732A90 Computational Statistics

Lab 6

Johannes Hedström & Mikael Montén

STIMA
Institutionen för datavetenskap
Linköpings universitet

2023-12-22

Contents

```r
library(ggplot2)
```

1 Question 1: Genetic algorithm

In this assignment, you will try to solve the n–queen problem using a genetic algorithm. Given an n by n chessboard, the task is to place n queens on it so that no queen is attacked by any other queen. You can read more about the problem at https://en.wikipedia.org/wiki/Eight_ queens_puzzle.

1.1 1

An individual in the population is a chessboard with some placement of the n queens on it. The first task is to code an individual. You are to consider three encodings for this question.

1.1.1 a

A collection (e.g., a list—but the choice of data structure is up to you) of n pairs denoting the coordinates of each queen, e.g., (5, 6) would mean that a queen is standing in row 5 and column 6. We are not using chess notation (in this case e5) as we do not want to limit n by 26 and also further work will be easier with a numerical notation

```r
chessboard_a <- function(n){

  rows <- sample(1:n,n) # sampling positions for the queens from 1:n
  cols <- sample(1:n,n)

  cbind(rows,cols)

}
```

1.1.2 b

On n numbers, where each number has log2 n binary digits—this number encodes the position of the queen in the given column. Notice that as queens cannot attack each other, in a legal configuration there can be only one queen per column. You can pad your binary representation with 0s if necessary.

```r
chessboard_b <- function(n){
  # sampling positions for the queens from 1:n
   lapply(sample(1:n,n), function(x) rev(as.numeric(intToBits(x))[1:ceiling(log2(n))]))
}
```

1.1.3 c

On n numbers, where each number is the row number of the queen in each column. Notice that this encoding differs from the previous one by how the row position is stored. Here it is an integer, in item 1b it was represented through its binary representation. This will induce different ways of crossover and mutating the state.

The tasks below, 2–7 are to be repeated for each of the three encodings above

```
chessboard_c <- function(n){

   sample(1:n,n) # sampling positions for the queens from 1:n

}
```

1.2 2

Define the function crossover(): for two chessboard layouts it creates a kid by taking columns 1, . . . , p from the first individual and columns p + 1, . . . , n from the second. Obviously, $0 < p \quad n/2$, and $p \in N$. Experiment with different values of p.

1.2.1 Encoding a

```
crossover_a <- function(board1,board2){
  p <- nrow(board1)/2
  # checking which rows in each board that contains values from 1:p and p+1:n
    idx1 <- which(board1[,2] %in% 1:p)
    idx2 <-  which(board2[,2] %in% (p+1):nrow(board2))

    kid <- rbind(board1[idx1,], board2[idx2,]) # adding the boards together
    kid

}
```

1.2.2 Encoding b

```
crossover_b <- function(board1,board2){
  p <- ceiling(log2(length(board1))/2) # getting p
  l <- list()
  for(i in 1:length(board1)){

    l[[i]] <- c(board1[[i]][1:p] ,board2[[i]][(p+1):log2(length(board2))]) # adding the first 1:p and p+

  }
  l
}
```

1.2.3 Encoding c

```
crossover_c <- function(board1,board2){
  p <- length(board1)/2 # finding the p value
  c(board1[1:p],board2[(p+1):length(board2)])  # adding the 2 boards together

}
```

1.3 3

Define the function mutate() that randomly moves a queen to a new position

1.3.1 Encoding a

```
mutate_a <- function(board, prob){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:nrow(board),1) # sampling which queen to switch

    pos <- sample((1:nrow(board)),1) # sampling a position for the sampled queen

    while (pos==board[idx,1]) {
      pos <- sample((1:nrow(board)),1)
    }

    board[idx,1] <- pos # adding the position to the board

    board
  }
}
```

1.3.2 Encoding b

```
mutate_b <- function(board, prob=1){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:length(board),1) # sampling which queen to switch

    pos <- as.numeric(intToBits(sample((1:length(board)),1))[1:ceiling(log2(length(board)))]) # sampling
```

```r
    while (all(pos == board[[idx]])) { # Checking that its a new queen
      pos <- as.numeric(intToBits(sample((1:length(board)),1))[1:ceiling(log2(length(board)))])
    }

    board[[idx]] <- pos # adding the position to the board

    board
  }
}
```

1.3.3 Encoding c

```r
mutate_c <- function(board, prob){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:length(board),1) # sampling which queen to switch

    pos <- sample((1:length(board)),1) # sampling a position for the sampled queen
    while (pos==board[idx]) {
      pos <- sample((1:length(board)),1)
    }

    board[idx] <- pos # adding the position to the board

    board
  }
}
```

1.4 4

Define a fitness function for a given configuration. Experiment with three: binary—is a solution or not;number of queens not attacked; $\begin{bmatrix} n \\ 2 \end{bmatrix}$-number) of pairs of queens attacking each other. If needed scale the value of the fitness function to [0, 1]. Experiment which could be the best one. Try each fitness function for each encoding method. You should not expect the binary fitness function to work well, explain why this is s

1.4.1 Encoding a

```r
fitnes_a_unkilled <- function(board) {
  unattacked_queens <- nrow(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) { # looping through every queen
   kill <- 0
    for (j in (i + 1):nrow(board)) {
      if (queens_attack(board[i, ], board[j, ])) { # checking if they can be killed
        kill <- 1

        break() # queen can be killed, dont need to check against other queens
      }
      }
   if(kill == 1){ # if killed then unattacked -1
     unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
}




fitnes_a_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) {  # looping through every queen
    for (j in (i + 1):nrow(board)) {
      if (queens_can_attack(board[i, ], board[j, ])) { # checking if they can be killed
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}



fitnes_a_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):nrow(board)) {
      if (queens_attack(board[i, ], board[j, ])) { # checking if they can be killed
        solution <- 0

        break() # queen can be killed, dont need to check against other queens
      }

    }
    if(solution == 0){
      break()
    }
  }

  solution
}
```

1.4.2 Encoding b

```
# Function to convert binary to integer
bitsToInt <- function(bits) {

  sum(bits * 2^(0:(length(bits)-1))) + 1 # Adding 1 as we must have log2(n) numbers and 1 is then repres
}
```

```r
fitnes_b_unkilled <- function(board) {
  unattacked_queens <- length(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can be k
        kill <- 1

        break() # queen can be killed, dont need to check against other queens
      }
    }
    if(kill == 1){ # if killed then unattacked -1
      unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
}


fitnes_b_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) {  # looping through every queen
    for (j in (i + 1):length(board)) {
      if (queens_can_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can
```

```
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}


fitnes_b_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can be k
        solution <- 0 # if its not a solution
        break()
      }
    }
    if(solution==0){
      break()
    }

  }
  solution
}
```

1.4.3 Encoding c

```
fitnes_c_unkilled <- function(board) {
  unattacked_queens <- length(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)
```

```r
    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(board[i], board[j],i,j)) { # checking if they can be killed
        kill <- 1

        break() # queen can be killed, dont need to check against other queens
      }
    }
    if(kill == 1){ # if killed then unattacked -1
      unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
}



fitnes_c_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) {  # looping through every queen
    for (j in (i + 1):length(board)) {
      if (queens_can_attack(board[i], board[j],i,j)) { # checking if they can be killed
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}
```

```
fitnes_c_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(board[i], board[j],i,j)) { # checking if they can be killed
        solution <- 0 # if its not a solution
        break()
      }
    }
     if(solution==0){
       break()
       }

  }
  solution
}
```

1.5 5

Implement a genetic algorithm that takes the choice of encoding, mutation probability, and fitness function as parameters. Your implementation should start with a random initial configuration. Each element of the population should have its fitness calculated. Do not forget to have in your code a limit for the number of iterations (but this limit should not be lower than 100, unless this causes running time issues, which should be clearly presented then), so that your code does not run forever. Count the number of pairs of queens attacking each other. At each iteration

1.5.1 a

Two individuals are randomly sampled from the current population, they are further used as parents (use sample()).

