Math475 Homework3

Muzhou liu

October 17, 2018

problem 1

a)

```
set.seed(12345)

num_nodes <- 15

A<- matrix(round(runif(num_nodes * num_nodes)), num_nodes, num_nodes)

diag(A) <- 0

A[lower.tri(A)] <- A[upper.tri(A)]


### c =1, p=0

n <- 15

set.seed(12345)
pi1 <- as.integer(runif(n)>0.5)


rxy <- function(init_par, asso_matrix){
  nc <- sum(init_par)
  np <- length(init_par)-nc
  vc <- sum(1-asso_matrix[which(init_par ==1),which(init_par ==1)]) - nc # subtract over counting diag
  vp <- sum(asso_matrix[which(init_par ==0),which(init_par ==0)]) #diag are 0 so ok
  dc <- nc^2-nc
  dp <- np^2-np
  d <- dc+dp

  sx <- sqrt((dc/d)*(dp/d))
  sy <- sqrt(((dc -vc+vp)/d)*((dp-vp+vc)/d))
  s2xy <- ((dc-vc)/d)-(dc/d)*((dc-vc+vp)/d)

  rxy <- s2xy/(sx*sy)

  if(nc==length(init_par)-1 | np == length(init_par)-1){ rxy <- -99}
  if(nc==length(init_par) | np==length(init_par)){rxy <- -100}

  return(rxy)

}
#asso_matrix <- A
#init_par <- c(1,rep(0,14))
```

```r
#rxy(c(1,1,1,rep(0,12)),A)

local_search_steps <- function(init_par, asso_matrix){
  init_pi <- init_par
  init_score <- rxy(init_pi,asso_matrix)
  score_record <- rep(0,length(init_par))
  for (i in 1:length(init_par)) {
    new_pi <- init_par
    new_pi[i] <- 1- init_par[i]
    #print(new_pi)
    score_record[i] <- rxy(new_pi,asso_matrix)
    #print(score_record)
  }

  #print(score_record)

  if(max(score_record, na.rm = TRUE) >init_score){
    steepest <- which.max(score_record)
    next_itr <- init_pi
    next_itr[steepest] <- 1- init_pi[steepest]
    return(next_itr)}
  else{return(init_pi)}
}


local_search_steps_itr <- function(init_par, asso_matrix, tol, max_step ){
 diff <- 2*tol
  num_itr <- 0
  pi_old <- init_par
  asso_matrix <- A

  while (diff > tol & num_itr < max_step) {
    next_itr <- local_search_steps(pi_old, asso_matrix)
    #print(next_itr)
    pi_new <- next_itr
    if(all(pi_new == pi_old)){
      num_itr <- num_itr+1
      return(list(message = "No neighborhood points increases R_xy ",
                  solution = pi_new, num_itr= num_itr,R_xy = rxy(pi_new,A)))
    }
    else{
      num_itr <- num_itr+1
      diff <- abs(rxy(pi_new,asso_matrix)-rxy(pi_old,asso_matrix)) #????
      pi_old <- pi_new
    }

  }

}

pi1 <- data.frame(seq(1:1000))
pi1 <- apply(pi1, 1, function(x){set.seed(x)
  return(as.integer(runif(15)>0.5))}) %>%
```

```
  t()%>%
  as.data.frame()

result_poola <- apply(pi1,1, function(x){return(local_search_steps_itr(x, A,0.001,1000))})

result_tablea <- rbind(lapply(result_poola,function(x){x$'R_xy'}),
                       lapply(result_poola,function(x){x$solution})) %>%
  t()%>%
  data.frame()


table(as.character(result_tablea$X1))
```

```
##
## 0.170940864689457  0.17884778999622  0.18424729466843 0.185382718087062
##                 8                 9                 8                 4
## 0.236722935456979 0.259311001817378 0.270907780694881 0.277819703334998
##                11                33                48                41
## 0.283730760852763 0.296112106049459 0.301554662061584  0.34219405926104
##                32               169                94               542
##              NULL
##                 1
```

```
# choose the best and the most common resutl
result_tablea[which(result_tablea$X1 == '0.34219405926104'),]$X2 %>%
  unique()
```

```
## [[1]]
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15
##   0   1   0   0   1   1   1   0   1   1   0   0   0   0   0
```

