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 b

Please check the code in appendix.

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

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

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

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

```
## [1]   4   2 128  64  16  32   8   1
```

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

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

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

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

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

(2) Crossover function:

Crossover function call result listed below. 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
## X3L    3    5    6    7    8    1    3    4
```

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

```
##     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## X1L    1    1    1    2    2    2    2    2
## X3L    3    4    8    1    2    3    5    6
```

```
## |-|-|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    2    2    2    2
## X3L    3    5    6    7    2    3    5    6
```

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

(3) Mutate functions:

We mutate the 3 encodings using the following functions.

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

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

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

(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.

```
########################## 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 <- function(config) {
  queen_num <- nrow(config)
  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(list(valid = valid,
              num_unattacked_queens = num_unattacked_queens,
              num_attacked_queens = queen_num - num_unattacked_queens))
}

fitness1 <- function(config) {
  return(fitness(config))
}

fitness2 <- function(config) {
  # convert binary encoding to (x,y) encoding
  mat <- decode_config(config, 2)
  config <- encode_config(mat, 1)
  return(fitness(config))
```

```
}

fitness3 <- function(config) {
  # convert vector position encoding to (x,y) encoding
  mat <- decode_config(config, 3)
  config <- encode_config(mat, 1)
  return(fitness(config))
}
```

(5)(6)(7):

The genetic_algorithm implemented as follows

```
#####################   genetic_algorithm  ###################################
genetic_algorithm <- function(method = 1,board_size=8){
  if (method == 1){
    configuration_1 <- init_configuration_1(board_size)
    val_1 <- fitness1(configuration_1)
  } else if (method == 2){
    configuration_1 <- init_configuration_2(board_size)
    val_1 <- fitness2(configuration_1)
  }else{
    configuration_1 <- init_configuration_3(board_size)
    val_1 <- fitness3(configuration_1)
  }

  configuration_2 <- NULL

  max_steps <- 1000
  steps <- 0

  num_attacked_queens_vector <- c(val_1$num_attacked_queens)

  while(val_1$num_attacked_queens != 0 && steps <= max_steps ) {

    if (is.null(configuration_2)){
      if (method == 1){
        configuration_2 <- init_configuration_1(board_size)
        val_2 <- fitness1(configuration_2)
      }else if(method == 2){
        configuration_2 <- init_configuration_2(board_size)
        val_2 <- fitness2(configuration_2)
      }else{
        configuration_2 <- init_configuration_3(board_size)
        val_2 <- fitness3(configuration_2)
      }
    }

    # cross over
    child_config <- crossover(configuration_1,
                              configuration_2,
                              p = p_val)

    # mutate
```

```
  if (method == 1){
    mutated_config <- mutate(child_config,method=1)
    val_child <- fitness1(mutated_config)

  } else if (method == 2){
    mutated_config <- mutate(child_config,method=2)
    val_child <- fitness2(mutated_config)
  } else {
    mutated_config <- mutate(child_config,method=3)
    val_child <- fitness3(mutated_config)
  }

  configs <- c(1,2,3)
  num_attacked_queens <- c(val_1$num_attacked_queens,
                   val_2$num_attacked_queens,
                   val_child$num_attacked_queens)
  df <- data.frame(config = configs,
                 num_attacked_queens = num_attacked_queens)

  custom_order <- order(df$num_attacked_queens)
  sorted_df <- df[custom_order, ]

  # choose minial 2 values
  if (sorted_df$config[1] == 1){
    configuration_1 <- configuration_1
    val_1 <- val_1
  }else if(sorted_df$config[1] == 2){
    configuration_1 <- configuration_2
    val_1 <- val_2
  }else {
    configuration_1 <- mutated_config
    val_1 <- val_child
  }

  if (sorted_df$config[2] == 1){
    configuration_2 <- configuration_1
    val_2 <- val_1
  }else if(sorted_df$config[2] == 2){
    configuration_2 <- configuration_2
    val_2 <- val_2
  }else {
    configuration_2 <- mutated_config
    val_2 <- val_child
  }

  num_attacked_queens_vector <- c(num_attacked_queens_vector,val_1$num_attacked_queens)
  steps <- steps + 1

}

# print the queen position if found the solution
if (val_1$num_attacked_queens == 0){
  print_board(configuration_1, 3)
```