1.5.2 b

One individual with the smallest fitness is selected from the current population, this will be the victim (use order()).

1.5.3 c

The two sampled parents are to produce a kid by crossover, and this kid should be mutated with probability mutprob (use crossover(), mutate()).

1.5.4 d

The victim is replaced by the kid in the population.

1.5.5 e

Do not forget to update the vector of fitness values of the population.

1.5.6 f

Remember the number of pairs of queens attacking each other at the given iteration.

1.6 6

If found return the legal configuration of queens.

1.7 7

Provide a plot of the number of pairs queens attacking each other at each iteration of the algorithm.

1.7.1 Encoding a, fitnes = pairs

```
# pairs population
pop_a_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}



## 7
genetic_a_pairs <- function(n, iter,pop_size,mutprob){
```

```r
  pop <- pop_a_p(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
 while(i <= iter){

   # a sampling two boards
   b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
   b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
   parent1 <- pop$pop[b1][[1]]
   parent2 <-  pop$pop[b2][[1]]

   victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

   # Doing the crossover, mutate and fitness step
   kid <- crossover_a(parent1,parent2)

   kid <- mutate_a(kid, mutprob)

   fit <- fitnes_a_pairs(kid)

   pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

   pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit



   pair_queens <- c(pair_queens,fit)

   if(fit == 0){
     print(fit)
     print(kid)

     mat <- matrix(0,n,n)

     for (j in 1:n) {
       mat[kid[j,1],kid[j,2]] <- 1

     }
     print(mat)
     i <- i + 1
     break("Legal configuration found")
   }

   i <- i + 1
 }

  # Plotting the number of pairs of queens attacking each other at each iteration
```

```
    plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pai
}
```

1.7.2 Encoding a, fitnes = unkilled

```
# unkilled population
pop_a_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_a_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_a_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_a(parent1,parent2)

    kid <- mutate_a(kid, mutprob)

    fit <- fitnes_a_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_a_pairs(kid)
    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j,1],kid[j,2]] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.3 Encoding a, fitnes = binary

```
# binary population
pop_a_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_a_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_a_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){
```

```r
    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]


    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim


    #
    kid <- crossover_a(parent1,parent2)

    kid <- mutate_a(kid, mutprob)

    fit <- fitnes_a_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_a_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j,1],kid[j,2]] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.4 Encoding b, fitness = pairs

```r
pop_b_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

## 7

genetic_b_pairs <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_p(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

    # Doing the crossover, mutate and fitness step
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_pairs(kid)

    pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit



    pair_queens <- c(pair_queens,fit)

    if(fit == 0){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.5 Encoding b, fitness = unkilled

```r
pop_b_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_b_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_b_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.6 Encoding b, fitness = binary

```
pop_b_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
```

```r
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_b_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]


    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim


    #
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_b_pairs(kid)

    pair_queens <- c(pair_queens,pair)


    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)
      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.7 Encoding c, fitness = pairs

```
# pairs population
pop_c_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

## 7


genetic_c_pairs <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_p(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]
    victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

    # Doing the crossover, mutate and fitness step
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)

    fit <- fitnes_c_pairs(kid)

    pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit



    pair_queens <- c(pair_queens,fit)

    if(fit == 0){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.8 Encoding c, fitness = unkilled

```
# Unkilled population
pop_c_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_c_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)

    fit <- fitnes_c_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_c_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.7.9 Encoding c, fitness = binary

```
# binary population
pop_c_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_c_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim

    #
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)
    fit <- fitnes_c_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_c_pairs(kid)

    pair_queens <- c(pair_queens,pair)


    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}
```

1.8 8

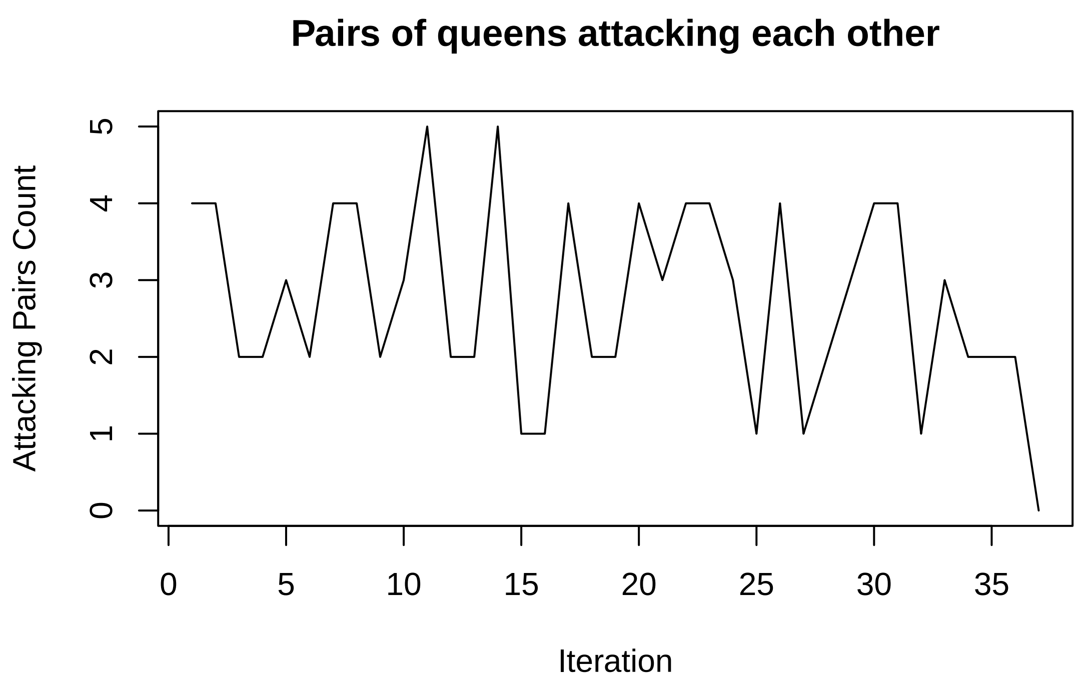
Run your code for n = 4, 8, 16 (if n = 16 requires too much computational time take a different n ∈ {10, 11, 12, 13, 14, 15}, but do not forget that this is not a power of 2 and more care is needed in the second encoding), the different encodings, objective functions, and mutprob= 0.1, 0.5, 0.9. Did you find a legal state?

Only showing 1 fitness per encoding to save space in the pdf, but all the encodings work with all fitness's.

1.8.1 Encoding a

```
###4
#genetic_a_unkilled(4,200,10,0.1)
genetic_a_unkilled(4,200,10,0.5)
```

```
## [1] 4
##      rows cols
## [1,]    2    1
## [2,]    4    2
## [3,]    1    3
## [4,]    3    4
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    1    0
## [2,]    1    0    0    0
## [3,]    0    0    0    1
## [4,]    0    1    0    0
```



```
#genetic_a_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_a_unkilled(8,200,10,0.1)
#genetic_a_unkilled(8,200,10,0.5)
#genetic_a_unkilled(8,200,10,0.9)

