Statistical Computing Algorithms

Chan Ho Ying	1155109018	
Choi Sen Hei	1155109412	
Chui Man Keung	1155110496	
Chan King Yeung	1155119394	
Lau Tsz Hin	1155110584	Group 7

The listed methods

Root finding (bisection, Newton-Raphson)

Optimization (genetic algorithm)

Numerical integration (Quadrature)

Monte Carlo integration (distribution sampling)

Question 1

1. implement both bisection and Newton-Raphson algorithms to find the maximal value of $\frac{log(x+x^2)}{1+x^3}$, where x>0

Bisection(Hand Calculation)

Let f(x) = ln(x+x^2)/1+x^3, where x>0

Here

x	2	1	0.5	0
f(x)	0.1991	0.3466	0.3589	“undefined”

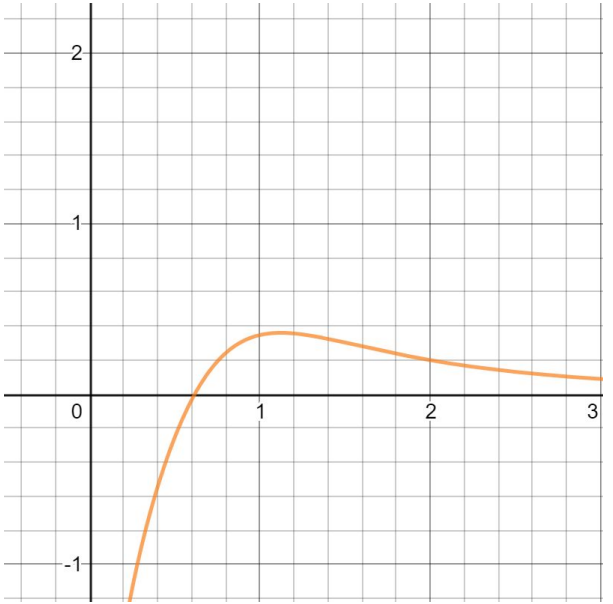
Let a1=2 , b1=1, c1=0.5 ,xi=(ai+bi)/2 , yi=(bi+ci)2

When f(x1) > f(a1) , x1 will be used for a2 next step.

When f(x1) > f(b1) , x1 will be used for b2 next step.

When f(y1) > f(b1) , y1 will be used for b2 next step.

When f(y1) > f(c1) , y1 will be used for c2 next step.



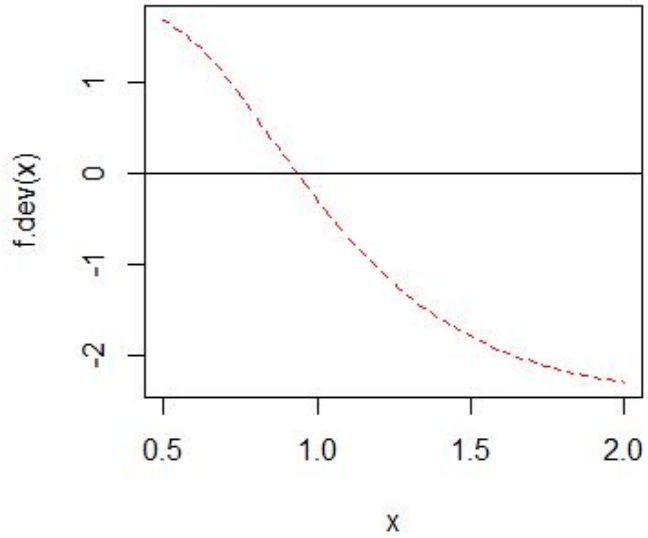
Bisection(Hand Calculation)

i	ai	f(ai)	xi	f(xi)	bi	f(bi)	yi	f(yi)	ci	f(ci)
1	1.5	0.3021	1.25	0.3502	1	0.3465	0.75	0.1913	0.5	-0.2557
2	1.25	0.3502	1.25	0.3502	1.25	0.3502	1	0.3465	0.75	0.1913
3	/	/	/	/	1.25	0.3502	1.125	0.3596	1	0.3465
4	/	/	/	/	1.125	0.3596	1.125	0.3596	1.125	0.3596

Therefore when **x=1.125** , **f(x)=0.3596** where is the maximal value of f(x).

