Lab6_report

Simon Jorstedt & Siddhesh Sreedar

2024-01-18

Question 1 - Genetic Algorithms

I have structured the code in such a way that we proceed with doing all the part for each encoding before going to the next encoding

Encoding-1

```
library(compositions)
```

```
## Warning: package 'compositions' was built under R version 4.3.2
```

```
## Welcome to compositions, a package for compositional data analysis.
## Find an intro with "? compositions"
```

```
##
## Attaching package: 'compositions'
```

```
## The following objects are masked from 'package:stats':
##
##     anova, cor, cov, dist, var
```

```
## The following object is masked from 'package:graphics':
##
##     segments
```

```
## The following objects are masked from 'package:base':
##
##     %*%, norm, scale, scale.default
```

```
### 1)

### a)
encoding1<-function(n){
  data<-list()
  for(i in 1:n){
    data[[i]]<-c(sample(n,1),sample(n,1))
    while(any(duplicated(data))== TRUE){
      data[[which(duplicated(data))]]<-c(sample(n,1),sample(n,1))
    }
```

```
  }
  return(data)
}

#layout1<-encoding1(8)
#layout2<-encoding1(8)


### 2)
crossover<-function(x,y){
  p<-sample(1:(length(x)/2),1)
  kid<-c(x[1:p],y[(p+1):length(x)])

  #while(any(duplicated(kid)==TRUE)){
  #  p<-sample((0:(length(x)/2)),1)
  #  kid<-c(x[1:p],y[(p+1):length(x)])
  #}
  return(kid)
}

#kid1<-crossover(layout1,layout2)

### 3)

mutate<-function(x){
  choice<-sample(length(x),1)
  a<-x[[choice]]
  x[[choice]]<-c(sample(length(x),1),sample(length(x),1))
  while(any(duplicated(x))== TRUE & all(x[[choice]] == a)){ #To ensure that the queen moves to a posito
    x[[which(duplicated(x))]]<-c(sample(length(x),1),sample(length(x),1))
  }
  return(x)
}

#mutate(layout1)

### 4)

fitness_binary<-function(x){
  a<-1

  #row check
  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  #column check
  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }
```

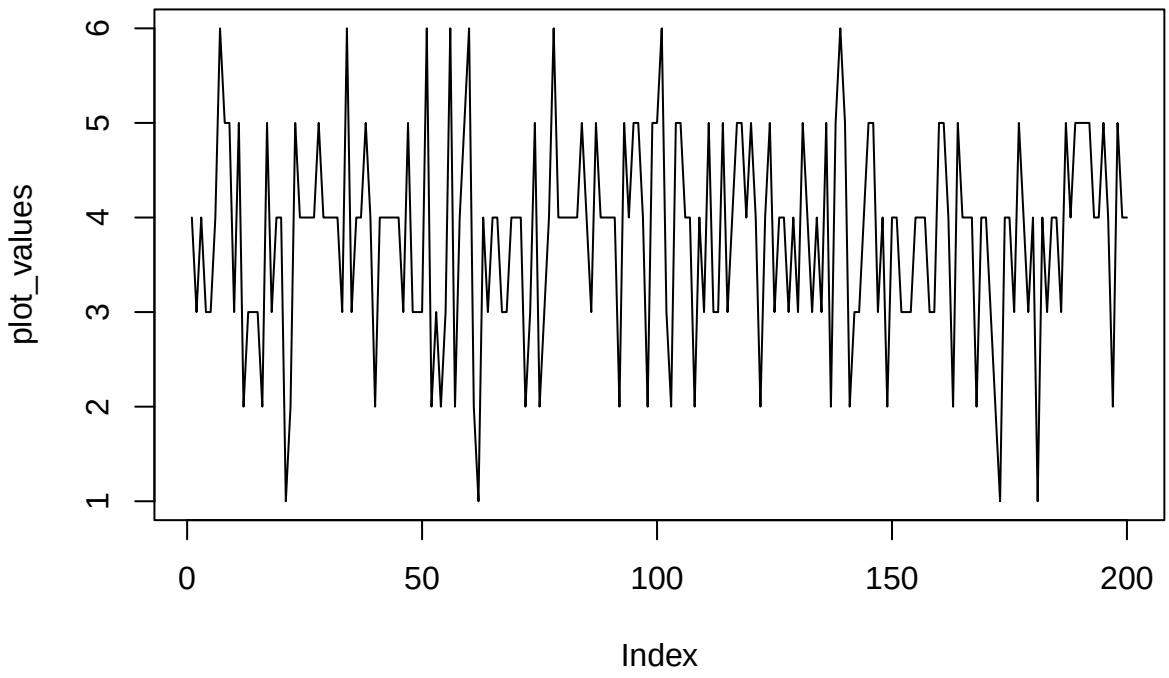
```
  if(any(duplicated(row))==TRUE){
    a<-0 #0 implies that it is not a solution
  }

  if(any(duplicated(col))==TRUE){
    a<-0
  }

  #diagonal check
  for(i in 1:length(x)){
    for(j in 1:length(x)){
      if(row[i]==row[j] & col[i]==col[j]){
        next
      }
      if(abs(row[i]-row[j])==abs(col[i]-col[j])){
        a<-0
      }
    }
  }

  return(a)
}

#fitness_binary(layout1)

fitness_attack<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }

  #non_attack<-length(x)
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==
        count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}
```

```r
#fitness_attack(layout1)

fitness_pair<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }

  #non_attack<-length(x)
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==a
      count[i]<-1
    }
  }
  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
  return(a-number)
}

#fitness_pair(layout1)


#For plotting
queens_attack<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==a
      count[i]<-1
    }
  }
  number<-sum(count,na.rm = TRUE)
  return(number)
}
```

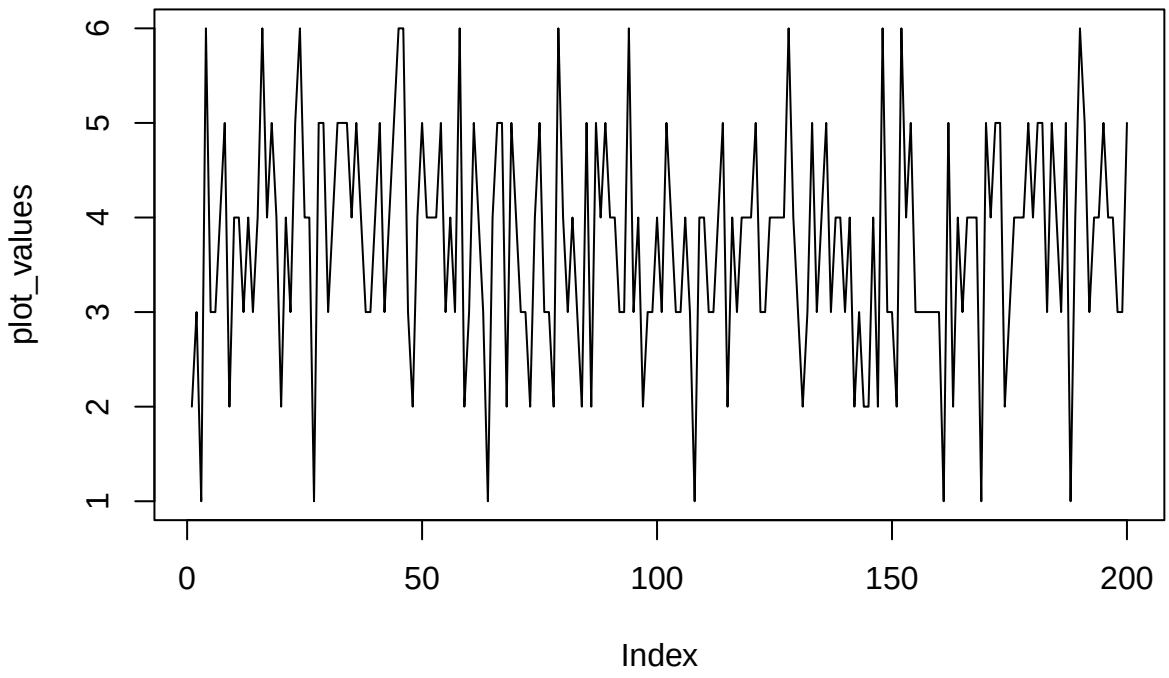
```r
#queens_attack(layout1)


### 5), 6), 7)

genetic_algo<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)


    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
    parent2<-population[[parent[2]]]

    kid<-crossover(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid

    plot_values[i]<-queens_attack(kid)

    if(queens_attack(kid)==0){
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}
```

8)

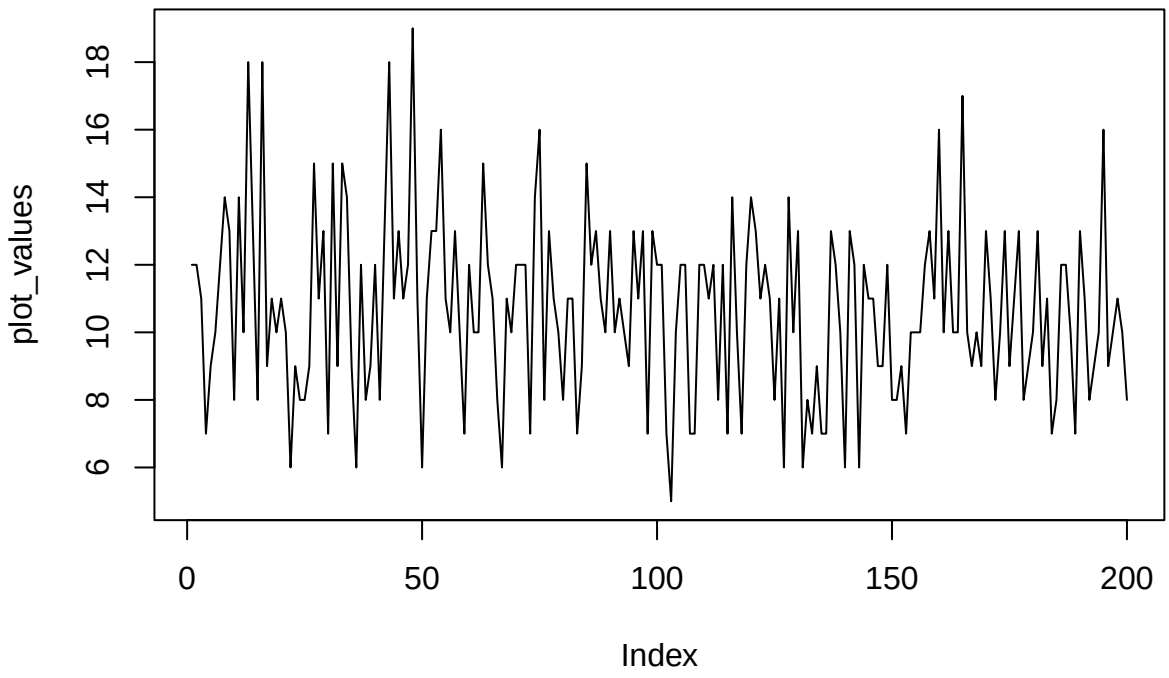
```
genetic_algo(encoding1,0.8,fitness_binary,4)
```



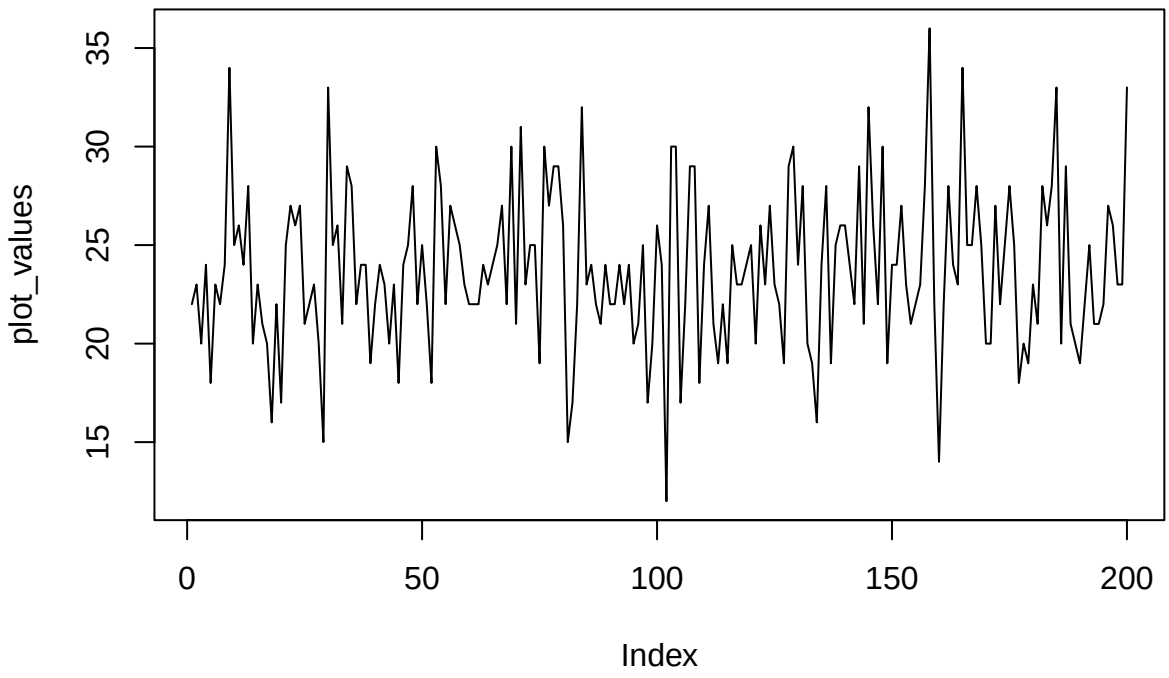

```
genetic_algo(encoding1,0.8,fitness_binary,4)
```



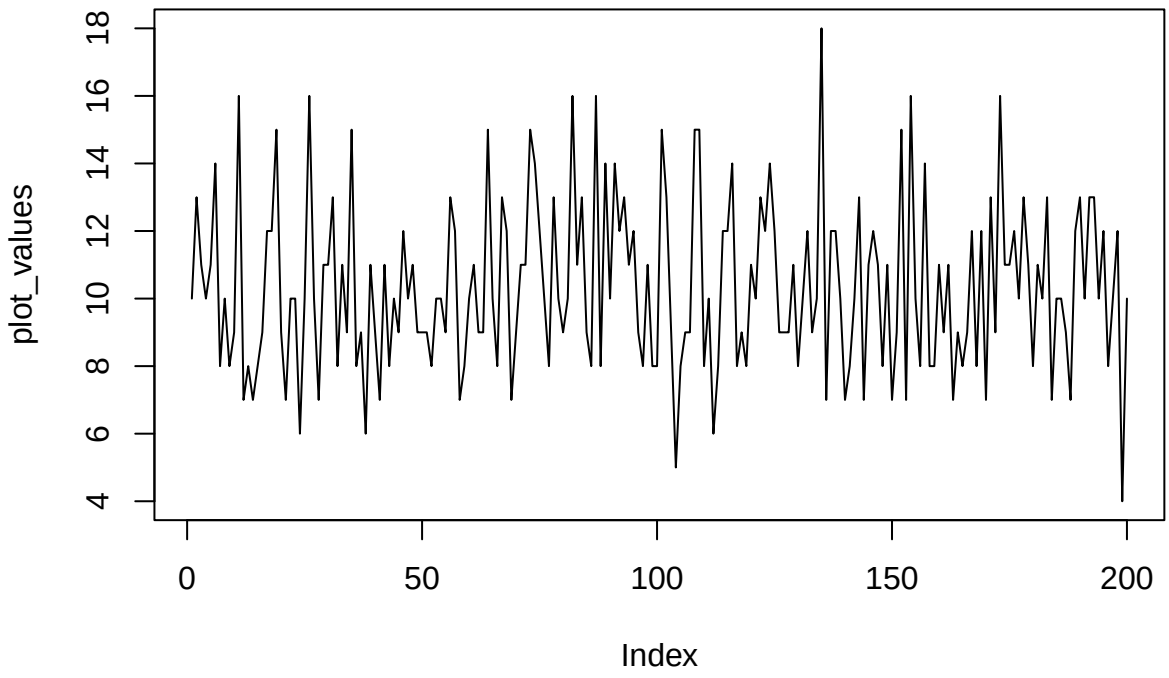

```
genetic_algo(encoding1,0.5,fitness_binary,8)
```



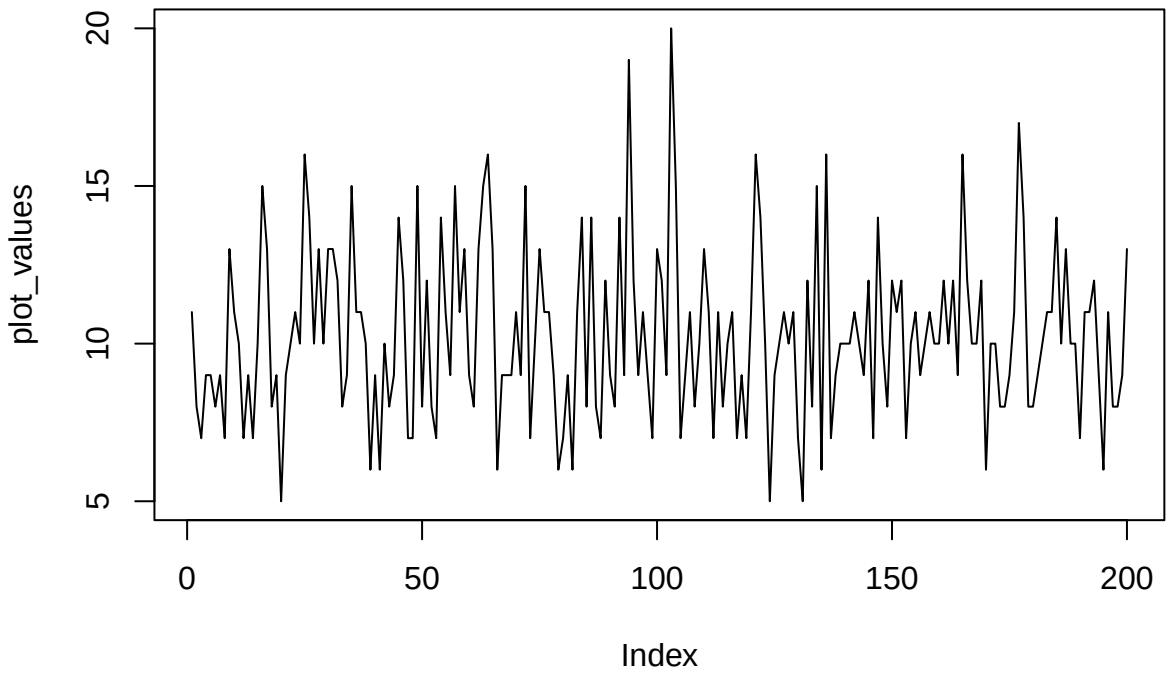

```
genetic_algo(encoding1,0.5,fitness_binary,16)
```



