



APSTA-GE 2352

Statistical Computing: Lecture 9

Klint Kanopka

New York University

NYUGreyArtGallery

SEYER CENTER

WASHINGTON St



Table of Contents

1. Statistical Computing - Week 9
 1. Table of Contents
 2. Announcements
 3. A Very Important Problem
2. Markov Chain Monte Carlo
 1. Markov Chain Monte Carlo (MCMC)
 2. Random Walks
 3. Markov Chains
 4. Google PageRank
 5. Dogs with MCMC
3. A (Much) Harder Problem
 1. Plan of Attack
 2. The Traveling Salesman with MCMC
4. Wrap Up
 1. Recap

Announcements

- PS4 is now late
 - Very spooky
 - How's it going?
- PS5 is up!
 - It's long
 - The code you write will take time to run

Check-In

- PolleEv.com/klintkanopka

A Very Important Problem

A Problem

Klint now finds that he can make even more money by selling both hot dogs and actual dogs. He finds that his daily revenue for selling x dozen hot dogs and y dozen actual dogs is:

$$R(x, y) = -5x^2 - 8y^2 - 2xy + 42x + 102y$$

If he can only obtain 10 dozen of either item in a day, what should he prepare to maximize revenue?

Four Potential Solutions:

1. Find an analytic solution (calculus)
2. Reframe as an optimization problem and find a numeric solution (`optim()`)
3. Guess random numbers until you find the right answer (terrible idea)
4. A secret fourth thing!

A Solution We Know!

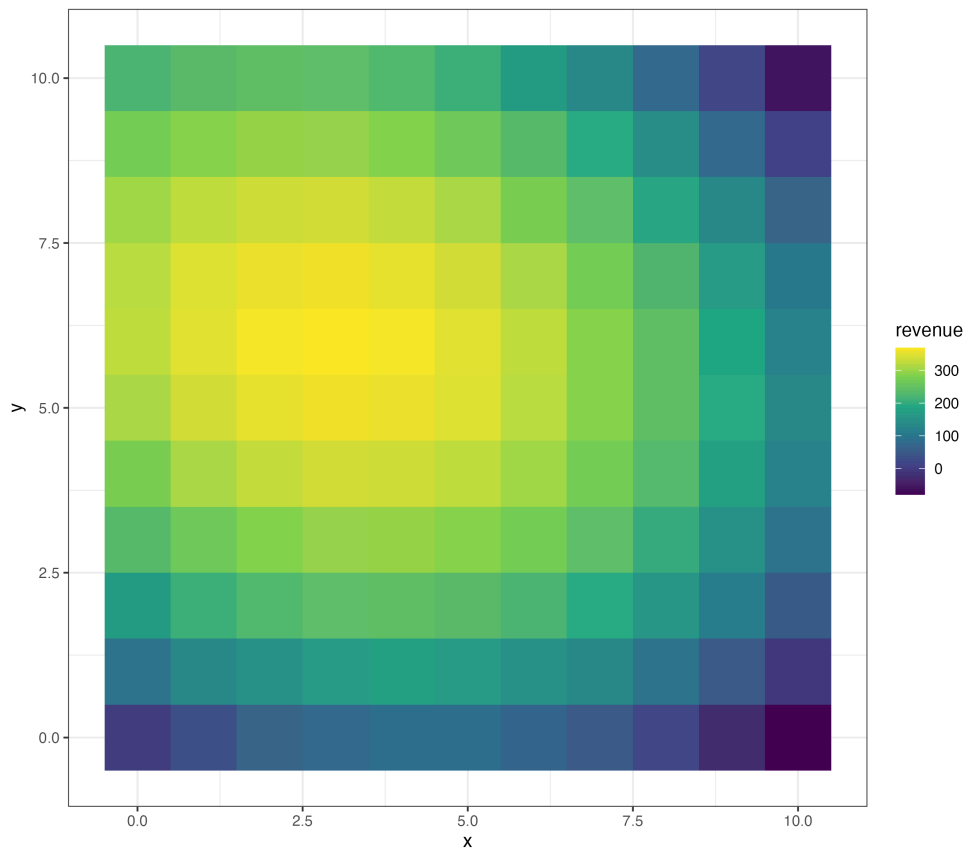
- We can write a function and maximize it to find the best values of x, y
- If you write this function carefully, you can just use `optim()` to spit out an answer

A variety of dogs with `optim()`

```
1 HotDogsDogs <- function(par){
2   x <- par[1]
3   y <- par[2]
4   R <- -5*x^2 -8*y^2 -2*x*y + 42*x + 102*y
5   return(R)
6 }
7
8 hotdogs <- expand.grid(x=0:10, y=0:10, revenue=0)
9
10 for (i in 1:nrow(hotdogs)){
11   hotdogs$revenue[i] <- HotDogsDogs(c(hotdogs$x[i], hotdogs$y[i]))
12 }
```


A variety of dogs with `optim()`

```
1  ggplot(hotdogs,  
2        aes(x = x,  
3            y = y,  
4            fill = revenue)) +  
5    geom_tile() +  
6    scale_fill_viridis_c() +  
7    coord_equal() +  
8    theme_bw()  
9  
10 out <- optim(  
11   c(1, 1),  
12   HotDogsDogs,  
13   lower = c(0, 0),  
14   upper = c(10, 10),  
15   method = "L-BFGS-B",  
16   control = list(fnscale = -1)  
17 )  
18  
19 out$par  
20  
21 # [1] 3 6
```



Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC)

- A super clever algorithm that fits somewhere between direct numerical optimization and just guessing random answers
- Conceptually really kind of weird, but practically pretty straightforward
- **Core idea:** We define a random walk across our search space that places higher probability of landing on “good” answers than “bad” answers. Then we take the walk for a while and find our answer from the places we go

