Computational Statistics Computer Lab 6 (Group 7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)

2024-01-28

Question 1: Genetic algorithm (Solved by Qinyuan Qi)

Answer:

(1) 3 Encodings :

We define 3 encodings accordingly which can generate chess board and put queens on the board randomly. We also define some functions to print out the board layout.

For simplicity, all encodings will be column based and can be converted to matrix for printing.

Please check the code in the appendix.

```
## [1] "Test Encoding 1"
```

```
##     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## X1L    1    1    1    1    1    2    2    2
## X2L    2    3    4    5    7    4    7    8
```

```
## |-|Q|Q|Q|Q|-|Q|-|
## |-|-|-|Q|-|-|Q|Q|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
```

```
## [1] "Test Encoding 2"
```

```
## [1]  2 16 32 64  4 16  8  8
```

```
## |-|-|-|-|-|-|-|-|
## |-|-|-|Q|-|-|-|-|
## |-|-|Q|-|-|-|-|-|
## |-|Q|-|-|-|Q|-|-|
## |-|-|-|-|-|-|Q|Q|
## |-|-|-|-|Q|-|-|-|
## |Q|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
```

```
## [1] "Test Encoding 3"
```

```
## [1] 7 1 5 8 6 2 3 4
```

```
## |-|Q|-|-|-|-|-|-|
## |-|-|-|-|-|Q|-|-|
## |-|-|-|-|-|-|Q|-|
## |-|-|-|-|-|-|-|Q|
```

```
## |-|-|Q|-|-|-|-|-|
## |-|-|-|-|Q|-|-|-|
## |Q|-|-|-|-|-|-|-|
## |-|-|-|Q|-|-|-|-|
```

(2) Crossover function:

Crossover function call result listed below. Output below shows that the crossover function works as expected. First 2 chess board are the parents and the last one is the child. For source code, please check the appendix.

```
## [1] "Test Crossover encoding 1"
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## X1L      1    1    1    1    1    2    2    2
## X1L.1    1    2    6    7    8    1    5    6
```

```
## |Q|Q|-|-|-|Q|Q|Q|
## |Q|-|-|-|Q|Q|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## X1L      1    1    1    1    1    2    2    2
## X1L.1    1    3    4    5    7    1    3    5
```

```
## |Q|-|Q|Q|Q|-|Q|-|
## |Q|-|Q|-|Q|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
```

```
## [1] "Child encoding 1"
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## X1L      1    1    1    1    1    2    2    2
## X1L.1    1    2    6    7    7    1    3    5
```

```
## |Q|Q|-|-|-|Q|Q|-|
## |Q|-|Q|-|Q|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
## |-|-|-|-|-|-|-|-|
```

(3) Mutate functions:

We mutate 3 encodings using the mutate function. The basic idea is to move the queen to a lower position (Y axis if not occupied. Check code in appendix for details.

```
## [1] "Test Mutate encoding 3"
```

```
## [1] 1 6 4 7 8 5 3 2
```

```
## [1] 2 6 4 5 1 8 7 3
```

(4):

To calc the fitness value of each encoding, we need to implement 3 different functions.

For the binary encoding, we restricted that every row must have a queen and only one queen to make check function easier to implement.

```
## [1] "Test Fitness encoding 3"
```

```
## [1] 7 8 3 6 2 4 5 1
```

```
## |-|-|-|-|-|-|-|Q|
## |-|-|-|-|Q|-|-|-|
## |-|-|Q|-|-|-|-|-|
## |-|-|-|-|-|Q|-|-|
## |-|-|-|-|-|-|Q|-|
## |-|-|-|Q|-|-|-|-|
## |Q|-|-|-|-|-|-|-|
## |-|Q|-|-|-|-|-|-|
```

```
## unsafe queens number: 5
```