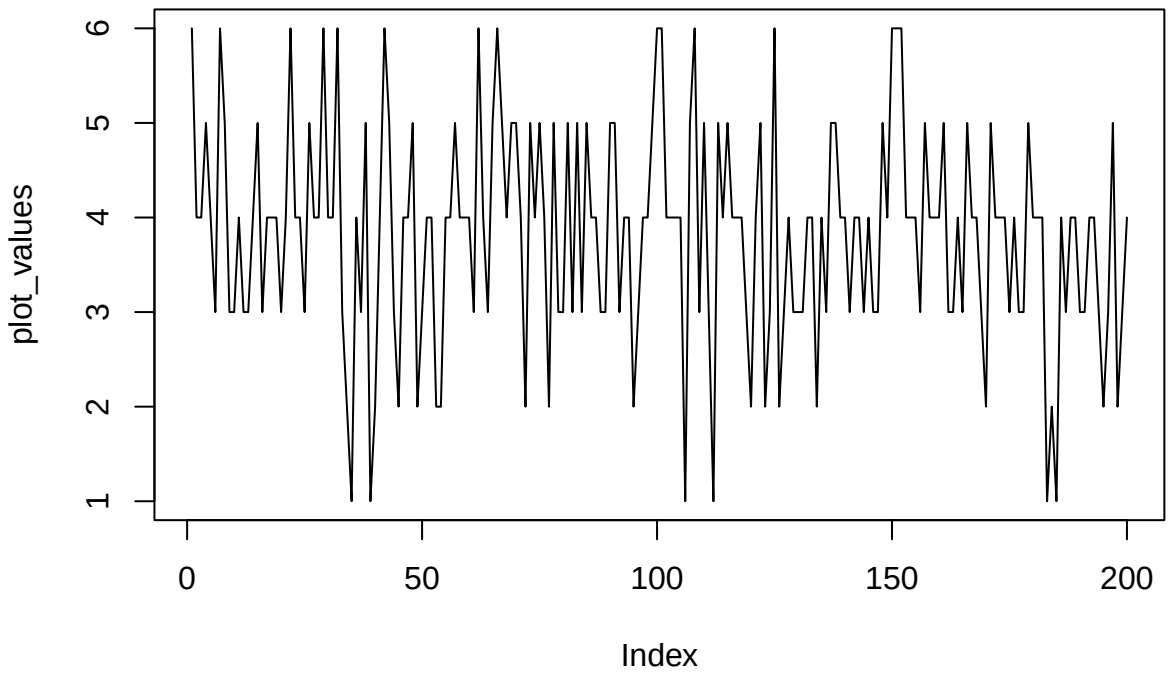

```
genetic_algo(encoding1,0.1,fitness_binary,8)
```



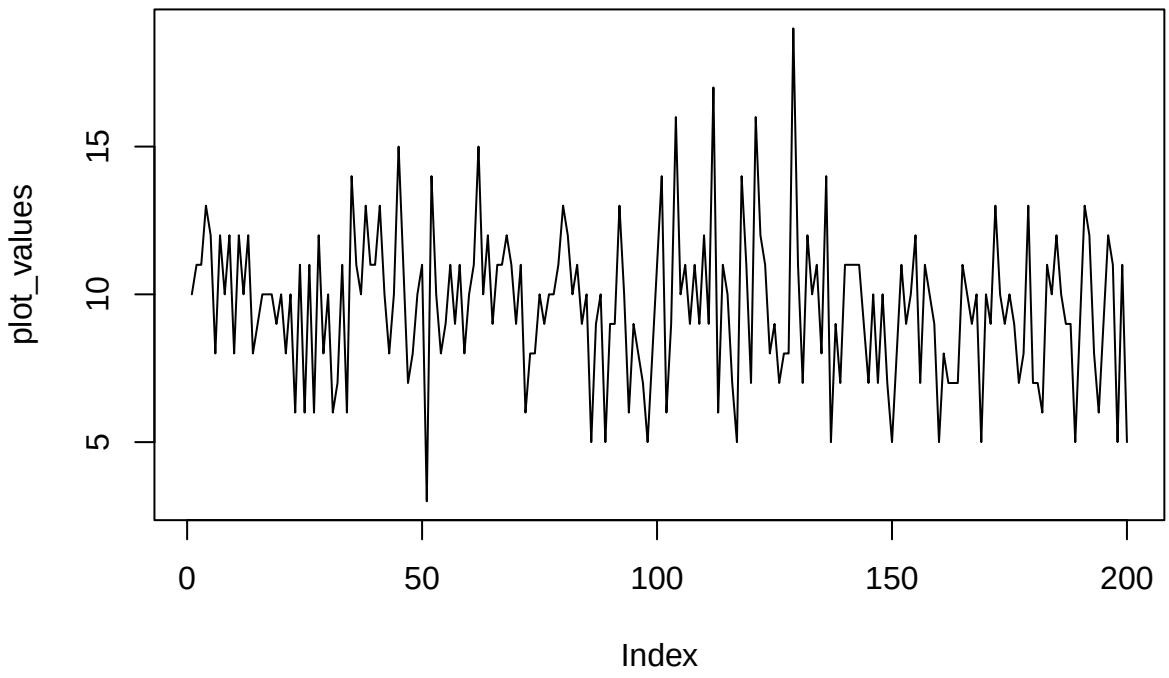

```
genetic_algo(encoding1,0.9,fitness_binary,8)
```



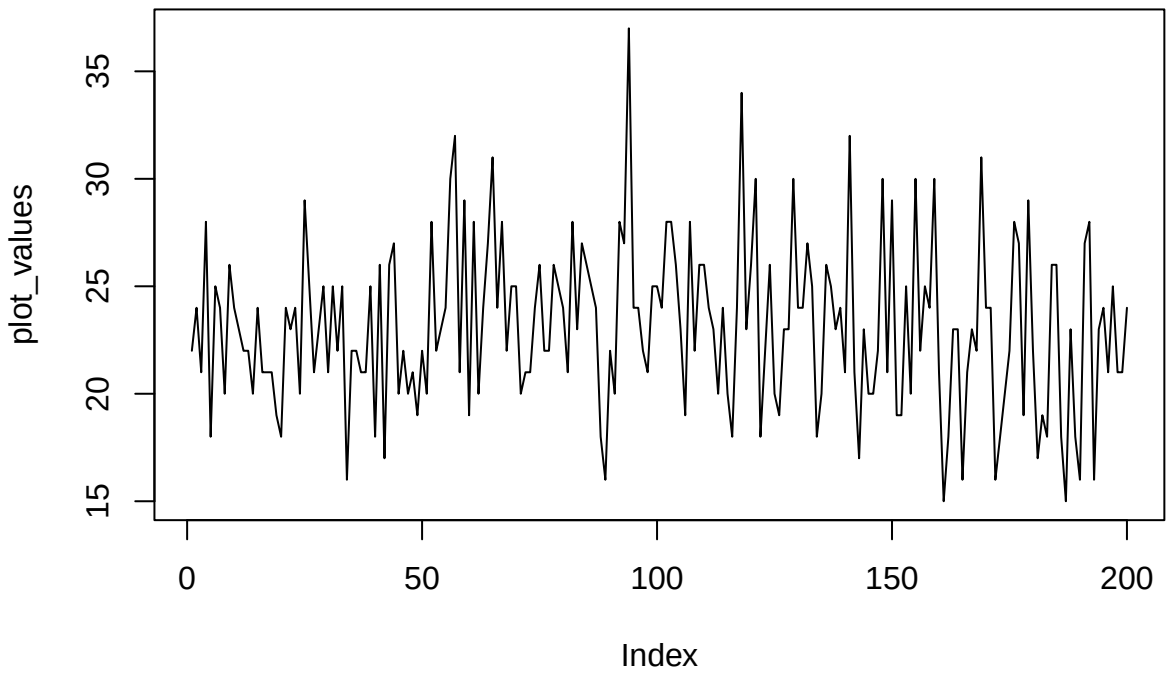

```
genetic_algo(encoding1,0.5,fitness_attack,4)
```



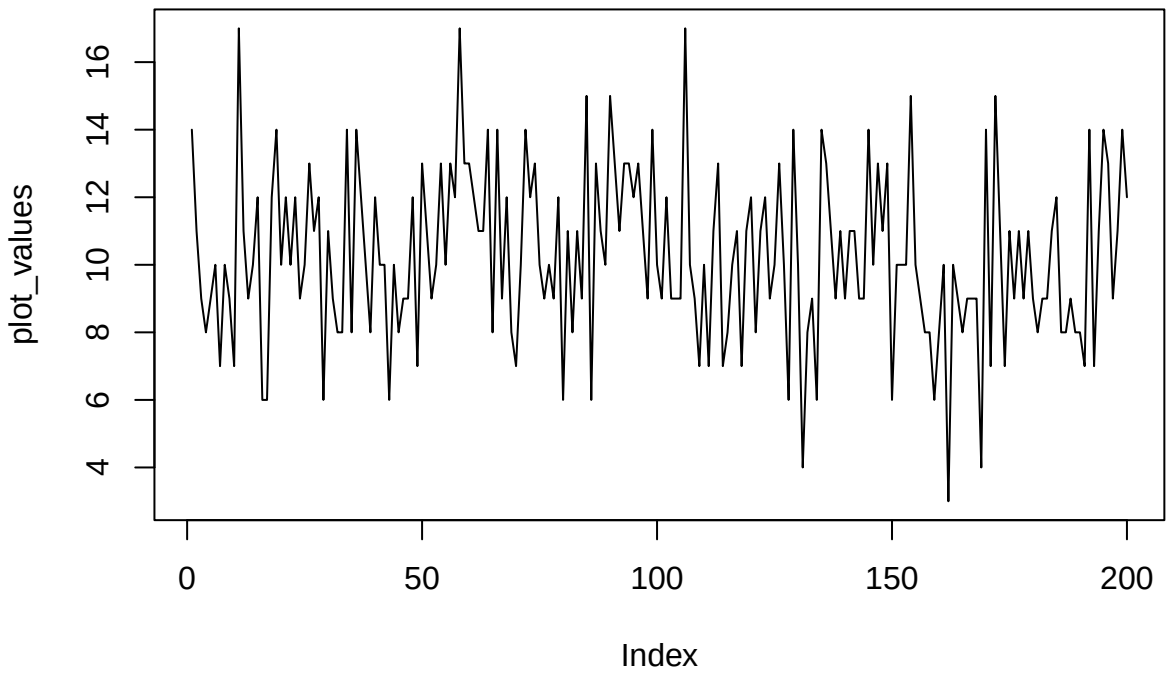

```
genetic_algo(encoding1,0.5,fitness_attack,8)
```



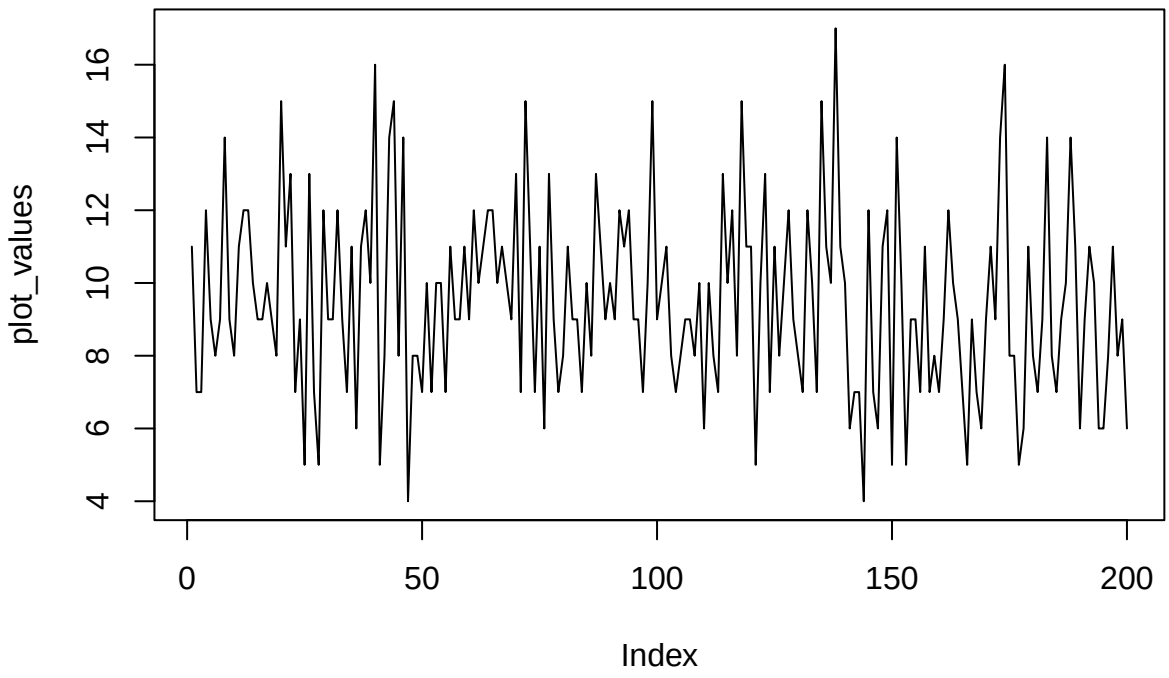

```
genetic_algo(encoding1,0.5,fitness_attack,16)
```



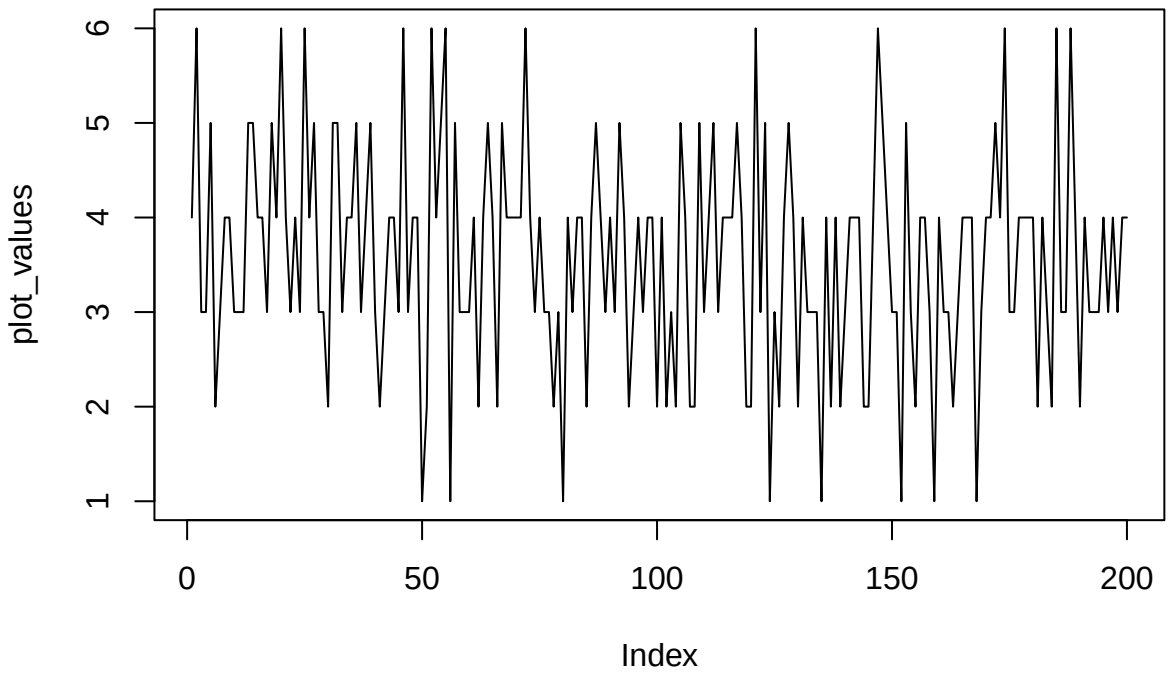

```
genetic_algo(encoding1,0.1,fitness_attack,8)
```



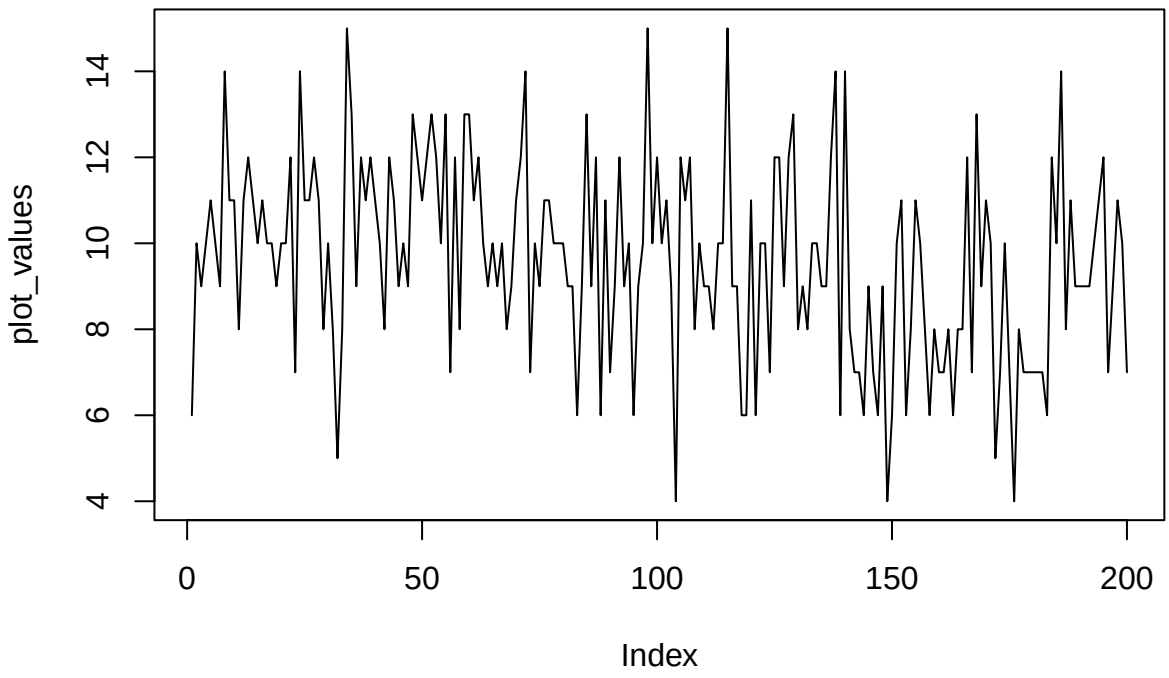

```
genetic_algo(encoding1,0.9,fitness_attack,8)
```