Bisection(By R Program)

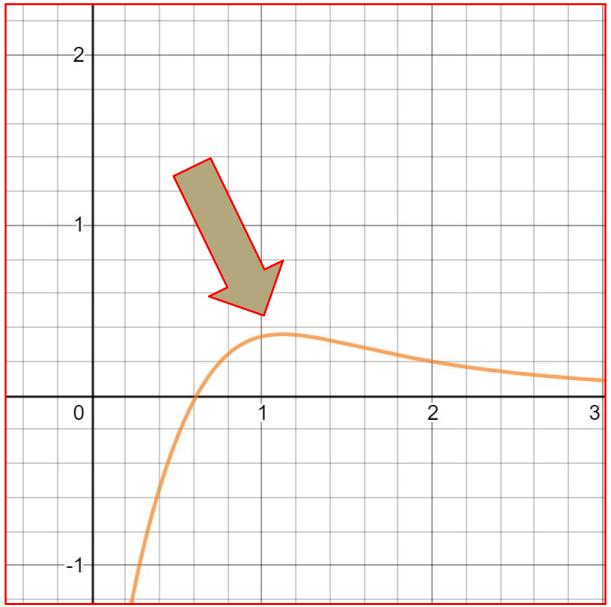
```
f.dev <- function(x){
(1+2*x)/((x+x^2)*(1+x^3))- log(x+x^2)*(3*x^2)/(1+x^3)^2
}

curve(f.dev, xlim=c(0.5,2), col='red', lwd=1.5, lty=2)
abline(h=0)
```



Bisection(By R Program)

```
bisection <- function(f.dev, a, b, z=0.001) {
if(f(a)*f(b)>0)
list(fail="fail to find root")
else{
repeat{
if(abs(b-a)<z) break
x<-(a+b)/2
if(f(a)*f(x)<0) b<-x else a<-x
}
list(root=(a+b)/2,func=f(x))
}
}
f <- function(x){
(1+2*x)/((x+x^2)*(1+x^3))-
log(x+x^2)*(3*x^2)/(1+x^3)^2
}
bisection(f,0.5,2)
$root
[1] 1.123657
```

x<-1.123657
log(x+x^2)/(1+x^3)
[1] 0.3595798



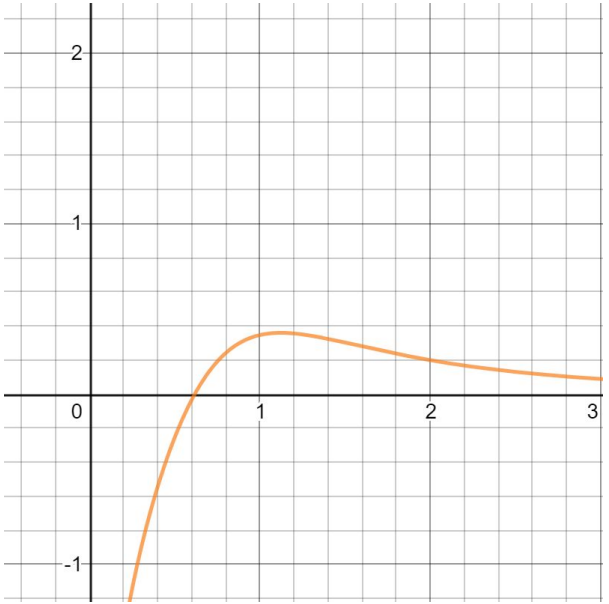
Newton-Raphson algorithms(Hand Calculation)

Let f(x) = ln(x+x^2)/1+x^3, where x>0

x	1.5	1	0.5	0
f(x)	0.3021	0.3466	0.3589	“undefined”

We need to find out f’(x) and f’’(x).

Let a=1.5 b=1, c=0.5 , x0=(a+b)/2 , y0=(b+c)/2
For xi=xi-1 - f'(xi-1)/f''(xi-1)



Newton-Raphson algorithms(Hand Calculation)

i	xi	f(xi)	f'(xi)	f''(xi)	dx (xi-1-xi)
0	1.25	0.3502	-0.1344	-0.7462	-0.1802
1	1.0698	0.3573	0.0858	-1.7939	0.0479
2	1.1176	0.3596	0.0082	-1.4593	0.0056
3	1.1232	0.3596	0.00016	-1.4228	0
4	1.1232	0.3596	0.00016	-1.4228	/

For xi=xi-1 - f'(xi-1)/f''(xi-1)

For this table, when **x=1.1232** , **f(x)=0.3596**, where is the maximal value of f(x).

Newton-Raphson algorithms(Hand Calculation)

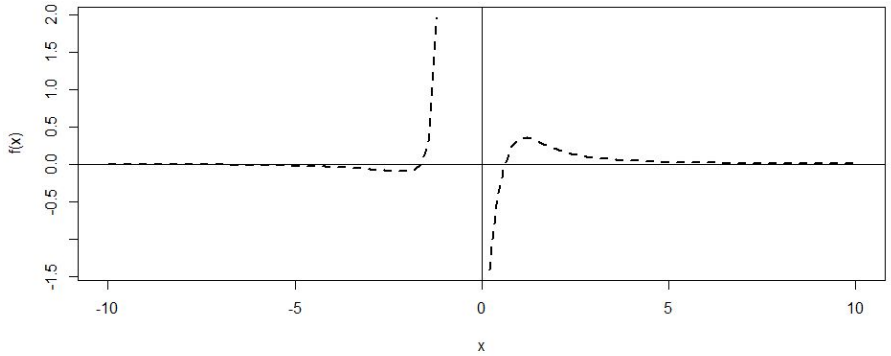
i	yi	f(yi)	f'(yi)	f''(yi)	dy (yi-1-yi)
0	0.75	0.1913	1.1126	-4.7262	0.2354
1	0.9854	0.3430	0.2655	-2.4825	0.1069
2	1.0923	0.3589	0.0473	-1.6312	0.0290
3	1.1213	0.3596	0.0029	-1.4351	0.0003
4	1.1233	0.3596	0.000008	-0.6176	0
5	1.1233	0.3596	0.000008	-0.6176	/

For yi=yi-1 - f'(yi-1)/f''(yi-1)

For this table, when **y=1.1233** , **f(y)=0.3596**, where is the maximal value of f(x).