Random Walks

- A process that describes a succession of random steps on some mathematical space
- Imagine a new way to pick a restaurant:
 1. Stand outside your apartment and flip a coin twice
 - If you get H,H, you walk one block north
 - If you get H,T, you walk one block south
 - If you get T,H you walk one block east
 - If you get T,T you walk one block west
 2. Repeat this a whole bunch of times
 3. After some number of steps, eat at the first restaurant you pass
- Sometimes called a “drunkard’s walk”

Markov Chains

- A Markov process is defined by two parts:
 1. A set of states
 2. A transition function that dictates the probability of moving from one state to any other
- Markov processes are “memoryless”
 - Predicting the next state relies only on the current state, not anything before!
 - Called the “Markov Property”
 - Sampling from a Markov process is a sequential process
- A Markov chain is a type of Markov process
 - Nobody really agrees on the exact definition
 - This doesn’t really matter

The Stationary Distribution

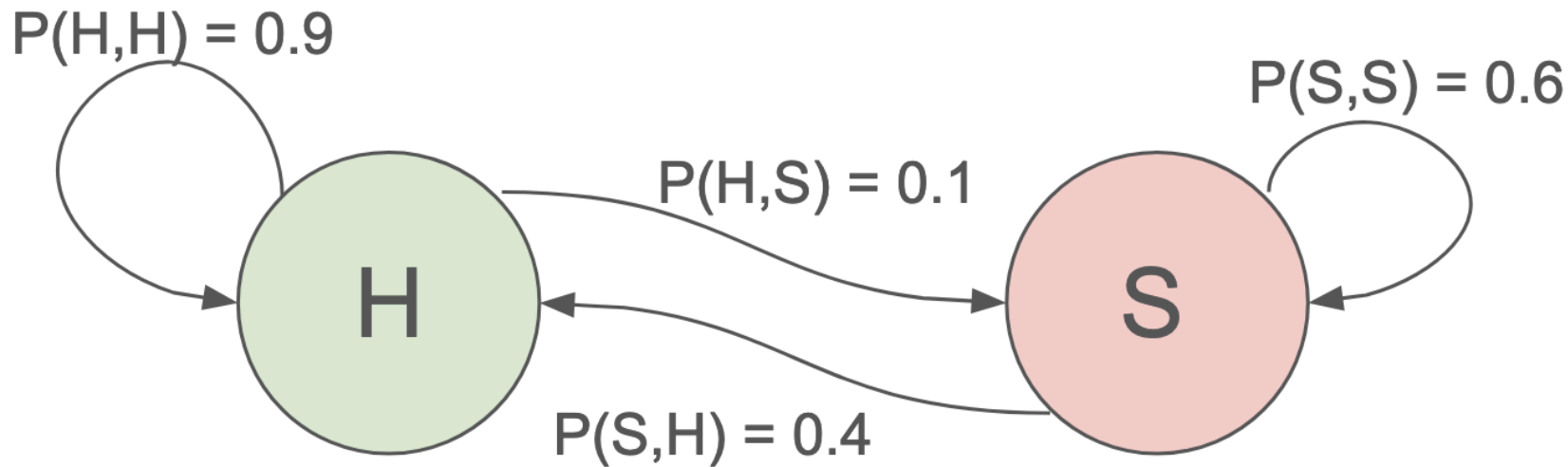
- **The Fundamental Theorem of Markov Chains:** Consider a Markov Chain that satisfies the following two conditions
 1. For all pairs of states s_i, s_j , it is possible to eventually get to state s_j if you start at s_i
 2. The chain is *aperiodic* (satisfied if there are no directed cycles and the states aren't bipartite)
- As the number of samples from the chain grows large, the probability of being in any particular state converges to a *stationary distribution* and this distribution is independent of time and starting state

Getting Sick as a Markov Chain

- Two states: Healthy (H) and Sick (S)
- Transition Probabilities:
 - $P(H, H) = 0.9$
 - $P(H, S) = 0.1$
 - $P(S, H) = 0.4$
 - $P(S, S) = 0.6$

$$T(S_1, S_2) = \begin{bmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{bmatrix}$$

Getting Sick as a Markov Chain



Taking the Walk

- If we want to know what percentage of the time we can expect to be sick, we just start somewhere and take the random walk
- If we sample from the Markov Chain for a while, eventually we converge to the stationary distribution
- The proportion of time we spend in each state is the proportion of time we expect to be in that state given the Markov Chain
- Also gives the probability a random observation is in that location
- Another approach is *power iteration*
 1. Pick some random state vector to start (doesn't matter where you start!)
 2. Multiply by the transition matrix a bunch of times
 3. The state vector converges to the stationary distribution

Google PageRank

- Google is Google because of PageRank
- **Core idea:** Important and relevant webpages have incoming links from important and relevant webpages
- **How to estimate:** Take a random walk through websites and see what sites are visited the most often
- **Core problem:** It's not possible to reach every possible website from every possible starting location—so it doesn't satisfy the fundamental theorem of Markov Chains!
- **Solution:** Teleportation! With some random chance, instead of clicking a link, teleport to a random website
- You'll implement this on a smallish graph in the homework

MCMC Problem Solving Process

1. Define your state space
2. Define the transition function
 - Be sure it can (eventually) make it to every possible state
 - Be sure it is aperiodic (no directed loops)
3. Start somewhere and sample from your Markov Chain a bunch of times
4. Throw away the samples in the beginning (burn-in)
5. Look at what's left

Today's approach will be a slightly simplified version of the Metropolis-Hastings algorithm, which will show up a lot in Bayesian inference, but it's not the only way to MCMC!

