a3q3

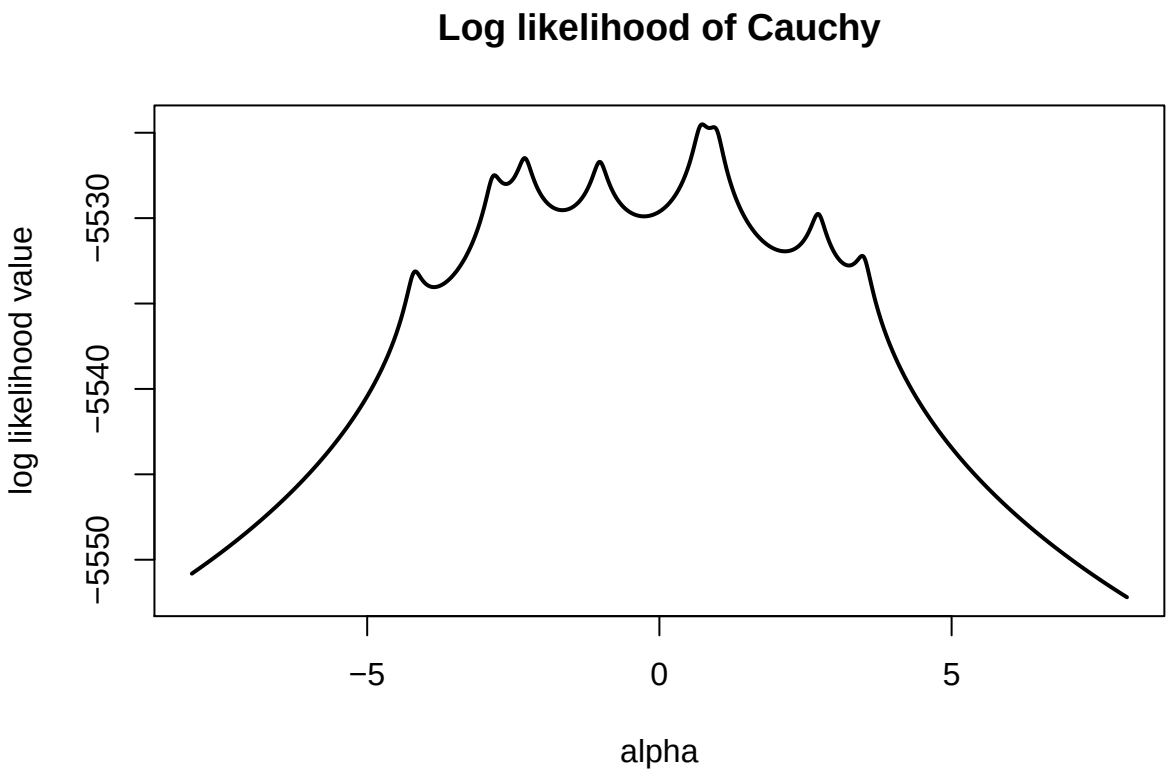
Jihoon Han

4/8/2021

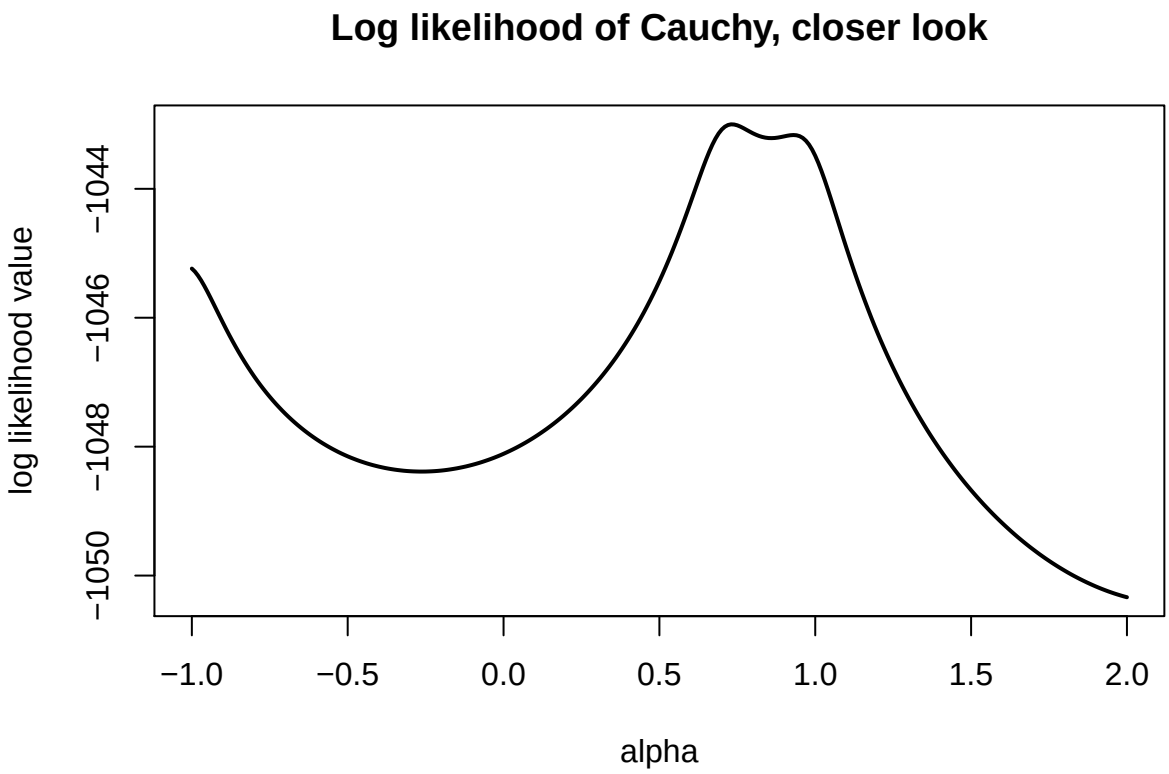
```r
# part a)
x <- c(-4.20, -2.85, -2.30, -1.02, 0.70, 0.98, 2.72, 3.50)
beta <- 0.1

log_likelihood <- function(alpha){
  n <- length(alpha)
  return_val <- c()
  for (i in 1:n) {
    return_val[i] <- n*log(beta)-n*log(pi)-sum(log((beta^2)+((x-alpha[i])^2)))
  }
  return(return_val)
}

alpha <-seq(-8,8,0.01)
l_likelihood <- log_likelihood(alpha)
plot(alpha,l_likelihood,ylab="log likelihood value",type="l",lwd=2,main="Log likelihood of Cauchy")
```




```
# Looking closer to the global max point.
alpha <-seq(-1,2,0.01)
l_likelihood <- log_likelihood(alpha)
plot(alpha,l_likelihood,ylab="log likelihood value",type="l",lwd=2,
     main="Log likelihood of Cauchy, closer look")
```



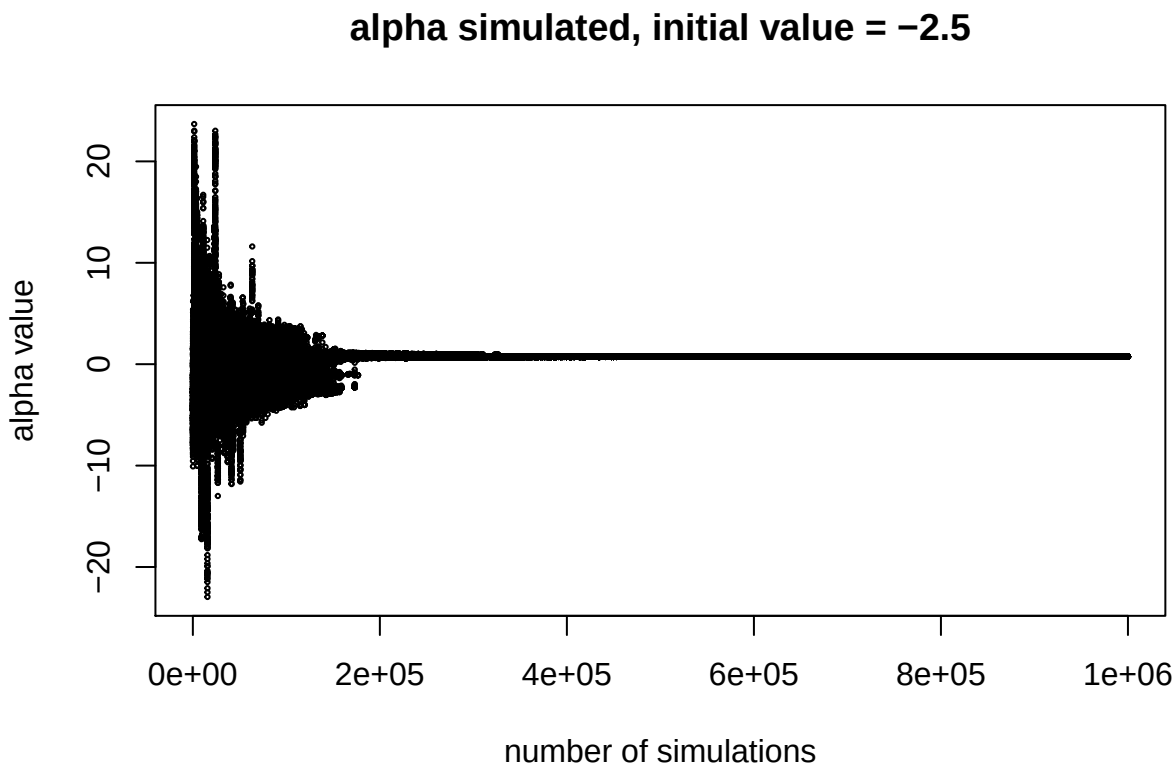
If we look at the log-likelihood closer, we can see that the alpha value that achieves the global maximum value of the log-likelihood is about 0.75.

```r
# part b)
# sim_alg is the simulated annealing algorithm function
sim_alg <- function(init, T, n) {
  # init means intial value.
  # T mean Temperature
  # a is alpha
  # new_a is the newly generated alpha
  a <- init
  f <- length(T)
  for (i in 1:n) {
    T_i <- T[1]*((T[f]/T[1])^((i-1)/n))
    new_a <- rnorm(1, a[i], 1)