Newton-Raphson algorithms(By R Program)

```
func <- function(x) {
(log(x+x^2))/(1+x^3))
}

curve(func, xlim=c(-10,10), col='black', lwd=2, lty=2, ylab='f(x)')
abline(h=0)
abline(v=0)
```



Newton-Raphson algorithms(By R Program)

```
f <- function(x){
  (2 * x + 1 ) / ((x^2 + x) * (x^3 + 1)) - ((3 * x^2 * log(x^2 + x)) / (x^3 + 1)^2)
}
#first derivative f'(x)
fdev <- function(x){
  (2 / (x + x^2) - (1 + 2*x) * (1 + 2 * x) / ( x + x^2)^2) / (1 + x^3) - (1 + 2 * x) / (x + x^2) * (3 * x^2) / (1 + x^3)^2 - (((1 + 2 * x) / (x + x^2) * (3 * x^2) + log(x + x^2) * (3 * (2 * x))) / (1 + x^3)^2 - log(x + x^2) * (3 * x^2) * (2 * (3 * x^2 * (1 + x^3))) / ((1 + x^3)^2)^2)
}
#second derivative f''(x)

x<-0.5
tol<-1e-5
root<- function(f, f.dev, x, tol){
  y=x
```

Newton-Raphson algorithms(By R Program)

#when f(y) is < tolerance i.e. e-05 , output the results

```
  while(abs(f(y)) > tol){
    y = y - f(y)/f.dev(y)
  }
  return(y)
}
root(f, f.dev, x, tol)
```

```
> root(f, fdev, x, tol)
[1] 1.123312

> x<-1.123312
> log(x + x^2) / (1 + x^3)
[1] 0.3595798
```

Newton-Raphson VS Bisection

Newton Raphson:

- converges much faster
- requires only a single function evaluation per iteration
- limitation upon the derivative of function in the question

Newton-Raphson VS Bisection

Bisection:

- Slow
- root is within the bound of interval → is guaranteed to converge but its rate of convergence is too slow.
- quite difficult to extend to use for systems of equations

Newton-Raphson VS Bisection

NR method is relatively more effective than Bisection method in terms of its order of convergence

Conclusion:

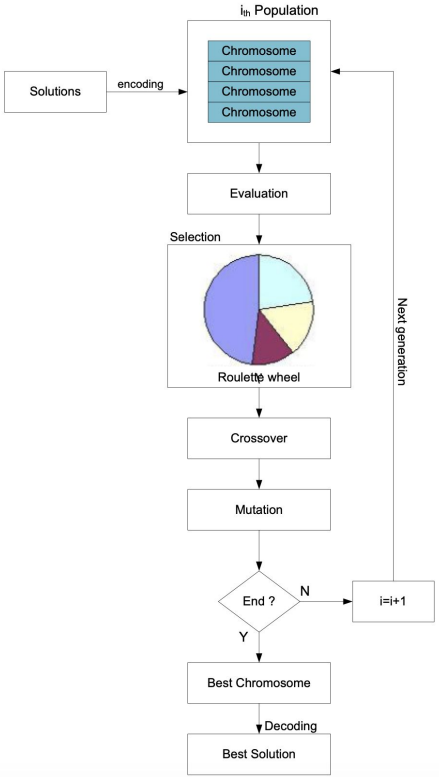
Using NR method is prefered when comparing with bisection method

Question 2

2. implement a genetic algorithm to maximize the following function with n=10:

$$f(\vec{x}) = \begin{cases} \left| \dfrac{\sum_{i=1}^{n} \cos^4(x_i) \; - \; 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, & if \;\; (\forall i,\, 0 \le x_i \le 10) \; and \; (\prod_{i=1}^{n} x_i \ge 0.75) \\ 0, & Otherwise \; (i.e. not \; feasible) \end{cases}$$

Genetic algorithm (Idea and Pseudocode)