### 16
```

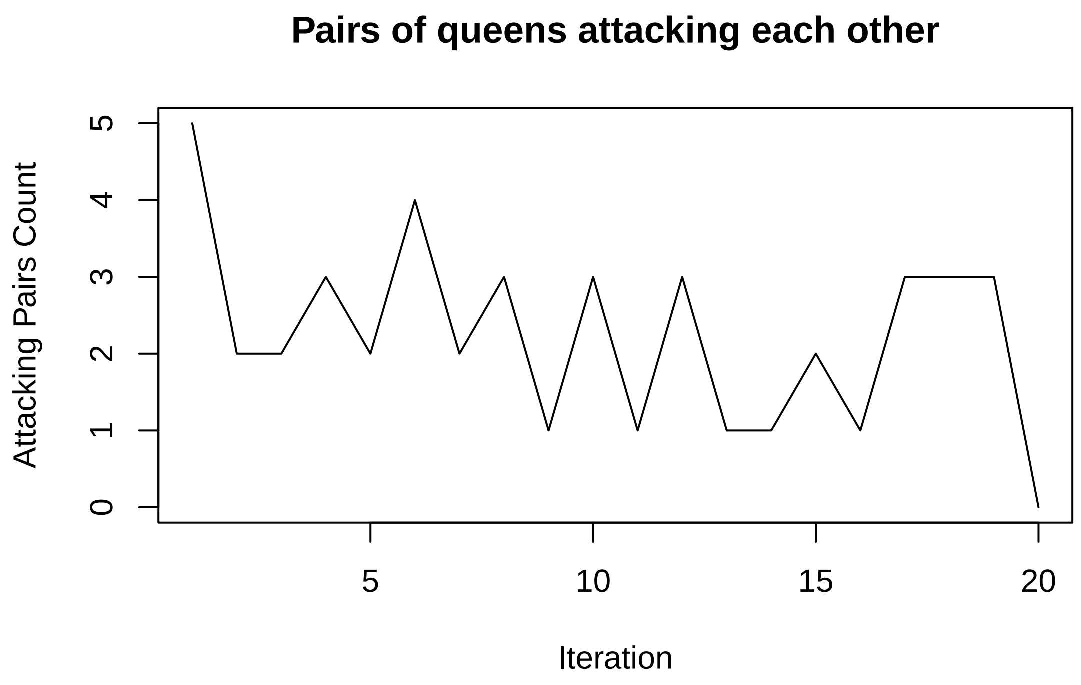
```
#genetic_a_unkilled(16,200,10,0.1)
#genetic_a_unkilled(16,200,10,0.5)
#genetic_a_unkilled(16,200,10,0.9)

###4
#genetic_a_pairs(4,200,10,0.1)
#genetic_a_pairs(4,200,10,0.5)
#genetic_a_pairs(4,200,10,0.9)

### 8
#genetic_a_pairs(8,200,10, 0.1)
#genetic_a_pairs(8,200,10, 0.5)
#genetic_a_pairs(8,200,10, 0.9)

### 16
#genetic_a_pairs(16,200,10,0.1)
#genetic_a_pairs(16,200,10,0.5)
#genetic_a_pairs(16,200,10,0.9)


###4
#genetic_a_binary(4,200,10,0.1)
#genetic_a_binary(4,200,10,0.5)
#genetic_a_binary(4,200,10,0.9)

### 8
#genetic_a_binary(8,200,10, 0.1)
#genetic_a_binary(8,200,10, 0.5)
#genetic_a_binary(8,200,10, 0.9)

### 16
#genetic_a_binary(16,200,10,0.1)
#genetic_a_binary(16,200,10,0.5)
#genetic_a_binary(16,200,10,0.9)
```

1.8.2 Encoding b

```
###4
#genetic_b_unkilled(4,200,10,0.1)
#genetic_b_unkilled(4,200,10,0.5)
#genetic_b_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_b_unkilled(8,200,10,0.1)
#genetic_b_unkilled(8,200,10,0.5)
```

```r
#genetic_b_unkilled(8,200,10,0.9)

### 16
#genetic_b_unkilled(16,200,10,0.1)
#genetic_b_unkilled(16,200,10,0.5)
#genetic_b_unkilled(16,200,10,0.9)

###4
#genetic_b_pairs(4,200,10,0.1)
genetic_b_pairs(4,200,10,0.5)
```

```
## [1] 0
## [[1]]
## [1] 1 0
##
## [[2]]
## [1] 1 1
##
## [[3]]
## [1] 0 0
##
## [[4]]
## [1] 0 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    1    0
## [2,]    1    0    0    0
## [3,]    0    0    0    1
## [4,]    0    1    0    0
```



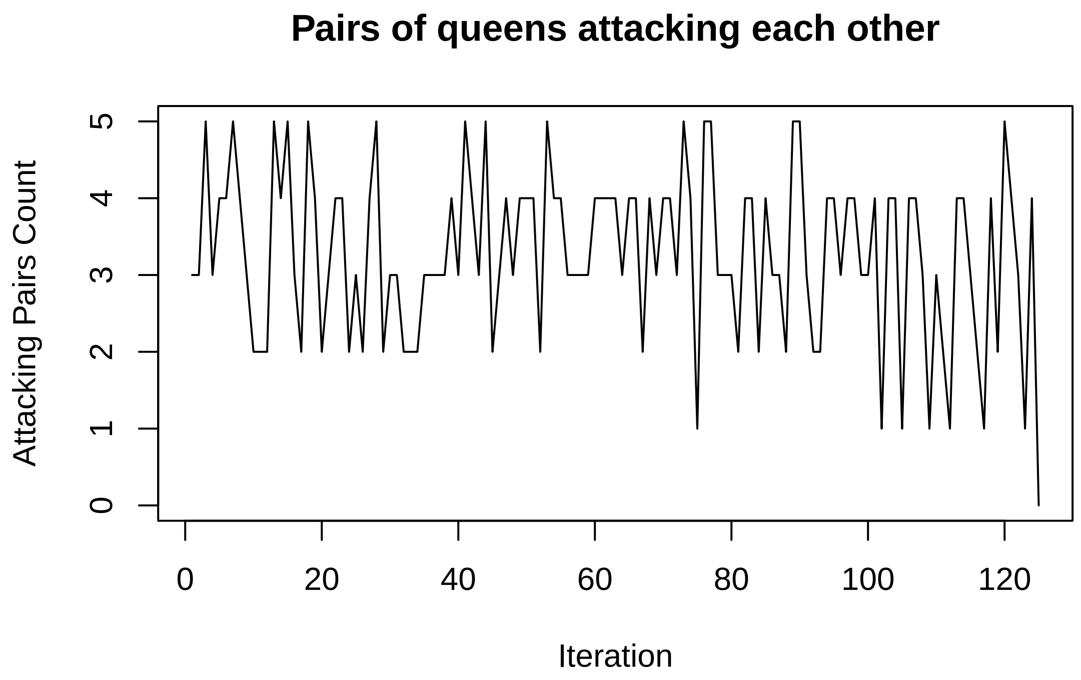
```
#genetic_b_pairs(4,200,10,0.9)

### 8
#genetic_b_pairs(8,200,10, 0.1)
#genetic_b_pairs(8,200,10, 0.5)
#genetic_b_pairs(8,200,10, 0.9)

### 16
#genetic_b_pairs(16,200,10,0.1)
#genetic_b_pairs(16,200,10,0.5)
#genetic_b_pairs(16,200,10,0.9)

###4
#genetic_b_binary(4,200,10,0.1)
#genetic_b_binary(4,200,10,0.5)
#genetic_b_binary(4,200,10,0.9)

### 8
#genetic_b_binary(8,200,10, 0.1)
#genetic_b_binary(8,200,10, 0.5)
#genetic_b_binary(8,200,10, 0.9)

### 16
#genetic_b_binary(16,200,10,0.1)
```

```
#genetic_b_binary(16,200,10,0.5)
#genetic_b_binary(16,200,10,0.9)
```

1.8.3 Encoding c