```
  }else{
    print("not found the solution")
  }

  #df <- data.frame(steps = 1:length(num_attacked_queens_vector),
  #                 num_attacked_queens = num_attacked_queens_vector)
  #ggplot(data=df, aes(x = steps, y = num_attacked_queens)) + geom_line()

}

p_val <- 3
genetic_algorithm(method=1,board_size=8)
```

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

(8):

```
p_val <- 2
genetic_algorithm(method=3,board_size=4)
p_val <- 4
genetic_algorithm(method=3,board_size=8)
p_val <- 8
genetic_algorithm(method=3,board_size=16)
```

(9):

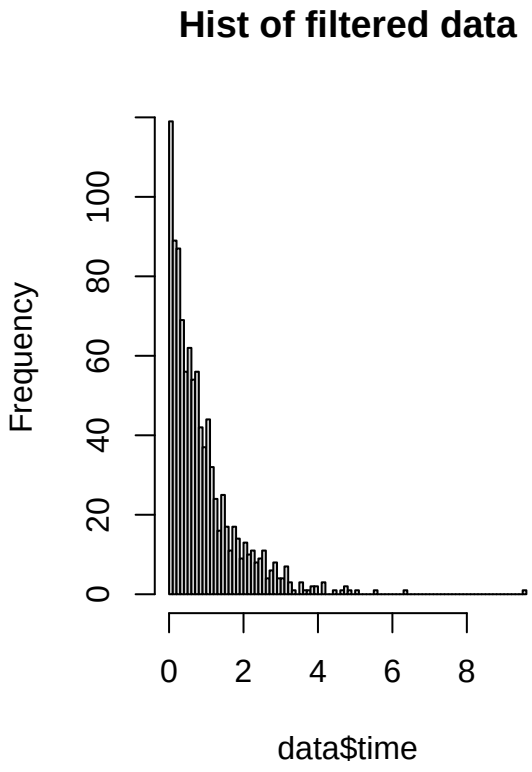
Since we did not get a reasonable plot, so we don't comment here.

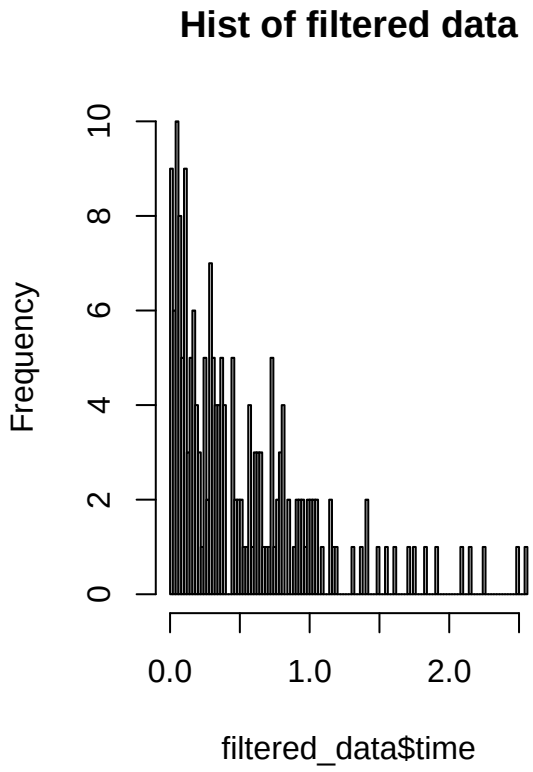
Question 2: EM algorithm (Solved by Satya Sai Naga Jaya Koushik Pilla)

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 likelihood function is same as the exponential distribution which is

$$L(x,\lambda) = \lambda e^{-\lambda x}$$

When cens=2 which means we find the failure after a period of time, the maximum time will be the check interval. The likelihood function is a truncated exponential distribution.

which is as follows.

$$L(\lambda) = \lambda e^{-\lambda x}$$

Likelihood function for the exponential distribution is as follows.

$$L(\lambda; x_1, x_2...x_n) = \prod f(x, \lambda) = \lambda^n exp(-\lambda \sum_{j=1}^{n} x_j)$$

So likelihood function for the truncated exponential distribution is as follows.

$$L(\lambda | X \leq c; x_1, x_2...x_n) = \prod P(X \leq x | X \leq c) = (\frac{\lambda^n}{c^n}) exp(-\lambda \sum_{j=1}^{n} x_j)$$

(3):

Since it's relative straight forward , we will use the likelihood function directly.

So we will derive the EM function using the likelihood function in 2.2.

E-Step:

Let's compute the expectation of likelihood as follows.

$$Q(\lambda, \lambda^t) = E(L(\lambda | X \leq c; x_1, x_2...x_n), \lambda^t)$$

M-Step:

In M Step , we need to maximum Q with respect to λ.

$$\lambda^{t+1} = argmax_\lambda Q(\lambda, \lambda^t)$$

$$Q(\lambda, \lambda^t) = E(L(\lambda | X \leq c; x_1, x_2...x_n), \lambda^t)$$

(4):

According to the output, we know that estimated lambda: 7.80785 and Number of iterations: 6