Back to Dogs

- We have a pretty easy to optimize function, so MCMC is way less efficient than solutions we already know
- Who cares, let's do it anyway

Setting Up

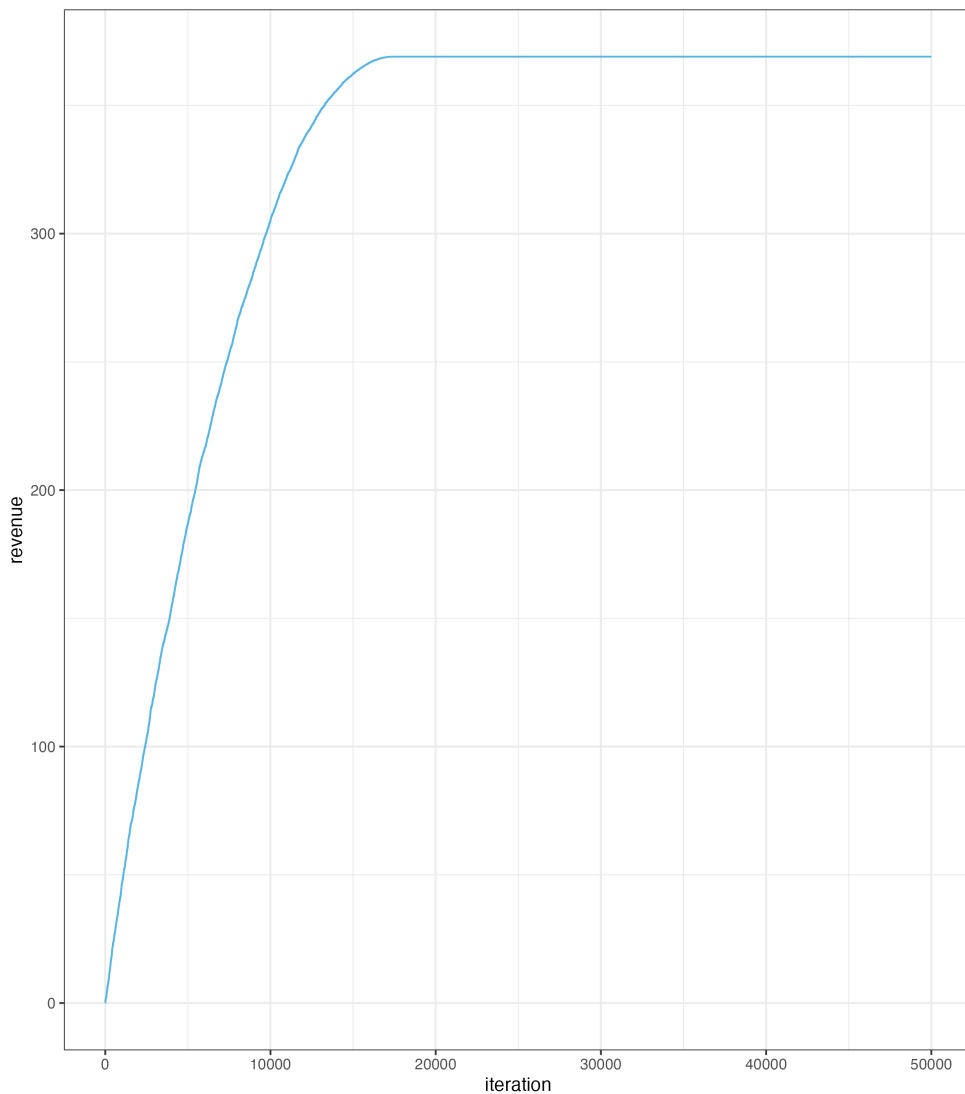
```
1 # First, we optimize the same loss function:
2
3 HotDogsDogs <- function(par){
4   x <- par[1]
5   y <- par[2]
6   R <- -5*x^2 -8*y^2 -2*x*y + 42*x + 102*y
7   return(R)
8 }
9
10 # Next, we propose new parameters:
11
12 ProposeParams <- function(params, lambda){
13   new_params <- params + rnorm(2, mean=0, sd=lambda)
14   return(new_params)
15 }
```

Dogs with MCMC

```
1 N_iter <- 5e4
2
3 revenue <- vector('numeric', length=N_iter)
4 params <- rep(0,2)
5 best <- params
6
7 param_history <- matrix(NA, nrow=N_iter, ncol=1+length(params))
8
9 lambda <- 1e-3
10 temp <- 0.025
11
12 for (i in 1:N_iter){
13   new_params <- ProposeParams(params, lambda)
14   if (HotDogsDogs(new_params) > HotDogsDogs(params) | runif(1) < temp){
15     params <- new_params
16   }
17   if (HotDogsDogs(params) > HotDogsDogs(best)){
18     best <- params
19   }
20   param_history[i,] <- c(i, params)
21   revenue[i] <- HotDogsDogs(params)
22 }
```