```
###4
#genetic_c_unkilled(4,200,10,0.1)
#genetic_c_unkilled(4,200,10,0.5)
#genetic_c_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_c_unkilled(8,200,10,0.1)
#genetic_c_unkilled(8,200,10,0.5)
#genetic_c_unkilled(8,200,10,0.9)

### 16
#genetic_c_unkilled(16,200,10,0.1)
#genetic_c_unkilled(16,200,10,0.5)
#genetic_c_unkilled(16,200,10,0.9)

###4
#genetic_c_pairs(4,200,10,0.1)
#genetic_c_pairs(4,200,10,0.5)
#genetic_c_pairs(4,200,10,0.9)

### 8
#genetic_c_pairs(8,200,10, 0.1)
#genetic_c_pairs(8,200,10, 0.5)
#genetic_c_pairs(8,200,10, 0.9)

### 16
#genetic_c_pairs(16,200,10,0.1)
#genetic_c_pairs(16,200,10,0.5)
#genetic_c_pairs(16,200,10,0.9)

###4
#genetic_c_binary(4,200,10,0.1)
genetic_c_binary(4,200,10,0.5)
```

```
## [1] 1
## [1] 2 4 1 3
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    1    0
## [2,]    1    0    0    0
## [3,]    0    0    0    1
## [4,]    0    1    0    0
```



```
#genetic_c_binary(4,200,10,0.9)

### 8
#genetic_c_binary(8,200,10, 0.1)
#genetic_c_binary(8,200,10, 0.5)
#genetic_c_binary(8,200,10, 0.9)

### 16
#genetic_c_binary(16,200,10,0.1)
#genetic_c_binary(16,200,10,0.5)
#genetic_c_binary(16,200,10,0.9)
```

1.9 9

Discuss which encoding and objective function worked best.

1.9.1 Encoding

The best encoding is encoding c, and the worst is encoding a. Only saving the rows works best as it reduces the memory saved and only has to go through 1 value instead of 2. We've encoded the bit encoding in the same way but as that needs to be transformed back and forward from binary to integer it takes a little bit longer, and its saved in a list and not a vector which also increases the calculation time.

1.9.2 Fitness

The worst objective is the binary, it can be fast to calculate the fitness but we don't get any information of improvement of our population and we hit a solution by chance as we don't know which is the best or worst in the population when taking a "victim".

Sometime the binary can find the correct solution super fast and in very few iterations, but we think that this is more based on "luck" and works best for small boards like 4x4.

Number of queens not attacked is the best as its in our implementation as its a bit quicker to run than pairs of queens even if pairs of queens can take less iterations, but pairs of queens get you more information of the fitness which then can help you to a faster improvement.

So in term of which is the best, considering running time, the number of unkilled queens is best, but in how many iterations it took to find a solution, the pairs of queen is the best.

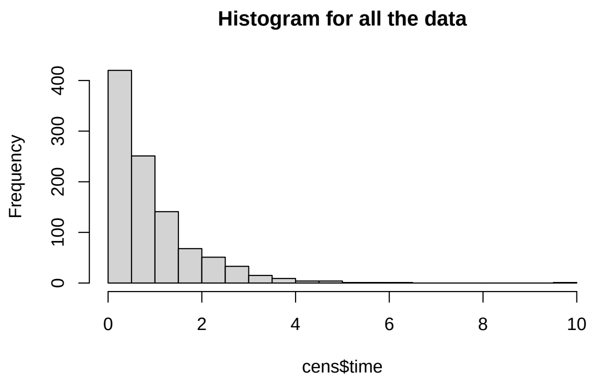
2 Question 2: EM algorithm

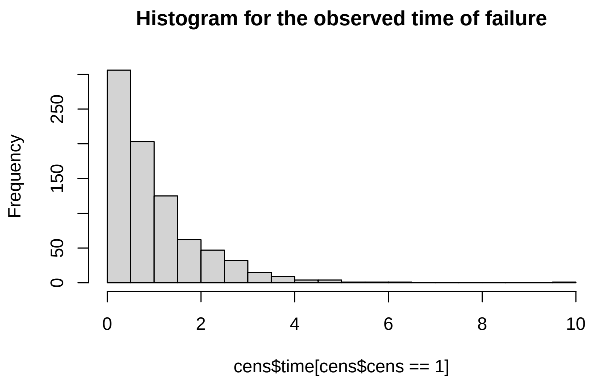
The data file censoredproc.csv contains the time after which a certain product fails. Some of these measurements are left-censored (cens=2)—i.e., we did not observe the time of failure, only that the product had already failed when checked upon. Status cens=1 means that the exact time of failure was observed.

2.1 1

Plot a histogram of the values. Do it for all of the data, and also when the censored observations are removed. Do the histograms remind of an exponential distribution?

```r
# loading the data
cens <- read.csv2('censoredproc.csv')

cens$time <- as.numeric(cens$time)

hist(cens$time, breaks=30 ,main='Histogram for all the data')
hist(cens$time[cens$cens==1], breaks=30 ,main='Histogram for the observed time of failure')
```





Both histograms are skewed to the right and the histogram over all the data looks very close to the exponential distribution.

2.2 2

Assume that the underlying data comes from an exponential distribution with parameter λ. This means that observed values come from the exponential λ distribution, while censored from a truncated exponential distribution. Write down the likelihood function.

The exponential distributon has CDF

$$P(X \leq x) = 1 - e^{-\lambda \cdot x}$$

. ("Exponential Distribution" 2023)

Which has likelihood

$$L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$

The truncated exponential distribution has CDF

$$P(X \leq x | X \leq c) = \frac{1 - e^{-\lambda \cdot x}}{1 - e^{-\lambda \cdot c}}$$

Which has likelihood

$$L(\lambda) = \prod_{j=1}^{m} \frac{\lambda e^{-\lambda z_j}}{1 - e^{-\lambda c_j}}$$

The combined CDF is

$$F_{x \leq c}(X) = 1 - e^{-\lambda \cdot x} \cdot \frac{1 - e^{-\lambda \cdot x}}{1 - e^{-\lambda \cdot c}}$$

Thus, the combined likelihood is

$$L_{x \leq c}(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} \prod_{j=1}^{m} \frac{\lambda e^{-\lambda z_j}}{1 - e^{-\lambda c_j}}$$

The log likelihood is

$$\log L_{x\leq c}(\lambda) = \sum_{i=1}^{n}(\log\lambda - \lambda x_i) + \sum_{j=1}^{m}\left(\log\lambda - \lambda z_j - \log\left(1 - e^{-\lambda c_j}\right)\right)$$

$$= n\ \log\lambda - \lambda\sum_{i=1}^{n} x_i + m\ \log\lambda - \lambda\sum_{j=1}^{m} z_j - \sum_{j=1}^{m}\log(1 - e^{-\lambda c_j})$$

The latent variable Z is the only variable that is affected by λ^k, which means the complete function is,

$$= n\ \log\lambda - \lambda\sum_{i=1}^{n} x_i + m\ \log\lambda^k - \lambda^k\sum_{j=1}^{m} z_j - \sum_{j=1}^{m}\log(1 - e^{-\lambda^k c_j})$$

("Expecatation-Maximization Algorithm" 2023)

2.3 3

The goal now is to derive an EM algorithm that estimates λ. Based on the above found likelihood function, derive the EM algorithm for estimating λ. The formula in the M–step can be differentiated, but the derivative is non–linear in terms of λ so its zero might need to be found numerically.

Our aim is here to derive

$$Q(\lambda, \lambda^k) = E[n\ \log\lambda - \lambda\sum_{i=1}^{n} x_i + m\ \log\lambda^k - \lambda^k\sum_{j=1}^{m} z_j - \sum_{j=1}^{m}\log(1 - e^{-\lambda^k c_j})|\lambda^k, Y] =$$

$$= n\ \log\lambda - \lambda\sum_{i=1}^{n} x_i + m\ \log\lambda^k - \lambda^k\ E[\sum_{j=1}^{m} z_j] - \sum_{j=1}^{m}\log(1 - e^{-\lambda^k c_j})$$

λ^k is known as it is the current estimated value for lambda, or the starting point when $k = 0$, and therefore we only have to maximize over λ. x_i, c_j is already observed and λ is not a random variable which is why we only need to compute the expected value for z_j

z_j is our latent observations and $E[Z;Y] = \frac{\int_0^y zf(z)\ dz}{F(y)} = (1 - e^{-\lambda y})^{-1}\int_0^y z\lambda e^{-\lambda z}dz = (1/(1 - e^{-\lambda * z})) * ((1/\lambda) - (z + (1/\lambda)) * e^{-\lambda * z})$ ("Truncated Distribution" 2023).

2.4 4

Implement the above in R. Take $\lambda_0 = 100$ as the starting value for the algorithm and stopping condition if the change in the estimate is less than 0.001. At what $\hat{\lambda}$ did the EM algorithm stop at? How many iterations were required?