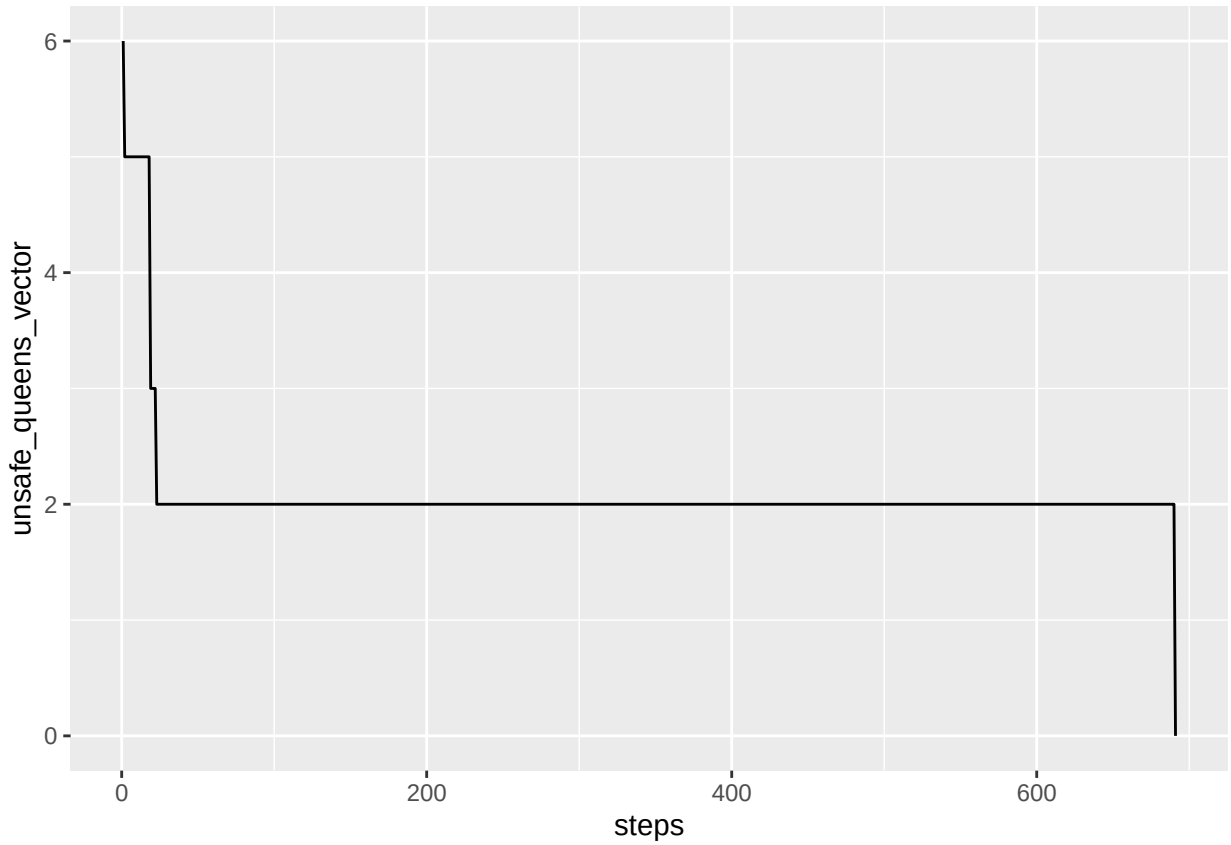
(5)(6)(7):

The genetic_algorithm implemented and listed in the appendix

(8):

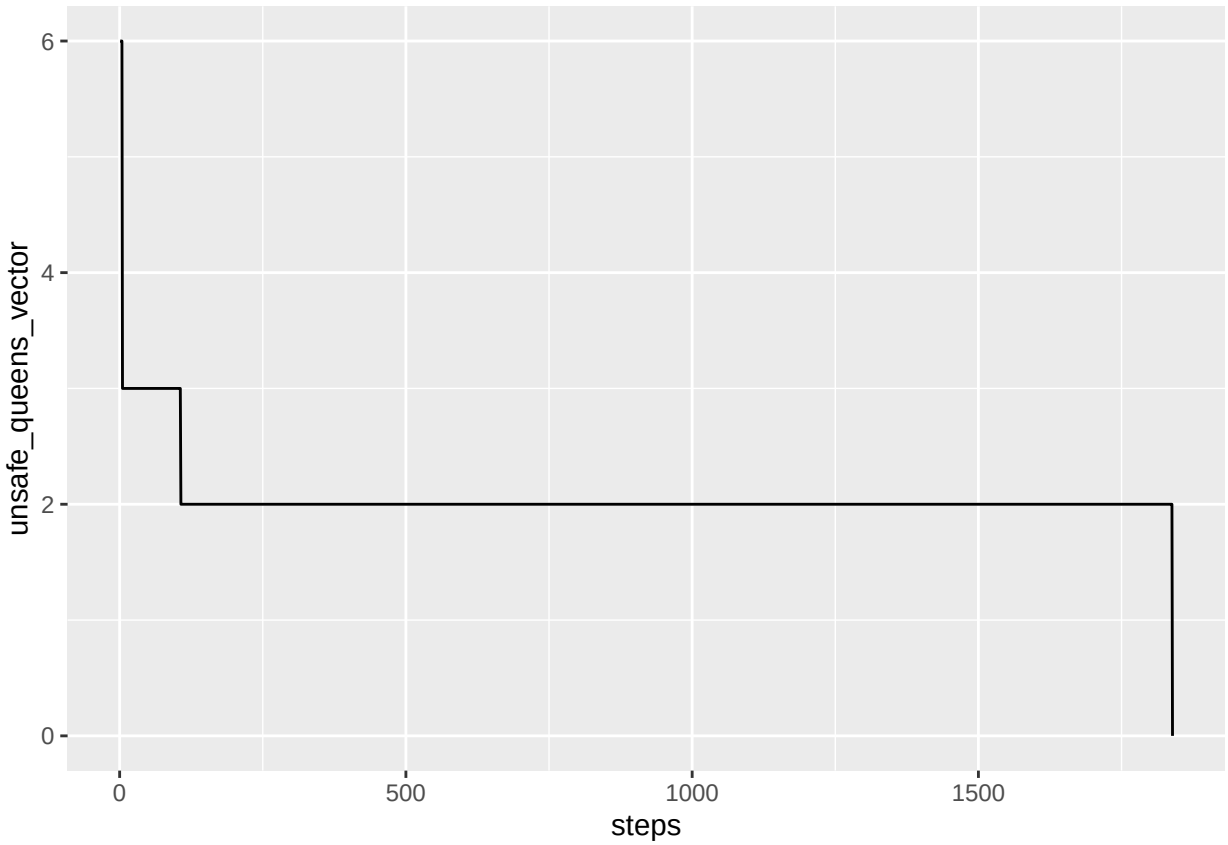
```
p_val <- 4
board_size <- 8
genetic_algorithm(method=3,max_steps = 1000)
```

```
## solution found after try  691  times
## |-|-|-|Q|-|-|-|-|
## |-|-|-|-|-|-|Q|-|
## |-|-|-|-|Q|-|-|-|
## |-|-|Q|-|-|-|-|-|
## |Q|-|-|-|-|-|-|-|
## |-|-|-|-|-|Q|-|-|
## |-|-|-|-|-|-|-|Q|
## |-|Q|-|-|-|-|-|-|
```




```
board_size <- 10
genetic_algorithm(method=3,max_steps = 10000)
```

```
## solution found after try  1839  times
## |-|-|-|-|-|-|-|Q|-|-|
## |-|-|-|-|Q|-|-|-|-|-|
## |Q|-|-|-|-|-|-|-|-|-|
## |-|-|-|-|-|Q|-|-|-|-|
## |-|-|-|-|-|-|-|-|Q|-|
## |-|Q|-|-|-|-|-|-|-|-|
## |-|-|-|Q|-|-|-|-|-|-|
## |-|-|-|-|-|-|Q|-|-|-|
## |-|-|-|-|-|-|-|-|-|Q|
## |-|-|Q|-|-|-|-|-|-|-|
```