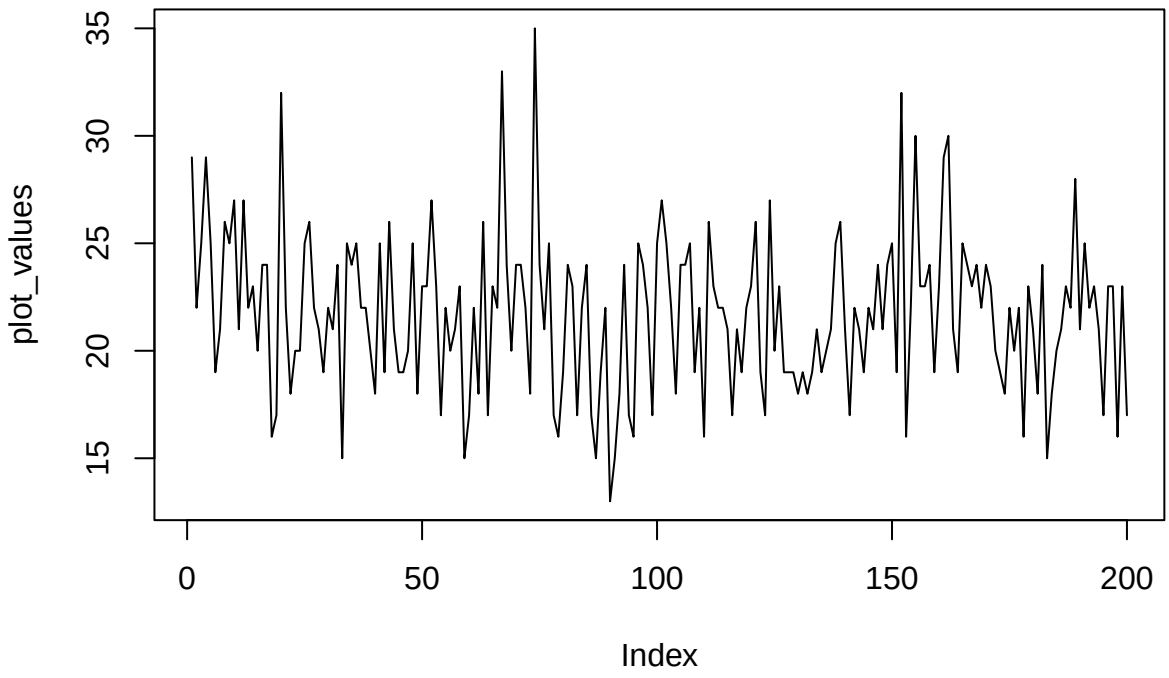

```
genetic_algo(encoding1,0.5,fitness_pair,4)
```



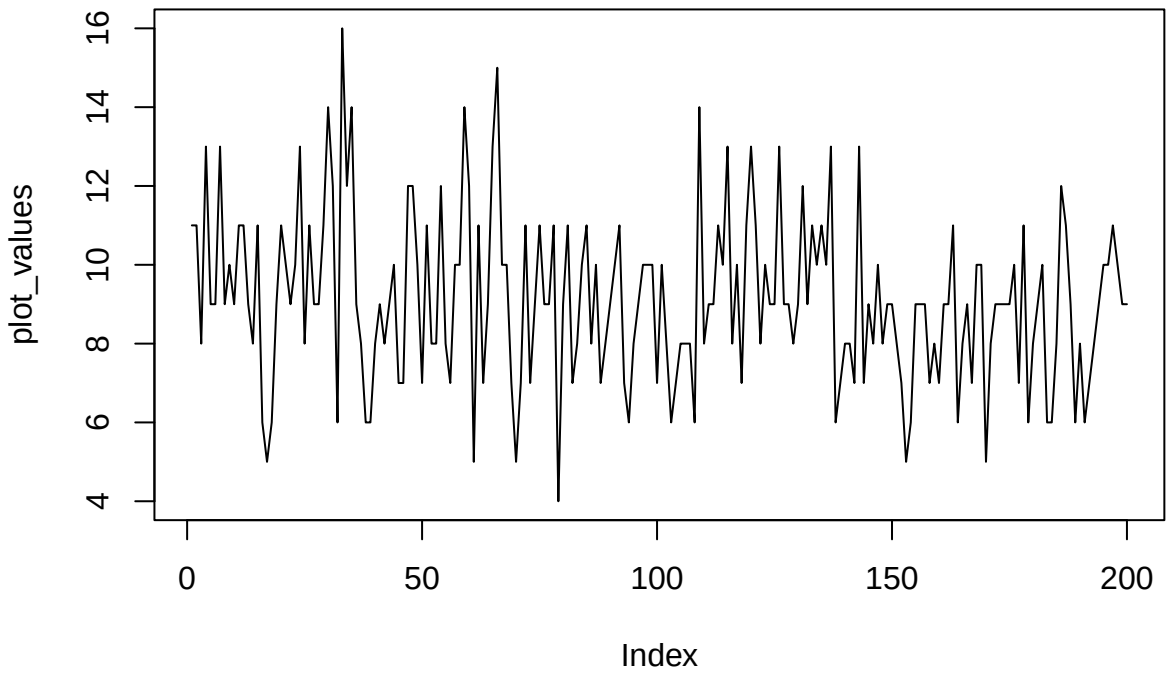

```
genetic_algo(encoding1,0.5,fitness_pair,8)
```



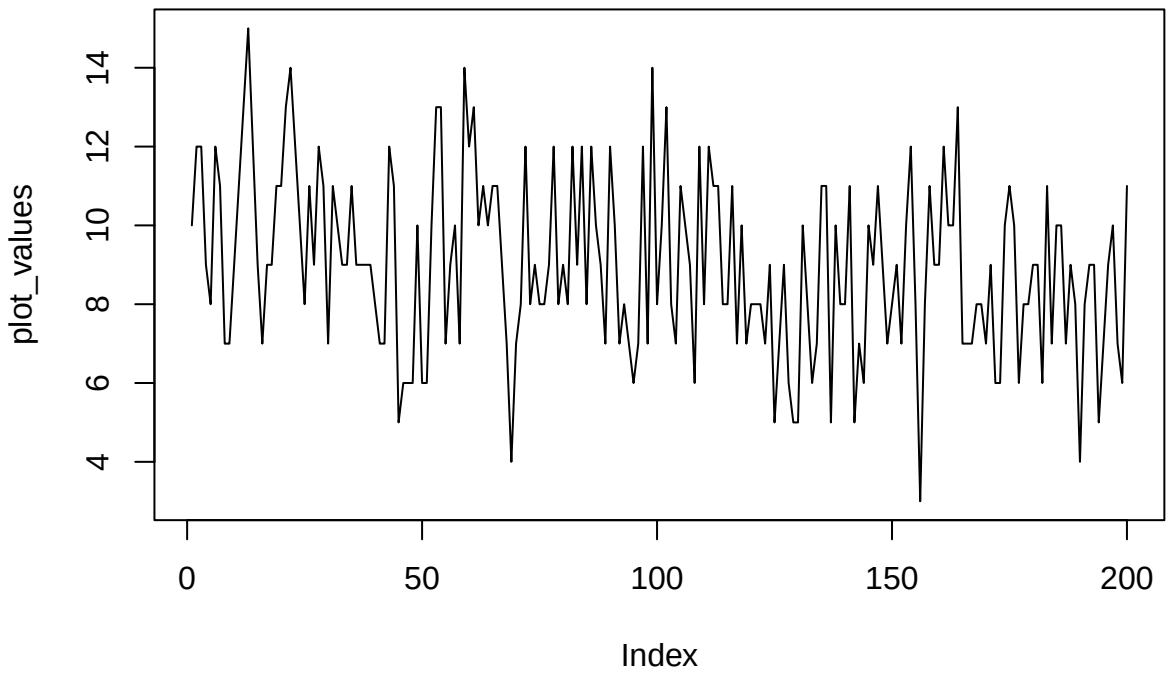

```
genetic_algo(encoding1,0.5,fitness_pair,16)
```




```
genetic_algo(encoding1,0.1,fitness_pair,8)
```




```
genetic_algo(encoding1,0.9,fitness_pair,8)
```



Couldn't find legal configuration for all via the first encoding when the number of iterations is 100. But after increasing the interations to 200, there are some that give the legal configuration. As we increase the value of "n", it becomes harder to find the legal configuration or we do get it if we keep running the function multiple times until a legal configuration is found. So we need to run the same function with the same inputs multiple times to get the legel configuration.

Encoding-2

```
toBits <- function (x, nBits = 8){
  tail(rev(as.numeric(intToBits(x))),nBits)
}

#toBits(8,5)

### 1)

### b)

encoding2<-function(n){
  index<-sample(n,n,replace = FALSE)
  output<-list()
  for(i in 1:n){
    #output[[i]]<-rep(0,n)
    output[[i]]<-toBits(index[i],log2(n))
  }
  return(output)
}
```

```r
#e1<-encoding2(8)
#e2<-encoding2(8)

#To convert bits to integer
bits_convert<-function(x){
  b<-c()
  for(i in 1:length(x)){
    c<-""
    for(j in 1:length(x[[1]])){
      c<-paste(c,x[[i]][j],sep = "")
    }
    b[i]<-unbinary(c)
  }
  b[b==0]<-length(x)
  return(b)
}

#bits_convert(e1)

#unbinary("111")
#b<-c()
#for(i in 1:8){
#  b[i]<-unbinary(paste(e1[[i]][1],e1[[i]][2],e1[[i]][3],sep = ""))
#}
# 0 implies 8


### 2)
crossover2<-function(x,y){
  kid<-list()
  for(i in 1:length(x)){
    p<-sample(1:round(log2(length(x))/2),1)
    kid[[i]]<-c(x[[i]][1:p],y[[i]][(p+1):log2(length(x))])
  }
  return(kid)
}

#crossover2(e1,e2)

### 3)
mutate2<-function(x){
  choice<-sample(length(x),1)
  value<-sample(length(x),1)
  x[[choice]]<-toBits(value,log2(length(x)))

  return(x)
}

#mutate2(e1)


### 4)
fitness_binary_2<-function(x){
```

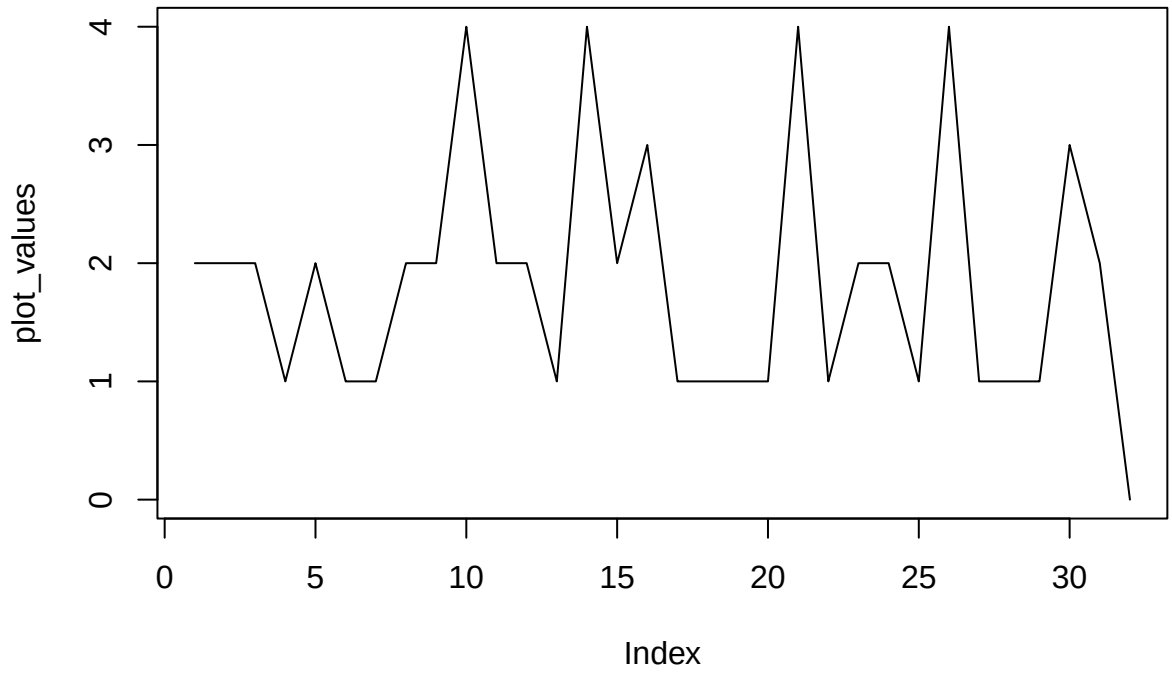
```
  x<-bits_convert(x)

  dim<-combn(length(x),2)

  a<-1 # if it is a legal config
  if(any(duplicated(x))==TRUE){
    a<-0
  }

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-0
    }else{
      count[i]<-1
    }
  }
  if(any(count==0)){
    a<-0
  }

  return(a)
}

fitness_attack_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}

fitness_pair_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
```

```
  return(a-number)
}


#For plotting
queens_attack_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  return(number)
}

### 5), 6), 7)
genetic_algo_2<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)


    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
    parent2<-population[[parent[2]]]

    kid<-crossover2(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate2(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid
```

```
    plot_values[i]<-queens_attack_2(kid)

    if(queens_attack_2(kid)==0){
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}
```

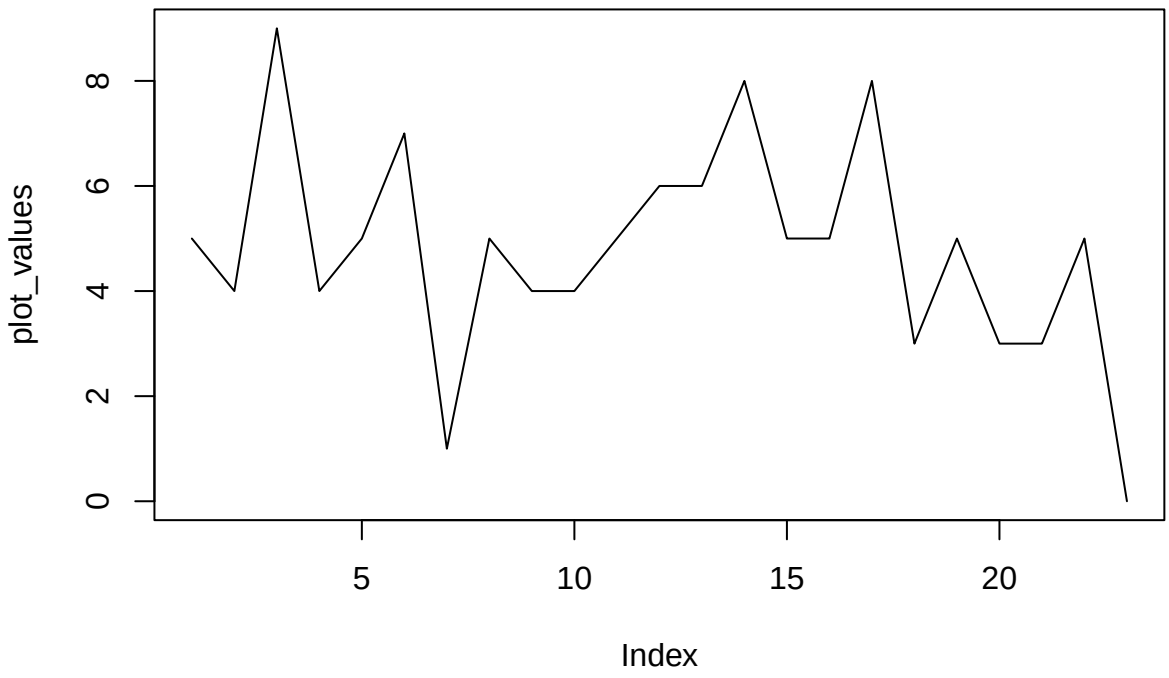
8)

```
genetic_algo_2(encoding2,0.8,fitness_binary_2,4)
```



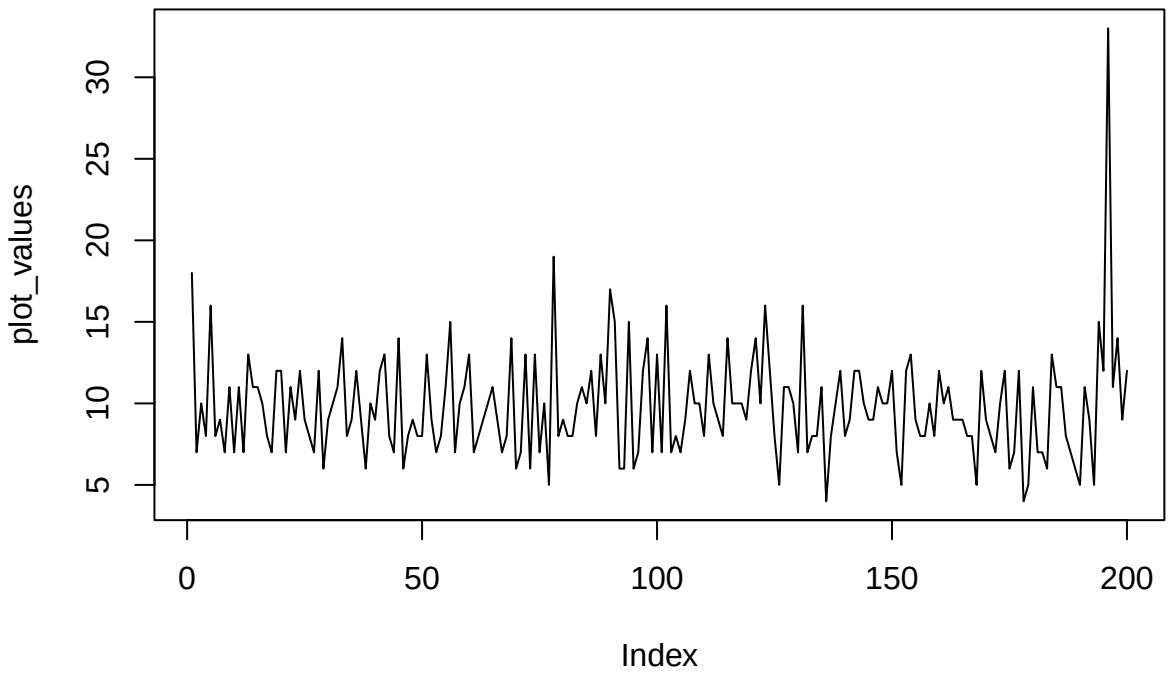

```
## [1] "legal configuration found"
```

```
genetic_algo_2(encoding2,0.5,fitness_binary_2,8)
```