```r
# starting values
m <- length(cens$time[cens$cens==2]) # number of censored product
n <- length(cens$time[cens$cens==1])
x  <-  cens$time[cens$cens==1]
```

```r
z  <- cens$time[cens$cens==2]
lambda <- 100
improv <- 0.001
max_iter <- 1000
iter <- 1
L <-c(-Inf)

# EM algorithm
Exp_llik <- function(lambda){
  (n * log(lambda)) - lambda * sum(x) +
  m * log(lambda) - (lambda * sum(q)) -
  (sum(log(1-exp(-lambda*z))))

}


# looping until
while(iter < max_iter){

   # Expectations
 q <- (1 /(1-exp(-lambda * z))) * ((1/lambda) - (z + (1/lambda)) * exp(-lambda * z))

 # loglikelihood
  L <- c(L,Exp_llik(lambda))

  if(abs(L[iter+1] - L[iter]) <= improv ){
    cat(" Lambda = ", lambda,"\n","Log likelihood = " ,L[iter+1] )# stopping criterion met
    break
  }
 # maximization

 # updating lambda numerically, using bfgs as the derivative can be found
   lambda <- optim(lambda,Exp_llik,method="BFGS", control = list(fnscale=-1))[[1]]

 iter <- iter+1
}
```

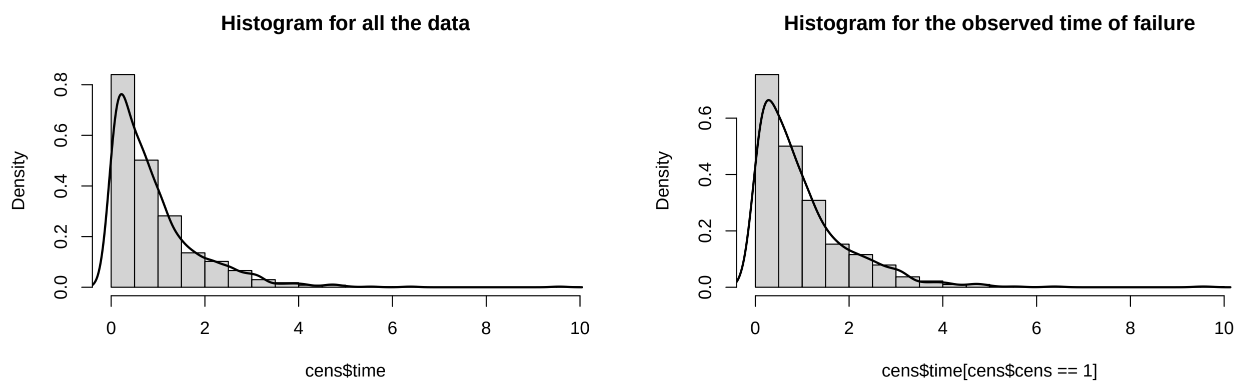
The algorithm stopped at $\hat{\lambda} = 1.0047$ and it took the algorithm 5 iterations to converge.

2.5 5

Plot the density curve of the $exp(\hat{\lambda})$ distribution over your histograms in task 1

```r
lambda <- 1.0047575694422
hist(cens$time, breaks=30 ,main='Histogram for all the data', probability = TRUE)
lines(density(cens$time, adjust = 1 / lambda), lwd = 2)
```

```r
hist(cens$time[cens$cens==1], breaks=30 ,main='Histogram for the observed time of failure', probability
lines(density(cens$time[cens$cens==1], adjust = 1 / lambda), lwd = 2)
```



2.6 6

Study how good your EM algorithm is compared to usual maximum likelihood estimation with data reduced to only the uncensored observations. To this end we will use a parametric bootstrap. Repeat 1000 times the following procedure:

2.6.1 a

Simulate the same number of data points as in the original data, from the exponential $\hat{\lambda}$ distribution. lambda

2.6.2 b

Randomly select the same number of points as in the original data for censoring. For each observation for censoring—sample a new time from the uniform distribution on [0,true time]. Remember that the observation was censored.

2.6.3 c

Estimate λ both by your EM-algorithm, and maximum likelihood based on the uncensored observations.

Bootstrap is done as follows

```r
bs_df <- matrix(NA, ncol = 2, nrow = 1000)
colnames(bs_df) <- c("time", "cens")

lambda_hat_mle <- c()
```

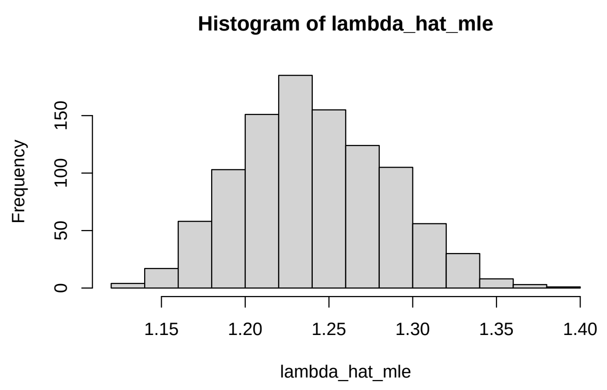
```r
m <- 189 # number of censored product
n <- 1000

# bootstrap
for (i in 1:n) {
  # a)
  bs_df[,1] <- rexp(n, rate = lambda)
  bs_df[,2] <- 0

  # b)
  bs_df[1:m,1] <- runif(m, min = 0, max = max(cens$time[cens$cens==2]))
  bs_df[1:m,2] <- 1

  # c)
  # estimate lambda by MLE on uncensored observations
  lambda_hat_mle[i] <- n/sum(bs_df[(m+1):n,1])
}
```

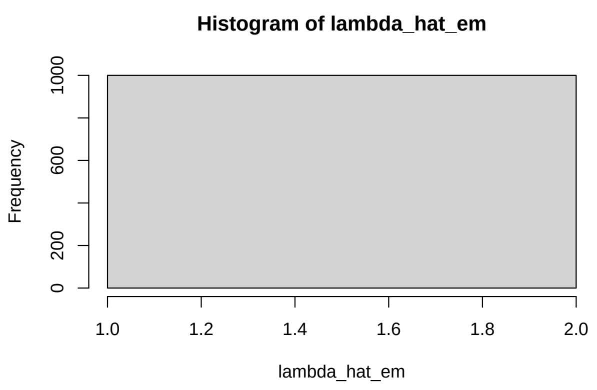
```r
bs_df <- as.data.frame(bs_df)
lambda_hat_em <- c()
for(i in 1:1000){
  # starting values
m <- length(bs_df$time[bs_df$cens==1]) # number of censored product
n <- length(bs_df$time[bs_df$cens==0])
x  <-  bs_df$time[bs_df$cens==0]
z  <- bs_df$time[bs_df$cens==1]
lambda <- 100
improv <- 0.001
max_iter <- 1000
iter <- 1
L <-c(-Inf)