```
estep <- function(lambda, x, c) {
  return(lambda / c * exp(-lambda * x))
}

# Function to compute the M-step
mstep <- function(lambda, x, c) {
  return(sum(x) / sum(c * exp(-lambda * x)))
}

# EM algorithm
em_algorithm <- function(initial_lambda, observed_data, truncation_point, max_iter = 100, tol = 0.001) {
  lambda_current <- initial_lambda

  for (iter in 1:max_iter) {
    # E-step
    expected_values <- estep(lambda_current, observed_data, truncation_point)

    # M-step
    lambda_next <- mstep(lambda_current, observed_data, expected_values)

    # Check for convergence
```

```
    if (abs(lambda_next - lambda_current) < tol) {
      break
    }

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

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

set.seed(12345)
truncation_point <- 2

# Initial guess for lambda
initial_lambda <- 100

# Run EM algorithm
result <- em_algorithm(initial_lambda, filtered_data, truncation_point)

# Print the result
cat("Estimated lambda:", result$lambda, "\n")
```

```
## Estimated lambda: 5.547934
```

```
cat("Number of iterations:", result$iterations, "\n")
```

```
## Number of iterations: 8
```

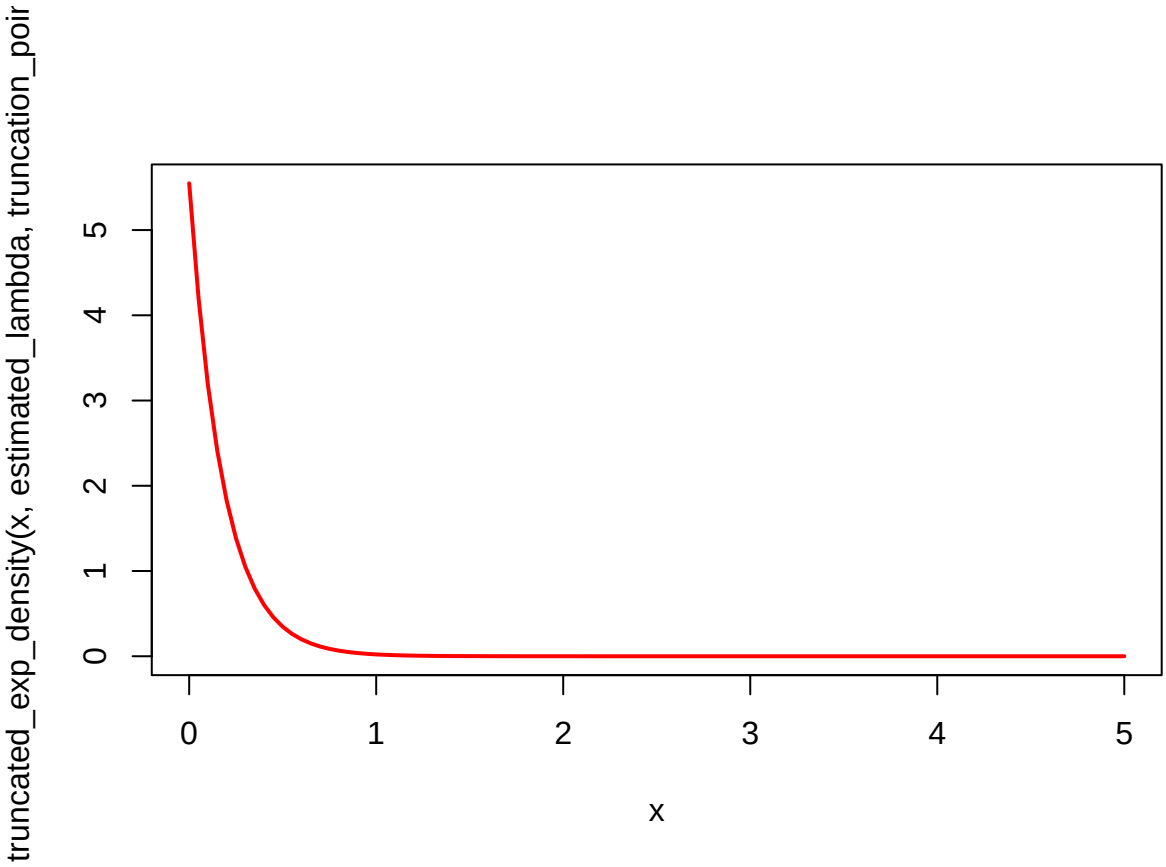
(5):

Density curve of truncated exp as follows.

