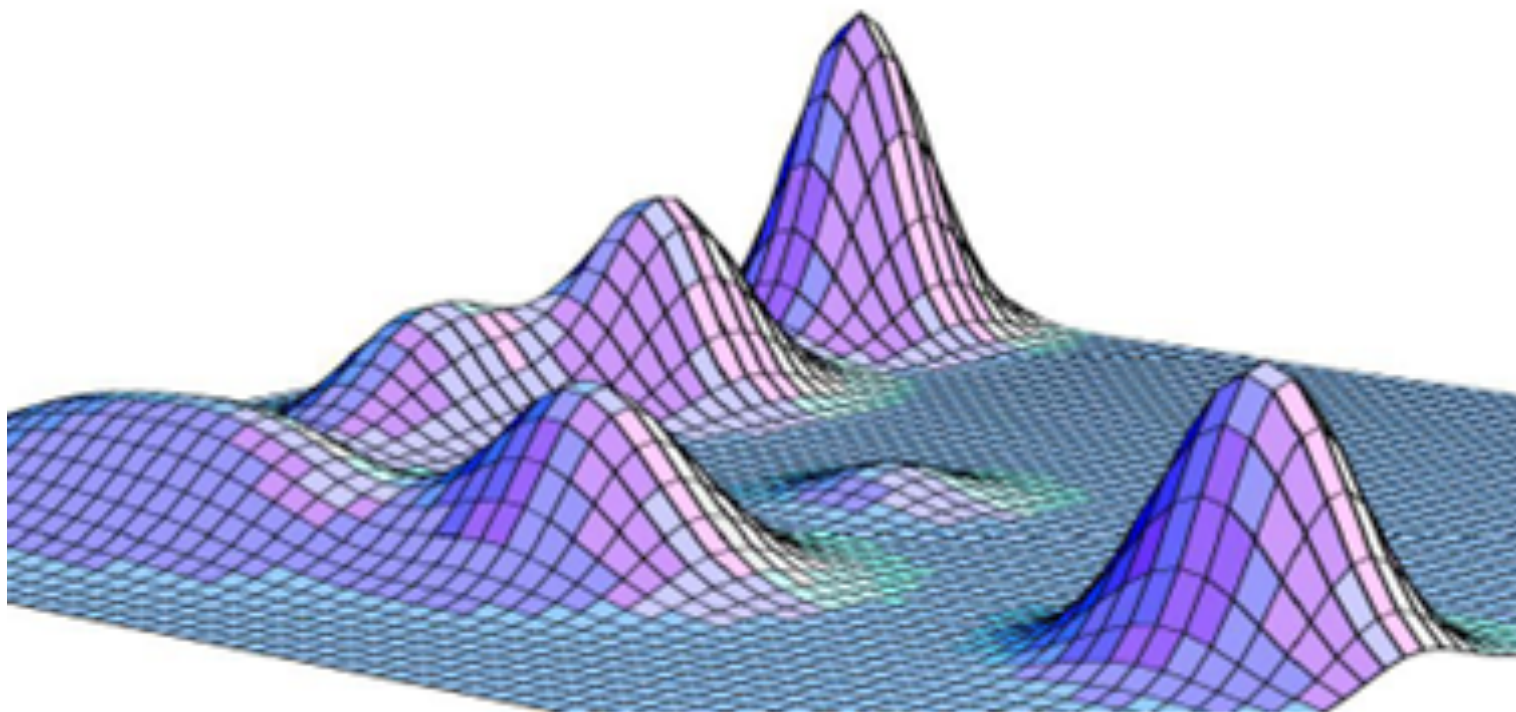
- Not really possible to have burn-in “too long” (but you need enough steps of data collection to obtain good estimates)
- A burn-in that is too short can cause real problems

We'll go over some ways to examine this later

# Running Multiple Chains

# Running Multiple Chains

- We've been talking about very simple examples
- Parameter space for "real" problems are likely quite complex (and unknowingly so)





# Running Multiple Chains

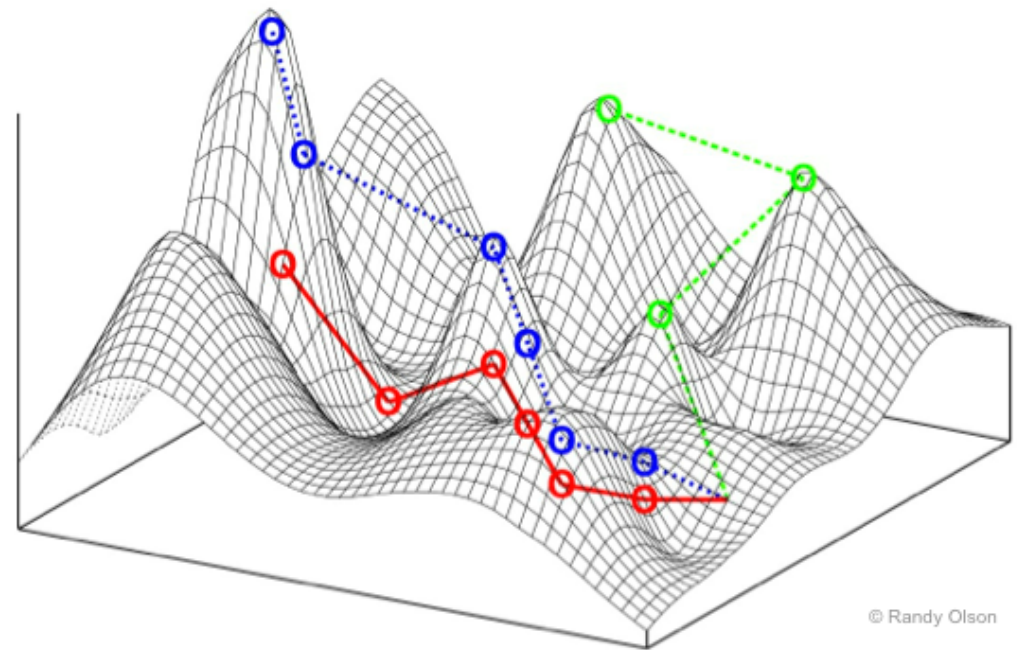
- Often more efficient to run multiple chains
  - Each starting in a different place
  - Each with a different heating value