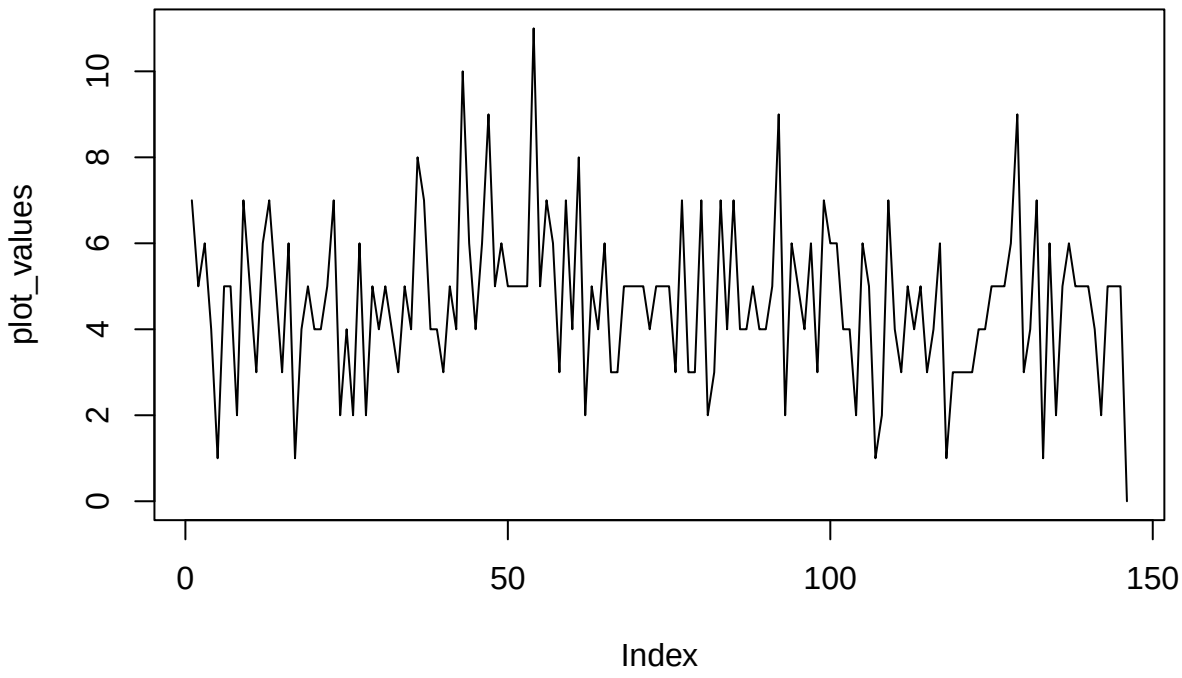



```
## [1] "legal configuration found"
```

```
genetic_algo_2(encoding2,0.5,fitness_binary_2,16)
```

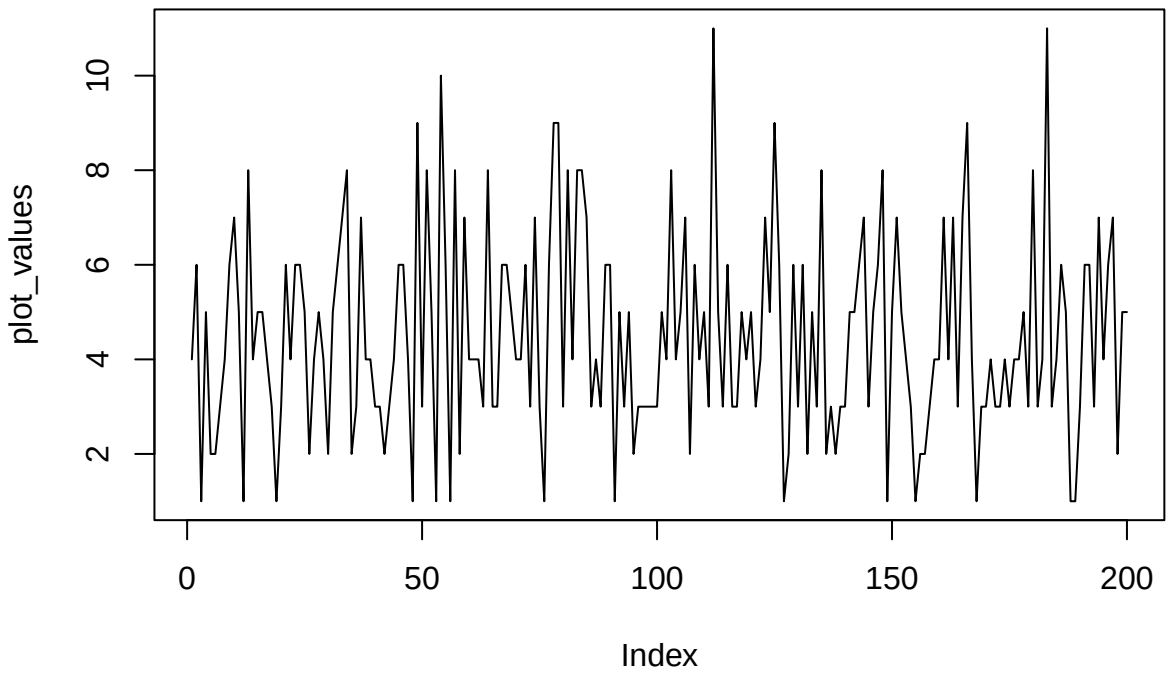



```
genetic_algo_2(encoding2,0.1,fitness_binary_2,8)
```

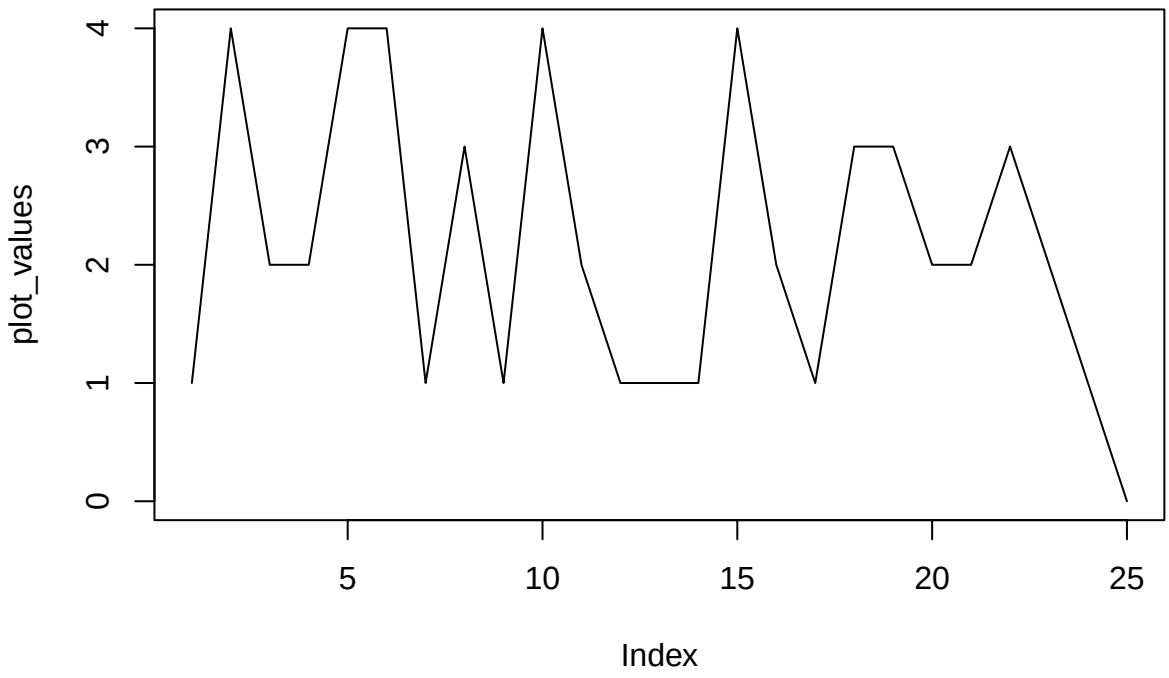



```
## [1] "legal configuration found"
```

```
genetic_algo_2(encoding2,0.9,fitness_binary_2,8)
```

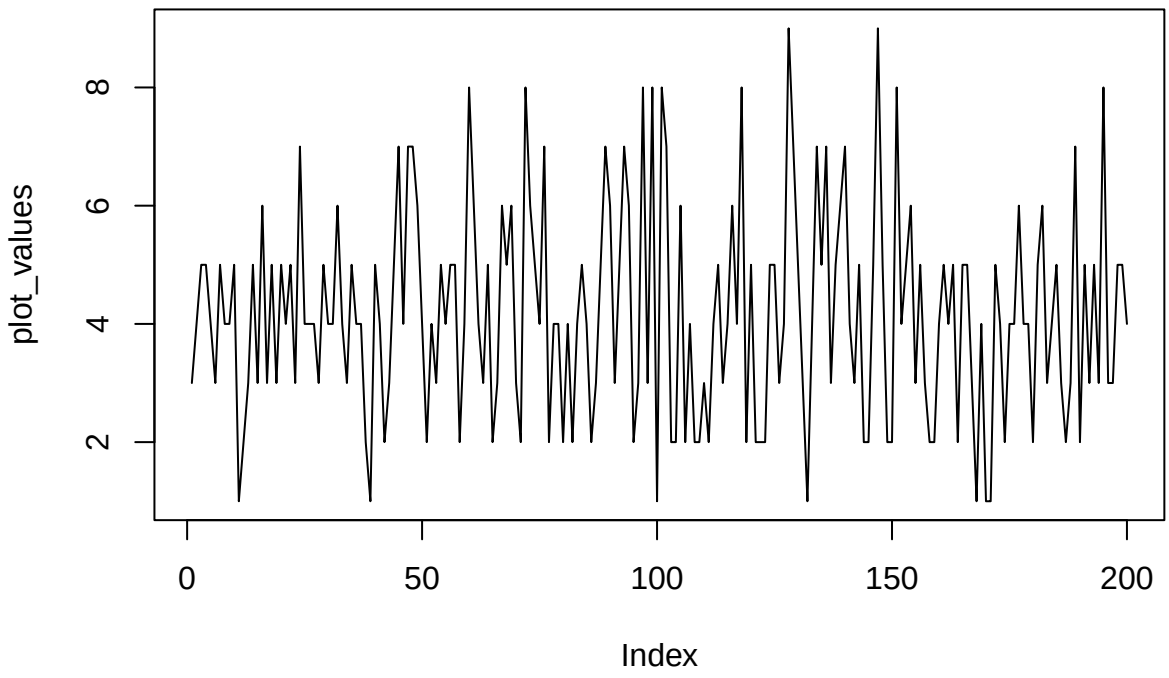



```
genetic_algo_2(encoding2,0.5,fitness_attack_2,4)
```

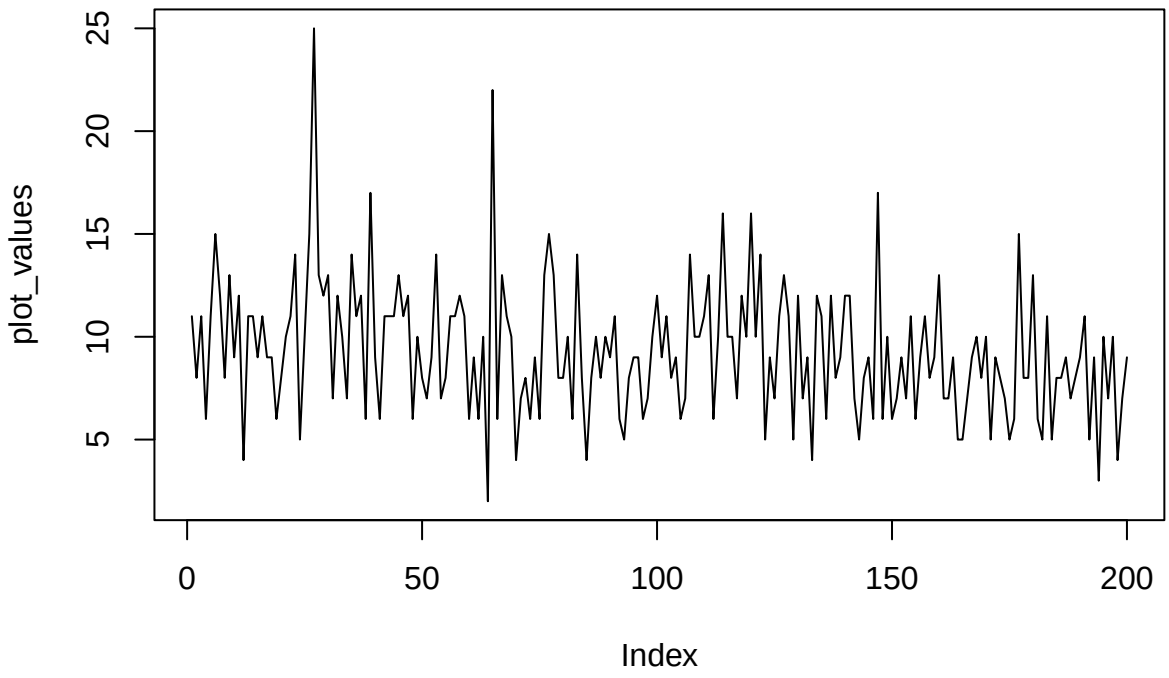



```
## [1] "legal configuration found"
```

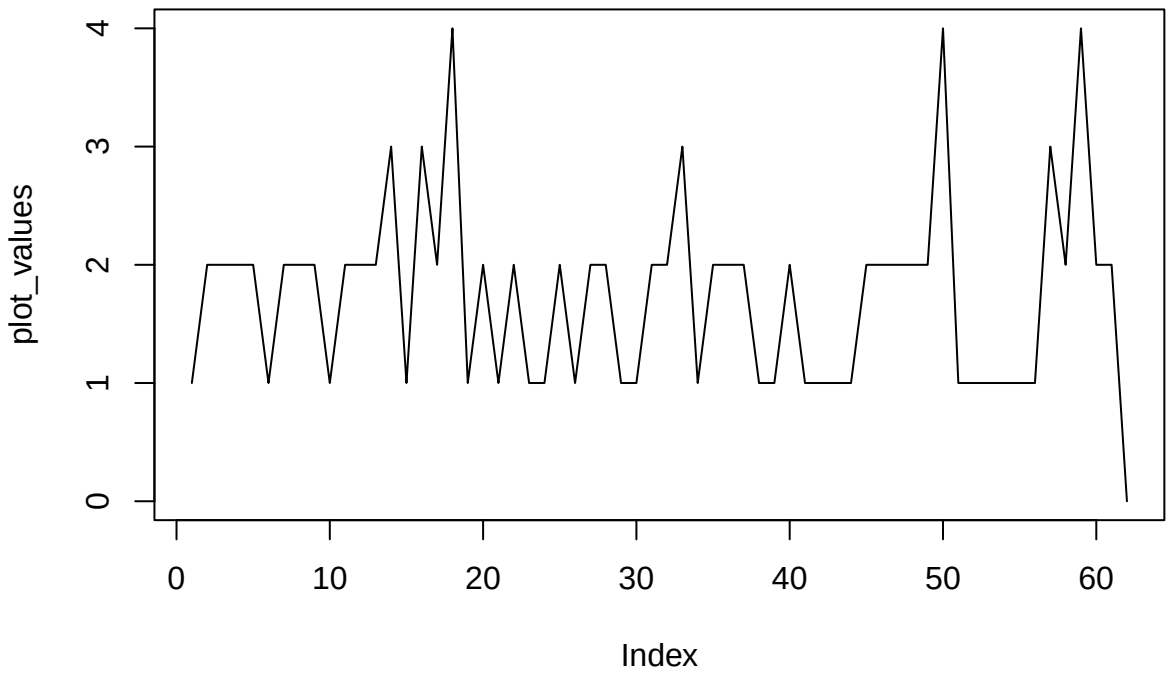
```
genetic_algo_2(encoding2,0.5,fitness_attack_2,8)
```




```
genetic_algo_2(encoding2,0.5,fitness_attack_2,16)
```

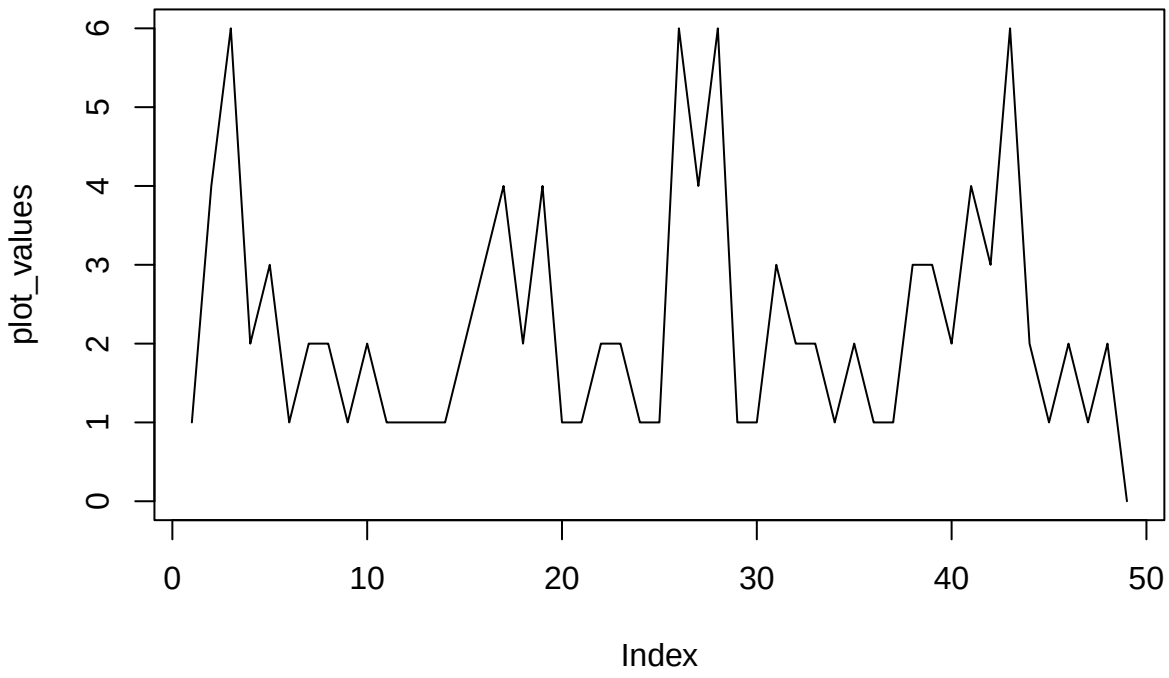



```
genetic_algo_2(encoding2,0.1,fitness_attack_2,4)
```



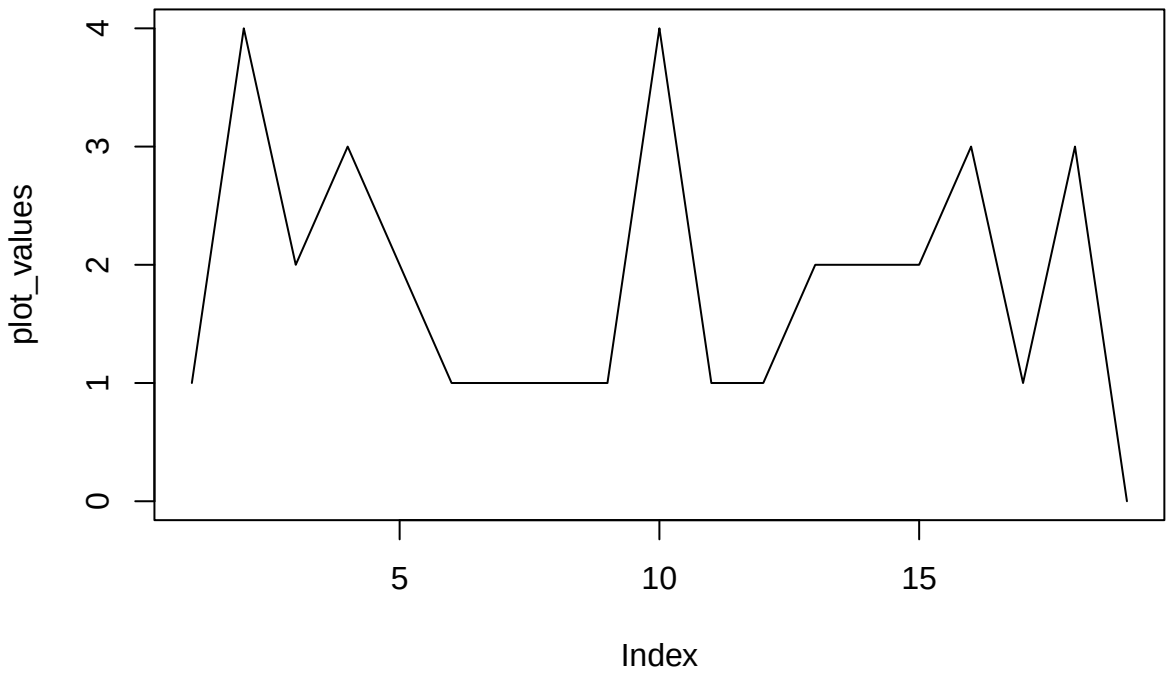

```
## [1] "legal configuration found"
```

```
genetic_algo_2(encoding2,0.9,fitness_attack_2,4)
```