(9):

Accoring to the code, we know that the 3rd one is the easiest to implement. Since it is a vector position encoding, which need to small memory space and also easy to implement the crossover and mutate function.

The 2nd one is the hardest to implement, since it is a binary encoding, to simplify, for every column, we only have one queen, which means the binary string will have the form as "00000001" , "10000000", "00010000". For "11001100" is forbidden, since if it's hard to generate a valid configuration even if we create chess board layout.

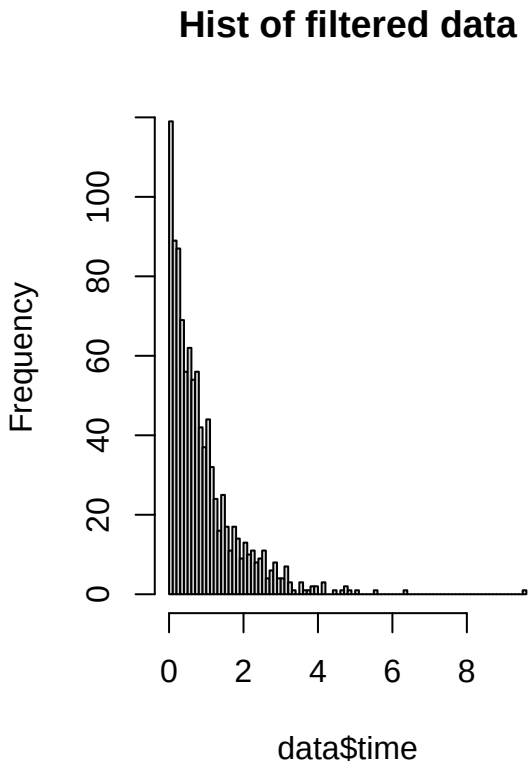
For the 1st one, easy to understand, but has the same problem as the 2nd one,because we can have multiple queens in the same column/rows even if we can random the layout using cross over and mutate function.and very easy to exceed the max try count.

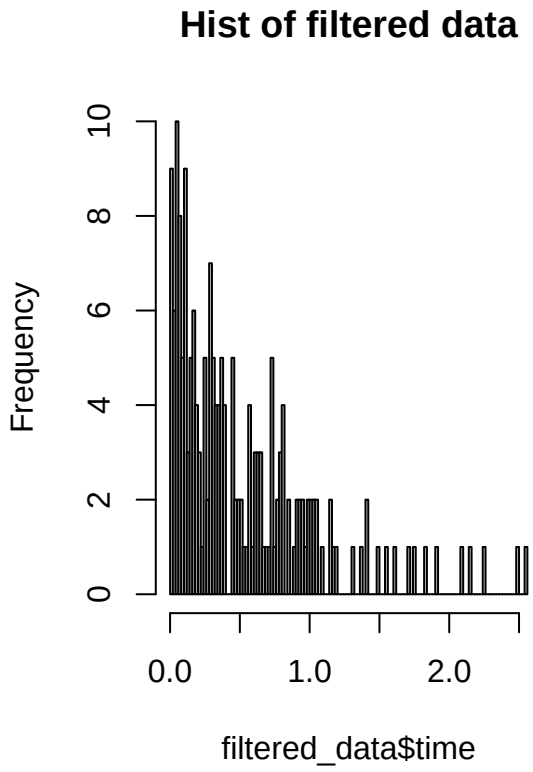
Question 2: EM algorithm (Solved by Satya Sai Naga Jaya Koushik Pilla)

Answer:

(1) Plot 2 histograms:

According to the plots generated, we found that the plot seems follow exponential distribution.