Will explore parameter space more efficiently than  
a single chain

# Running Multiple Chains

- Have them swap periodically
  - Have one main chain, and the rest “workers”
  - Every so often, compare the probability of positions of each chain
  - If a worker chain has a higher probability than the main chain, switch parameter values between chains

Allows main chain to “jump”  
to peaks that other chains have  
found



# Gibbs Sampling

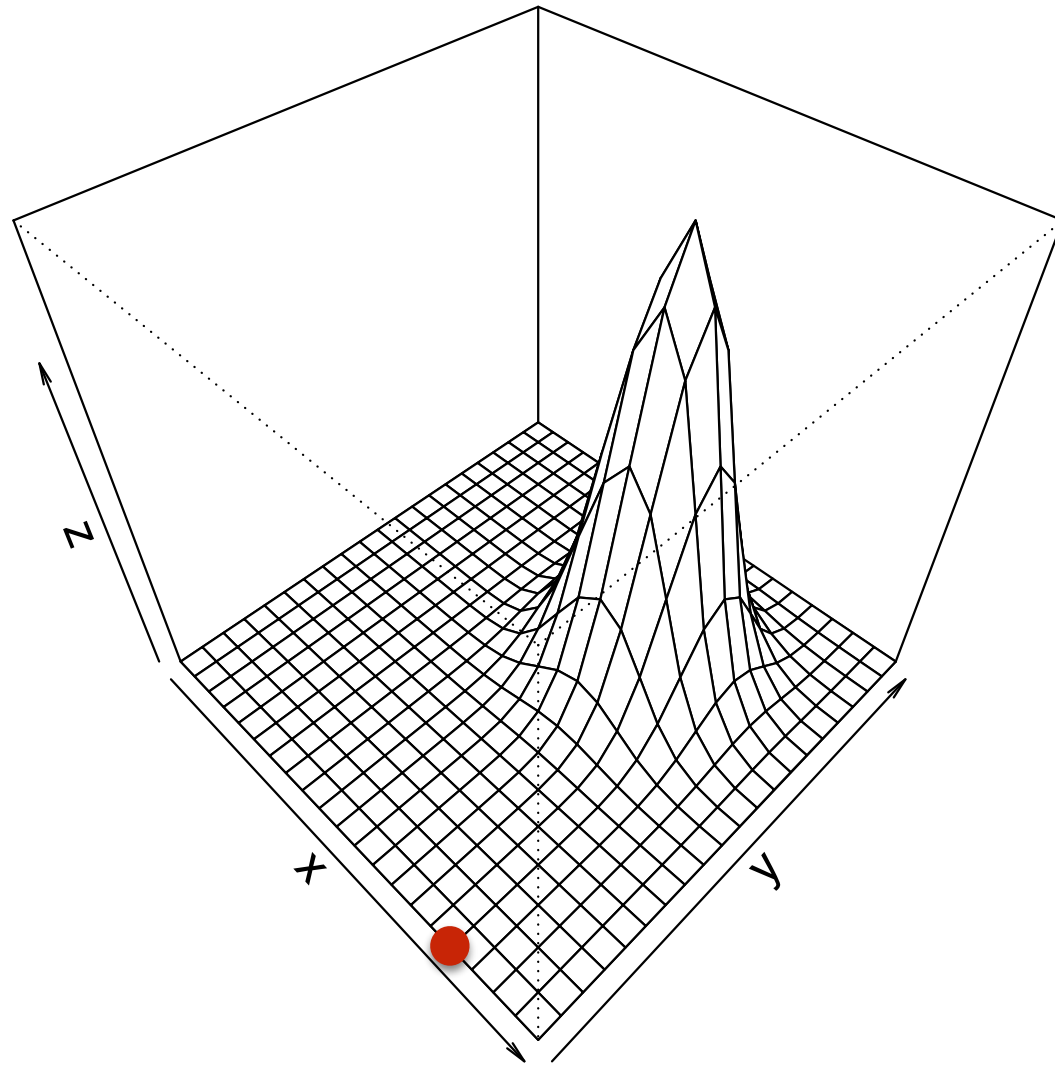
# Gibbs Sampling

- The Metropolis-Hastings algorithm can be very inefficient
  - Many unselected proposals, particularly once on a peak
  - Uses a lot of computing power to not do very much
- Gibbs sampling developed to fix this (just another algorithm for sampling MCMC chains)