    r <- exp((log_likelihood(new_a) - log_likelihood(a[i]))/T_i)
    if (r >= 1) {
      a[i+1] <- new_a
    }
    else {
      u <- runif(1)
      if (u <= r) {
        a[i+1] <- new_a
      }
      else {
        a[i+1] <- a[i]
      }
```

```
    }
  }
  return(a)
}

T <- c(10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001)
n <- 10^6
init <- 0

# MLE for initial value alpha = -2.5
init <- -2.5
alpha_vals <- sim_alg(init,T,n)
plot(1:length(alpha_vals), alpha_vals,
     main="alpha simulated, initial value = -2.5",
     xlab = "number of simulations", ylab = "alpha value",cex =0.3)
```



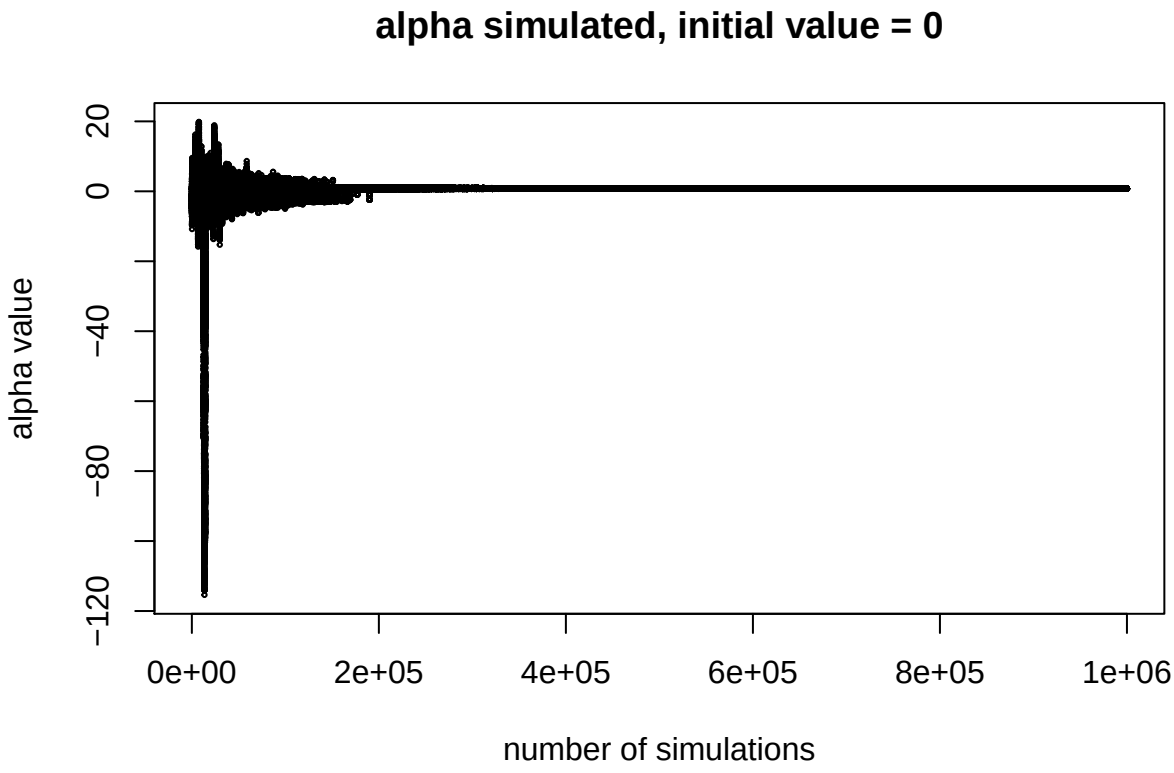

```
MLE_0 <- alpha_vals[n]
MLE_0
```

```
## [1] 0.7328211
```

```
tail(alpha_vals)
```

```
## [1] 0.7328211 0.7328211 0.7328211 0.7328211 0.7328211 0.7328211
```

```r
# MLE for initial value alpha = 0
init <- 0
alpha_vals <- sim_alg(init,T,n)
plot(1:length(alpha_vals), alpha_vals,
     main="alpha simulated, initial value = 0",
     xlab = "number of simulations", ylab = "alpha value",cex =0.3)
```



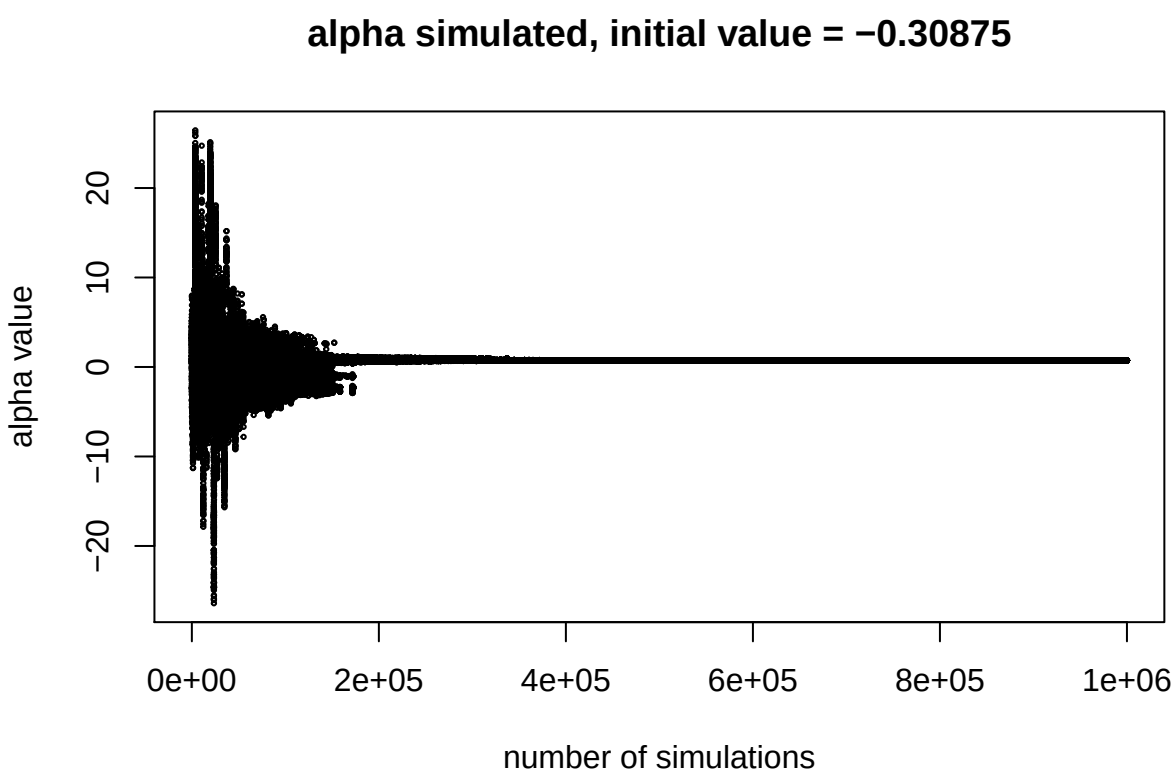

```r
MLE_0 <- alpha_vals[n]
MLE_0
```

```
## [1] 0.7328407
```

```r
tail(alpha_vals)
```

```
## [1] 0.7328407 0.7328407 0.7328407 0.7328407 0.7328407 0.7328407
```

```r
 # MLE for initial value alpha = -0.30875
init <- -0.30875
alpha_vals <- sim_alg(init,T,n)
plot(1:length(alpha_vals), alpha_vals,