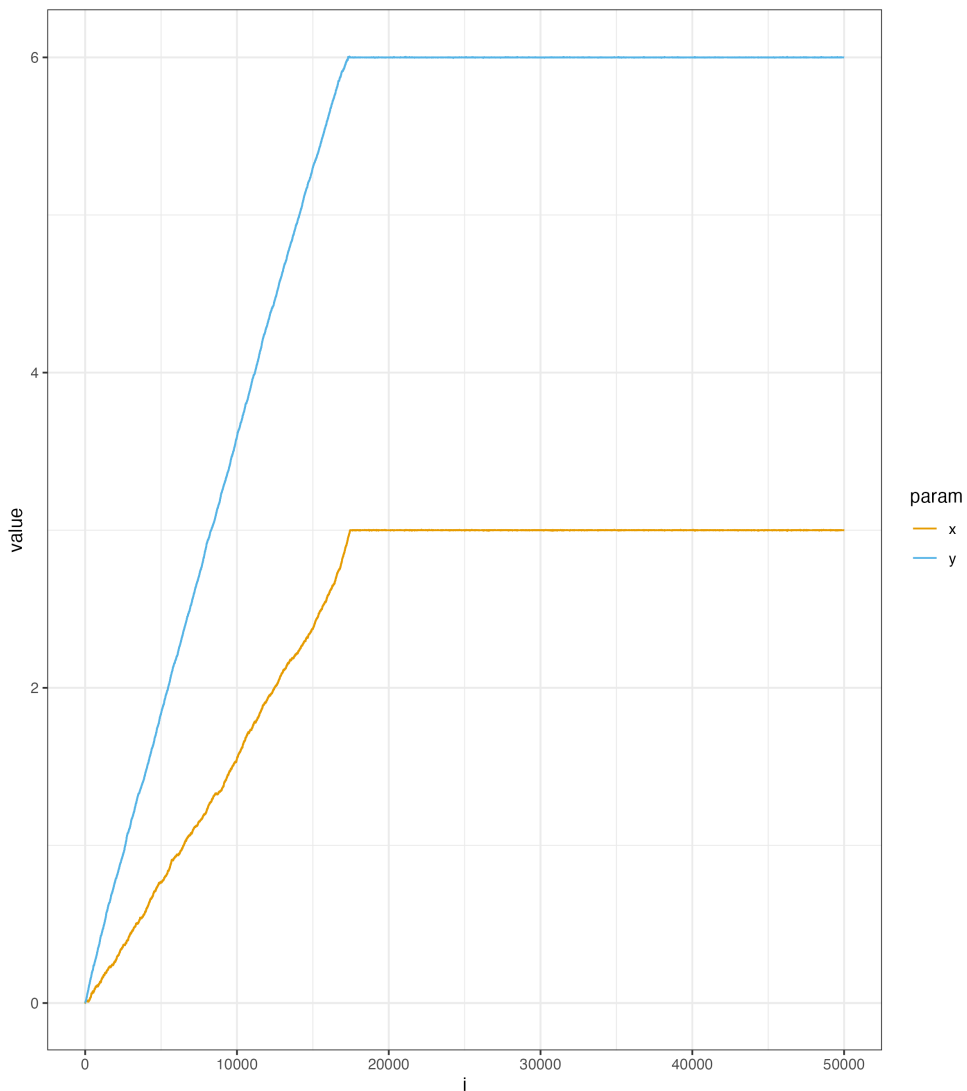

MCMC Results

```
1 conv <- data.frame(  
2   i = 1:N_iter,  
3   revenue=revenue  
4 )  
5  
6 ggplot(conv, aes(x=i, y=revenue)) +  
7   geom_line(color=okabeito_colors(2)) +  
8   labs(x='iteration', y='revenue') +  
9   theme_bw()
```



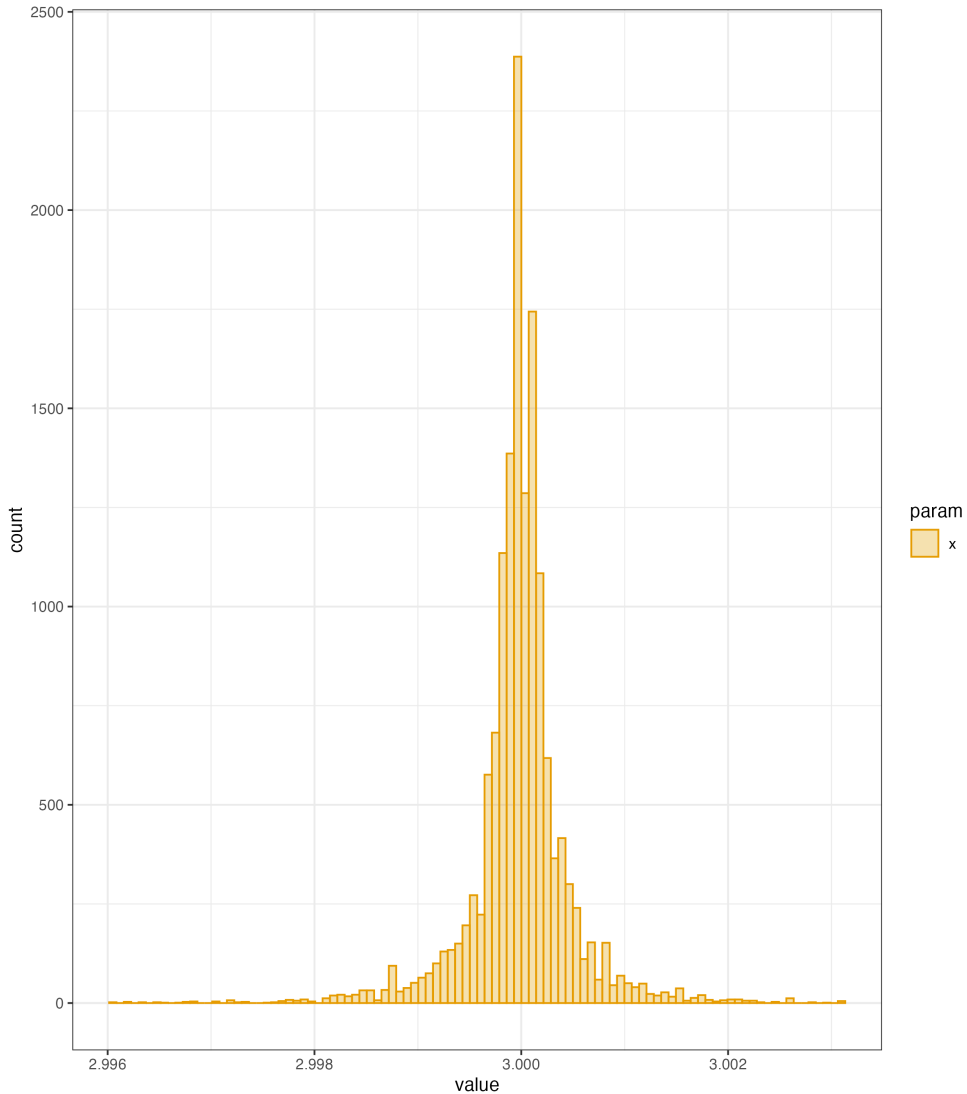
MCMC Results

```
1 param_history <- data.frame(param_history)
2 names(param_history) <- c('i', 'x', 'y')
3
4 param_history |>
5   pivot_longer(-i, names_to='param') |>
6   ggplot(aes(x=i, y=value, color=param)) +
7   geom_line() +
8   scale_color_okabeito() +
9   theme_bw()
```