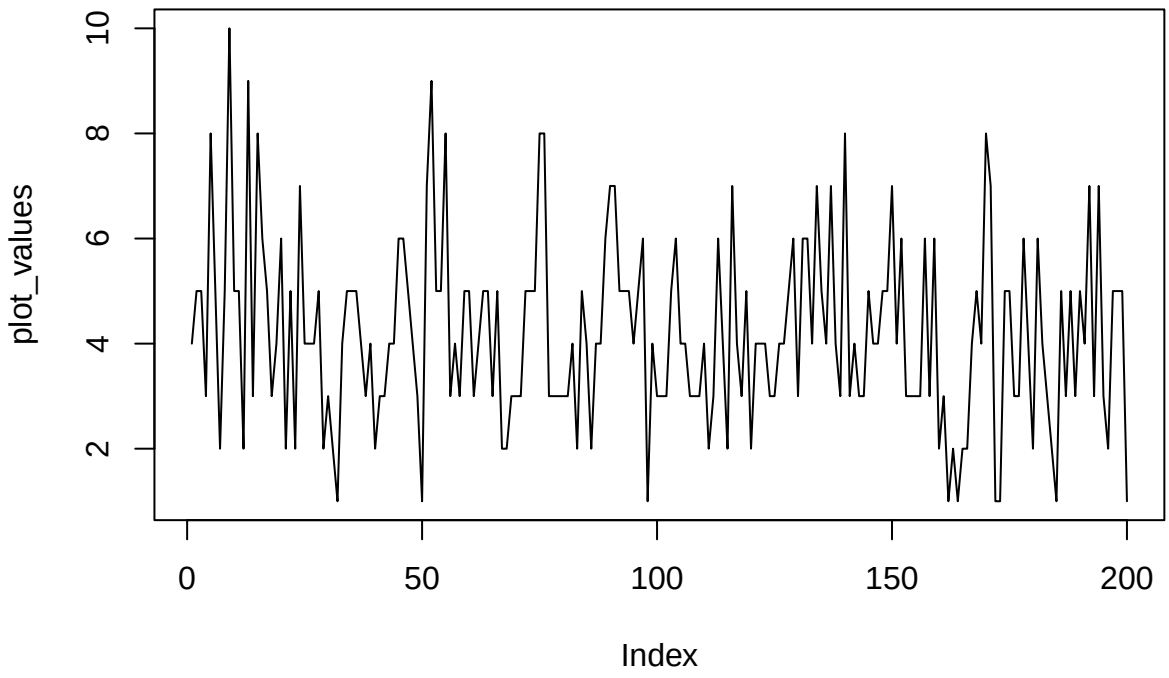

```
## [1] "legal configuration found"
```

```
genetic_algo_2(encoding2,0.5,fitness_pair_2,4)
```

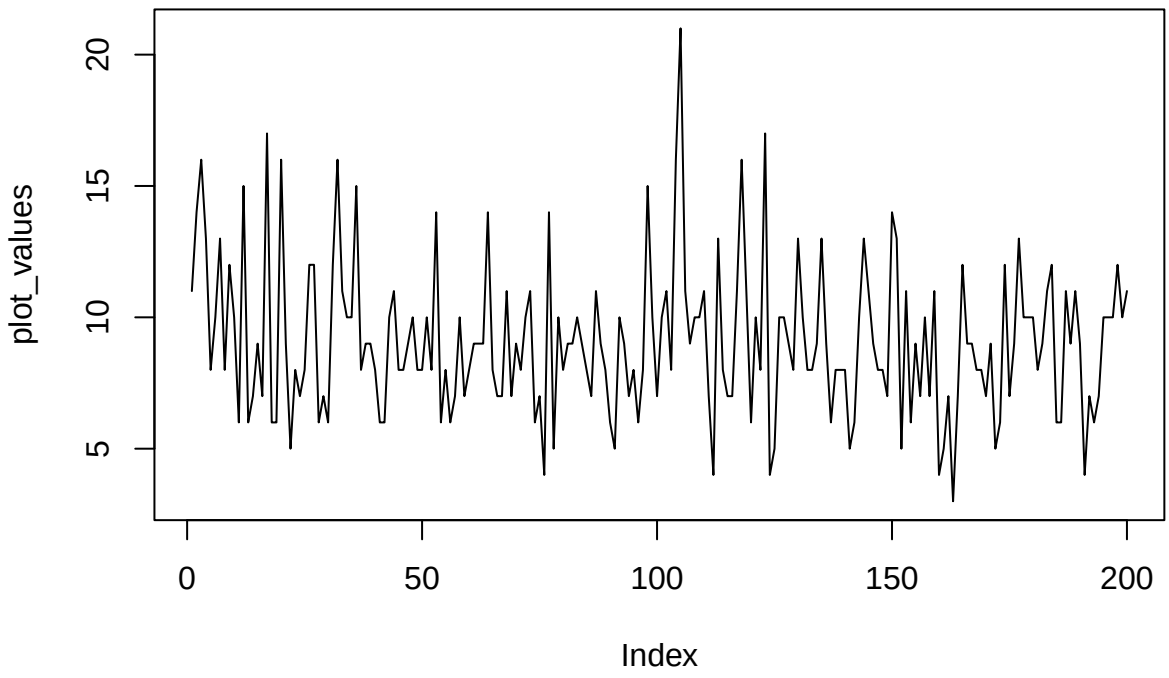



```
## [1] "legal configuration found"
```

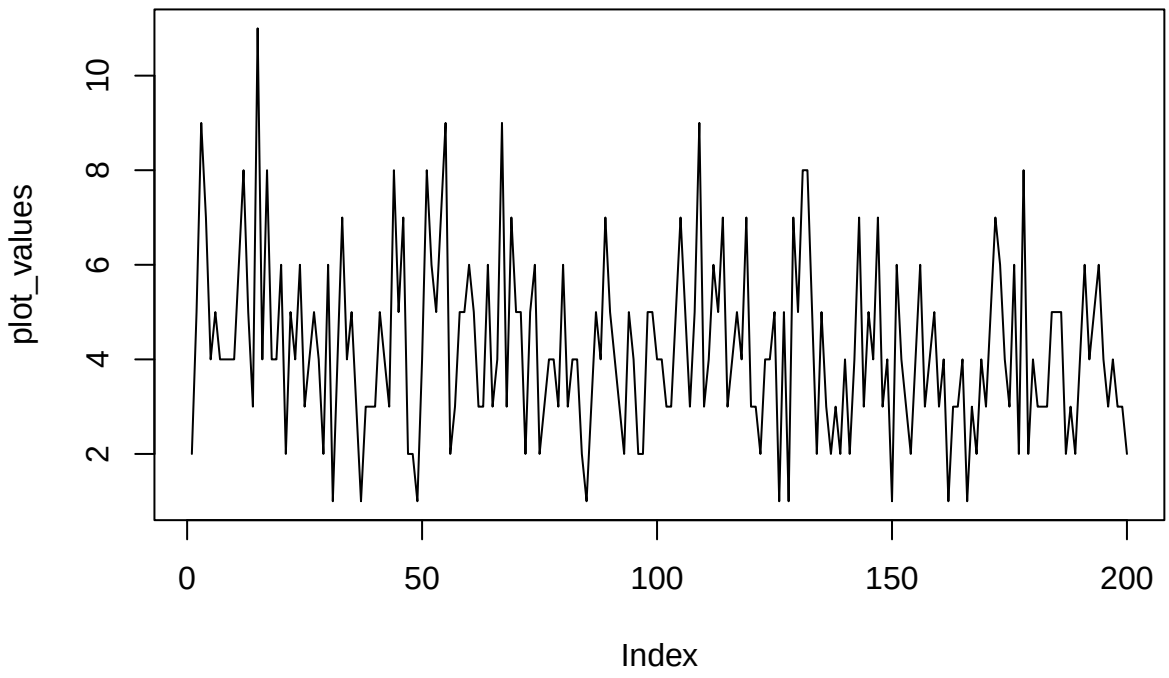
```
genetic_algo_2(encoding2,0.5,fitness_pair_2,8)
```



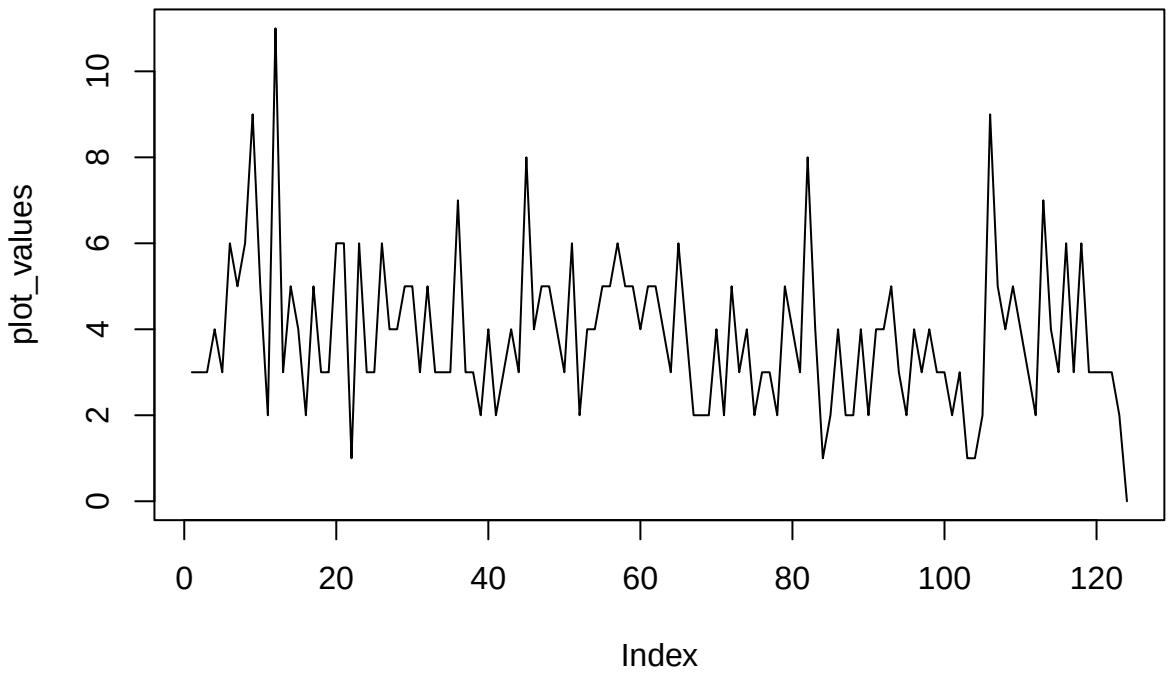

```
genetic_algo_2(encoding2,0.5,fitness_pair_2,16)
```




```
genetic_algo_2(encoding2,0.1,fitness_pair_2,8)
```




```
genetic_algo_2(encoding2,0.9,fitness_pair_2,8)
```




```
## [1] "legal configuration found"
```

We find more legal configurations for this for different input changes as compared to encoding-1. We can sometimes repeat the same function multiple times to get the legal configuration if not found in the first try.

Encoding-3

```
### 1)

### c)

encoding3<-function(n){
  output<-sample(n,n,replace = FALSE)
  return(output)
}

l1<-encoding3(8)
l2<-encoding3(8)

### 2)

crossover3<-function(x,y){
  p<-sample(1:(length(x)/2),1)
  kid<-c(x[1:p],y[(p+1):length(x)])
  return(kid)
}
```

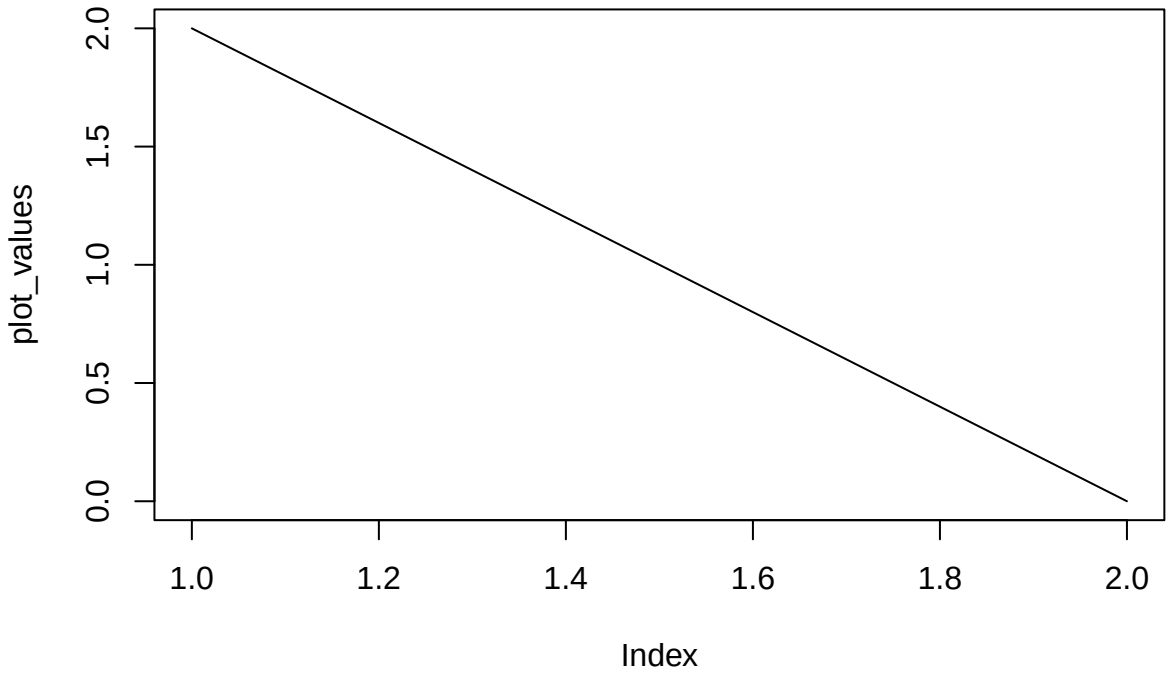
```r
#crossover3(l1,l2)

### 3)
mutate3<-function(x){
  choice<-sample(length(x),1)
  x[choice]<-sample(length(x),1)
  return(x)
}

#mutate3(l1)

### 4)
fitness_binary_3<-function(x){
  dim<-combn(length(x),2)

  a<-1 # if it is a legal config
  if(any(duplicated(x))==TRUE){
    a<-0
  }

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-0
    }else{
      count[i]<-1
    }
  }
  if(any(count==0)){
    a<-0
  }

  return(a)
}


fitness_attack_3<-function(x){
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}
```

```
fitness_pair_3<-function(x){
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
  return(a-number)
}

#fitness_pair_3(l1)


#For plotting
queens_attack_3<-function(x){
  dim<-combn(length(x),2)

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  return(number)
}

#queens_attack_3(l1)

### 5), 6), 7)
genetic_algo_3<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)
```

```
    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
    parent2<-population[[parent[2]]]

    kid<-crossover3(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate3(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid

    plot_values[i]<-queens_attack_3(kid)

    if(queens_attack_3(kid)==0){
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}
```

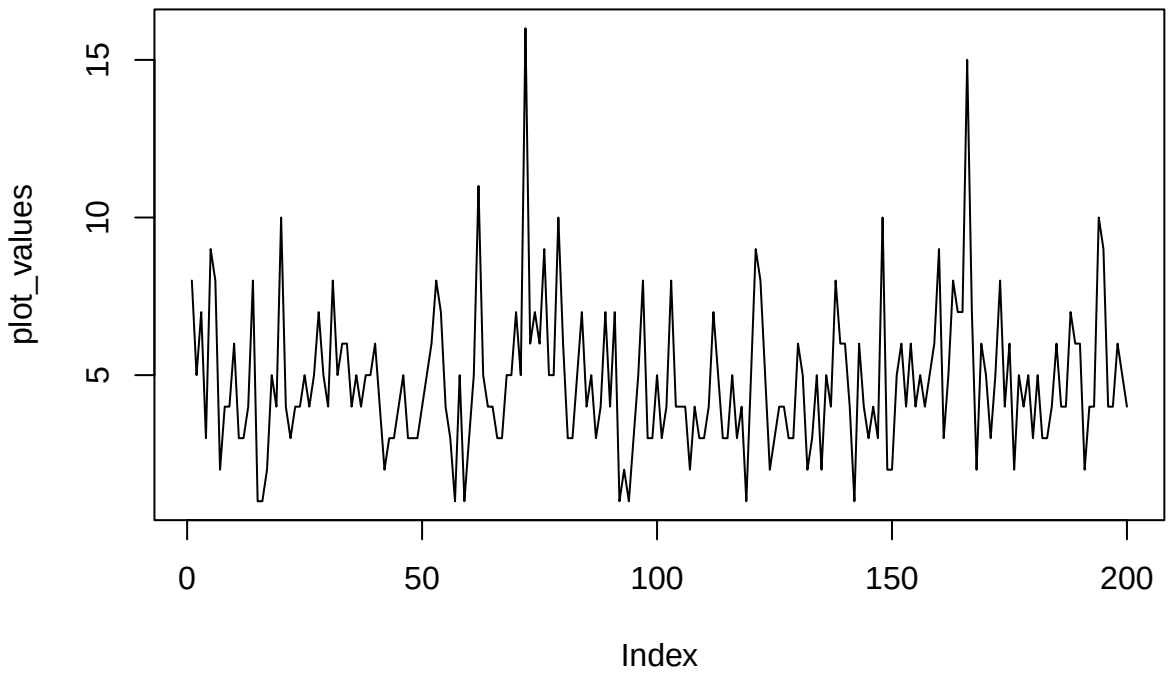
8)

```
genetic_algo_3(encoding3,0.5,fitness_binary_3,4)
```