```
estimated_lambda <- result$lambda

truncated_exp_density <- function(x, lambda, c) {
  return(lambda * exp(-lambda * x) / (1 - exp(-lambda * c)))
}

x_values <- seq(0, 5, length.out = 100)
curve(truncated_exp_density(x, estimated_lambda, truncation_point), col = "red", lwd = 2, add = FALSE, i
```



(6):

The code as follows.

```r
# Function to calculate MLE for truncated exponential distribution
mle_truncated_exp <- function(data, c) {
  n <- length(data)
  lambda_hat <- n / sum(data)
  return(lambda_hat)
}

# Function to generate bootstrap samples
generate_bootstrap_samples <- function(data, num_bootstrap) {
  bootstrap_samples <- vector("list", length = num_bootstrap)
  for (i in 1:num_bootstrap) {
    bootstrap_samples[[i]] <- sample(data, replace = TRUE)
  }
  return(bootstrap_samples)
}

# Function to perform MLE and parametric bootstrap
perform_bootstrap <- function(observed_data, truncation_point, num_bootstrap) {
  # MLE for original data
  lambda_hat_original <- mle_truncated_exp(observed_data, truncation_point)

  # Generate bootstrap samples
  bootstrap_samples <- generate_bootstrap_samples(observed_data, num_bootstrap)
```

```r
  # Perform MLE for each bootstrap sample
  lambda_hat_bootstrap <- sapply(bootstrap_samples, function(bootstrap_sample) {
    mle_truncated_exp(bootstrap_sample, truncation_point)
  })

  return(list(lambda_hat_original = lambda_hat_original, lambda_hat_bootstrap = lambda_hat_bootstrap))
}

# Set the number of bootstrap samples
num_bootstrap <- 1000

# Perform MLE and parametric bootstrap
bootstrap_results <- perform_bootstrap(filtered_data, truncation_point, num_bootstrap)

# Print the results
cat("Original MLE Estimate:", bootstrap_results$lambda_hat_original, "\n")
```

```
## Original MLE Estimate: 0.004196919
```

```r
cat("Mean of Bootstrap Estimates:", mean(bootstrap_results$lambda_hat_bootstrap), "\n")
```

```
## Mean of Bootstrap Estimates: 0.005351708
```

Appendix: Code for this report

```
########################## Init code for question 1 ############################
rm(list = ls())
library(ggplot2)
library(bitops)
set.seed(12345)

########################## Config Encoding Function 1.1 ############################
# First encoding: n pairs encoding
init_configuration_1 <- function(board_size = 8) {
  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(board_size = 8,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)

}

# Third Encoding: vector position encoding
init_configuration_3 <- function(board_size = 8) {
```

```
  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 <- dim(config)[2]
    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 <- length(config)
      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 <- length(config)
    mat <- matrix(0, nrow = n, ncol = n)
    for (i in 1:n) {
      mat[config[i],i] <- 1
    }
  }

  return(mat)
}

# convert  matrix to encodings
encode_config <- function(mat,method = 1) {
  n <- nrow(mat)[1]
  if (method == 1){#(x,y) encoding
    config <- data.frame(x = c(), y = c())
    for(i in 1:n){
      for(j in 1:n){
        if (mat[i,j] == 1){
          config$x <- c(config$x,i)
          config$y <- c(config$y,j)
        }
      }
    }
  }else if (method == 2) {#binary encoding
    configuration <- rep(0, n)
    # n rows
    for (i in 1:n) {
      for(j in 1:n) {
        config[i] <- config[i] + mat[i,j] * 2^(n - j)
      }
```