# Gibbs Sampling

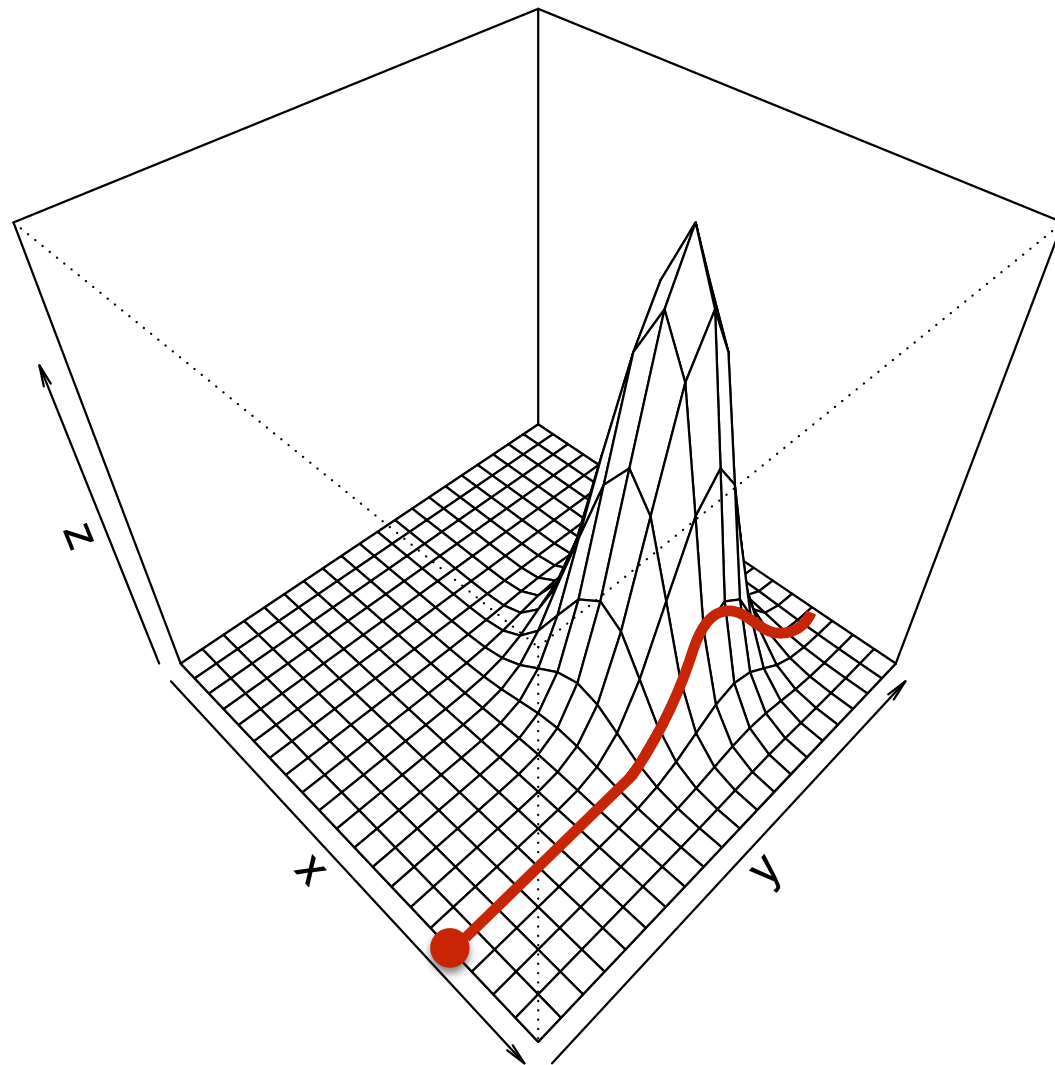
- Once one parameter is updated, calculate associated probabilities **across the range** of the next parameter (x- and y-values in our case)
- Pick the value for that parameter with the highest probability, based on current value for previous parameter
- Repeat across parameters and values
  - **Every** step increases the probability (with exceptions based on heating values and such) so **all steps accepted**