(2):

The general CDF form of an exponential distribution is:

$$F(x,\lambda) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

So PDF of an exponential distribution is derivative of F on x:

$$f(x,\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

According to the plot, we know that time is greater than 0, so we omit the condition x < 0.

We have 2 types of observations here, one is censored(cens=1) and the other is not censored(cens=2).

When cens=1 which means we find the failure immediately, so the contribution of a single censered point x_i to the likelihood is: $f(x_i, \lambda)$

When cens=2 which means we find the failure after a period of time,this time will be the check interval. which means the contribution of a single uncensered point x_j to the likelihood is: $f(x_j|X < x_j)$, which is given by.

$$F(x_j|X < x_j) = \frac{F(x_j)}{f(x_j)} = \frac{\lambda e^{-\lambda x_j}}{1 - e^{-\lambda x_j}}$$

So the overrall likelihood function is the product of two likelihood for all points in our dataset.

Let's set U is the set of uncensored points(original exp distribution) set and C is the set of censored points set(which need to consider conditional distribution). Then we have

$$L(\lambda, U, C) = \prod_U f(x_i, \lambda) \prod_C f(x_i | X \leq x_j) = \prod_U \lambda e^{-\lambda x_i} \prod_C \frac{\lambda e^{-\lambda x_j}}{1 - e^{-\lambda x_j}}$$

Apply log on both sides ,we have the log likelihood function as follows.

$$logL(\lambda, U, C) = \sum_U log(\lambda e^{-\lambda x_i}) + \sum_C log(\frac{\lambda e^{-\lambda x_j}}{1 - e^{-\lambda x_j}})$$

$$logL(\lambda, U, C) = \sum_U (log(\lambda) - \lambda x_i) + \sum_C (log(\lambda) - \lambda x_j - log(1 - e^{-\lambda x_j}))$$

(3):

We derivative on likelihood function in 2.2, and we get the following function.

$$\frac{d}{d\lambda} logL(\lambda, U, C) = \sum_U (1 - x_i) + \sum_C (1 - x_j) + \sum_C \frac{x_j e^{-\lambda x_j}}{1 - e^{-\lambda x_j}}$$

After rerange the terms, we get the λ expression as follows.|U| and |C| are the number of uncensored points and censored points respectively.

$$\lambda = \frac{|U| + |C|}{\sum_U x_i + \sum_C x_j - \sum_C \frac{x_j e^{-\lambda x_j}}{1 - e^{-\lambda x_j}}}$$

which can be used in E-Step in our EM algorithm.

(4):

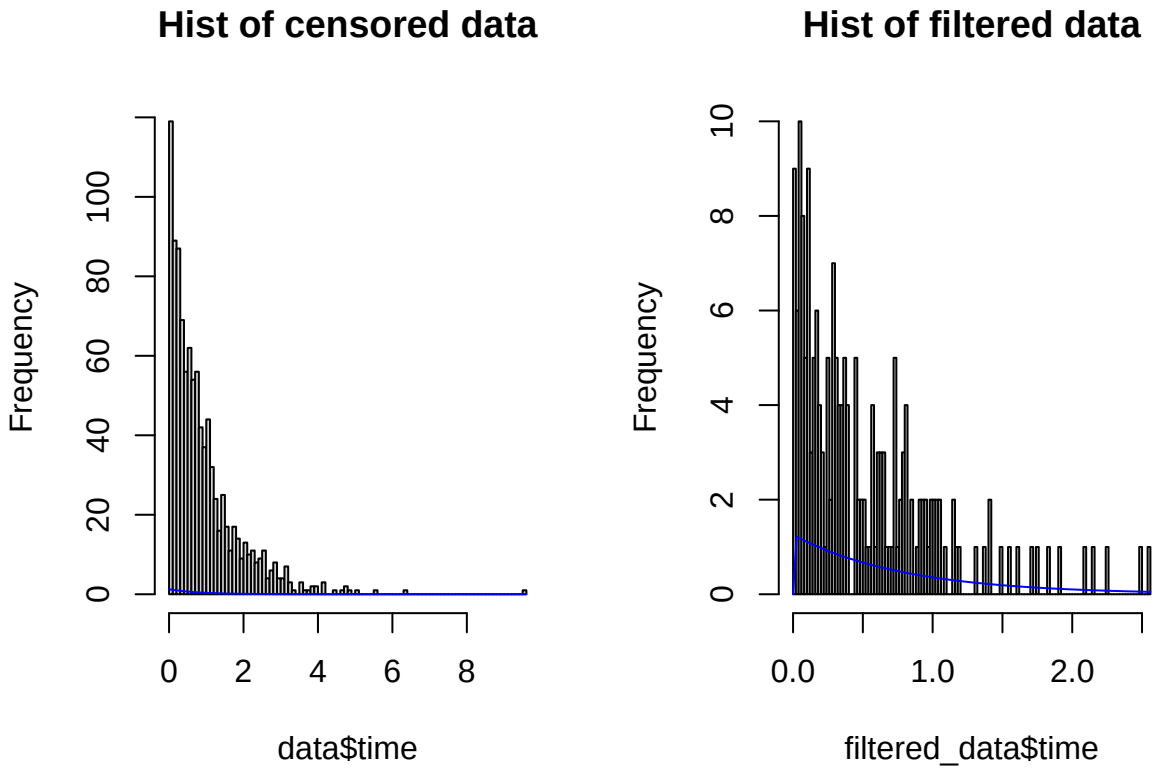
According to the formula in 2.3, we can implement the EM algorithm. The result is as follows.

```
## Estimated lambda: 1.258075
```