```
    }
  }else if (method == 3){#vector position encoding
    configuration <- rep(0, n)
    for(j in 1:n){
      for(i in 1:n){
        if (mat[i,j] == 1){
          config[j] <- i
        }
      }
    }
  }

  return(config)
}

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

  n <- dim(config)[1]
  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(board_size)
configuration
print_board(configuration, 1)

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

```
print_board(configuration, 2)

print("Test Encoding 3")
configuration <- init_configuration_3(board_size)
configuration
print_board(configuration, 3)
########################## Crossover Function ############################
crossover <- function(config1, config2,method = 1, p = 4){
  if (method == 1){
    boardsize <- ncol(config1)
  }else{
    boardsize <- length(config1)
  }

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

  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)

    if (mutate_config[col_to_mutate] + 1 > boardsize){
      mutate_config[col_to_mutate] <- 1
    }else{
      mutate_config[col_to_mutate] <- mutate_config[col_to_mutate] + 1
    }
  }

  return(mutate_config)
}

########################### Test Mutate functions ############################
print("Test Mutate encoding 3")
config1 <- init_configuration_3(board_size)
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
```

```r
# check whole config is valid or not and return
# attacked queens number and unattacked queens number
fitness <- function(config) {
  queen_num <- nrow(config)
  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(list(valid = valid,
              num_unattacked_queens = num_unattacked_queens,
              num_attacked_queens = queen_num - num_unattacked_queens))
}

fitness1 <- function(config) {
  return(fitness(config))
}

fitness2 <- function(config) {
  # convert binary encoding to (x,y) encoding
  mat <- decode_config(config, 2)
  config <- encode_config(mat, 1)
  return(fitness(config))
}

fitness3 <- function(config) {
  # convert vector position encoding to (x,y) encoding
  mat <- decode_config(config, 3)
  config <- encode_config(mat, 1)
  return(fitness(config))
}
#####################   genetic_algorithm  ###################################
genetic_algorithm <- function(method = 1,board_size=8){
  if (method == 1){
    configuration_1 <- init_configuration_1(board_size)
    val_1 <- fitness1(configuration_1)
  } else if (method == 2){
    configuration_1 <- init_configuration_2(board_size)
    val_1 <- fitness2(configuration_1)
  }else{
    configuration_1 <- init_configuration_3(board_size)
    val_1 <- fitness3(configuration_1)
  }
```

```
configuration_2 <- NULL

max_steps <- 1000
steps <- 0

num_attacked_queens_vector <- c(val_1$num_attacked_queens)

while(val_1$num_attacked_queens != 0 && steps <= max_steps ) {

  if (is.null(configuration_2)){
    if (method == 1){
      configuration_2 <- init_configuration_1(board_size)
      val_2 <- fitness1(configuration_2)
    }else if(method == 2){
      configuration_2 <- init_configuration_2(board_size)
      val_2 <- fitness2(configuration_2)
    }else{
      configuration_2 <- init_configuration_3(board_size)
      val_2 <- fitness3(configuration_2)
    }
  }

  # cross over
  child_config <- crossover(configuration_1,
                            configuration_2,
                            p = p_val)

  # mutate
  if (method == 1){
    mutated_config <- mutate(child_config,method=1)
    val_child <- fitness1(mutated_config)

  } else if (method == 2){
    mutated_config <- mutate(child_config,method=2)
    val_child <- fitness2(mutated_config)
  } else {
    mutated_config <- mutate(child_config,method=3)
    val_child <- fitness3(mutated_config)
  }

  configs <- c(1,2,3)
  num_attacked_queens <- c(val_1$num_attacked_queens,
                    val_2$num_attacked_queens,
                    val_child$num_attacked_queens)
  df <- data.frame(config = configs,
                   num_attacked_queens = num_attacked_queens)

  custom_order <- order(df$num_attacked_queens)
  sorted_df <- df[custom_order, ]

  # choose minial 2 values
  if (sorted_df$config[1] == 1){
    configuration_1 <- configuration_1
```

```
      val_1 <- val_1