Pseudocode for GA

1. Start
2. Generate initial population
3. Compute their fitness
4. Repeat Selection, Crossover and Mutation
5. Stop

Reference: https://arxiv.org/ftp/arxiv/papers/1308/1308.4675.pdf

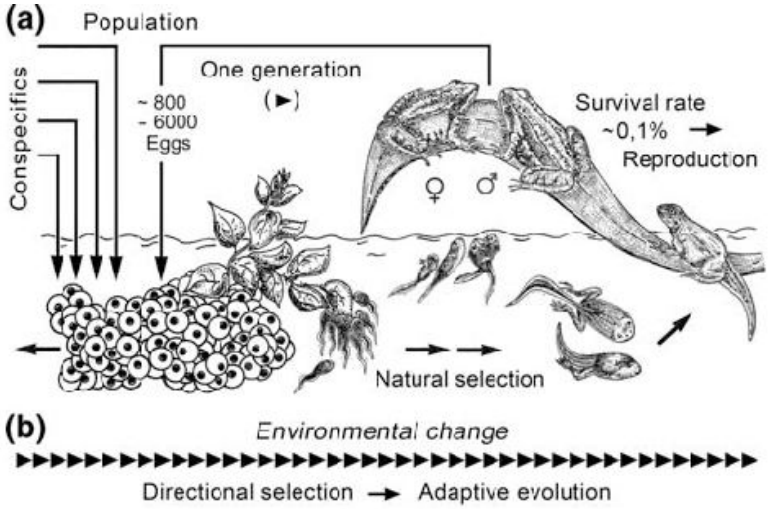
Pre-Process for the Objective Function f(x)

```
################################################################################

# define basic parameters
n <- 10
m <- 100
lb <- 0
ub <- 10
probCrossover <- 0.99
probMutatuin <- 0.2
set.seed(3011)

################################################################################

# objective function
f <- function(x) {
  # throw an exception: product of x >= 0.75
  if(prod(x) < 0.75)
    return(0)
  else
    return(abs((sum(cos(x) ^ 4) - 2 * prod(cos(x) ^ 2)) / sqrt(sum(x ^ 2 * 1:n))))
}

################################################################################
```

$$\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, \quad if \ (\forall i, 0 \le x_i \le 10) \ and \ (\prod_{i=1}^{n} x_i \ge 0.75)$$

Generate Initial Population

```
#####################################################################

# original population
intialPopulation <- matrix(runif(n * m, lb, ub), nr = n, nc = m)
rownames(intialPopulation) <- paste0("Cromosome", 1:n)
colnames(intialPopulation) <- paste0("Solution", 1:m)

#####################################################################

# evaluate the fitness score
fitnessEval <- function(population) {

  fitness <- c()

  for(i in 1:ncol(population)) {
    fitness <- c(fitness, f(population[, i]))
  }

  return(fitness)
}

#############################################
```



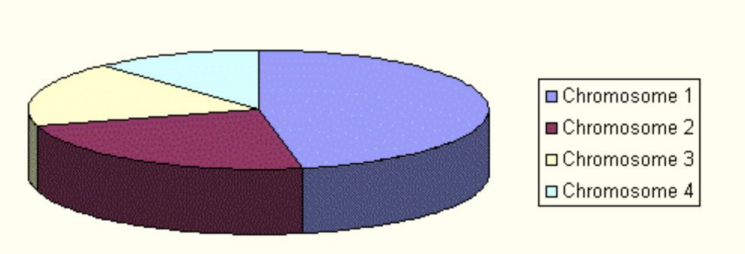

```
> head(initialPopulation)
           Solution1  Solution2 Solution3 Solution4  Solution5 Solution6
Cromosome1 2.5332022 4.18271053  3.744392 3.6043213 5.34458684 0.1825068
Cromosome2 0.8765101 2.12262579  7.452201 1.7445796 4.62805059 3.5074002
Cromosome3 0.1548834 2.77182563  9.744714 4.7971930 4.12785411 4.2825206
Cromosome4 1.0931321 7.81456373  8.398711 8.1533433 0.07370348 9.3171692
Cromosome5 3.9543806 9.01292099  4.638192 9.1407973 0.13496198 9.9777543
Cromosome6 6.5033149 0.04582352  7.098797 0.9079936 7.64217440 0.9032987
```

```
> fitnessEval(initialPopulation)
 [1] 0.11336280 0.08461128 0.05361489 0.05909271 0.08738563 0.10118555
```

Selection and Fitness Score

```
################################################################################

# selection from population
selection <- function(population) {

  fitness <- fitnessEval(population)
  probSelect <- fitness / sum(fitness)
  cumProbSelect <- cumsum(probSelect)

  iter = 0

  repeat{
    ran <- runif(m, 0, 1)
    index <- c()

    for(i in 1:m) {
      for(j in 1:m) {
        if(ran[i] <= cumProbSelect[j]) {
          index <- c(index, j)
          break
        }
      }
    }

    newSelectionPopulation <- c()
    for(i in 1:m) {
      newSelectionPopulation <- cbind(newSelectionPopulation, population[, index[i]])
    }

    temp <- sum(fitnessEval(newSelectionPopulation))
    temp2 <- sum(fitness)

    if(temp > temp2)
      break
    else
      iter = iter + 1

    if(temp2 - temp > 0.3 | iter > 20)
      return(population)
  }

  colnames(newSelectionPopulation) <- paste0("Solution", 1:m)

  return(newSelectionPopulation)
}

x = selection(intialPopulation)
compare()

################################################################################
```