# Gibbs Sampling



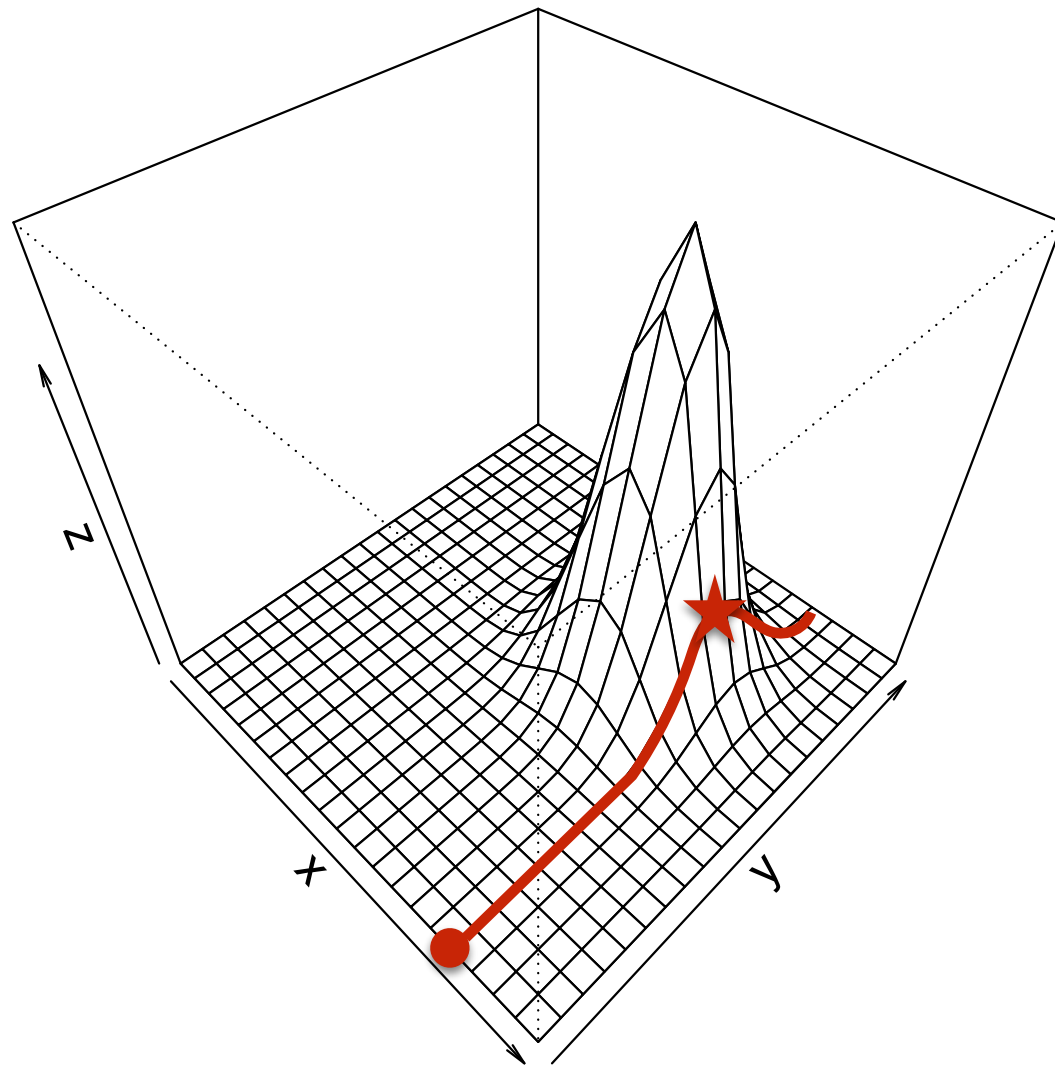
- Pick a starting value for  $x$

# Gibbs Sampling



- Pick a starting value for  $x$
- Calculate range of  $y$  values given that  $x$