# EM algorithm
Exp_llik <- function(lambda){
  (n * log(lambda)) - lambda * sum(x) +
  m * log(lambda) - (lambda * sum(q)) -
  (sum(log(1-exp(-lambda*z))))

}


# looping until
while(iter < max_iter){

   # Expectations
 q <- (1 /(1-exp(-lambda * z))) * ((1/lambda) - (z + (1/lambda)) * exp(-lambda * z))

 # loglikelihood
  L <- c(L,Exp_llik(lambda))
  if(abs(L[iter+1] - L[iter]) <= improv ){
    cat(" Lambda = ", lambda,"\n","Log likelihood = " ,L[iter+1] )# stopping criterion met
    break
  }
 # maximization

 # updating lambda numerically, using bfgs as the derivative can be found
   lambda <- optim(lambda,Exp_llik,method="BFGS", control = list(fnscale=-1))[[1]]
   lambda_hat_em[i] <- lambda

 iter <- iter+1
}
}
```

```
##  Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026 Lambda =  1.051917
##  Log likelihood =  -745.0026
```

Compare the distributions of the estimates of λ from the two methods. Plot the histograms, report whether they both seem unbiased, and what is the variance of the estimators.

```r
hist(lambda_hat_mle)
hist(lambda_hat_em)
```





```
var(lambda_hat_mle)
```

```
## [1] 0.001927185
```

```
var(lambda_hat_em)
```

```
## [1] 0
```

3 References

"Expecatation-Maximization Algorithm." 2023. Wikipedia. https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm.

"Exponential Distribution." 2023. Wikipedia. https://en.wikipedia.org/wiki/Exponential_distribution.

"Truncated Distribution." 2023. Wikipedia. https://en.wikipedia.org/wiki/Truncated_distribution.

4 Appendix

```r
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning=FALSE, fig.width = 6, fig.height = 4)
library(ggplot2)
chessboard_a <- function(n){

  rows <- sample(1:n,n) # sampling positions for the queens from 1:n
  cols <- sample(1:n,n)

  cbind(rows,cols)

}




chessboard_b <- function(n){
  # sampling positions for the queens from 1:n
   lapply(sample(1:n,n), function(x) rev(as.numeric(intToBits(x))[1:ceiling(log2(n))]))
}



chessboard_c <- function(n){

   sample(1:n,n) # sampling positions for the queens from 1:n

}



crossover_a <- function(board1,board2){
  p <- nrow(board1)/2
  # checking which rows in each board that contains values from 1:p and p+1:n
    idx1 <- which(board1[,2] %in% 1:p)
    idx2 <-  which(board2[,2] %in% (p+1):nrow(board2))

    kid <- rbind(board1[idx1,], board2[idx2,]) # adding the boards together
    kid
```

```
}

crossover_b <- function(board1,board2){
  p <- ceiling(log2(length(board1))/2) # getting p
  l <- list()
  for(i in 1:length(board1)){

    l[[i]] <- c(board1[[i]][1:p] ,board2[[i]][(p+1):log2(length(board2))]) # adding the first 1:p and p+

  }
  l
}



crossover_c <- function(board1,board2){
  p <- length(board1)/2 # finding the p value
  c(board1[1:p],board2[(p+1):length(board2)])  # adding the 2 boards together

}


mutate_a <- function(board, prob){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:nrow(board),1) # sampling which queen to switch

    pos <- sample((1:nrow(board)),1) # sampling a position for the sampled queen

    while (pos==board[idx,1]) {
      pos <- sample((1:nrow(board)),1)
    }

    board[idx,1] <- pos # adding the position to the board

    board
  }
}

mutate_b <- function(board, prob=1){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:length(board),1) # sampling which queen to switch

    pos <- as.numeric(intToBits(sample((1:length(board)),1))[1:ceiling(log2(length(board)))]) # sampling
    while (all(pos == board[[idx]])) { # Checking that its a new queen
      pos <- as.numeric(intToBits(sample((1:length(board)),1))[1:ceiling(log2(length(board)))])
    }

    board[[idx]] <- pos # adding the position to the board

    board
  }
}


mutate_c <- function(board, prob){
  if(runif(1) > prob){
    board
  }else{

    idx <- sample(1:length(board),1) # sampling which queen to switch

    pos <- sample((1:length(board)),1) # sampling a position for the sampled queen
    while (pos==board[idx]) {
      pos <- sample((1:length(board)),1)
    }

    board[idx] <- pos # adding the position to the board

    board
  }
}


fitnes_a_unkilled <- function(board) {
  unattacked_queens <- nrow(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) { # looping through every queen
   kill <- 0
    for (j in (i + 1):nrow(board)) {
      if (queens_attack(board[i, ], board[j, ])) { # checking if they can be killed
        kill <- 1
        break() # queen can be killed, dont need to check against other queens
      }
      }
   if(kill == 1){ # if killed then unattacked -1
     unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
}




fitnes_a_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) {  # looping through every queen
    for (j in (i + 1):nrow(board)) {
      if (queens_can_attack(board[i, ], board[j, ])) { # checking if they can be killed
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}



fitnes_a_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1[1] == queen2[1]
    same_column <- queen1[2] == queen2[2]
    same_diagonal <- abs(queen1[1] - queen2[1]) == abs(queen1[2] - queen2[2])

    return(same_row || same_column || same_diagonal)
  }

  for (i in 1:(nrow(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):nrow(board)) {
      if (queens_attack(board[i, ], board[j, ])) { # checking if they can be killed
        solution <- 0

        break() # queen can be killed, dont need to check against other queens
      }

    }
    if(solution == 0){
      break()
    }
  }

  solution
}


# Function to convert binary to integer
bitsToInt <- function(bits) {

  sum(bits * 2^(0:(length(bits)-1))) + 1 # Adding 1 as we must have log2(n) numbers and 1 is then repres
}



fitnes_b_unkilled <- function(board) {
  unattacked_queens <- length(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can be k
```

```r
        kill <- 1

        break() # queen can be killed, dont need to check against other queens
      }
    }
    if(kill == 1){ # if killed then unattacked -1
      unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
}


fitnes_b_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) {  # looping through every queen
    for (j in (i + 1):length(board)) {
      if (queens_can_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}


fitnes_b_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)
    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(bitsToInt(board[[i]]), bitsToInt(board[[j]]),i,j)) { # checking if they can be k
        solution <- 0 # if its not a solution
        break()
      }
    }
    if(solution==0){
      break()
    }

  }
  solution
}


fitnes_c_unkilled <- function(board) {
  unattacked_queens <- length(board)  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(board[i], board[j],i,j)) { # checking if they can be killed
        kill <- 1

        break() # queen can be killed, dont need to check against other queens
      }
    }
    if(kill == 1){ # if killed then unattacked -1
      unattacked_queens <- unattacked_queens - 1
    }


  }

  unattacked_queens
```

```
}



fitnes_c_pairs <- function(board) {
  attacking_pairs <- 0 # starting the count of attacking pairs