MCMC Results

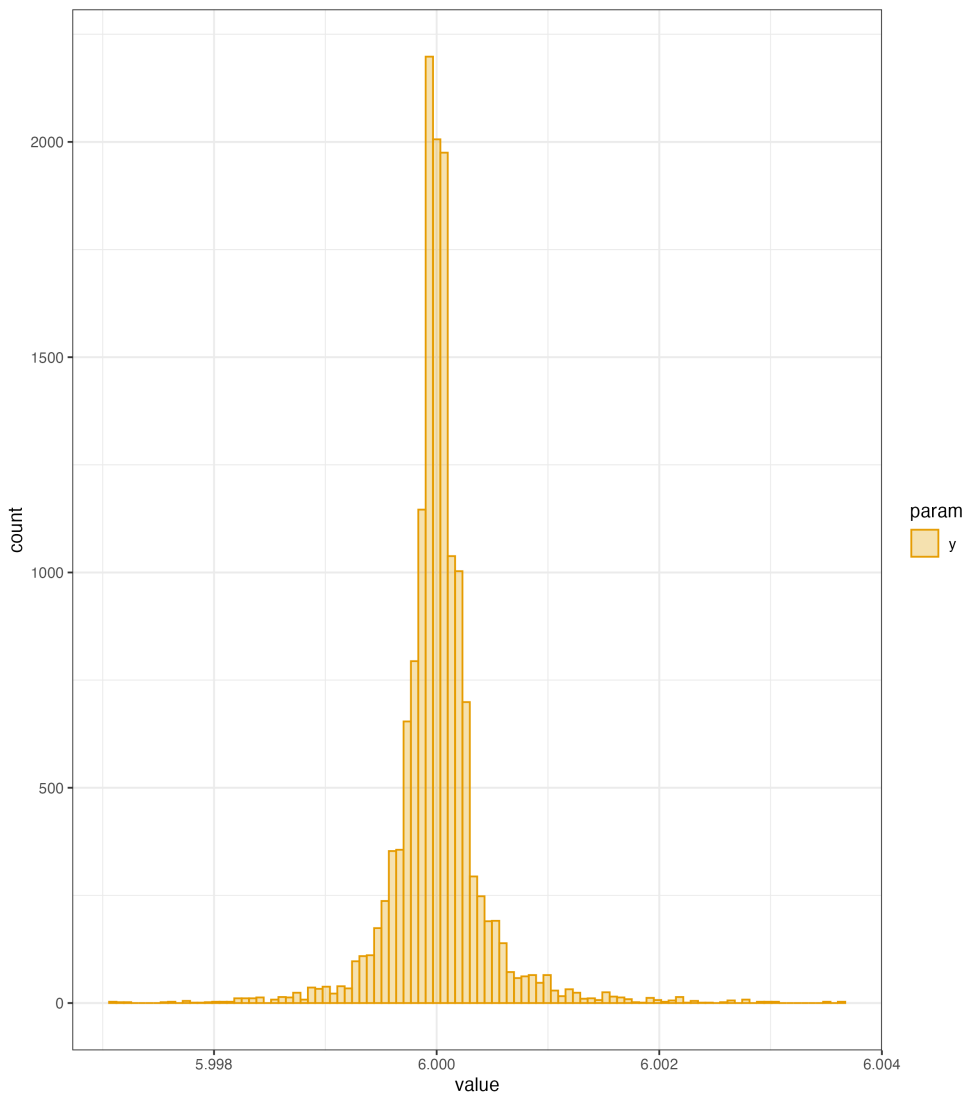
```
1 param_history |>
2   pivot_longer(-i, names_to='param') |>
3   filter(i >= 35000) |>
4   filter(param=='x') |>
5   ggplot(aes(x=value,
6             color=param,
7             fill=param)) +
8   geom_histogram(alpha=0.3, bins=100) +
9   scale_color_okabeito() +
10  scale_fill_okabeito() +
11  theme_bw()
```



MCMC Results

```
1 param_history |>
2   pivot_longer(-i, names_to='param') |>
3   filter(i >= 35000) |>
4   filter(param=='y') |>
5   ggplot(aes(x=value,
6             color=param,
7             fill=param)) +
8   geom_histogram(alpha=0.3, bins=100) +
9   scale_color_okabeito() +
10  theme_bw()

12 colMeans(param_history[35000:N_iter, 2:3])
13
14 #           x           y
15 # 2.999987 6.000010
16
17 best
18
19 # [1] 3.000002 6.000000
```



What About Temperature?