     main="alpha simulated, initial value = -0.30875",
     xlab = "number of simulations", ylab = "alpha value",cex =0.3)
```




```
MLE_30875 <- alpha_vals[n]
MLE_30875
```

```
## [1] 0.7327837
```

```
tail(alpha_vals)
```

```
## [1] 0.7327837 0.7327837 0.7327837 0.7327837 0.7327837 0.7327837
```

As we can see, for all initial values, it seems like it converges to the correct alpha that achieves global maximum, which is 0.73.

```
# part c)
library(GA)

# decode function converts the x decimal values to binary values
decode <- function(x, N){
  bvals <- c()
  for (i in 1:N) {
    bvals[[i]] <- decimal2binary(x[i]*(10^6), 25)
  }
  return(bvals)
}
```

```r
# encode function converts the x binary representation
# values to decimal values
encode <- function(binary_rep, N) {
  dvals <- c()
  for (i in 1:N) {
    dvals[i] <- binary2decimal(binary_rep[[i]]/(10^6))
  }
  return(dvals)
}

# crossover mutates the binary_representation of x and
# crossover the binaries.
crossover <- function(binary_rep, row1, row2, split_point) {
  b1 <- binary_rep[[row1]]
  b2 <- binary_rep[[row2]]
  b1_former <- b1[1:split_point]
  b2_former <- b2[1:split_point]
  b1_latter <- b1[(split_point+1):25]
  b2_latter <- b2[(split_point+1):25]
  return(rbind(c(b1_former, b2_latter),
               c(b2_former, b1_latter)))
}