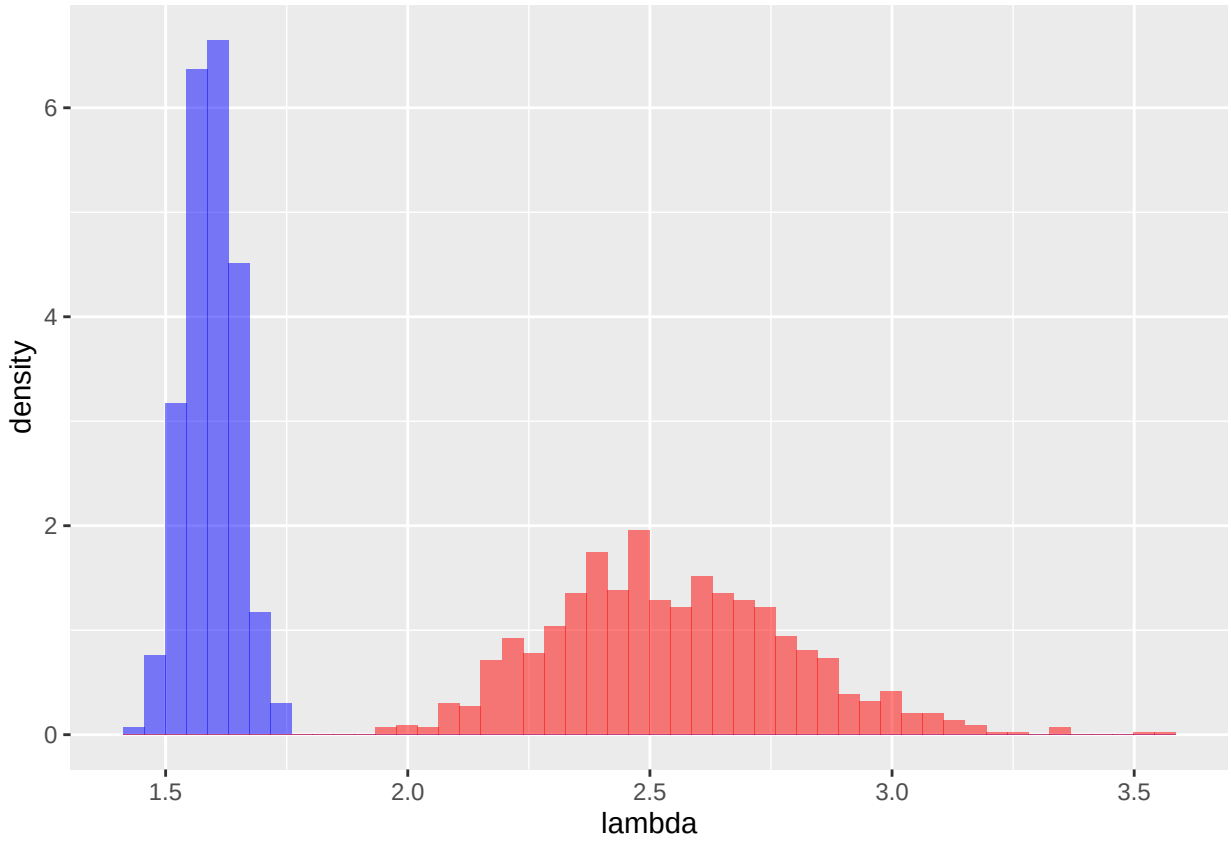
```
## Number of iterations: 6
```

(5):

Density curve are as follows.


Hist of censored data
Frequency
0
20
40
60
80
100
0
2
4
6
8
data$time
Hist of filtered data
Frequency
0
2
4
6
8
10
0.0
1.0
2.0
filtered_data$time

(6):


density
0
2
4
6
1.5
2.0
2.5
3.0
3.5
lambda

```
## EM lambda : 1.258075
```

```
## Boot EM lambda: 1.593501
```

```
## Boot MLE lambda: 2.54651
```

```
## Boot EM lambda variance: 0.002830399
```

```
## Boot MLE lambda variance: 0.06115264
```

According to the output, the variance are csmall, MLE estimator has a smaller variance. But both are biased estimators.