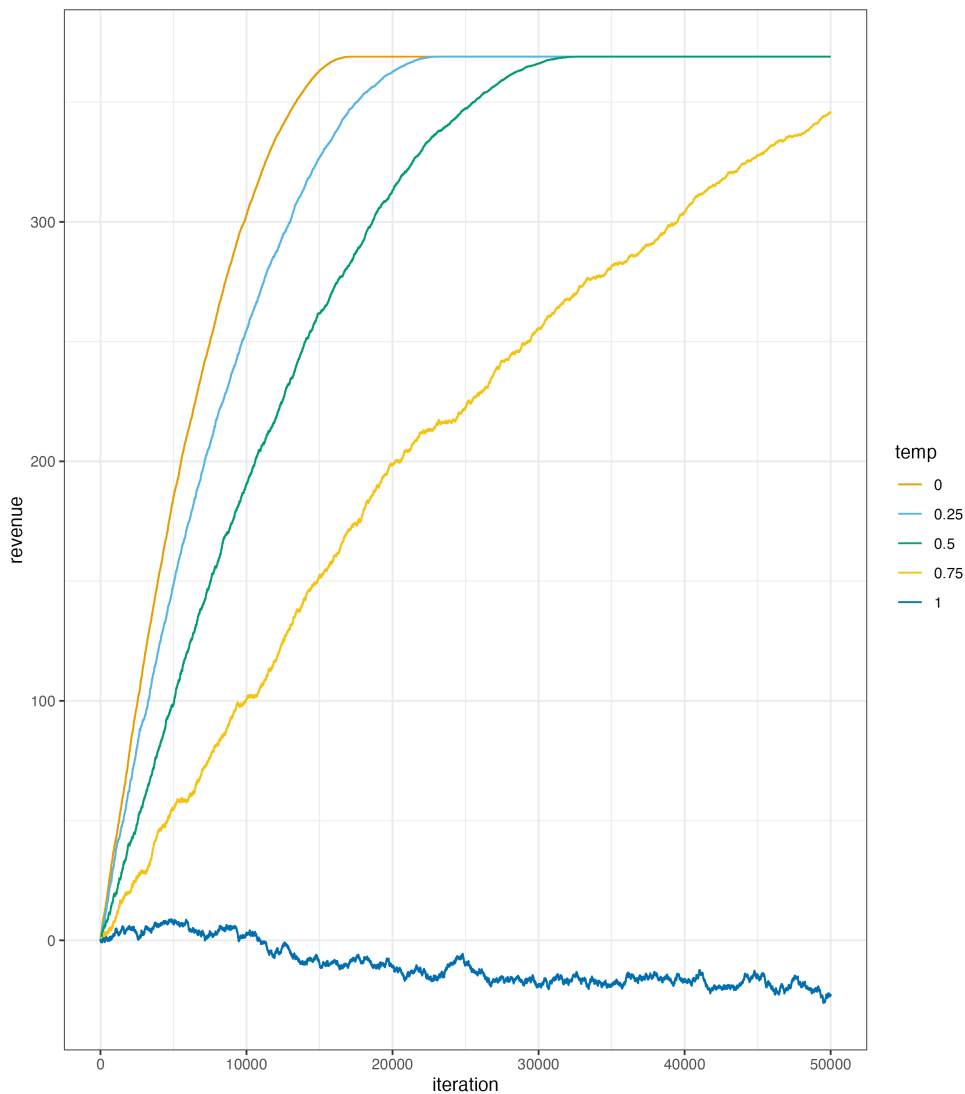
```
1 N_iter <- 5e4
2 lambda <- 1e-3
3 Ts <- seq(from=0, to=1, by=0.25)
4 conv <- best <- T_param_history <- vector('list', length=length(Ts))
5
6 for (j in seq_along(Ts)){
7   revenue <- vector('numeric', length=N_iter)
8   params <- rep(0,2)
9   best[[j]] <- params
10  param_history_tmp <- matrix(NA, nrow=N_iter, ncol=2+length(params))
11  for (i in 1:N_iter){
12    new_params <- ProposeParams(params, lambda)
13    if (HotDogsDogs(new_params) > HotDogsDogs(params) | runif(1) < Ts[j]){
14      params <- new_params
15    }
16    if (HotDogsDogs(params) > HotDogsDogs(best[[j]])){
17      best[[j]] <- params
18    }
19    param_history_tmp[i,] <- c(i, Ts[j], params)
20    revenue[i] <- HotDogsDogs(params)
21  }
22  conv[[j]] <- data.frame(i = 1:N_iter, revenue=revenue, T=Ts[j])
23  T_param_history[[j]] <- param_history_tmp
24 }
```

MCMC Results

```
1 conv <- do.call('rbind', conv)
2 T_param_history <- do.call('rbind', T_param_history)
3 T_param_history <- data.frame(T_param_history)
4 names(T_param_history) <- c('i', 'T', 'x', 'y')
5
6 best
7
8 # [[1]]
9 # [1] 3.000006 6.000006
10
11 # [[2]]
12 # [1] 2.999985 6.000006
13
14 # [[3]]
15 # [1] 3.000041 6.000000
16
17 # [[4]]
18 # [1] 1.870241 4.682416
19
20 # [[5]]
21 # [1] 0.16900318 0.01831097
```