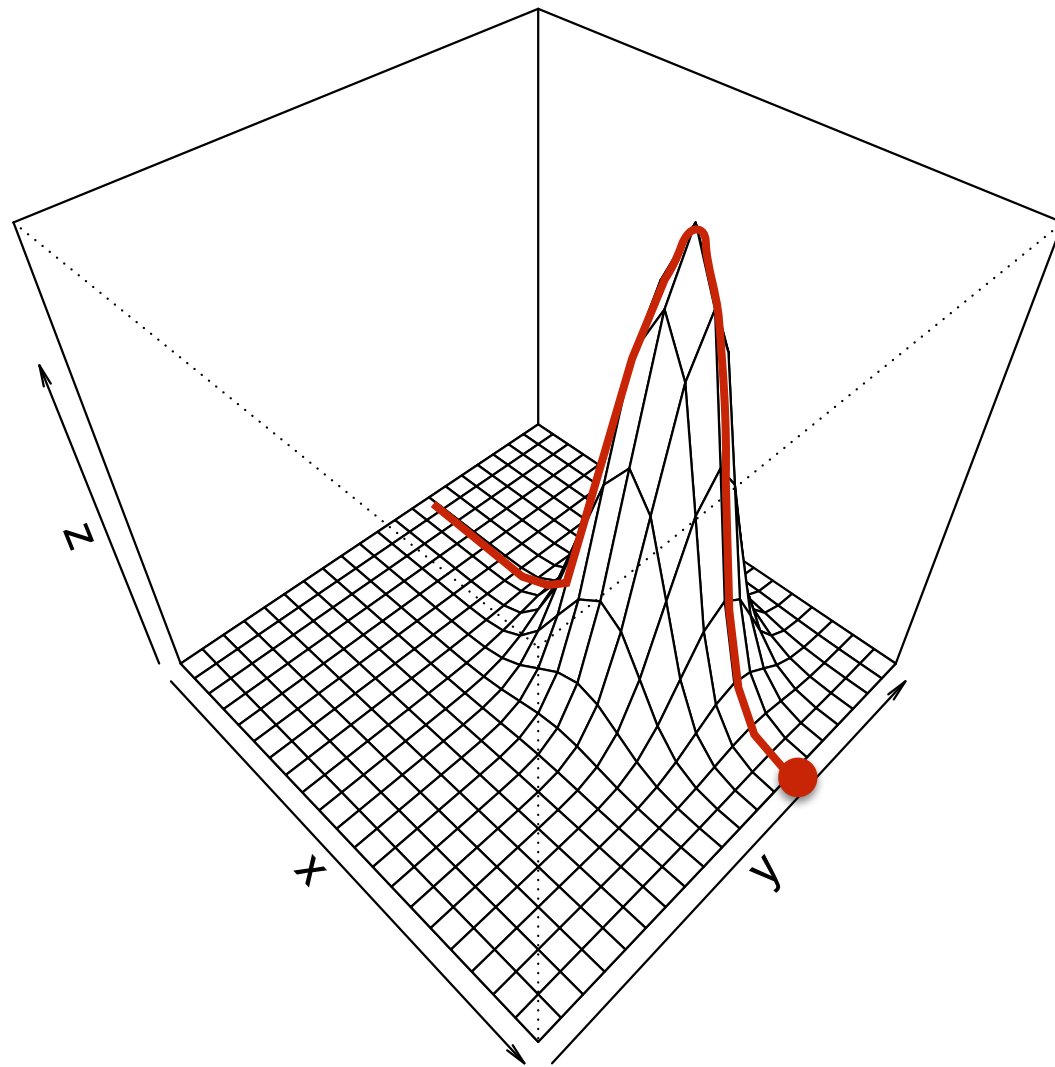
# Gibbs Sampling



- Pick a starting value for  $x$
- Calculate range of  $y$  values given that  $x$
- Find the  $y$  value with the highest probability, and choose that as next value



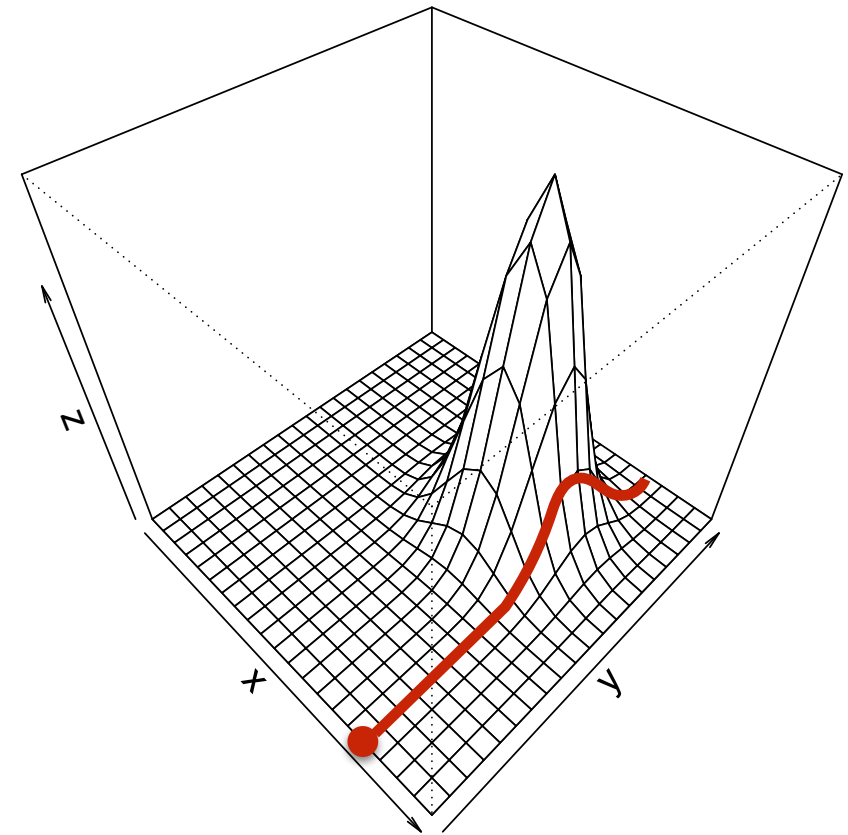
# Gibbs Sampling



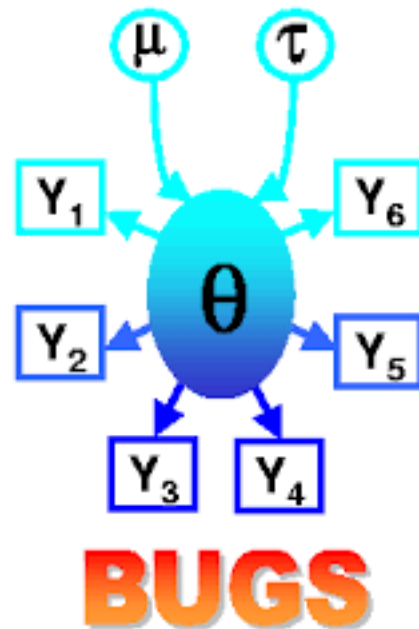
- Pick a starting value for  $x$
- Calculate range of  $y$  values given that  $x$
- Find the  $y$  value with the highest probability, and choose that as next value
- Use that as the next  $y$  value and calculate the corresponding range of  $x$  values
- etc.

# Gibbs Sampling

- In many situations, this will be much more efficient than just Metropolis-Hastings Algorithm
- Primary method in 80s, 90s, and early 2000s