Appendix: Code for this report

```
########################## Init code for question 1 ############################
rm(list = ls())
library(ggplot2)



########################## Config Encoding Function 1.1 ############################
# First encoding: n pairs encoding
init_configuration_1 <- function() {

  configuration <- data.frame(x = c(), y = c())
  queen_count <- 0
  for (i in 1:board_size) {
    for(j in 1:board_size) {
      isQueen <- sample(c(0, 1), 1)
      if (queen_count < board_size) {
        if (isQueen == 1){
          configuration <- rbind(configuration, c(i, j))
          queen_count <- queen_count + 1
        }
      }
    }
  }
  return(t(configuration))
}

# Second Encoding: binary encoding
# if board_size = 8, then the max n is 2^8 - 1 = 255
init_configuration_2 <- function(max_try_count = 1000) {

  max_value <- 2^board_size - 1
  queen_count <- 0
  configuration <- rep(0, board_size)
  while(queen_count < board_size && max_try_count > 0) {
    number <- sample(0:max_value, 1)

    new_queen <- sum(as.numeric(intToBits(number)))
    if (queen_count + new_queen <= board_size && new_queen == 1) {
      queen_count <- queen_count + new_queen
      configuration[queen_count] <- number
    }
    max_try_count <- max_try_count - 1
  }

  if(max_try_count == 0) {
    print("Error: cannot generate a configuration")
  }

  return(configuration)

}
```

```r
# Third Encoding: vector position encoding
init_configuration_3 <- function() {

  configuration <- sample(1:board_size, board_size)
  return(configuration)
}

# convert encodings to matrix
decode_config <- function(config, method = 1) {
  if (method == 1){#(x,y) encoding
    n <- board_size
    mat <- matrix(0, nrow = n, ncol = n)
    for (i in 1:n) {
      mat[config[1,i], config[2,i]] <- 1
    }
  }else if (method == 2) {#binary encoding
      n <- board_size
      mat <- matrix(0, nrow = n, ncol = n)
      for (j in 1:n) {
        number <- config[j]
        for (i in n:1) {
          mat[i, j] <- number %% 2
          number <- number %/% 2
        }
      }
  }else if (method == 3){#vector position encoding
    n <- board_size
    mat <- matrix(0, nrow = n, ncol = n)
    for (i in 1:n) {
      mat[config[i],i] <- 1
    }
  }

  return(mat)
}

# convert  matrix to encoding 1 (x,y) pair
encode_config <- function(mat) {
  n <- board_size
  #(x,y) encoding
  config <- data.frame(x = rep(0,n), y = rep(0,n))
  index = 1
  for(i in 1:n){
    for(j in 1:n){
      if (mat[i,j] == 1){
        config[index,1] <- i
        config[index,2] <- j
        index <- index + 1
      }
    }
  }
  config <- t(config)
  return(config)
```

```
}

# print board common function
print_board <- function(configuration, method = 1) {
  config <- decode_config(configuration, method)

  n <- board_size
  for(i in 1:n) {
    for(j in 1:n) {
      if(config[i, j] == 1) {
        if (j == 1) {
          cat("|Q|")
        }else if (j == n){
          cat("Q|", "\n")
        }else{
          cat("Q|")
        }
      } else {
        if (j == 1) {
          cat("|-|")
        }else if (j == n){
          cat("-|", "\n")
        }else{
          cat("-|")
        }
      }
    }
  }
}
##########################Test Function Call ############################
board_size <- 8

print("Test Encoding 1")
configuration <- init_configuration_1()
configuration

print_board(configuration, 1)

print("Test Encoding 2")
configuration <- init_configuration_2()
configuration

print_board(configuration, 2)

print("Test Encoding 3")
configuration <- init_configuration_3()
configuration

print_board(configuration, 3)
########################## Crossover Function ############################
crossover <- function(config1, config2,method = 1, p = 4){

  child_config <- config1
```

```
  if (method == 1){
    child_config[,(p+1):board_size] = config2[,(p+1):board_size]
  }else{
    for(i in (p+1):board_size){
      child_config[i] <- config2[i]
    }
  }
  return(child_config)
}
########################## Test Crossover Function Call ##############################
print("Test Crossover encoding 1")
config1 <- init_configuration_1()
config1
print_board(config1, 1)
config2 <- init_configuration_1()
config2
print_board(config2, 1)

print("Child encoding 1")
child_config <- crossover(config1, config2, 1, 4)
child_config
print_board(child_config, 1)
########################## Mutate functions ############################
mutate <- function(config,method = 1) {

  boardsize <- board_size

  mutate_config <- config

  if (method == 1){
    processed <- FALSE
    mat <- decode_config(mutate_config, 1)
    while(!processed){
      col_to_mutate <- sample(1:boardsize, 1)
      # move the queen to lower position (y-1) if not occupied
      if (mutate_config[2,col_to_mutate] + 1 <= boardsize &&
          mat[mutate_config[1,col_to_mutate],mutate_config[2,col_to_mutate] + 1] == 0){