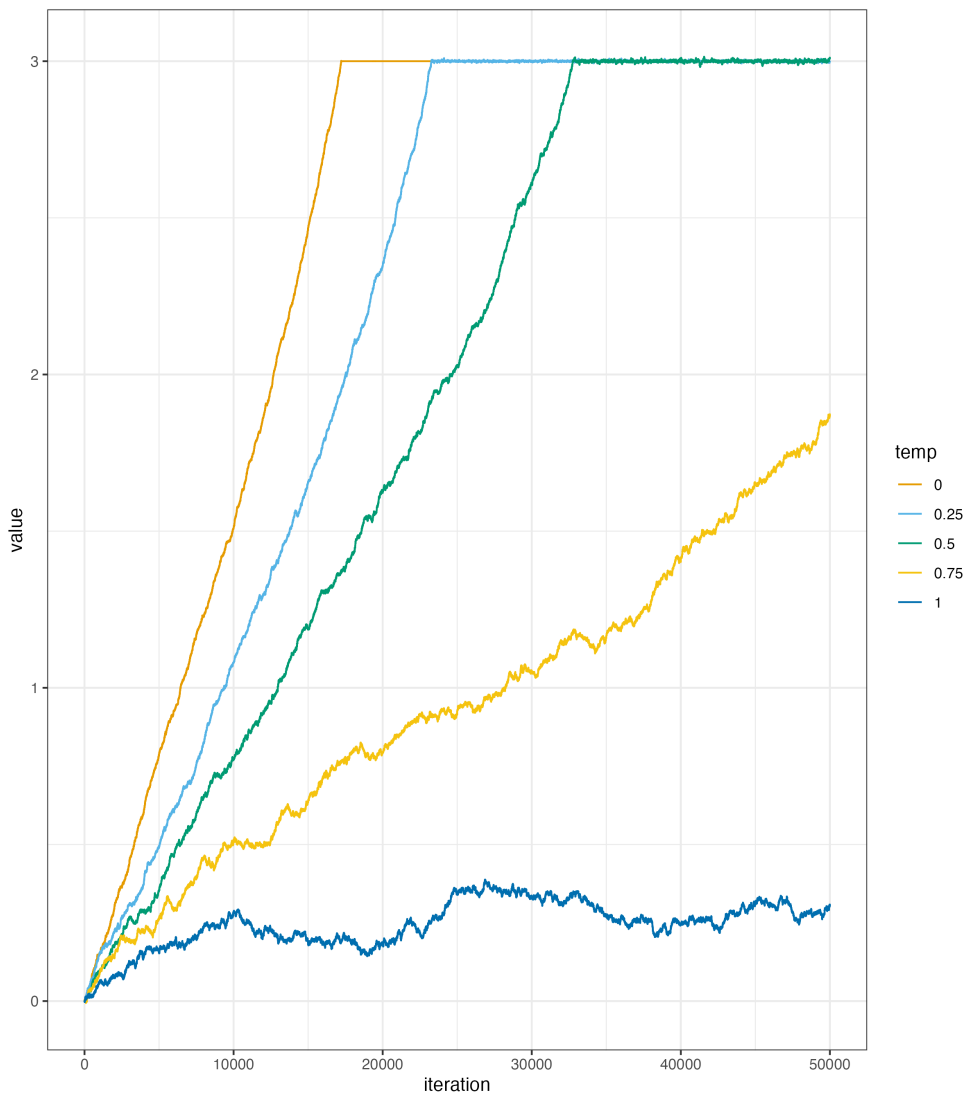
MCMC Results

```
1  ggplot(conv,
2      aes(x=i,
3          y=revenue,
4          color=as.character(T))) +
5  geom_line() +
6  labs(x='iteration',
7      y='revenue',
8      color='temp') +
9  scale_color_okabeito() +
10 theme_bw()
```



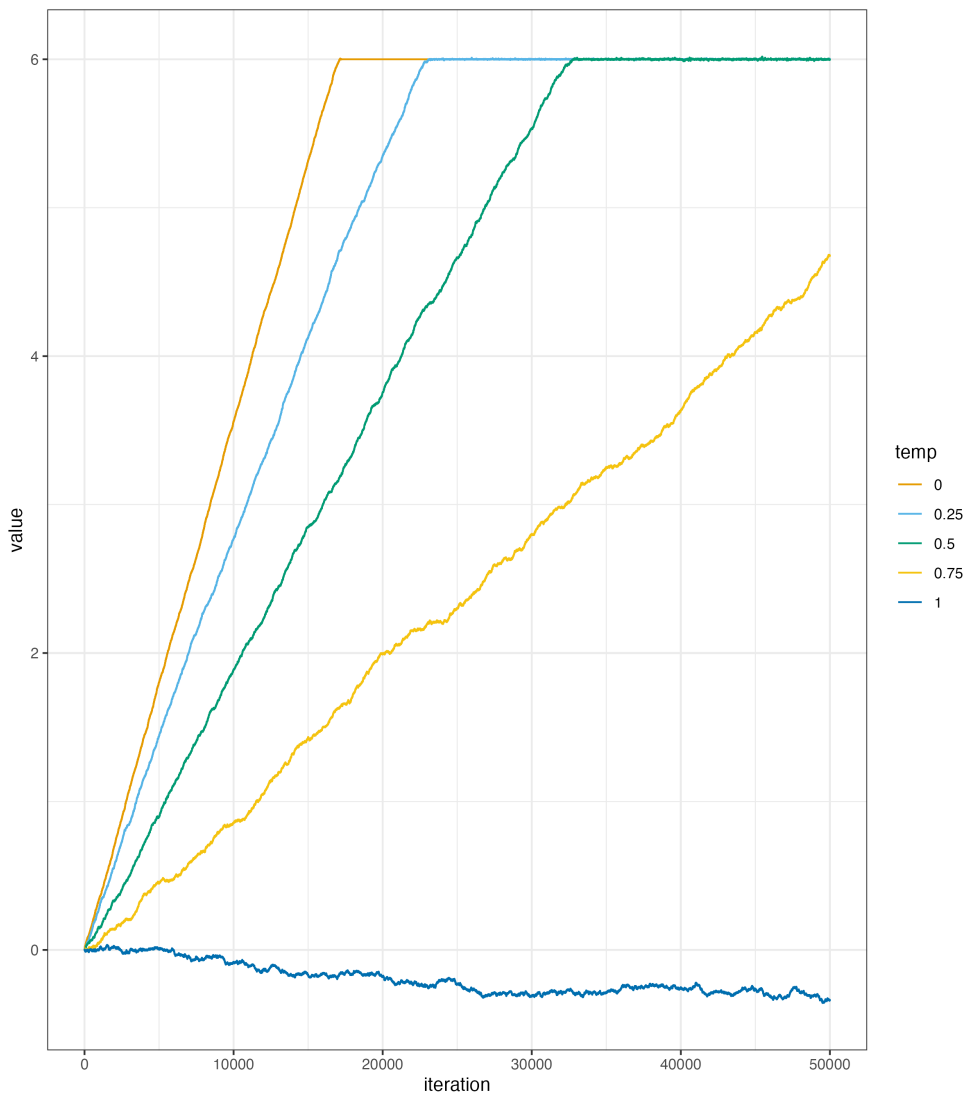
MCMC Results

```
1 T_param_history |>
2   pivot_longer(c(x,y),
3                 names_to='param',
4                 values_to='value') |>
5   filter(param == 'x') |>
6   ggplot(aes(x=i,
7              y=value,
8              color=as.character(T))) +
9   geom_line() +
10  labs(x='iteration', color='temp') +
11  scale_color_okabeito() +
12  theme_bw()
```