Roulette Wheel Selection



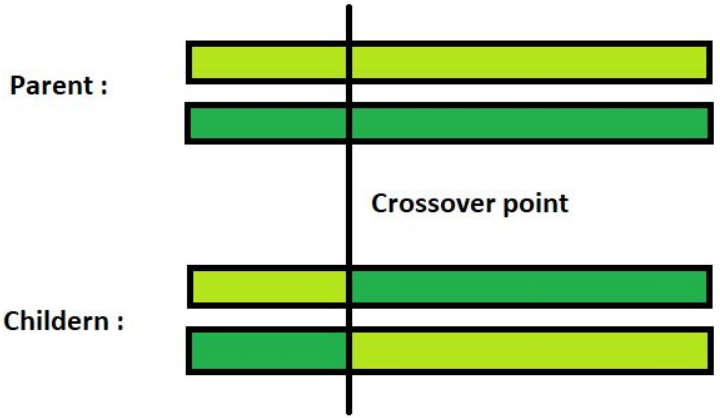
Reference: https://www.obitko.com/tutorials/genetic-algorithms/selection.php

Crossover

```
crossover <- function(population, probCrossover) {

  iter = 0

  repeat{
    sexPartner <- sample(1:m)

    newCrossoverPopulation <- c()

    for(i in seq(1, m, 2)) {
      if(runif(1, 0, 1) < probCrossover)
        newCrossoverPopulation <- cbind(newCrossoverPopulation, crossoverADT(sexPartner[i:(i + 1)], population))
      else
        newCrossoverPopulation <- cbind(newCrossoverPopulation, population[, sexPartner[i]], population[, sexPartner[(i + 1)]])
    }

    temp <- sum(fitnessEval(newCrossoverPopulation))
    temp2 <- sum(fitnessEval(population))

    if(temp > temp2)
      break
    else
      iter = iter + 1

    if(temp2 - temp > 0.3 | iter > 20)
      return(population)
  }

  colnames(newCrossoverPopulation) <- paste0("Solution", 1:m)
  rownames(newCrossoverPopulation) <- paste0("Cromosome", 1:n)

  return(newCrossoverPopulation)
}
```

```
# crossover from population
crossoverADT <- function(set, population) {

  index <- sample(1:(n - 1), 1)

  tempvector <- population[1:index, set[1]]
  tempvector2 <- population[(index + 1):n, set[1]]

  tempvector3 <- population[1:index, set[2]]
  tempvector4 <- population[(index + 1):n, set[2]]

  newSol <- c(tempvector, tempvector4)
  newSol2 <- c(tempvector3, tempvector2)

  return(cbind(newSol, newSol2))
}
```

```
> crossover(x, probCrossover)
           Solution1 Solution2  Solution3 Solution4 Solution5 Solution6 Solution7 Solution8 Solution9 Solution10
Cromosome1  2.793816 9.4378422 4.18271053 3.6043213 2.5332022 9.4378422 9.4378422  3.744392 9.4378422   2.793816
Cromosome2  1.850924 0.1026439 2.12262579 1.7445796 0.8765101 0.1026439 0.1026439 7.452201 0.1026439   1.850924
Cromosome3  9.541761 3.7675276 2.77182563 4.7971930 0.1548834 3.7675276 9.7447139 3.767528 3.7675276   9.541761
Cromosome4  3.869359 6.4624206 7.81456373 8.1533433 6.4624206 1.0931321 8.3987106 6.462421 6.4624206   3.869359
Cromosome5  2.915124 4.1424598 9.01292099 9.1407973 4.1424598 8.9543806 4.6381915 4.142460 4.1424598   2.915124
Cromosome6  9.684185 2.8716899 0.04582352 0.9079936 2.8716899 6.5033149 7.0987968 2.871690 2.8716899   9.684185
Cromosome7  6.085711 2.4774582 7.15283849 7.7543854 2.4774582 9.8368572 9.9495752 2.477458 2.4774582   6.085711
Cromosome8  4.647248 2.4056230 6.57970378 2.7017526 2.4056230 0.2504151 7.2174144 2.405623 2.4056230   4.647248
Cromosome9  7.133505 2.1784833 7.89407557 0.6824263 7.1335054 1.0831396 7.7064728 7.133505 2.1784833   7.133505
Cromosome10 4.112735 5.9615679 8.84394499 8.2571374 4.1127350 7.2606301 9.5519769 4.112735 5.9615679   4.112735
> initialPopulation
           Solution1  Solution2 Solution3 Solution4  Solution5 Solution6 Solution7 Solution8 Solution9 Solution10
Cromosome1 2.5332022 4.18271053  3.744392 3.6043213 5.34458684 0.1825068  8.894970  2.793816  7.217438  9.4378422
Cromosome2 0.8765101 2.12262579  7.452201 1.7445796 4.62805059 3.5074002  7.229408  1.850924  1.672605  0.1026439
Cromosome3 0.1548834 2.77182563  9.744714 4.7971930 4.12785411 4.2825206  9.181520  9.541761  1.525048  3.7675276
Cromosome4 1.0931321 7.81456373  8.398711 8.1533433 0.07370348 9.3171692 3.740069  3.869359  3.333599  6.4624206
Cromosome5 3.9543806 9.01292099  4.638192 9.1407973 0.13496198 9.9777543 9.052893  2.915124  0.700382  4.1424598
Cromosome6 6.5033149 0.04582352  7.098797 0.9079936 7.64217440 0.9032987 5.425081  9.684185  6.824790  2.8716899
Cromosome7 9.8368572 7.75438542  9.949575 7.1528385 9.66870801 0.2399181  7.196991  6.085711  7.627525  2.4774582
Cromosome8 0.2504151 2.70175257  7.217414 6.5797038 8.97387309 1.4544940  9.139718  4.647248  2.921399  2.4056230
Cromosome9 1.0831396 0.68242633  7.706473 7.8940756 1.30074935 7.0827335  5.606471  2.178483  1.967558  7.1335054
Cromosome10 7.2606301 8.25713743  9.551977 8.8439450 1.92124122 7.6296159  3.425094  5.961568  2.516759  4.1127350
>
```