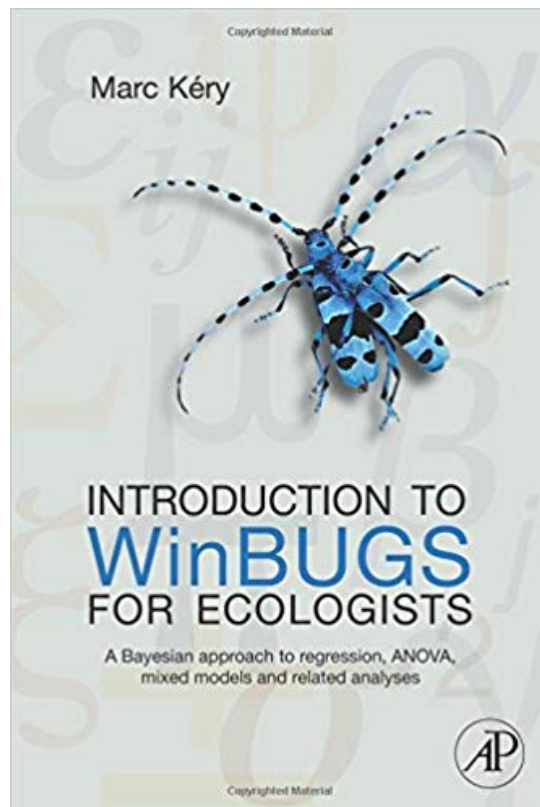


# Gibbs Sampling

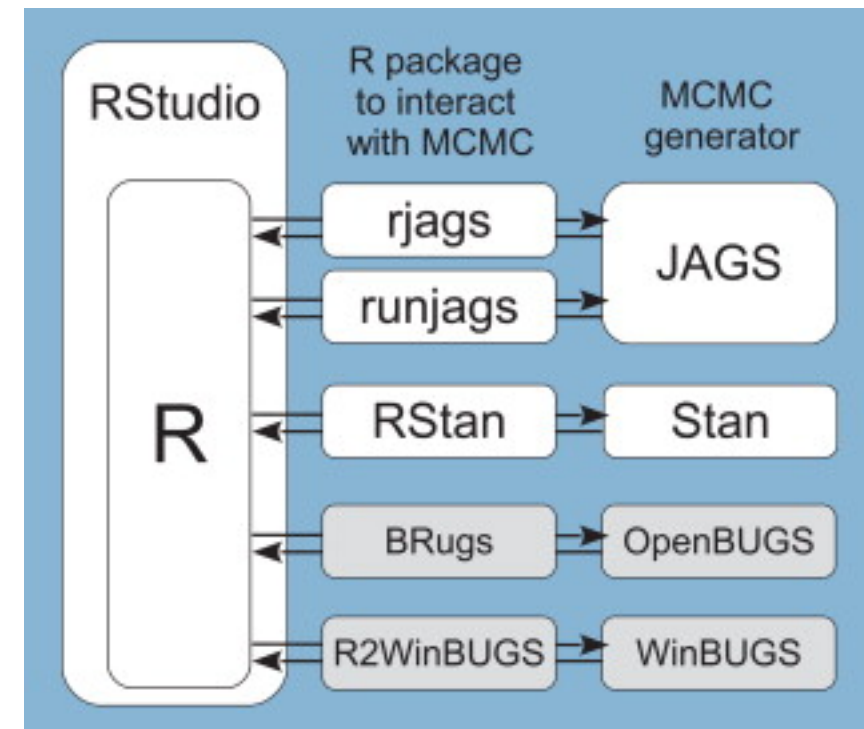


WinBUGS

openBUGS



## JAGS



# Hamiltonian Monte Carlo

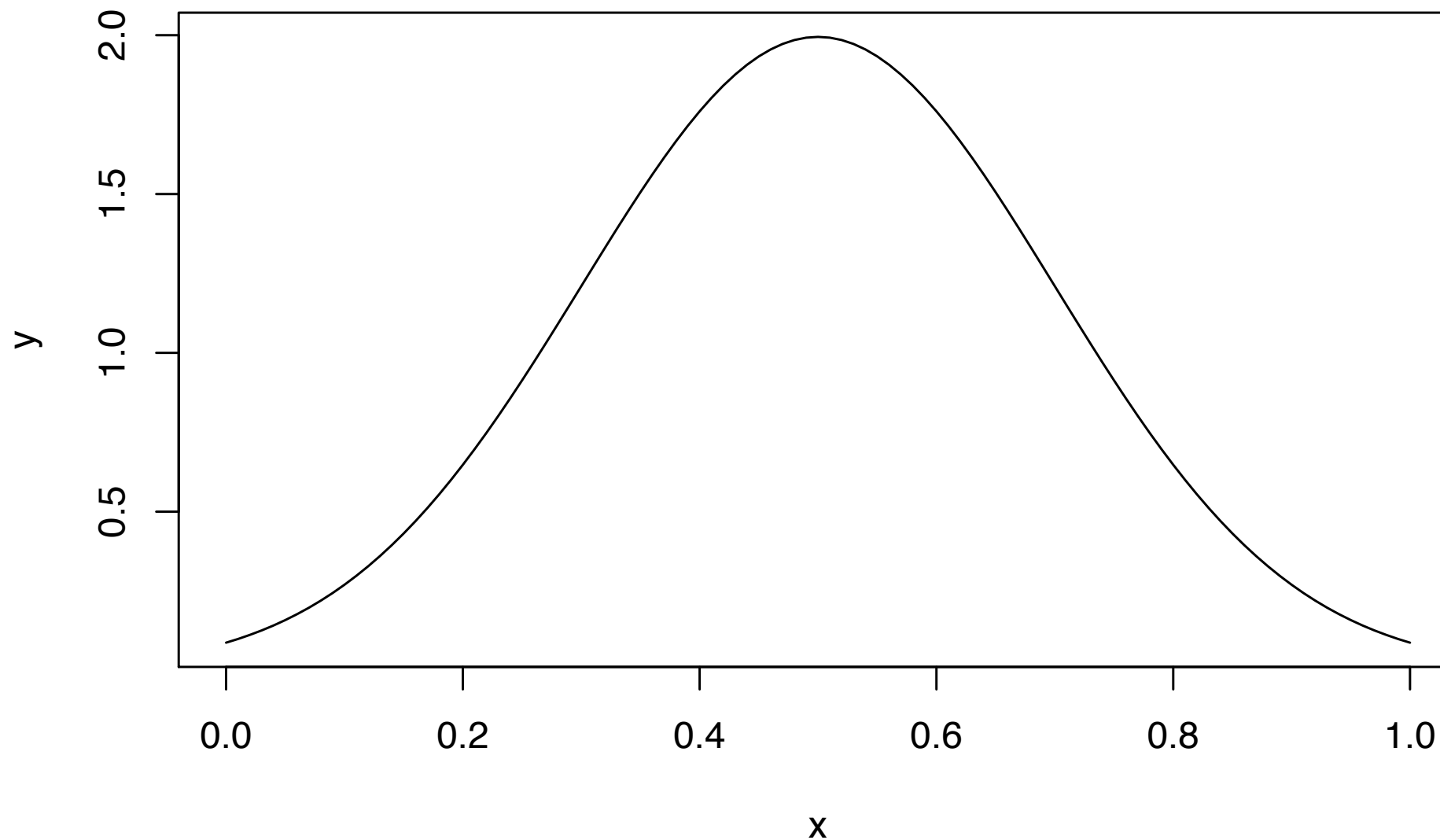


# Hamiltonian Monte Carlo

- A physics simulation