MCMC Results

```
1 T_param_history |>
2   pivot_longer(c(x,y),
3                 names_to='param',
4                 values_to='value') |>
5   filter(param == 'y') |>
6   ggplot(aes(x=i,
7              y=value,
8              color=as.character(T))) +
9   geom_line() +
10  labs(x='iteration', color='temp') +
11  scale_color_okabeito() +
12  theme_bw()
```



A (Much) Harder Problem

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

- Known as the “Traveling Salesman Problem” (TSP)
- NP-Hard Problem
- We think (but have not proven) that there are no polynomial-time algorithms to solve NP-Hard problems
- The issue is that this problem is non-convex, meaning that we’re not trying to find the min/max on some smooth curve that’s a function of some variables!
- This makes optimization potentially trickier than what we’ve dealt with so far!

Plan of Attack

- Write down a function to minimize
- Decide how to explore the solution space
- MCMC the hell out of it!

The Traveling Salesman with MCMC

First, we generate a matrix that holds distances between locations:

```
1  n_locations <- 10
2  dist_mat <- matrix(0, nrow=n_locations, ncol=n_locations)
3
4  for (i in 1:n_locations){
5    for (j in 1:n_locations){
6      if (i == j){
7        dist_mat[i,j] <- 0
8      } else if (j > i){
9        dist_mat[i,j] <- runif(1, min=1, max=10)
10     } else if (j < i) {
11       dist_mat[i,j] <- dist_mat[j,i]
12     }
13   }
14 }
15
16 dist_mat[1:3, 1:3]
17
18 #           [,1]      [,2]      [,3]
19 # [1,] 0.000000 4.388056 8.869981
20 # [2,] 4.388056 0.000000 3.998735
21 # [3,] 8.869981 3.998735 0.000000
```

The Traveling Salesman with MCMC

Next, we need a loss function to evaluate the total trip length of our proposed solution:

```
1 TripDist <- function(path, dist_matrix){
2   total_dist <- 0
3   for (i in seq_along(path)){
4     if (i == length(path)) {
5       total_dist <- total_dist + dist_mat[path[i], path[1]]
6     } else {
7       total_dist <- total_dist + dist_mat[path[i], path[i+1]]
8     }
9   }
10  return(total_dist)
11 }
12
13 TripDist(1:10, dist_mat)
14
15 # [1] 51.36853
```

The Traveling Salesman with MCMC

Finally, we need a "transition matrix"—something to draw our new proposal from:

```
1 PermutePath <- function(path){
2   new_path <- path
3   p <- sample(1:length(path), 1)
4   if (p == length(path)){
5     new_path[p] <- path[1]
6     new_path[1] <- path[p]
7   } else {
8     new_path[p] <- path[p+1]
9     new_path[p+1] <- path[p]
10  }
11  return(new_path)
12 }
13
14 PermutePath(1:10)
15
16 # [1] 1 2 3 4 5 6 7 8 10 9
```


The Traveling Salesman with MCMC

Now we implement MCMC:

```
1 points <- 1:n_locations
2 N_iter <- 1e6
3 Ts <- c(0, 0.001, 0.005, 0.01, 0.05)
4 conv <- best <- vector('list', length=length(Ts))
5 for (j in seq_along(Ts)){
6   dist <- vector('numeric', length=N_iter)
7   path <- sample(points)
8   best[[j]] <- path
9   for (i in 1:N_iter){
10    new_path <- PermutePath(path)
11    if (TripDist(new_path, dist_mat) < TripDist(path, dist_mat) | runif(1) < Ts[j]){
12      path <- new_path
13    }
14    if (TripDist(path, dist_matrix) < TripDist(best[[j]], dist_mat)){
15      best[[j]] <- path
16    }
17    dist[i] <- TripDist(path, dist_mat)
18  }
19  conv[[j]] <- data.frame(i = 1:N_iter, dist=dist, T=Ts[j])
20 }
```

Traveling Salesman Results by Temperature

```
1  for (j in 1:length(Ts)) {  
2    print(Ts[j])  
3    print(best[[j]])  
4    print(TripDist(best[[j]], dist_mat))  
5  }  
6  
7  # [1] 0  
8  # [1] 7 9 5 3 10 2 1 6 8 4  
9  # [1] 35.75039  
10 # [1] 0.001  
11 # [1] 7 9 2 10 3 5 6 1 8 4  
12 # [1] 32.90462  
13 # [1] 0.005  
14 # [1] 2 10 9 1 6 3 5 8 4 7  
15 # [1] 32.65007  
16 # [1] 0.01  
17 # [1] 8 4 7 2 10 9 1 6 3 5  
18 # [1] 32.65007  
19 # [1] 0.05  
20 # [1] 6 1 9 10 2 7 4 8 5 3  
21 # [1] 32.65007
```

Traveling Salesman Results by Temperature

```
1 conv <- do.call('rbind', conv)
2
3 ggplot(conv,
4         aes(x=i,
5             y=dist,
6             color=as.character(T))) +
7   geom_line() +
8   facet_grid(T~.) +
9   labs(x='iteration',
10        y='distance',
11        color='temp') +
12   theme_bw()
```



Wrap Up

Recap

- Markov Chain Monte Carlo is an exceptionally powerful algorithm!
- It has way more applications than we talked about today
- Take a course in Bayesian inference if you want to do more MCMC
- There are also tons of great extensions to MCMC
 - Simulated annealing
 - Hamiltonian Monte Carlo
 - Gibbs Sampling
- This stuff is challenging, but you'll practice and implement it in your homework. Please ask questions!

Final Thoughts

- [PollEv.com/klintkanopka](https://pollev.com/klintkanopka)