        mutate_config[2,col_to_mutate] <-  mutate_config[2,col_to_mutate] + 1
        processed = TRUE
      }else if (mat[mutate_config[1,col_to_mutate],1] == 0){
        mutate_config[2,col_to_mutate] <- 1
        processed <- TRUE
      }
    }
  }else if (method == 2){
    # current num * 2 or become 1 using bit operation
    col_to_mutate <- sample(1:boardsize, 1)
    queen_integer <- mutate_config[col_to_mutate]
    if (queen_integer == 1){
      queen_integer <- 2^(boardsize-1)
    }else{
      queen_integer <- bitwShiftR(queen_integer,1)
    }
```

```
    mutate_config[col_to_mutate] <- queen_integer
  }else if (method == 3){
    #col_to_mutate <- sample(1:boardsize, 1)

    #mutate_config[col_to_mutate] <- mutate_config[col_to_mutate] + 1

    #if (mutate_config[col_to_mutate] > boardsize){
    #  mutate_config[col_to_mutate] <- 1
    #}
    mutate_config = sample(1:board_size, board_size)
  }

  return(mutate_config)
}
########################## Test Mutate functions ############################
print("Test Mutate encoding 3")
config1 <- init_configuration_3()
config1
mutate_config <- mutate(config1, 3)
mutate_config
########################## Common Function ############################
# common functions for fitness function
# check attack between queen position
is_attack <- function(queen1, queen2) {
  return(
    queen1[1] == queen2[1] || queen1[2] == queen2[2] ||
    abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])
  )
}

########################## Fitness Function ############################
# fitness function to handle all the encodings, other encodings will be
# converted to this encoding.(X,Y) Location Encoding

# check whole config is valid or not and return
# attacked queens number and unattacked queens number
fitness_base <- function(config) {
  queen_num <- board_size
  attacked_queens <- c()

  for (i in 1:(queen_num - 1)) {
    for (j in (i + 1):queen_num) {
      if (is_attack(config[i,], config[j,])) {
        attacked_queens <- c(attacked_queens, i, j)
      }
    }
  }

  unattacked_queens <- setdiff(1:queen_num, unique(attacked_queens))
  num_unattacked_queens <- length(unattacked_queens)
  valid <- (length(unique(attacked_queens)) == 0)

  return(queen_num - num_unattacked_queens)
```

```
  #return(list(valid = valid,
  #            num_unattacked_queens = num_unattacked_queens,
  #            num_attacked_queens = queen_num - num_unattacked_queens))
}

fitness <- function(config, method = 1) {
  if (method == 1){
    return(fitness_base(t(config)))
  }else if (method == 2){
    # convert encoding to (x,y) encoding
    mat <- decode_config(config, method)
    config <- encode_config(mat)
    return(fitness_base(t(config)))
  }else if (method == 3){
    # convert vector position encoding to (x,y) encoding
    mat <- decode_config(config, 3)
    config <- encode_config(mat)
    return(fitness_base(t(config)))
  }
}


fitness1111 <- function(config, method = 1) {
  if (method == 1){
    return(fitness_base(t(config)))
  }else if (method == 2){
    # convert encoding to (x,y) encoding
    mat <- decode_config(config, method)
    config <- encode_config(mat)
    return(fitness_base(t(config)))
  }else if (method == 3){
    # convert vector position encoding to (x,y) encoding
    mat <- decode_config(config, 3)
    config <- encode_config(mat)
    return(fitness_base(t(config)))
  }
}
########################## Test Code for Fitness Function ############################
print("Test Fitness encoding 3")
set.seed(123)
config1 <- init_configuration_3()
print(config1)
print_board(config1, 3)
cat ("unsafe queens number:",fitness(config1,3),"\n")

######################   genetic_algorithm  ###################################
genetic_algorithm <- function(method = 1,max_steps = 10000){

  configuration_1 <- init_configuration_1()
  val_1 <- fitness(configuration_1,method)

  configuration_2 <- NULL
```

```
steps <- 0

unsafe_queens_vector <- c()

while(val_1 != 0 && steps <= max_steps ) {
   if (is.null(configuration_2)){
     if (method == 1){
       configuration_2 <- init_configuration_1()
       val_2 <- fitness(configuration_2,1)
     }else if(method == 2){
       configuration_2 <- init_configuration_2()
       val_2 <- fitness(configuration_2,2)
     }else{
       configuration_2 <- init_configuration_3()
       val_2 <- fitness(configuration_2,3)
     }
   }

  # cross over

  child_config <- crossover(configuration_1,
                            configuration_2,
                            method = method,
                            p = p_val)

  # mutate
  mutated_config <- mutate(child_config,method=method)
  val_child <- fitness(mutated_config,method)


  configs = list(configuration_1,
                 configuration_2,
                 mutated_config)

  unsafe_queens_num <- c(val_1,
                     val_2,
                     val_child)
  df <- data.frame(unsafe_queens = unsafe_queens_num,