  # Function to check if two queens can attack each other
  queens_can_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) {  # looping through every queen
    for (j in (i + 1):length(board)) {
      if (queens_can_attack(board[i], board[j],i,j)) { # checking if they can be killed
        attacking_pairs <- attacking_pairs + 1
      }
    }
  }

  attacking_pairs
}


fitnes_c_binary <- function(board) {
  solution <- 1  # starting the count

  # Function to check if two queens can attack each other
  queens_attack <- function(queen1, queen2,i,j) {
    # Queens can attack each other if they are in the same row, column, or diagonal
    same_row <- queen1 == queen2
    same_diagonal <- abs(queen1 - queen2) == abs(i - j)

    return(same_row  | same_diagonal)
  }

  for (i in 1:(length(board) - 1)) { # looping through every queen
    kill <- 0
    for (j in (i + 1):length(board)) {
      if (queens_attack(board[i], board[j],i,j)) { # checking if they can be killed
        solution <- 0 # if its not a solution
        break()
      }
    }
     if(solution==0){
       break()
       }

  }
  solution
}

# pairs population
pop_a_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}



## 7
genetic_a_pairs <- function(n, iter,pop_size,mutprob){
     pop <- pop_a_p(n,pop_size)

     fitness_vec <- c() # fitness vector
     i <- 0 # iterations
     pair_queens <- c()
    while(i <= iter){

      # a sampling two boards
      b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
      b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
      parent1 <- pop$pop[b1][[1]]
      parent2 <-  pop$pop[b2][[1]]

      victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

      # Doing the crossover, mutate and fitness step
      kid <- crossover_a(parent1,parent2)

      kid <- mutate_a(kid, mutprob)

      fit <- fitnes_a_pairs(kid)

      pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

      pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit
      pair_queens <- c(pair_queens,fit)

      if(fit == 0){
        print(fit)
        print(kid)

        mat <- matrix(0,n,n)

        for (j in 1:n) {
          mat[kid[j,1],kid[j,2]] <- 1

        }
        print(mat)
        i <- i + 1
        break("Legal configuration found")
      }

      i <- i + 1
    }

     # Plotting the number of pairs of queens attacking each other at each iteration
    plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pai
}

# unkilled population
pop_a_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_a_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_a_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_a(parent1,parent2)

    kid <- mutate_a(kid, mutprob)

    fit <- fitnes_a_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_a_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j,1],kid[j,2]] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}


# binary population
pop_a_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
```

```r
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_a(n)
    fit_vec[i] <-  fitnes_a_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_a_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_a_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]


    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim


    #
    kid <- crossover_a(parent1,parent2)

    kid <- mutate_a(kid, mutprob)

    fit <- fitnes_a_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_a_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)
      for (j in 1:n) {
        mat[kid[j,1],kid[j,2]] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}


pop_b_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

## 7

genetic_b_pairs <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_p(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

    # Doing the crossover, mutate and fitness step
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_pairs(kid)

    pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit



    pair_queens <- c(pair_queens,fit)

    if(fit == 0){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}


pop_b_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}
```

```r
genetic_b_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_b_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }
  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}


pop_b_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_b(n)
    fit_vec[i] <-  fitnes_b_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_b_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_b_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]


    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim


    #
    kid <- crossover_b(parent1,parent2)

    kid <- mutate_b(kid, mutprob)

    fit <- fitnes_b_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_b_pairs(kid)
    pair_queens <- c(pair_queens,pair)


    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[bitsToInt(kid[[j]]),j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}

# pairs population
pop_c_p <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_pairs(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

## 7


genetic_c_pairs <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_p(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){
    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=TRUE)[1]] # selecting a victim

    # Doing the crossover, mutate and fitness step
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)

    fit <- fitnes_c_pairs(kid)

    pop$pop[[order(pop$fit, decreasing=TRUE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=TRUE)[1]] <- fit



    pair_queens <- c(pair_queens,fit)

    if(fit == 0){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}

# Unkilled population
pop_c_u <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_unkilled(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_c_unkilled <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_u(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    victim <- pop$"pop"[order(pop$fit, decreasing=FALSE)[1]] # selecting a victim

    #
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)

    fit <- fitnes_c_unkilled(kid)

    pop$pop[[order(pop$fit, decreasing=FALSE)[1]]] <- kid

    pop$fit[order(pop$fit, decreasing=FALSE)[1]] <- fit

    pair <- fitnes_c_pairs(kid)

    pair_queens <- c(pair_queens,pair)

    if(fit == n){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1
      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}

# binary population
pop_c_b <-function(n,pop_size){
  lis <- list()
  fit_vec <- c()
  for (i in 1:pop_size) {
    lis$"pop"[[i]] <- chessboard_c(n)
    fit_vec[i] <-  fitnes_c_binary(lis$"pop"[[i]] )
  }

  lis$fit <- fit_vec
  lis
}

genetic_c_binary <- function(n, iter,pop_size,mutprob){
  pop <- pop_c_b(n,pop_size)

  fitness_vec <- c() # fitness vector
  i <- 0 # iterations
  pair_queens <- c()
  while(i <= iter){

    # a sampling two boards
    b1 <- sample(1:length(pop$pop),1) # sampling 1 parent
    b2 <- sample((1:length(pop$pop))[-b1],1) # sampling another parent
    parent1 <- pop$pop[b1][[1]]
    parent2 <-  pop$pop[b2][[1]]

    idx_vic <- sample(1:length(pop$pop),1)
    victim <- pop$"pop"[idx_vic] # selecting a victim

    #
    kid <- crossover_c(parent1,parent2)

    kid <- mutate_c(kid, mutprob)
    fit <- fitnes_c_binary(kid)

    pop$pop[[idx_vic]] <- kid

    pop$fit[idx_vic] <- fit


    pair <- fitnes_c_pairs(kid)

    pair_queens <- c(pair_queens,pair)


    if(fit == 1){
      print(fit)
      print(kid)

      mat <- matrix(0,n,n)

      for (j in 1:n) {
        mat[kid[j],j] <- 1

      }
      print(mat)
      i <- i + 1
      break("Legal configuration found")
    }

    i <- i + 1
  }

  # Plotting the number of pairs of queens attacking each other at each iteration
  plot(1:(i), pair_queens, type = "l", xlab = "Iteration", ylab = "Attacking Pairs Count", main = "Pairs
}


###4
#genetic_a_unkilled(4,200,10,0.1)
genetic_a_unkilled(4,200,10,0.5)
#genetic_a_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_a_unkilled(8,200,10,0.1)
#genetic_a_unkilled(8,200,10,0.5)
#genetic_a_unkilled(8,200,10,0.9)

### 16
#genetic_a_unkilled(16,200,10,0.1)
```

```
#genetic_a_unkilled(16,200,10,0.5)
#genetic_a_unkilled(16,200,10,0.9)

###4
#genetic_a_pairs(4,200,10,0.1)
#genetic_a_pairs(4,200,10,0.5)
#genetic_a_pairs(4,200,10,0.9)

### 8
#genetic_a_pairs(8,200,10, 0.1)
#genetic_a_pairs(8,200,10, 0.5)
#genetic_a_pairs(8,200,10, 0.9)

### 16
#genetic_a_pairs(16,200,10,0.1)
#genetic_a_pairs(16,200,10,0.5)
#genetic_a_pairs(16,200,10,0.9)


###4
#genetic_a_binary(4,200,10,0.1)
#genetic_a_binary(4,200,10,0.5)
#genetic_a_binary(4,200,10,0.9)