Gen_alg <- function(N, iterations, mutation_prob) {
  popn <- runif(N, 1, 31)
  for (t in 1:iterations) {
    binary_rep <- decode(popn,N)
    # selecting parents
    best_fit_row <- which.max(lapply(popn, log_likelihood))
    least_fit_row <- which.min(lapply(popn, log_likelihood))
    binary_rep[[least_fit_row]] <- binary_rep[[best_fit_row]]
    # crossover
    for (i in seq(1, N, 2)) {
      split_point <- sample(c(1:24), 1)
      crossover_val <- crossover(binary_rep, i, i+1, split_point)
      binary_rep[[i]] <- crossover_val[1,]
      binary_rep[[i+1]] <- crossover_val[2,]
      # print(binary_rep)
    }
    # mutation
    for (i in 1:N) {
      u <- runif(1)
      if (u < mutation_prob) {
        mutation_point <- sample(c(1:25), 1)
        binary_rep[[i]][mutation_point] <- (1 - binary_rep[[i]][mutation_point])
      }
    }
    # update the new t+1 generation by encoding
    popn <- encode(binary_rep,N)
  }
  return(popn)
}
```

```r
# We take 10000 generations

# Mutation probability 0.1
Gen_alg(10,10000,0.1)
```

```
##  [1] 0.732671 0.732671 0.732671 2.829823 0.732671 0.732671 0.732671 0.732671
##  [9] 0.732671 0.732671
```

```r
Gen_alg(20,10000,0.1)
```

```
##  [1] 0.732772 0.732775 2.829924 0.732772 0.732768 0.732772 0.732900 0.732772
##  [9] 0.732768 0.732772 0.732781 0.749153 0.730659 0.748771 0.732772 0.994916
## [17] 0.732772 0.732768 0.732788 0.732772
```