- Represent parameter state as a particle in  $n$ -dimensional space
- Flick particle around frictionless  $-\log$ -posterior
- Record positions

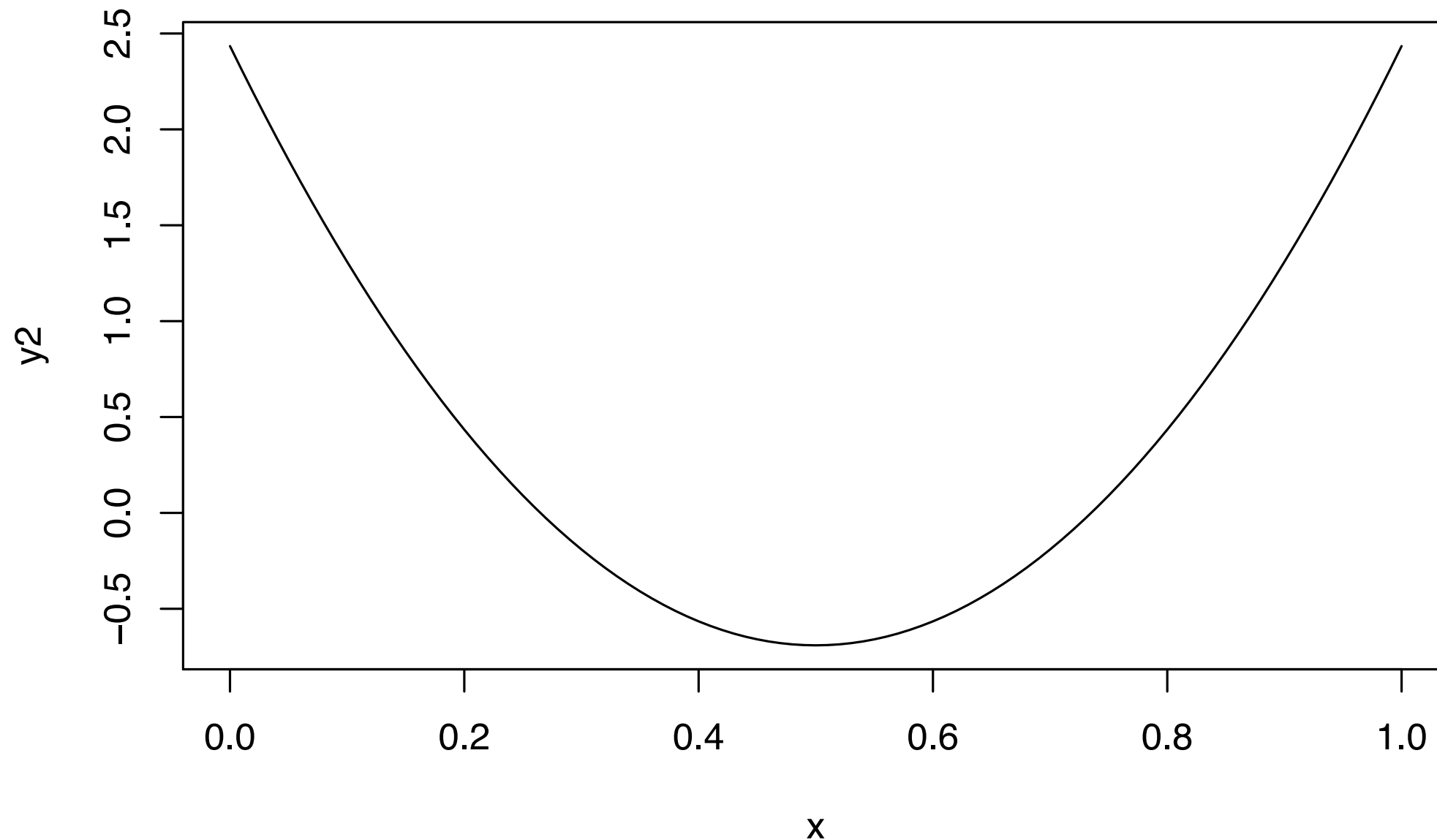
# Hamiltonian Monte Carlo

```
x = seq(from = 0, to = 1, by = 0.01)  
y = dnorm(x, mean = 0.5, sd = 0.2)  
plot(y ~ x, type = "l")
```