Reference: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

Mutation

```
################################################################################

# mutation of population
mutation <- function(population, probMutatuin) {

  if(runif(1, 0, 1) < probMutatuin) {
    post <- sample(1:n,1)

    for(i in 1:post) {
      index <- sample(1:n, 1)
      population[index] <- runif(1, 0, n)
    }
  }

  return(population)
}

x = mutation(x, probMutatuin)
compare()

################################################################################
```

```
> (x = crossover(x, probCrossover))
            Solution1 Solution2 Solution
Cromosome1   4.182711 3.7443925 2.793816
Cromosome2   2.122626 0.1026439 0.102643
Cromosome3   4.797193 9.5417611 3.767527
Cromosome4   8.153343 3.8693590 6.462420
Cromosome5   9.140797 2.9151239 4.142459
Cromosome6   9.684185 0.9079936 2.871689
Cromosome7   2.477458 7.7543854 2.477458
Cromosome8   2.405623 2.4056230 2.701752
Cromosome9   7.133505 7.1335054 2.178483
Cromosome10  7.260630 5.9615679 4.112735
> mutation(x, probMutatuin)
[1] "Yes"
            Solution1 Solution2 Solution
Cromosome1   2.449859 3.7443925 2.793816
Cromosome2   2.122626 0.1026439 0.102643
Cromosome3   6.589532 9.5417611 3.767527
Cromosome4   6.674933 3.8693590 6.462420
Cromosome5   3.357719 2.9151239 4.142459
Cromosome6   9.684185 0.9079936 2.871689
Cromosome7   2.477458 7.7543854 2.477458
Cromosome8   9.714982 2.4056230 2.701752
Cromosome9   7.133505 7.1335054 2.178483
Cromosome10  7.260630 5.9615679 4.112735
>
```

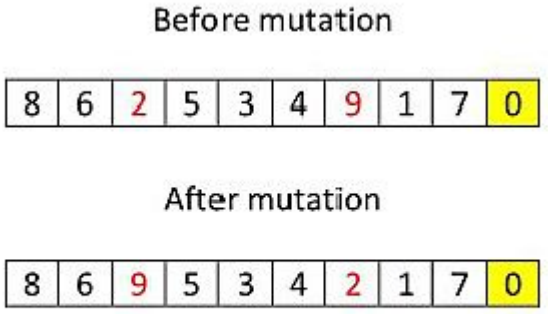
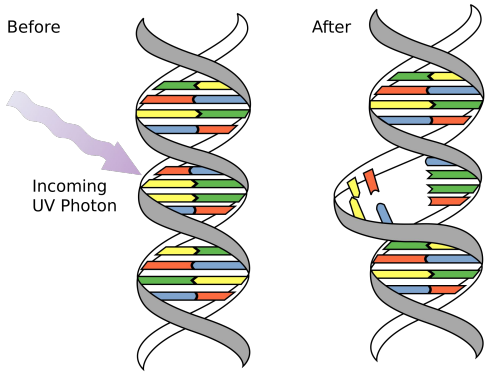