```r
Gen_alg(30,10000,0.1)
```

```
##  [1] 0.962163 3.124831 0.673600 0.732746 0.732769 0.730723 0.732778 2.846435
##  [9] 0.731736 0.994916 0.733121 0.731007 0.732259 0.724580 0.732772 0.732772
## [17] 0.732704 0.732738 0.601668 0.732787 0.732704 0.670334 0.753504 0.732770
## [25] 9.154372 0.732780 0.732786 0.224353 0.732769 0.732491
```

```r
# Mutation probability 0.2
Gen_alg(10,10000,0.2)
```

```
##  [1] 0.732770 0.732772 0.732768 0.732773 0.724580 0.732736 0.732780 0.732787
##  [9] 9.121315 0.732772
```

```r
Gen_alg(20,10000,0.2)
```

```
##  [1] 0.729689 1.781349 0.749155 0.732260 0.737123 0.976480 1.781344 0.732772
##  [9] 0.746093 0.732772 0.667258 0.994425 0.732772 0.736996 0.732769 0.671268
## [17] 0.750430 0.993894 0.732260 0.601700
```

```r
Gen_alg(30,10000,0.2)
```

```
##  [1] 0.732999 9.113027 4.044351 0.233062 0.205411 1.796679 1.772543 0.731749
##  [9] 1.255002 0.732746 2.698853 0.732177 0.724575 0.667103 0.601785 0.732260
## [17] 0.667200 0.996987 0.976485 2.829919 0.732772 0.732780 0.801380 0.732895
## [25] 0.740960 0.601700 2.830056 0.733003 1.276970 0.726754
```

```r
# Mutation probability 0.3
Gen_alg(10,10000,0.3)
```

```
##  [1] 0.732772 0.732775 0.667140 0.763491 0.749163 4.335917 0.732772 0.732900
##  [9] 0.732771 0.732772
```

```
Gen_alg(20,10000,0.3)
```

```