                   configid = c(1,2,3))

  custom_order <- order(df$unsafe_queens, decreasing = FALSE)
  sorted_df <- df[custom_order, ]

  configuration_1 <- configs[[sorted_df$configid[1]]]
  configuration_2 <- configs[[sorted_df$configid[2]]]
  val_1 <- fitness(configuration_1,method)
  val_2 <- fitness(configuration_2,method)

  unsafe_queens_vector <- c(unsafe_queens_vector,sorted_df$unsafe_queens[1])
  steps <- steps + 1
}

#print the queen position if found the solution
```

```
  if (unsafe_queens_vector[length(unsafe_queens_vector)] == 0){
   cat("solution found after try ",steps," times\n")
   print_board(configuration_1, method)
  }else{
   print(paste("No solution found after try ",max_steps," times"))
  }

  df <- data.frame(steps = 1:length(unsafe_queens_vector),
                   unsafe_queens_vector = unsafe_queens_vector)
  ggplot(data=df, aes(x = steps, y = unsafe_queens_vector)) + geom_line()
}
p_val <- 4
board_size <- 8
genetic_algorithm(method=3,max_steps = 1000)

board_size <- 10
genetic_algorithm(method=3,max_steps = 10000)

########################## Init code for question 1 ############################
rm(list = ls())
library(ggplot2)
set.seed(123415)
################################## (2.1) #######################################
# Load data
data <- read.csv("censoredproc.csv",
                 sep = ";", header = TRUE)

filtered_data <- data[data$cens ==2,]
layout(matrix(c(1:2), 1, 2))
# plot the data
hist(data$time, breaks = 100, main="Hist of filtered data")
# plot the filtered data
hist(filtered_data$time, breaks = 100, main="Hist of filtered data")
log_likelihood <- function(lambda, data) {
  censor_data <- data[data$cens == 1,]
  unsensor_data <- data[data$cens == 2,]

  ll_censor <- sum(log(lambda) - censor_data$time * lambda)
  ll_unsensor <- sum(
                 log(lambda) -
                 unsensor_data$time * lambda -
                 log(1 - exp(-lambda * unsensor_data$time)))
  ll <- ll_censor + ll_unsensor
  ll
}

# EM algorithm
em_algorithm <- function(init_lambda=100,
                         data,
                         max_iter = 1000,
                         epsilon = 0.001) {

  lambda <- init_lambda
```

```r
  ll_values <- c()

  for (iter in 1:max_iter) {
    # E-step
    ll_values <- c(ll_values, log_likelihood(lambda, data))

    # M-step
    censored <- data[data$cens == 1,]
    uncensored <- data[data$cens == 2,]

    lambda_next <- nrow(data) / (
         sum(censored$time) +
         sum(uncensored$time) -
         sum(
              (uncensored$time * exp(-lambda * uncensored$time)) / (1-exp(-lambda * uncensored$time))
            )
         )

    # Check for convergence
    if (abs(lambda_next - lambda) < epsilon) {
      break
    }

    # Update lambda for the next iteration
    lambda <- lambda_next
  }

  return(list(lambda = lambda, iterations = iter))
}

# Initial guess for lambda
initial_lambda <- 100

# Run EM algorithm
em_result <- em_algorithm(initial_lambda, data)

# Print the result
cat("Estimated lambda:", em_result$lambda, "\n")
cat("Number of iterations:", em_result$iterations, "\n")
layout(matrix(c(1:2), 1, 2))
hist(data$time, breaks = 100,main="Hist of censored data")
curve(dexp(x, rate = em_result$lambda), add = TRUE, col = "blue")

hist(filtered_data$time, breaks = 100, main="Hist of filtered data")
curve(dexp(x, rate = em_result$lambda), add = TRUE, col = "blue")

bootstrap <- function(lambda, data, num_bootstrap) {

  em_lambda <- numeric(num_bootstrap)
  mle_lambda <- numeric(num_bootstrap)

  for (i in 1:num_bootstrap) {
    boot_samples <- rexp(nrow(data), rate = lambda)
```

```
    censored <- numeric(nrow(data))

    censor_index <- sample(which(data$cens == 2), sum(data$cens == 2))
    boot_samples[censor_index] <- runif(length(censor_index),0, boot_samples[censor_index])

    censored[censor_index]  <- 2
    censored[-censor_index] <- 1
    em_df <- data.frame(time = boot_samples, cens = censored)
    mle_df <- subset(em_df, cens == 2)

    em_lambda[i] <- em_algorithm(data = em_df)$lambda
    mle_lambda[i] <- 1 / mean(mle_df$time)
  }
  return (list(em_lambda = em_lambda, mle_lambda = mle_lambda))
}

# Set the number of bootstrap samples
num_bootstrap <- 1000

# perform bootstrap
boot_result <- bootstrap(lambda = em_result$lambda, data = data, num_bootstrap = 1000)

# plot
ggplot() +
geom_histogram(aes(x = boot_result$em_lambda, y = ..density..),
                 bins = 50, fill = "blue", alpha = 0.5) +
geom_histogram(aes(x = boot_result$mle_lambda, y = ..density..),
                 bins = 50, fill = "red", alpha = 0.5) +
xlab("lambda")




# Print the result
cat("EM lambda :", em_result$lambda, "\n")
cat("Boot EM lambda:", mean(boot_result$em_lambda), "\n")
cat("Boot MLE lambda:", mean(boot_result$mle_lambda), "\n")

cat("Boot EM lambda variance:", var(boot_result$em_lambda), "\n")
cat("Boot MLE lambda variance:", var(boot_result$mle_lambda), "\n")
```