### 8
#genetic_a_binary(8,200,10, 0.1)
#genetic_a_binary(8,200,10, 0.5)
#genetic_a_binary(8,200,10, 0.9)

### 16
#genetic_a_binary(16,200,10,0.1)
#genetic_a_binary(16,200,10,0.5)
#genetic_a_binary(16,200,10,0.9)

###4
#genetic_b_unkilled(4,200,10,0.1)
#genetic_b_unkilled(4,200,10,0.5)
#genetic_b_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_b_unkilled(8,200,10,0.1)
#genetic_b_unkilled(8,200,10,0.5)
#genetic_b_unkilled(8,200,10,0.9)

### 16
#genetic_b_unkilled(16,200,10,0.1)
#genetic_b_unkilled(16,200,10,0.5)
```

```
#genetic_b_unkilled(16,200,10,0.9)

###4
#genetic_b_pairs(4,200,10,0.1)
genetic_b_pairs(4,200,10,0.5)
#genetic_b_pairs(4,200,10,0.9)

### 8
#genetic_b_pairs(8,200,10, 0.1)
#genetic_b_pairs(8,200,10, 0.5)
#genetic_b_pairs(8,200,10, 0.9)

### 16
#genetic_b_pairs(16,200,10,0.1)
#genetic_b_pairs(16,200,10,0.5)
#genetic_b_pairs(16,200,10,0.9)

###4
#genetic_b_binary(4,200,10,0.1)
#genetic_b_binary(4,200,10,0.5)
#genetic_b_binary(4,200,10,0.9)

### 8
#genetic_b_binary(8,200,10, 0.1)
#genetic_b_binary(8,200,10, 0.5)
#genetic_b_binary(8,200,10, 0.9)

### 16
#genetic_b_binary(16,200,10,0.1)
#genetic_b_binary(16,200,10,0.5)
#genetic_b_binary(16,200,10,0.9)





###4
#genetic_c_unkilled(4,200,10,0.1)
#genetic_c_unkilled(4,200,10,0.5)
#genetic_c_unkilled(4,200,10,0.9)

# with mutprob > 0.1 we will always find a legal configuration

### 8
#genetic_c_unkilled(8,200,10,0.1)
#genetic_c_unkilled(8,200,10,0.5)
#genetic_c_unkilled(8,200,10,0.9)

### 16
```

```
#genetic_c_unkilled(16,200,10,0.1)
#genetic_c_unkilled(16,200,10,0.5)
#genetic_c_unkilled(16,200,10,0.9)

###4
#genetic_c_pairs(4,200,10,0.1)
#genetic_c_pairs(4,200,10,0.5)
#genetic_c_pairs(4,200,10,0.9)

### 8
#genetic_c_pairs(8,200,10, 0.1)
#genetic_c_pairs(8,200,10, 0.5)
#genetic_c_pairs(8,200,10, 0.9)

### 16
#genetic_c_pairs(16,200,10,0.1)
#genetic_c_pairs(16,200,10,0.5)
#genetic_c_pairs(16,200,10,0.9)

###4
#genetic_c_binary(4,200,10,0.1)
genetic_c_binary(4,200,10,0.5)
#genetic_c_binary(4,200,10,0.9)

### 8
#genetic_c_binary(8,200,10, 0.1)
#genetic_c_binary(8,200,10, 0.5)
#genetic_c_binary(8,200,10, 0.9)

### 16
#genetic_c_binary(16,200,10,0.1)
#genetic_c_binary(16,200,10,0.5)
#genetic_c_binary(16,200,10,0.9)

# loading the data
cens <- read.csv2('censoredproc.csv')

cens$time <- as.numeric(cens$time)

hist(cens$time, breaks=30 ,main='Histogram for all the data')
hist(cens$time[cens$cens==1], breaks=30 ,main='Histogram for the observed time of failure')
# starting values
m <- length(cens$time[cens$cens==2]) # number of censored product
n <- length(cens$time[cens$cens==1])
x  <-  cens$time[cens$cens==1]
z  <- cens$time[cens$cens==2]
lambda <- 100
improv <- 0.001
max_iter <- 1000
```

```r
iter <- 1
L <-c(-Inf)

# EM algorithm
Exp_llik <- function(lambda){
  (n * log(lambda)) - lambda * sum(x) +
  m * log(lambda) - (lambda * sum(q)) -
  (sum(log(1-exp(-lambda*z))))

}


# looping until
while(iter < max_iter){

   # Expectations
 q <- (1 /(1-exp(-lambda * z))) * ((1/lambda) - (z + (1/lambda)) * exp(-lambda * z))

 # loglikelihood
  L <- c(L,Exp_llik(lambda))

  if(abs(L[iter+1] - L[iter]) <= improv ){
    cat(" Lambda = ", lambda,"\n","Log likelihood = " ,L[iter+1] )# stopping criterion met
    break
  }
 # maximization

 # updating lambda numerically, using bfgs as the derivative can be found
   lambda <- optim(lambda,Exp_llik,method="BFGS", control = list(fnscale=-1))[[1]]

 iter <- iter+1
}
lambda <- 1.0047575694422
hist(cens$time, breaks=30 ,main='Histogram for all the data', probability = TRUE)
lines(density(cens$time, adjust = 1 / lambda), lwd = 2)

hist(cens$time[cens$cens==1], breaks=30 ,main='Histogram for the observed time of failure', probability
lines(density(cens$time[cens$cens==1], adjust = 1 / lambda), lwd = 2)
bs_df <- matrix(NA, ncol = 2, nrow = 1000)
colnames(bs_df) <- c("time", "cens")

lambda_hat_mle <- c()



m <- 189 # number of censored product
n <- 1000

# bootstrap
```

```r
for (i in 1:n) {
  # a)
  bs_df[,1] <- rexp(n, rate = lambda)
  bs_df[,2] <- 0

  # b)
  bs_df[1:m,1] <- runif(m, min = 0, max = max(cens$time[cens$cens==2]))
  bs_df[1:m,2] <- 1

  # c)
  # estimate lambda by MLE on uncensored observations
  lambda_hat_mle[i] <- n/sum(bs_df[(m+1):n,1])
}



bs_df <- as.data.frame(bs_df)
lambda_hat_em <- c()
for(i in 1:1000){
  # starting values
m <- length(bs_df$time[bs_df$cens==1]) # number of censored product
n <- length(bs_df$time[bs_df$cens==0])
x  <-  bs_df$time[bs_df$cens==0]
z  <- bs_df$time[bs_df$cens==1]
lambda <- 100
improv <- 0.001
max_iter <- 1000
iter <- 1
L <-c(-Inf)

# EM algorithm
Exp_llik <- function(lambda){
  (n * log(lambda)) - lambda * sum(x) +
  m * log(lambda) - (lambda * sum(q)) -
  (sum(log(1-exp(-lambda*z))))

}


# looping until
while(iter < max_iter){

   # Expectations
 q <- (1 /(1-exp(-lambda * z))) * ((1/lambda) - (z + (1/lambda)) * exp(-lambda * z))

 # loglikelihood
  L <- c(L,Exp_llik(lambda))

  if(abs(L[iter+1] - L[iter]) <= improv ){
    cat(" Lambda = ", lambda,"\n","Log likelihood = " ,L[iter+1] )# stopping criterion met
    break
  }
 # maximization

 # updating lambda numerically, using bfgs as the derivative can be found
   lambda <- optim(lambda,Exp_llik,method="BFGS", control = list(fnscale=-1))[[1]]
   lambda_hat_em[i] <- lambda

 iter <- iter+1
}
}

hist(lambda_hat_mle)
hist(lambda_hat_em)
var(lambda_hat_mle)
var(lambda_hat_em)
```