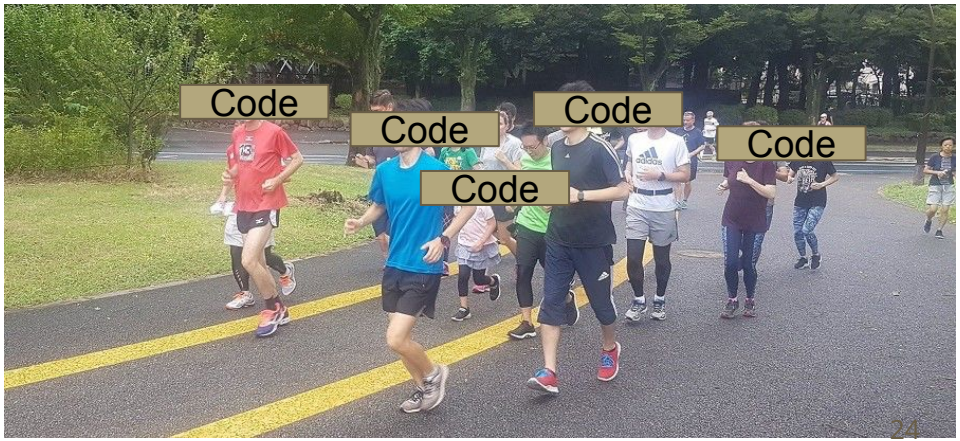
Figure 6. Mutation process

Reference:
https://figshare.com/articles/Using_the_Genetic_Algorithm_for_the_Optimization_of_Dynamic_School_Bus_Routing_Problem-Figure_6_Mutation_process/6274883/1



Run the Programme!

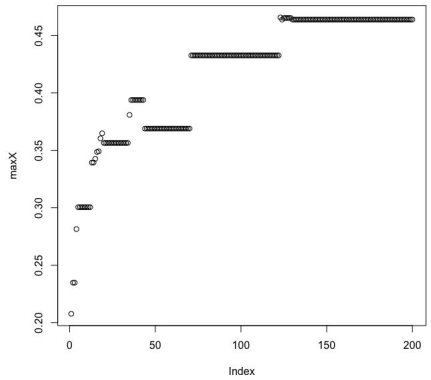
```
x <- mutation(crossover(selection(intialPopulation), probCrossover), probMutatuin)
maxX <- c()
meanX <- c()
for(i in 1:200){
  x <- mutation(crossover(selection(x), probCrossover), probMutatuin)
  maxX <- c(maxX,max(fitnessEval(x)))
  meanX <- c(meanX, mean(fitnessEval(x)))
  print(paste0("Loop", i," Finish"))
}
```

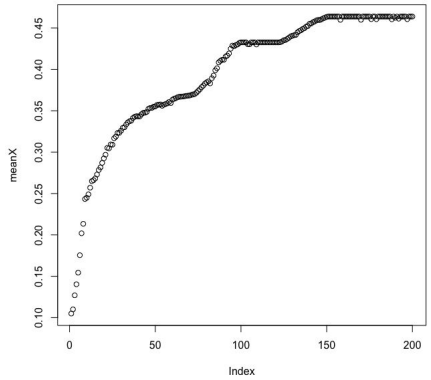


Output of the Programme

Best Solution is 0.4657127 with 200 generations

- Baised due to randomisation
- Increase the population size and the number of generations



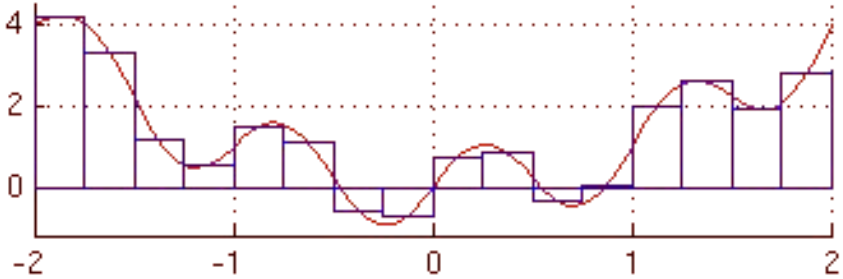


Question 3

3. implement both a quadrature method and a Monte Carlo method to estimate the integral: $\int_0^{+\infty} \frac{|\cos(x)|}{x} e^{-(\log(x)-3)^2}\, dx$

Quadrature method

```
#####################################################################

f <- function(x) {
  # throw an exception: x must be greater than 0 for log(x) function
  if(x <= 0)
    return(0)
  else
    return(abs(cos(x)) / x * exp(-(log(x) - 3) ^ 2))
}

rectangleRule <- function(lb, ub, n) {
  result = 0
  width <- (ub - lb) / n
  for(i in 1:n)
    result <- result + width * f(lb + (i - 1) * width)
  return(result)
}

rectangleRule(0, 1e6, 100000)

#####################################################################
```



Reference: https://en.wikipedia.org/wiki/File:Integration_rectangle.png

Solution is 1.044556

- Approximate only
- Motivate Monte Carlo method

A Monte Carlo method

General Monte Carlo method:

$$\int f(X)P(X)dX = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

We have P(X) = $\int_0^{\infty} \frac{1}{x\sqrt{\pi}} e^{-(\ln(x)-3)^2} dx = 1$

Where P(X) is normalized, $\int P(X)dX = 1$

Where f(X) = $\int |\cos(x)| \sqrt{\pi}$ dx

Obviously, X~log-Normal(3,½^2) with fx(x) = $\frac{1}{x\sqrt{\pi}} e^{-(\ln(x)-3)^2}$,x>0