    }else if(sorted_df$config[1] == 2){
      configuration_1 <- configuration_2
      val_1 <- val_2
    }else {
      configuration_1 <- mutated_config
      val_1 <- val_child
    }

    if (sorted_df$config[2] == 1){
      configuration_2 <- configuration_1
      val_2 <- val_1
    }else if(sorted_df$config[2] == 2){
      configuration_2 <- configuration_2
      val_2 <- val_2
    }else {
      configuration_2 <- mutated_config
      val_2 <- val_child
    }

    num_attacked_queens_vector <- c(num_attacked_queens_vector,val_1$num_attacked_queens)
    steps <- steps + 1

  }

  # print the queen position if found the solution
  if (val_1$num_attacked_queens == 0){
    print_board(configuration_1, 3)
  }else{
    print("not found the solution")
  }

  #df <- data.frame(steps = 1:length(num_attacked_queens_vector),
  #                 num_attacked_queens = num_attacked_queens_vector)
  #ggplot(data=df, aes(x = steps, y = num_attacked_queens)) + geom_line()

}

p_val <- 3
genetic_algorithm(method=1,board_size=8)
p_val <- 2
genetic_algorithm(method=3,board_size=4)
p_val <- 4
genetic_algorithm(method=3,board_size=8)
p_val <- 8
genetic_algorithm(method=3,board_size=16)
############################ Init code for question 1 ############################
rm(list = ls())
library(ggplot2)
set.seed(12345)
################################### (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")
estep <- function(lambda, x, c) {
  return(lambda / c * exp(-lambda * x))
}

# Function to compute the M-step
mstep <- function(lambda, x, c) {
  return(sum(x) / sum(c * exp(-lambda * x)))
}

# EM algorithm
em_algorithm <- function(initial_lambda, observed_data, truncation_point, max_iter = 100, tol = 0.001) {
  lambda_current <- initial_lambda

  for (iter in 1:max_iter) {
    # E-step
    expected_values <- estep(lambda_current, observed_data, truncation_point)

    # M-step
    lambda_next <- mstep(lambda_current, observed_data, expected_values)

    # Check for convergence
    if (abs(lambda_next - lambda_current) < tol) {
      break
    }

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

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

set.seed(12345)
truncation_point <- 2

# Initial guess for lambda
initial_lambda <- 100

# Run EM algorithm
result <- em_algorithm(initial_lambda, filtered_data, truncation_point)

# Print the result
cat("Estimated lambda:", result$lambda, "\n")
cat("Number of iterations:", result$iterations, "\n")
```

```r
estimated_lambda <- result$lambda

truncated_exp_density <- function(x, lambda, c) {
  return(lambda * exp(-lambda * x) / (1 - exp(-lambda * c)))
}

x_values <- seq(0, 5, length.out = 100)
curve(truncated_exp_density(x, estimated_lambda, truncation_point), col = "red", lwd = 2, add = FALSE, 

# Function to calculate MLE for truncated exponential distribution
mle_truncated_exp <- function(data, c) {
  n <- length(data)
  lambda_hat <- n / sum(data)
  return(lambda_hat)
}

# Function to generate bootstrap samples
generate_bootstrap_samples <- function(data, num_bootstrap) {
  bootstrap_samples <- vector("list", length = num_bootstrap)
  for (i in 1:num_bootstrap) {
    bootstrap_samples[[i]] <- sample(data, replace = TRUE)
  }
  return(bootstrap_samples)
}

# Function to perform MLE and parametric bootstrap
perform_bootstrap <- function(observed_data, truncation_point, num_bootstrap) {
  # MLE for original data
  lambda_hat_original <- mle_truncated_exp(observed_data, truncation_point)

  # Generate bootstrap samples
  bootstrap_samples <- generate_bootstrap_samples(observed_data, num_bootstrap)

  # Perform MLE for each bootstrap sample
  lambda_hat_bootstrap <- sapply(bootstrap_samples, function(bootstrap_sample) {
    mle_truncated_exp(bootstrap_sample, truncation_point)
  })

  return(list(lambda_hat_original = lambda_hat_original, lambda_hat_bootstrap = lambda_hat_bootstrap))
}

# Set the number of bootstrap samples
num_bootstrap <- 1000

# Perform MLE and parametric bootstrap
bootstrap_results <- perform_bootstrap(filtered_data, truncation_point, num_bootstrap)

# Print the results
cat("Original MLE Estimate:", bootstrap_results$lambda_hat_original, "\n")
cat("Mean of Bootstrap Estimates:", mean(bootstrap_results$lambda_hat_bootstrap), "\n")
```