##  [1] 0.600652 0.749159 0.663110 0.730139 0.667239 0.732769 0.735592 0.679018
##  [9] 0.986705 0.732775 9.121392 0.735857 0.730721 0.724575 1.516624 0.732768
## [17] 0.732775 0.485477 0.222741 0.683611
```

```
Gen_alg(30,10000,0.3)
```

```
##  [1] 0.733016 9.164634 1.781353 0.732233 0.735058 0.995016 0.732748 2.303583
##  [9] 0.994374 5.840725 0.732778 0.730917 0.722175 0.782148 0.724545 1.781349
## [17] 2.862771 3.090019 0.662397 0.730660 3.833417 2.860914 2.371161 0.732769
## [25] 1.713257 2.714991 2.821773 0.736310 0.658019 0.198597
```

When we have low mutation(mutation probability = 0.1),
we can observe that most of the population has the value 0.732772,
which is the global maximal alpha.
However, since the mutation probability is low, we introduce less randomness,
and this might cause some generation being stuck in a different "local" maximal value.
As we have higher mutation probability (mutation probability = 0.3),
we introduce much more randomness, and this causes the population values
having different numbers rather than the global optimal value, 0.73.
However, we can identify that there are some values still having near
0.73.

As the number of population increases, we can identify that the population
has much more randomness.
However, when the mutation is low, it still has the majority of population values
being near 0.73.