# Hamiltonian Monte Carlo

```
y2 = -log(y)  
plot(y2 ~ x, type = "l")
```



# Hamiltonian Monte Carlo



1. Starting position is random draw from priors for each parameter
2. Flick particle (push sled) in a randomly chosen direction, with a randomly selected amount of momentum



# Hamiltonian Monte Carlo



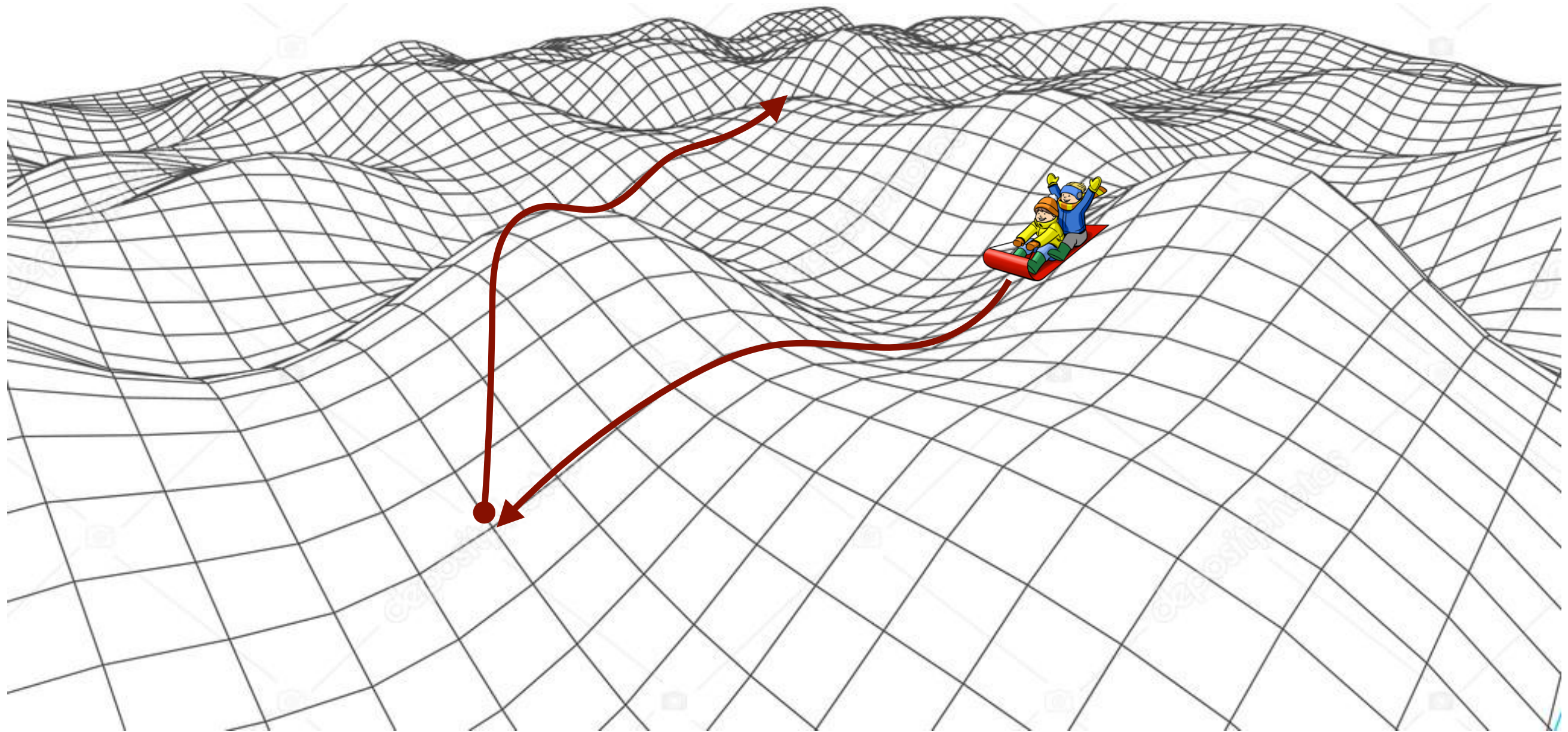
1. Starting position is random draw from priors for each parameter
2. Flick particle (push sled) in a randomly chosen direction, with a randomly selected amount of momentum
3. Calculate probabilities at steps along path to identify local gradient and changes therein
  - Adjust speed and path accordingly

# Hamiltonian Monte Carlo



4. After a pre-determined number of steps\* stop sled and calculate probability (this is proposed step, not any in between)
  - Use algorithm to determine whether or not to take proposed step
5. Repeat many times

# Hamiltonian Monte Carlo



# **Video**

## **(Naïve HMC)**

<https://chi-feng.github.io/mcmc-demo/>

# Hamiltonian Monte Carlo

- When do you take estimate (stop the particle)?
- NUTS: No U-turn sampler
  - Start trajectory in both directions
  - Run until it starts to turn around



## Video

<https://chi-feng.github.io/mcmc-demo/>

# Hamiltonian Monte Carlo

- Does not have *burn-in*
- Has *warmup* instead
  - Stan is optimizing model behaviour for your model (step size, mass of particle, etc.)

**Questions?**