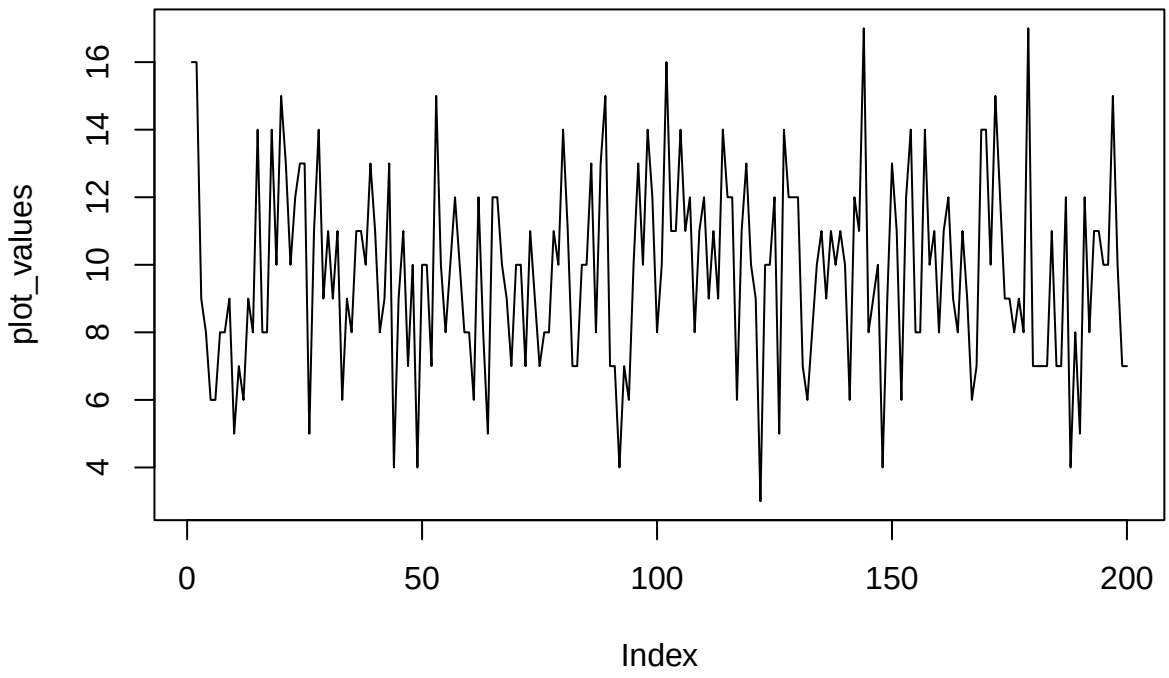



```
## [1] "legal configuration found"
```

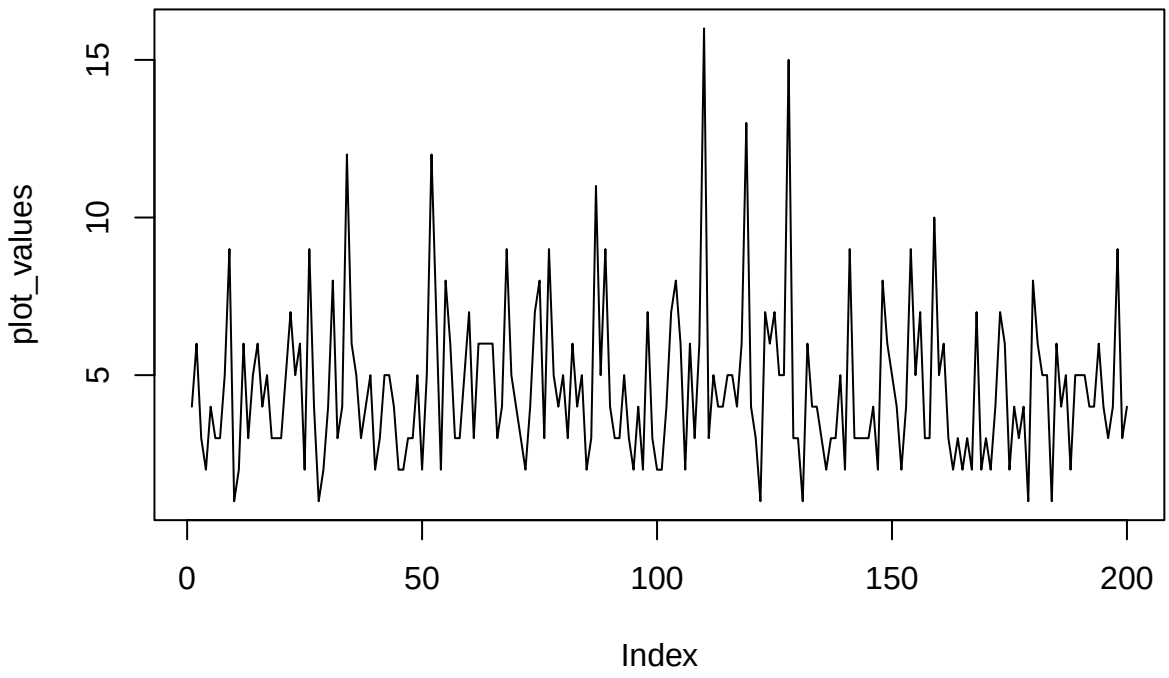
```
genetic_algo_3(encoding3,0.5,fitness_binary_3,8)
```



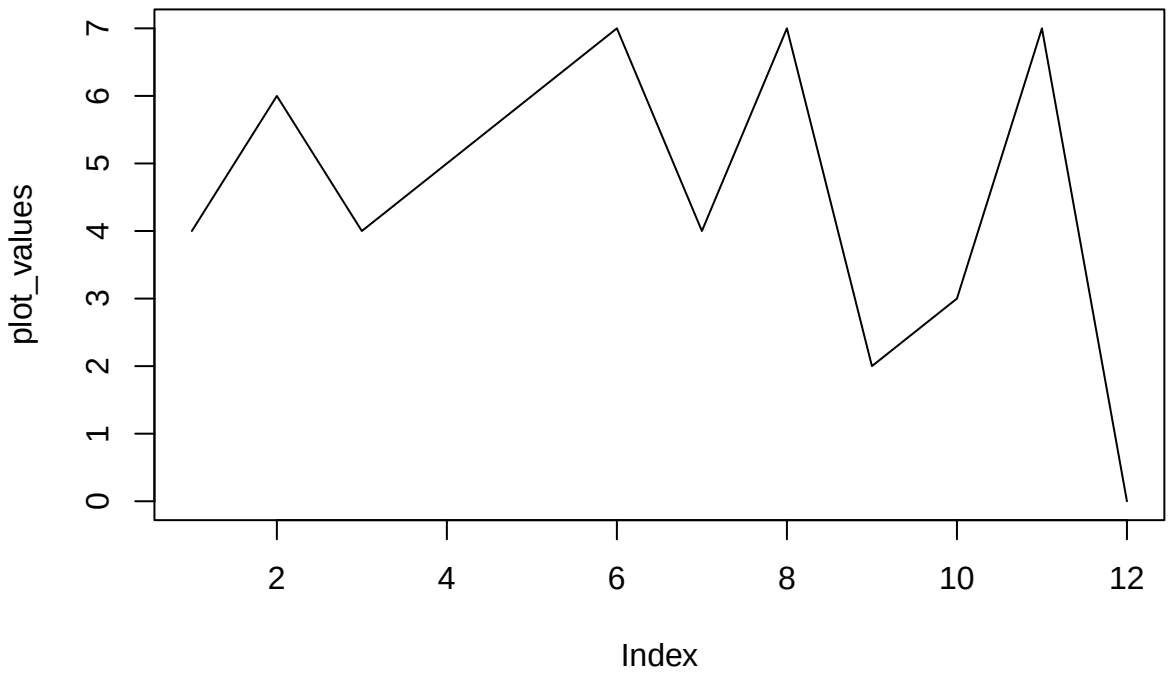

```
genetic_algo_3(encoding3,0.5,fitness_binary_3,16)
```




```
genetic_algo_3(encoding3,0.1,fitness_binary_3,8)
```

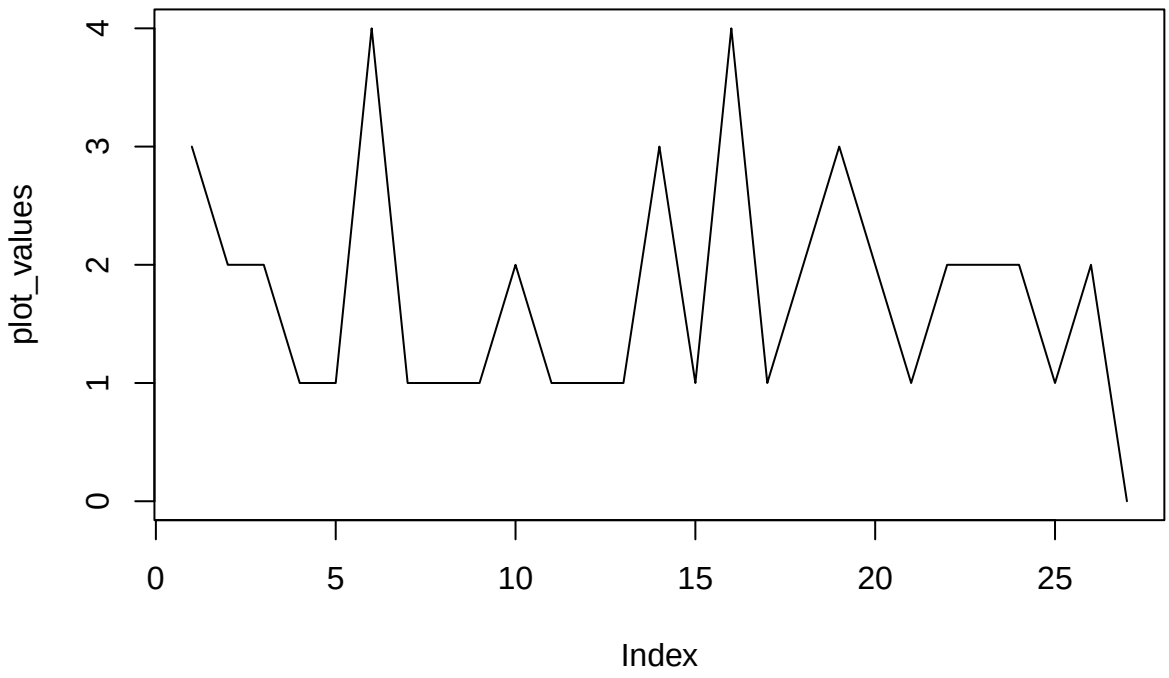



```
genetic_algo_3(encoding3,0.9,fitness_binary_3,8)
```



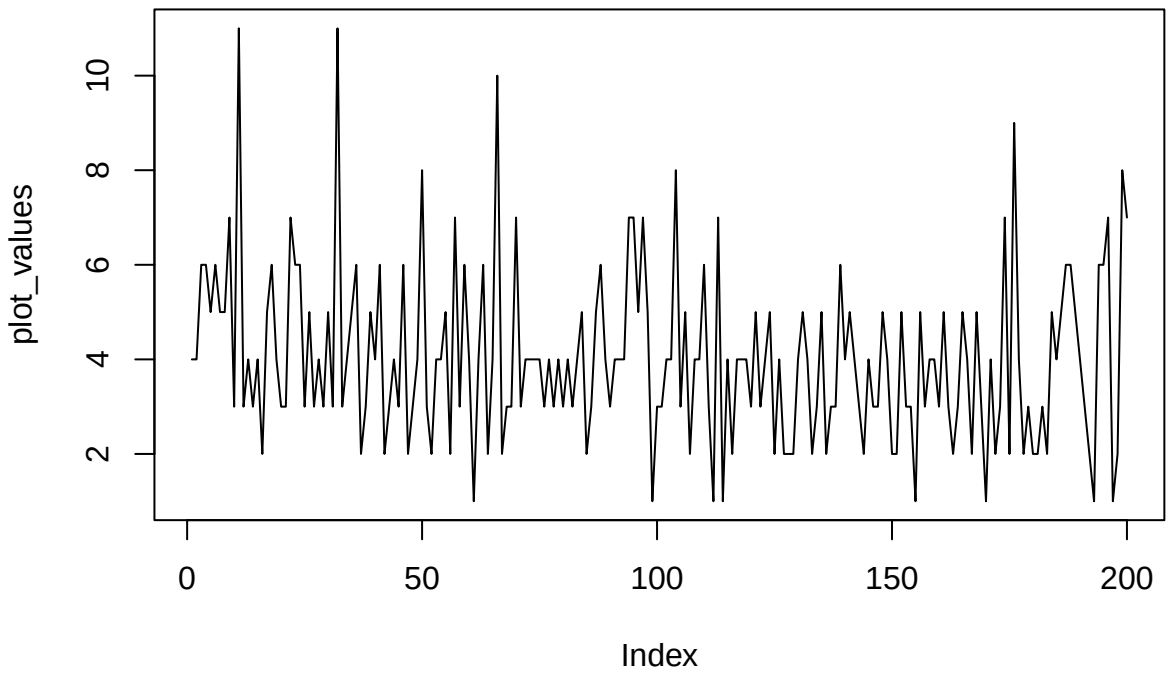

```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.5,fitness_attack_3,4)
```

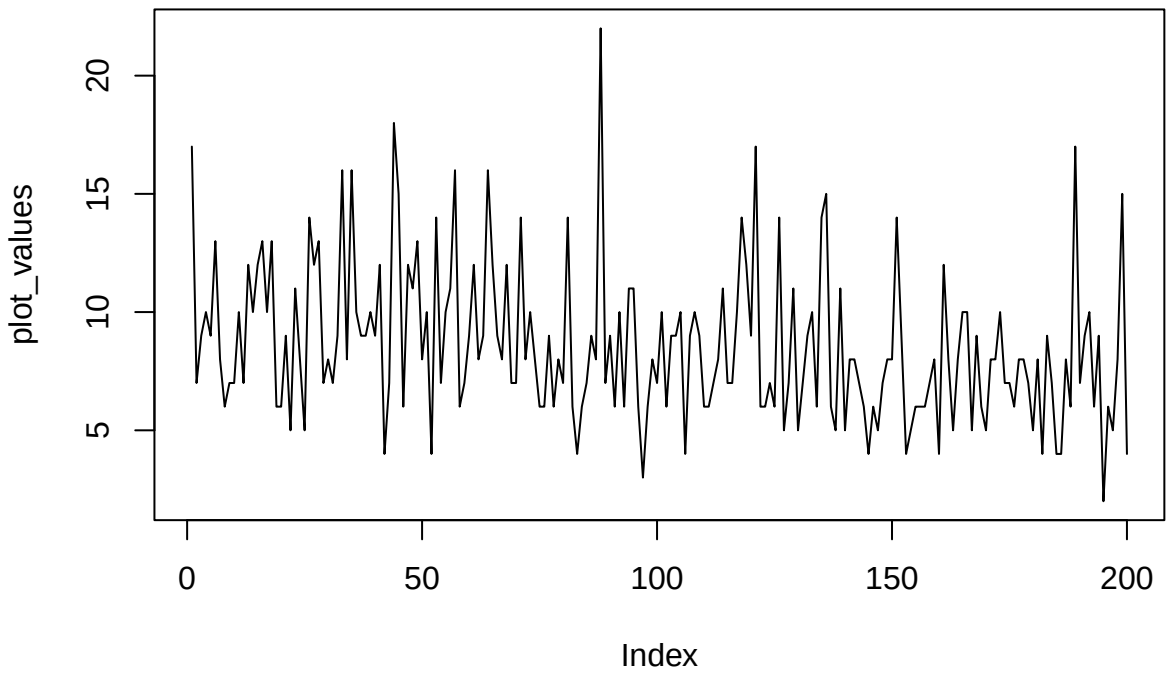



```
## [1] "legal configuration found"
```

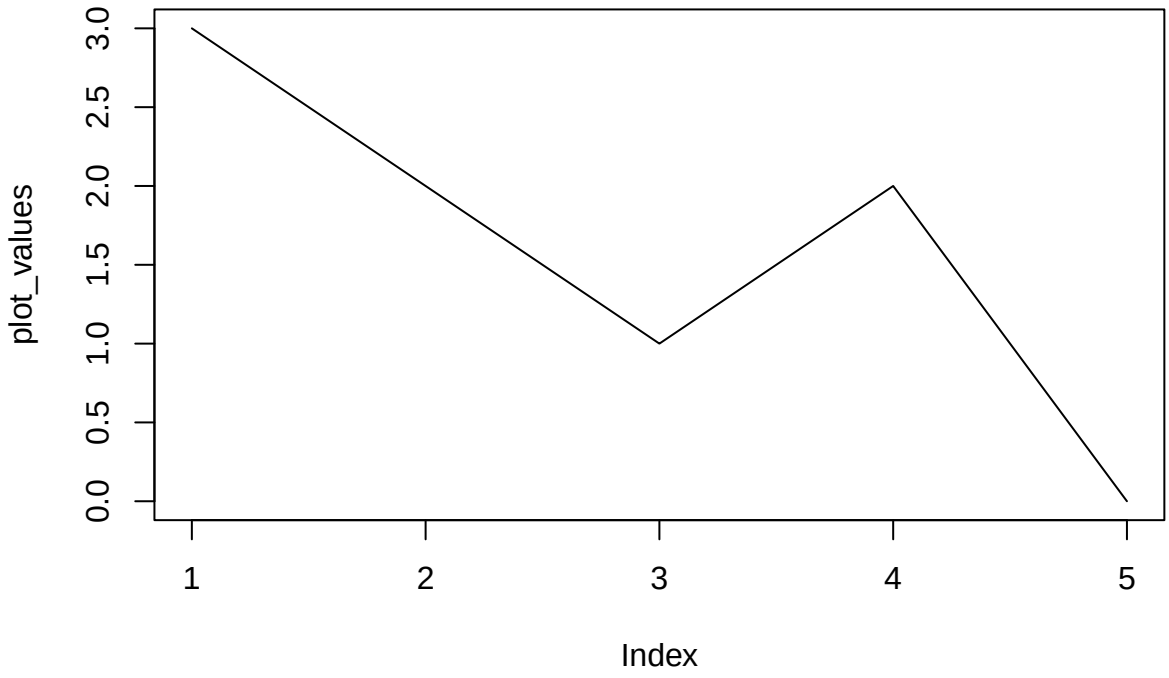
```
genetic_algo_3(encoding3,0.5,fitness_attack_3,8)
```




```
genetic_algo_3(encoding3,0.5,fitness_attack_3,16)
```