PDF of log-normal distributiion:

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x-\mu)^2}{\sqrt{2\sigma^2}}}$$

A Monte Carlo method

```
> x<-rlnorm(100000,3,1/2)
> I<-function(x){abs(cos(x))/x*exp(-(log(x)-3)^2)}
> f<-function(x){abs(cos(x))*sqrt(pi)}
> p<-function(x){1/(sqrt(pi)*x)*exp(-(log(x)-3)^2)}
> result<-mean(f(x))
> result
[1] 1.126292
```

Question 4

4. calculate the correlation between x&y of the following distribution:

$$p(x, y) \propto \sin(\pi x)\sin^{20}(\pi x^2) + \sin(\pi y)\sin^{20}(2\pi y^2),\ \ 0 \leq x, y \leq 1$$

Using Monte Carlo method to solve

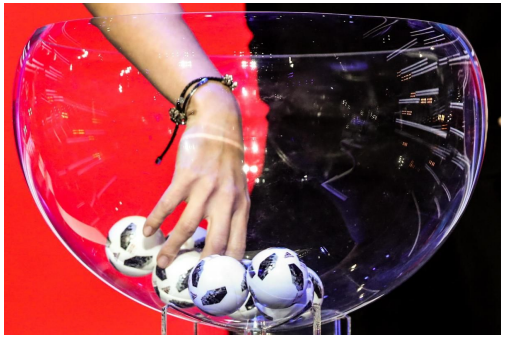
Correlation formula:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

- if we want to find correlation
- we have to find E(X),E(Y) and E(XY) first

Procedure before calculation

- Set n=100000
- When n is large, the result of estimator will be more concentrate
- So the final result obtained will be more accurate



- Which means draw 100000 samples of x and y uniformly

Find E(X) and Var(X)

```
x=runif(n)                                  #calculate for each sample of x

EX=mean(x*f(x,y))/mean(f(x,y))              #Find E(X)

varX=mean(x^2*f(x,y))/mean(f(x,y))-mux^2    #Find Var(X)
```

Find E(X) and Var(X)

- **mean f(x,y)** acts as normalizer for estimation

Find E(X) and Var(X)

E(X)	Var(X)
0.593464	0.057048

Find E(Y) and Var(Y)

```
y=runif(n)                                          #calculate for each sample of y

EY=mean(y*f(x,y))/mean(f(x,y))                      #Find E(Y)

varY=mean(y^2*f(x,y))/mean(f(x,y))-muy^2            #Find Var(Y)
```

Find E(Y) and Var(Y)

E(Y)	Var(Y)
0.535163	0.054156

Find E(XY) and Cov(X,Y)

EXY=mean(x*y*f(x,y))/mean(f(x,y))

#Find E(XY)

covXY=EXY-EX*EY

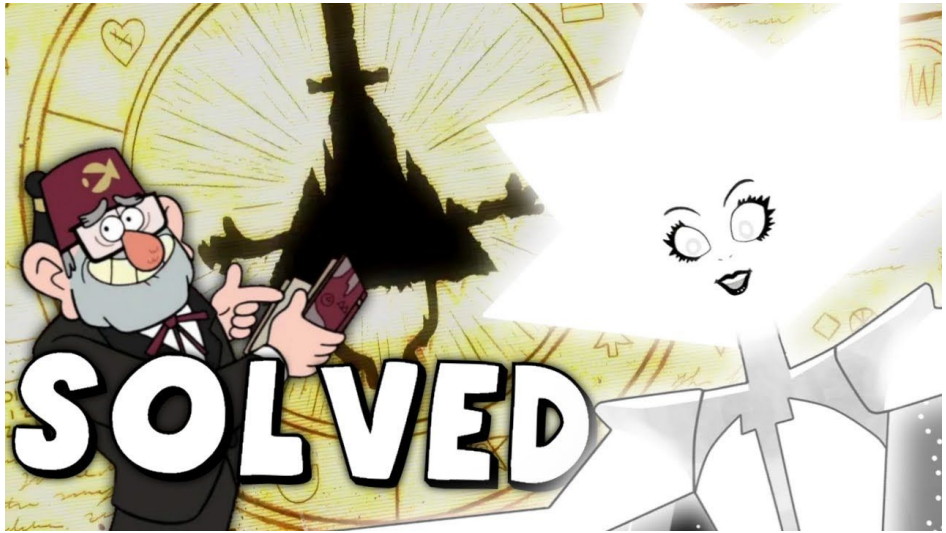
#Find Cov(X,Y)

Find E(XY) and Cov(X,Y)

E(XY)	Cov(X,Y)
0.314158	-0.003442

Find Correlation

corr=covXY/sqrt(varX*varY) #Find Correlation



Find Correlation

Correlation	-0.061936

END