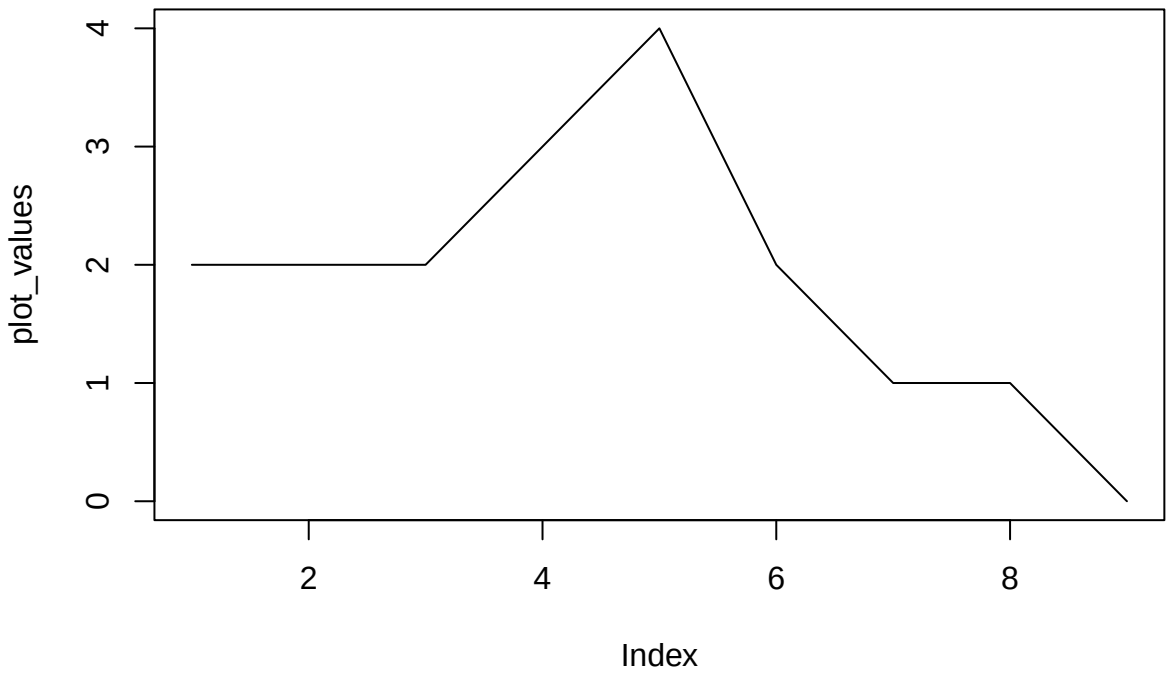



```
genetic_algo_3(encoding3,0.1,fitness_attack_3,4)
```



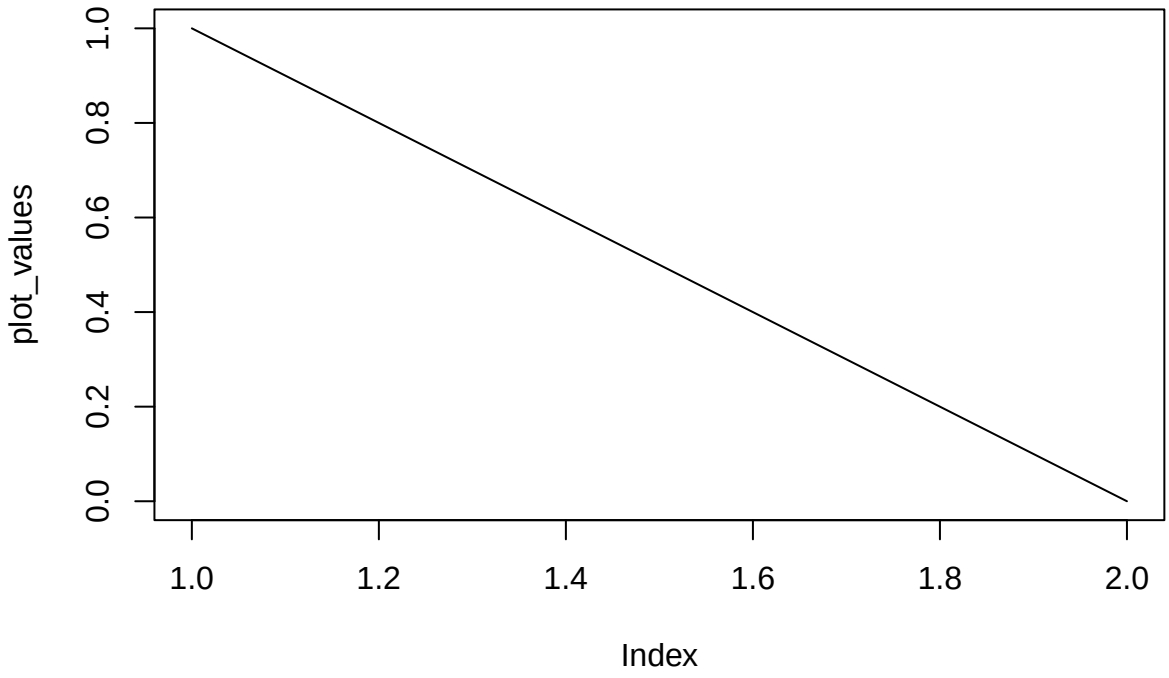

```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.9,fitness_attack_3,4)
```



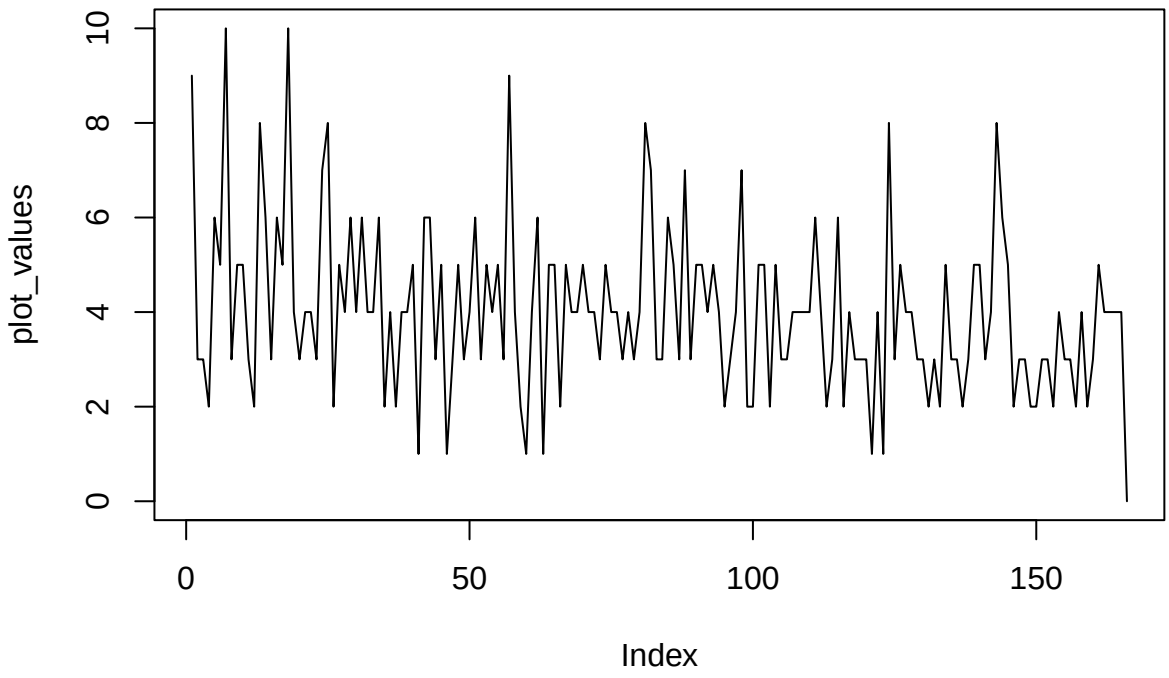

```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.5,fitness_pair_3,4)
```



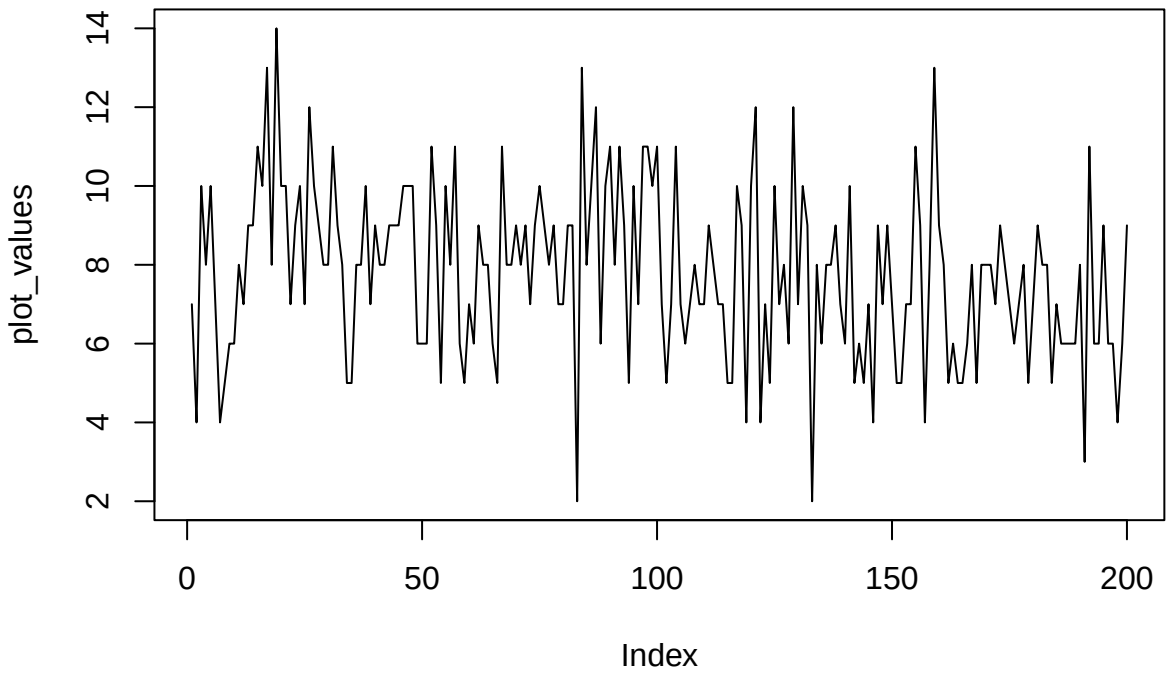

```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.5,fitness_pair_3,8)
```

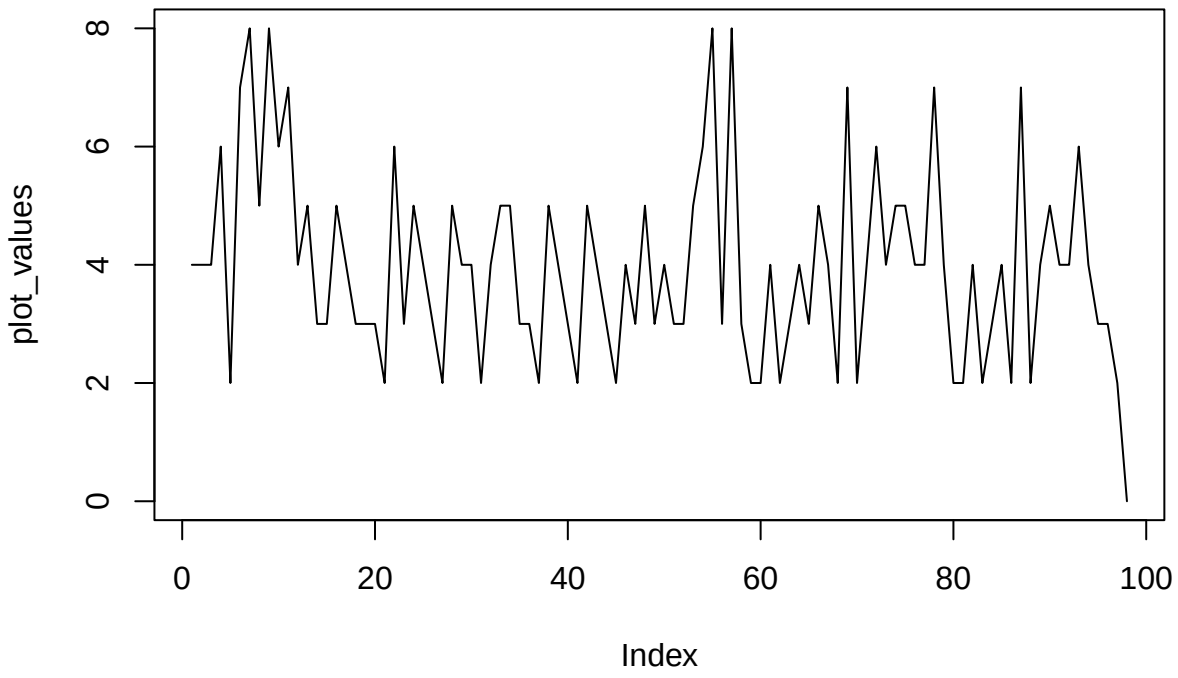



```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.5,fitness_pair_3,16)
```

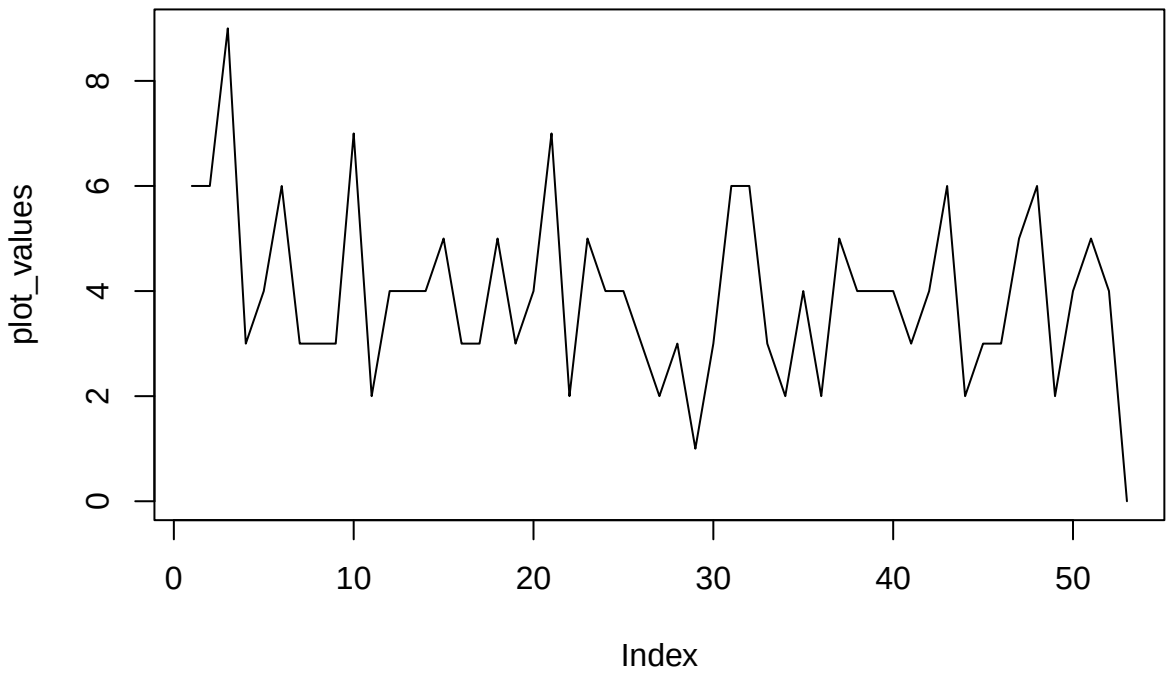



```
genetic_algo_3(encoding3,0.1,fitness_pair_3,8)
```




```
## [1] "legal configuration found"
```

```
genetic_algo_3(encoding3,0.9,fitness_pair_3,8)
```




```
## [1] "legal configuration found"
```

We find the most legal configurations for this for different input changes. We can sometimes repeat the same function multiple times to get the legal configuration if not found in the first try.

9) In terms of selecting which encoding would be the best, encoding 2 and 3 worked best to help find the legal configuration. But if we compare encoding 2 and 3, encoding 3 seems to provide more legal configurations based on the above inputs.

In terms of the fitness/objective function, the binary function doesn't work well as it just provides an output of 1 and 0, and doesn't say much about how good or bad each chessboard layout is so it isn't a very useful fitness function in this case. Even though we do get some legal configuration using this, the other fitness functions get the legal configuration in lesser iterations. While both the "number of queens not attacked" function and the "pairs of queens attacking each other" function seem to perform equally good to help get the legal configuration.

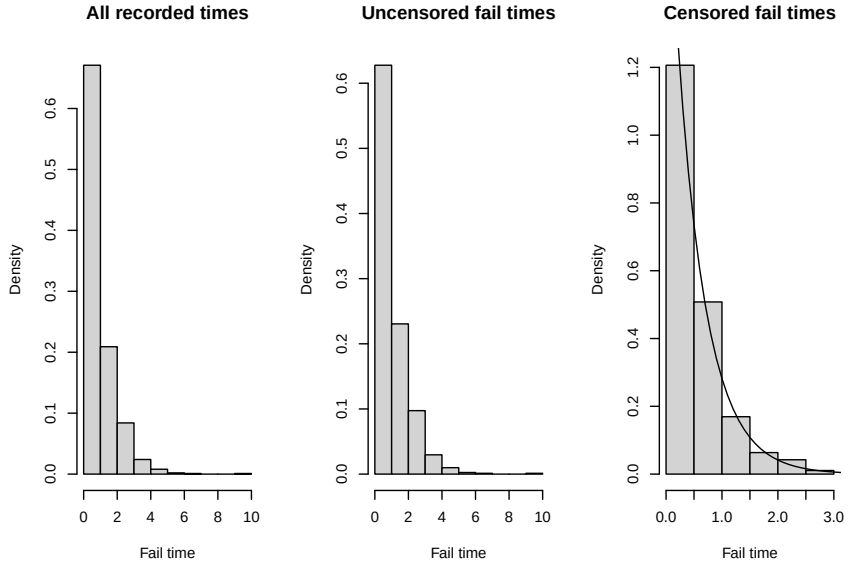
Question 2 - EM Algorithm

A certain type of product fails after random times. We are given a dataset of observations of these fail times. Most observations represent the exact fail time, but for some observations, the time that a product was discovered to have failed is recorded instead. In other words, these observations are censored. Assuming that fail times and time until discovery are independent, this would imply that the uncensored observations can be modeled by some probability distribution $X \sim D$, and the censored observations can be modeled

by the distribution of a random variable $X + Y$ where Y is a random variable from the distribution of the time between a product failing and it being discovered. In the instructions, it is stated that the censored measurements are left censored, which makes sense in this context.

Question 2.1

Figure 2.1: Density of fail times



All three histograms in Figure 2.1 appear approximately exponential. The density for all recorded times is not significantly different from the uncensored fail times, but it should be noted that the censored fail times only make up about 23 % of the observations.

Question 2.2

We now assume that all uncensored observations x_i, $i = 1, \cdots, n$, pairwise independently come from an exponential distribution with rate λ, and thus density $f(x) = \lambda e^{-\lambda x}$. For the censored observations, we will model the underlying true fail times z_i, $i = 1, \cdots, m$, each conditioned on the censored value y_i, $i = 1, \cdots, m$, as observations of a truncated exponential distribution:

$$Z|Y = y_i \sim \text{TruncExp}(\lambda, 0, y_i)$$

with density

$$g(z|y) = \left\{ \begin{array}{cc} \frac{\lambda \exp(-\lambda z)}{1-\exp(-\lambda y)} & 0 \leq z \leq y \\ 0 & \text{otherwise} \end{array} \right\}.$$

Thus the Likelihood of our observed data is

$$L(\lambda; X, Z|Y) = \left(\prod_{i=1}^{n} f(x_i) \right) \cdot \left(\prod_{i=1}^{m} g(Z_i|y_i) \right).$$

Observe that the Likelihood above is a random variable, since it is a function of the random variables $Z_i|y_i$, $i = 1, \cdots, m$.

Question 2.3

Because of our uncertainty in the true fail times z_i, we will employ the EM algorithm to estimate the MLE of λ. In the E step, we will construct the function $Q(\lambda, \lambda^{(t)}) = E_*[\log L(\lambda; X, Z|Y)]$, where the $*$ indicates that we take the expectation with respect to $Z|Y = y_i \sim \text{TruncExp}(\lambda^{(t)}, 0, y_i)$. This function is quite intricate, and can in the end be formulated as

$$Q(\lambda, \lambda^{(t)}) = n \cdot \ln \lambda - n + m \ln \lambda - \lambda \cdot \sum_{i=1}^{m} E_*[Z|Y = y_i] - \sum_{i=1}^{m} \ln \Big(1 - \exp(-\lambda y_i)\Big).$$

For clarity, n and m are the number of *un*censored and censored observations respectively in the data. The first two terms in expression above are the log likelihood contribution from the uncensored data, and the other terms are the log likelihood contribution from the censored data. The expectation in the expression above is found to be

$$E_*[Z|Y = y_i] = \frac{1 - \exp(-\lambda^{(t)} y_i) - y_i \lambda^{(t)} \exp(-\lambda^{(t)} y_i)}{\lambda^{(t)} \Big(1 - \exp(-\lambda^{(t)} y_i)\Big)} = \frac{1}{\lambda^{(t)}} - y_i \Big(\exp(\lambda^{(t)} y_i) - 1\Big)^{-1}.$$

In the M step of the EM algorithm, we simply optimise $Q(\lambda, \lambda^{(t)})$ with respect to λ, and set $\lambda^{(t+1)} = \text{argmax}_\lambda Q(\lambda, \lambda^{(t)})$.

Question 2.4

Now we implement the EM-algorithm.

```
# Question 2.4

Expected_Trunc_Exp <- function(rate, b=Inf){
  # Returns the expected value of an exp distribution truncated on [0, b].

  numerator <- (-exp(-rate*b) + 1 + -b*rate*exp(-rate*b))
  denominator <- (rate*(1-exp(-rate*b)))

  return(numerator / denominator)
}

Q <- function(lambda, lambda_t){
  # Function for the expected value of the Likelihood of the data

  censored_data <- df_prodfails %>% filter(censored == "yes")
  n <- df_prodfails %>% filter(censored == "no") %>% extract2(1) %>% length()
  m <- censored_data$time %>% length()

  # This term represents the log likelihood contribution from the fully
  # observed data to the expectation
  term1 <- n * log(lambda) - n

  # The following terms make up the log likelihood contribution from the
  # censored observations to the expectation
  term2 <- m * log(lambda)
```

```r
  term3 <- - lambda * sum(Expected_Trunc_Exp(lambda_t, b = censored_data$time))

  term4 <- - sum(log(1-exp(-lambda * censored_data$time)))

  return(term1 + term2 + term3 + term4)
}


# The EM Algorithm
EM_algorithm <- function(lambda_start = 100, epsilon = 0.001){
  # Setup
  k <- 0
  k_max <- 100
  lambda_t = lambda_start
  stop <- FALSE

  # Keep iterating until convergence is reached
  while(stop != TRUE){
    k <- k + 1

    # E Step: Q is already (implicitly) constructed

    # M Step
    lambda_new <- optimise(f = Q,
                           lambda_t = lambda_t,
                           maximum = T,
                           interval=c(0, 10^3))$maximum

    # Evaluate stop conditions
    stop <- k <= k_max | abs(lambda_t - lambda_new) < epsilon

    # Accept the new lambda_t (if the algorithm has not been stopped)
    lambda_t <- lambda_new
  }

  cat("Optimal lambda:", lambda_new, "\n")
  cat("Iterations:", k)
}

EM_algorithm()
```

```
## Optimal lambda: 548.1614
## Iterations: 1
```

When we implement the EM algorithm above, we run into an interesting error. The estimate of λ tends to infinity (bounded by an arbitrary parameter interval we set). This is because our formulation of $Q(\lambda, \lambda^{(t)})$ grows strictly with λ. This is likely due to some error in our derivation.

Appendix

```
library(compositions)

### 1)

### a)
encoding1<-function(n){
  data<-list()
  for(i in 1:n){
    data[[i]]<-c(sample(n,1),sample(n,1))
    while(any(duplicated(data))== TRUE){
      data[[which(duplicated(data))]]<-c(sample(n,1),sample(n,1))
    }
  }
  return(data)
}

#layout1<-encoding1(8)
#layout2<-encoding1(8)


### 2)
crossover<-function(x,y){
  p<-sample(1:(length(x)/2),1)
  kid<-c(x[1:p],y[(p+1):length(x)])

  #while(any(duplicated(kid)==TRUE)){
  #  p<-sample((0:(length(x)/2)),1)
  #  kid<-c(x[1:p],y[(p+1):length(x)])
  #}
  return(kid)
}

#kid1<-crossover(layout1,layout2)

### 3)

mutate<-function(x){
  choice<-sample(length(x),1)
  a<-x[[choice]]
  x[[choice]]<-c(sample(length(x),1),sample(length(x),1))
  while(any(duplicated(x))== TRUE & all(x[[choice]] == a)){ #To ensure that the queen moves to a posito
    x[[which(duplicated(x))]]<-c(sample(length(x),1),sample(length(x),1))
  }
  return(x)
}

#mutate(layout1)

### 4)

fitness_binary<-function(x){
```

```
  a<-1

  #row check
  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  #column check
  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }

  if(any(duplicated(row))==TRUE){
    a<-0 #0 implies that it is not a solution
  }

  if(any(duplicated(col))==TRUE){
    a<-0
  }

  #diagonal check
  for(i in 1:length(x)){
    for(j in 1:length(x)){
      if(row[i]==row[j] & col[i]==col[j]){
        next
      }
      if(abs(row[i]-row[j])==abs(col[i]-col[j])){
        a<-0
      }
    }
  }

  return(a)
}

#fitness_binary(layout1)

fitness_attack<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }

  #non_attack<-length(x)
```

```
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==a
        count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}


#fitness_attack(layout1)

fitness_pair<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }

  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }

  #non_attack<-length(x)
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==a
      count[i]<-1
    }
  }
  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
  return(a-number)
}

#fitness_pair(layout1)


#For plotting
queens_attack<-function(x){
  dim<-combn(length(x),2)

  row<-c()
  for(i in 1:length(x)){
    row[i]<-x[[i]][1]
  }
```

```
  col<-c()
  for(i in 1:length(x)){
    col[i]<-x[[i]][2]
  }
  count<-c()
  for(i in 1:ncol(dim)){
    if(row[dim[1,i]]==row[dim[2,i]] | col[dim[1,i]]==col[dim[2,i]] | abs(row[dim[1,i]]-row[dim[2,i]])==a
      count[i]<-1
    }
  }
  number<-sum(count,na.rm = TRUE)
  return(number)
}

#queens_attack(layout1)


### 5), 6), 7)

genetic_algo<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)


    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
    parent2<-population[[parent[2]]]

    kid<-crossover(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid

    plot_values[i]<-queens_attack(kid)
```

```
    if(queens_attack(kid)==0){
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}

genetic_algo(encoding1,0.8,fitness_binary,4)
genetic_algo(encoding1,0.8,fitness_binary,4)
genetic_algo(encoding1,0.5,fitness_binary,8)
genetic_algo(encoding1,0.5,fitness_binary,16)
genetic_algo(encoding1,0.1,fitness_binary,8)
genetic_algo(encoding1,0.9,fitness_binary,8)


genetic_algo(encoding1,0.5,fitness_attack,4)
genetic_algo(encoding1,0.5,fitness_attack,8)
genetic_algo(encoding1,0.5,fitness_attack,16)
genetic_algo(encoding1,0.1,fitness_attack,8)
genetic_algo(encoding1,0.9,fitness_attack,8)


genetic_algo(encoding1,0.5,fitness_pair,4)
genetic_algo(encoding1,0.5,fitness_pair,8)
genetic_algo(encoding1,0.5,fitness_pair,16)
genetic_algo(encoding1,0.1,fitness_pair,8)
genetic_algo(encoding1,0.9,fitness_pair,8)
toBits <- function (x, nBits = 8){
  tail(rev(as.numeric(intToBits(x))),nBits)
}

#toBits(8,5)

### 1)

### b)

encoding2<-function(n){
  index<-sample(n,n,replace = FALSE)
  output<-list()
  for(i in 1:n){
    #output[[i]]<-rep(0,n)
    output[[i]]<-toBits(index[i],log2(n))
  }
  return(output)
}

#e1<-encoding2(8)
#e2<-encoding2(8)
```

```r
#To convert bits to integer
bits_convert<-function(x){
  b<-c()
  for(i in 1:length(x)){
    c<-""
    for(j in 1:length(x[[1]])){
      c<-paste(c,x[[i]][j],sep = "")
    }
    b[i]<-unbinary(c)
  }
  b[b==0]<-length(x)
  return(b)
}

#bits_convert(e1)

#unbinary("111")
#b<-c()
#for(i in 1:8){
#  b[i]<-unbinary(paste(e1[[i]][1],e1[[i]][2],e1[[i]][3],sep = ""))
#}
# 0 implies 8


### 2)
crossover2<-function(x,y){
  kid<-list()
  for(i in 1:length(x)){
    p<-sample(1:round(log2(length(x))/2),1)
    kid[[i]]<-c(x[[i]][1:p],y[[i]][(p+1):log2(length(x))])
  }
  return(kid)
}

#crossover2(e1,e2)

### 3)
mutate2<-function(x){
  choice<-sample(length(x),1)
  value<-sample(length(x),1)
  x[[choice]]<-toBits(value,log2(length(x)))

  return(x)
}

#mutate2(e1)


### 4)
fitness_binary_2<-function(x){
  x<-bits_convert(x)

  dim<-combn(length(x),2)
```

```
  a<-1 # if it is a legal config
  if(any(duplicated(x))==TRUE){
    a<-0
  }

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-0
    }else{
      count[i]<-1
    }
  }
  if(any(count==0)){
    a<-0
  }

  return(a)
}

fitness_attack_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}

fitness_pair_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
  return(a-number)
}
```

```
#For plotting
queens_attack_2<-function(x){
  x<-bits_convert(x)
  dim<-combn(length(x),2)

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  return(number)
}

### 5), 6), 7)
genetic_algo_2<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)


    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
    parent2<-population[[parent[2]]]

    kid<-crossover2(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate2(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid

    plot_values[i]<-queens_attack_2(kid)

    if(queens_attack_2(kid)==0){
```

```
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}

genetic_algo_2(encoding2,0.8,fitness_binary_2,4)
genetic_algo_2(encoding2,0.5,fitness_binary_2,8)
genetic_algo_2(encoding2,0.5,fitness_binary_2,16)
genetic_algo_2(encoding2,0.1,fitness_binary_2,8)
genetic_algo_2(encoding2,0.9,fitness_binary_2,8)


genetic_algo_2(encoding2,0.5,fitness_attack_2,4)
genetic_algo_2(encoding2,0.5,fitness_attack_2,8)
genetic_algo_2(encoding2,0.5,fitness_attack_2,16)
genetic_algo_2(encoding2,0.1,fitness_attack_2,4)
genetic_algo_2(encoding2,0.9,fitness_attack_2,4)


genetic_algo_2(encoding2,0.5,fitness_pair_2,4)
genetic_algo_2(encoding2,0.5,fitness_pair_2,8)
genetic_algo_2(encoding2,0.5,fitness_pair_2,16)
genetic_algo_2(encoding2,0.1,fitness_pair_2,8)
genetic_algo_2(encoding2,0.9,fitness_pair_2,8)

### 1)

### c)

encoding3<-function(n){
  output<-sample(n,n,replace = FALSE)
  return(output)
}

l1<-encoding3(8)
l2<-encoding3(8)

### 2)

crossover3<-function(x,y){
  p<-sample(1:(length(x)/2),1)
  kid<-c(x[1:p],y[(p+1):length(x)])
  return(kid)
}

#crossover3(l1,l2)

### 3)
```

```
mutate3<-function(x){
  choice<-sample(length(x),1)
  x[choice]<-sample(length(x),1)
  return(x)
}

#mutate3(l1)

### 4)
fitness_binary_3<-function(x){
  dim<-combn(length(x),2)

  a<-1 # if it is a legal config
  if(any(duplicated(x))==TRUE){
    a<-0
  }

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-0
    }else{
      count[i]<-1
    }
  }
  if(any(count==0)){
    a<-0
  }

  return(a)
}


fitness_attack_3<-function(x){
  dim<-combn(length(x),2)
  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }
  attack<-dim[,which(count==1)]
  a<-ncol(dim)
  if(any(dim(attack)>0)==TRUE){
    a<-abs(length(unique(c(unique(attack[1,]),unique(attack[2,]))))-length(x))
  }
  return(a)
}


fitness_pair_3<-function(x){
  dim<-combn(length(x),2)
  count<-c()
```

```
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  a<-factorial(length(x))/(factorial(length(x)-2)*2)
  return(a-number)
}

#fitness_pair_3(l1)


#For plotting
queens_attack_3<-function(x){
  dim<-combn(length(x),2)

  count<-c()
  for(i in 1:ncol(dim)){
    if(abs(x[dim[1,i]]-x[dim[2,i]])==abs(dim[1,i]-dim[2,i])){
      count[i]<-1
    }
  }

  number<-sum(count,na.rm = TRUE)
  return(number)
}

#queens_attack_3(l1)

### 5), 6), 7)
genetic_algo_3<-function(encoding,mutation,fitness,n){
  population<-list()
  fitness_value<-c()
  for(i in 1:100){
    population[[i]]<-encoding(n)
    fitness_value[i]<-fitness(population[[i]])
  }

  plot_values<-c()
  for(i in 1:200){
    #print(i)
    worst_index <- which.min(fitness_value)

    #best<-population[[tail(order(fitness_value),1)]]
    best<-population[[which.max(fitness_value)]]
    #plot_values[i]<-queens_attack(best)


    parent<-sample(1:100,2)
    victim<-population[[order(fitness_value)[1]]]
    parent1<-population[[parent[1]]]
```

```
    parent2<-population[[parent[2]]]

    kid<-crossover3(parent1,parent2)

    if(runif(1,0,1)>=mutation){
      kid<-mutate3(kid)
    }
    fitness_kid <- fitness(kid)

    population[[worst_index]] <- kid
    fitness_value[worst_index] <- fitness_kid

    plot_values[i]<-queens_attack_3(kid)

    if(queens_attack_3(kid)==0){
      plot(plot_values,type="l")
      return("legal configuration found")
    }
    #population[[order(fitness_value)[1]]]<-kid
    #fitness_value[order(fitness_value)[1]]<-fitness_pair(kid)
  }
  plot(plot_values,type="l")
  #return(queens_attack(kid))
}
genetic_algo_3(encoding3,0.5,fitness_binary_3,4)
genetic_algo_3(encoding3,0.5,fitness_binary_3,8)
genetic_algo_3(encoding3,0.5,fitness_binary_3,16)
genetic_algo_3(encoding3,0.1,fitness_binary_3,8)
genetic_algo_3(encoding3,0.9,fitness_binary_3,8)


genetic_algo_3(encoding3,0.5,fitness_attack_3,4)
genetic_algo_3(encoding3,0.5,fitness_attack_3,8)
genetic_algo_3(encoding3,0.5,fitness_attack_3,16)
genetic_algo_3(encoding3,0.1,fitness_attack_3,4)
genetic_algo_3(encoding3,0.9,fitness_attack_3,4)


genetic_algo_3(encoding3,0.5,fitness_pair_3,4)
genetic_algo_3(encoding3,0.5,fitness_pair_3,8)
genetic_algo_3(encoding3,0.5,fitness_pair_3,16)
genetic_algo_3(encoding3,0.1,fitness_pair_3,8)
genetic_algo_3(encoding3,0.9,fitness_pair_3,8)

# Setup

library(magrittr)
library(dplyr)

# Read in the data, and create a column with clear indication whether an
# observation is censored or not. To be clear, cens=2 indicates that an
# observation *is* censored, and thus "yes".
df_prodfails <- read.csv("censoredproc.csv", sep = ";") %>%
```

```r
  dplyr::mutate(censored = c("no", "yes")[(cens == 2)+1])
# Question 2.1

# Create plot grid
layout(mat=matrix(c(1,2, 3), nrow = 1, byrow = T))

hist(df_prodfails$time,
     freq = F,
     main = "All recorded times",
     xlab = "Fail time")

hist(df_prodfails %>% filter(censored == "no") %>% extract2(1),
     freq=F,
     main = "Uncensored fail times",
     xlab = "Fail time")

hist(df_prodfails %>% filter(censored == "yes") %>% extract2(1),
     freq=F,
     main = "Censored fail times",
     xlab = "Fail time")

# Add exponential density
lambda_estim <- 1/(df_prodfails %>% filter(censored == "yes") %>% extract2(1) %>% mean())
x_seq <- seq(0, 10, 0.1)
points(x_seq, dexp(x_seq, rate=lambda_estim), type="l")

# Set figure title
mtext("Figure 2.1: Density of fail times", side=3, outer=TRUE, line=-1)

# Question 2.4

Expected_Trunc_Exp <- function(rate, b=Inf){
  # Returns the expected value of an exp distribution truncated on [0, b].

  numerator <- (-exp(-rate*b) + 1 + -b*rate*exp(-rate*b))
  denominator <- (rate*(1-exp(-rate*b)))

  return(numerator / denominator)
}

Q <- function(lambda, lambda_t){
  # Function for the expected value of the Likelihood of the data

  censored_data <- df_prodfails %>% filter(censored == "yes")
  n <- df_prodfails %>% filter(censored == "no") %>% extract2(1) %>% length()
  m <- censored_data$time %>% length()

  # This term represents the log likelihood contribution from the fully
  # observed data to the expectation
  term1 <- n * log(lambda) - n

  # The following terms make up the log likelihood contribution from the
  # censored observations to the expectation
```

```r
  term2 <- m * log(lambda)

  term3 <- - lambda * sum(Expected_Trunc_Exp(lambda_t, b = censored_data$time))

  term4 <- - sum(log(1-exp(-lambda * censored_data$time)))

  return(term1 + term2 + term3 + term4)
}


# The EM Algorithm
EM_algorithm <- function(lambda_start = 100, epsilon = 0.001){
  # Setup
  k <- 0
  k_max <- 100
  lambda_t = lambda_start
  stop <- FALSE

  # Keep iterating until convergence is reached
  while(stop != TRUE){
    k <- k + 1

    # E Step: Q is already (implicitly) constructed

    # M Step
    lambda_new <- optimise(f = Q,
                           lambda_t = lambda_t,
                           maximum = T,
                           interval=c(0, 10^3))$maximum

    # Evaluate stop conditions
    stop <- k <= k_max | abs(lambda_t - lambda_new) < epsilon

    # Accept the new lambda_t (if the algorithm has not been stopped)
    lambda_t <- lambda_new
  }

  cat("Optimal lambda:", lambda_new, "\n")
  cat("Iterations:", k)
}

EM_algorithm()
```