b)

```
annealing_step <- function(init_par, asso_matrix, temp){
  init_pi <- init_par
  init_score <- rxy(init_pi,asso_matrix)
  score_record <- rep(0,length(pi))
  for (i in 1:length(init_par)) {
    new_pi <- init_par
    new_pi[i] <- 1- init_par[i]
    #print(new_pi)
    score_record[i] <- rxy(new_pi,asso_matrix)
    #print(score_record)
  }

  rand_choice <- sample.int(length(init_par),1)
  #print(rand_choice)

  if(score_record[rand_choice]>init_score ){
    new_pi <- init_pi
    new_pi[rand_choice] <-1 - init_pi[rand_choice]
    #print(new_pi)
    return(new_pi)
  }
```

```r
  else {
    p_a <- min(1, exp((-init_score+score_record[rand_choice]/temp)))
    if(rbinom(1,1,p_a)==1){return(new_pi)}
    else{return(init_pi)}
  }

}



annealing_step_itr <- function(init_par,asso_matrix, max_iter, temps){
  num_iter <- 0
  old_pi <- init_par
  while(num_iter < max_iter){
    next_iteration <- annealing_step(old_pi,asso_matrix,temps[num_iter+1])
    new_pi<- next_iteration
    num_iter <- num_iter+1
    #print(num_iter)
    old_pi <- new_pi
    #print(c(new_pi,rxy(new_pi,A)))
  }
  return(list(solution = new_pi, R_xy = rxy(new_pi,asso_matrix )))
}


pi2 <- data.frame(seq(1:1000))
pi2 <- apply(pi2, 1, function(x){set.seed(x)
  return(as.integer(runif(15)>0.5))}) %>%
  t()%>%
  as.data.frame()

result_poolb <- apply(pi2,1, function(x){return(annealing_step_itr(x, A,1000,(1000:1)/1000))})

result_tableb <- rbind(lapply(result_poolb,function(x){x$'R_xy'}),
                       lapply(result_poolb,function(x){x$solution})) %>%
  t()%>%
  data.frame()


table(as.character(result_tableb$X1))
```

```
##
##   0.14213381090374 0.191086497040005 0.195064898086275 0.206474160483505
##                 12                39                 1                77
## 0.206596288177169 0.210467464226522 0.218496722494452 0.223437465906151
##               100                 1                86                 1
## 0.232379000772445 0.244313637439979 0.259311001817378 0.263221442307993
##                78                41                11                 3
## 0.265876574214751 0.270907780694881 0.277819703334998  0.28301965507089
##                22                35                 3                 1
## 0.283730760852763 0.293142326612384 0.296112106049459 0.301554662061584
##                50               100                94                37
## 0.336787657027282  0.34219405926104
##               102               106
```

```r
# choose the best the most common resutl
result_tableb[which(result_tableb$X1 == '0.34219405926104'),]$X2 %>%
  unique()
```

```
## [[1]]
##   V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15
##    0   1   0   0   1   1   1   0   1   1   0   0   0   0   0
```

c)

```r
 # optim(sample(c(0,1),15,replace=T),
 #       function(x){rxy(x,A)},
 #       function(x){annealing_step(x,A,(1000:1)/1000)},
 #       method = "SANN",
 #       control = list(maxit=1000,temp=1000:1,fnscale=-1,tmax=1000))


pi3 <- data.frame(seq(1:1000))
pi3 <- apply(pi3, 1, function(x){set.seed(x)
  return(as.integer(runif(15)>0.5))}) %>%
  t()%>%
  as.data.frame()

result_poolc <- apply(pi3 ,1, function(x){return(optim(x,
      function(x){rxy(x,A)},
      function(x){annealing_step(x,A,(1000:1)/1000)},
      method = c("SANN"),
      control = list(maxit=1000,temp=1000:1,fnscale=-1,tmax=1000))[1:2])})

result_tablec <- rbind(lapply(result_poolc,function(x){x$'value'}),
                       lapply(result_poolc,function(x){x$par})) %>%
  t()%>%
  data.frame()


table(as.character(result_tablec$X1))
```

```
##
## 0.259311001817378 0.270907780694881 0.277819703334998  0.27979769384754
##                15                59                10                 3
##  0.28301965507089 0.283730760852763 0.284144864626421 0.287677980891231
##                 1                84                 1                 1
## 0.289629360056618 0.293142326612384 0.296112106049459 0.299664563428365
##                 2                 1               314                 5
## 0.301554662061584 0.308327406296755  0.34219405926104
##                93                18               393
```

```r
# choose the best the most common resutl
result_tableb[which(result_tableb$X1 == '0.34219405926104'),]$X2 %>%
  unique()
```

```
## [[1]]
##   V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15
##    0   1   0   0   1   1   1   0   1   1   0   0   0   0   0
```

Problem 2

a)

```r
Simpson_rule <- function(fun, up, lo, n ){
   if(lo>up){
     c <- up
     up <- lo
     lo <- c
   }
   f <- function(x){eval(parse(text=paste(fun)))}
   xi_s <- lo+ c(1:n)/n *(up-lo)
   h <- (up-lo)/n
   sum <-0
   for (i in 1:(n-1)) {
     sum <- sum+ h/6 * (f(xi_s[i])+4* f((xi_s[i]+xi_s[i+1])/2) + f(xi_s[i+1]))
   }

   return(sum)
}

# mu=0, sigma=1 n = 1000
  Simpson_rule('1/(sqrt(1/1000)*sqrt(2*pi))* exp(-((log(x)-0)^2)/(2/1000))',10000,0,100000)
```

```
## [1] 0.9082
```

```r
# mu=1, sigma=1 n = 1000
  Simpson_rule('1/(sqrt(1/1000)*sqrt(2*pi))* exp(-((log(x)-1)^2)/(2/1000))',10000,0,100000)
```

```
## [1] 2.72
```

```r
# mu=1, sigma=2 n = 1000
  Simpson_rule('1/(sqrt(2/1000)*sqrt(2*pi))* exp(-((log(x)-1)^2)/(4/1000))',20000,0,100000)
```

```
## [1] 2.722
```

```r
# mu=2, sigma=2 n = 1000
  Simpson_rule('1/(sqrt(2/1000)*sqrt(2*pi))* exp(-((log(x)-2)^2)/(4/1000))',20000,0,100000)
```

```
## [1] 7.396
```

b)

```r
laplace_appx <- function(mu, sigma,n) {
  return(1/(sigma*sqrt(2*pi/n)) * sqrt(2*pi/n) *sqrt(1/(abs(-1/exp(2*mu)*sigma^2))) )
}


laplace_appx(0,1,1000)
```

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

```r
laplace_appx(1,1,1000)
```

```
## [1] 2.718
```

```r
laplace_appx(2,1,1000)
```

```
## [1] 7.389
```

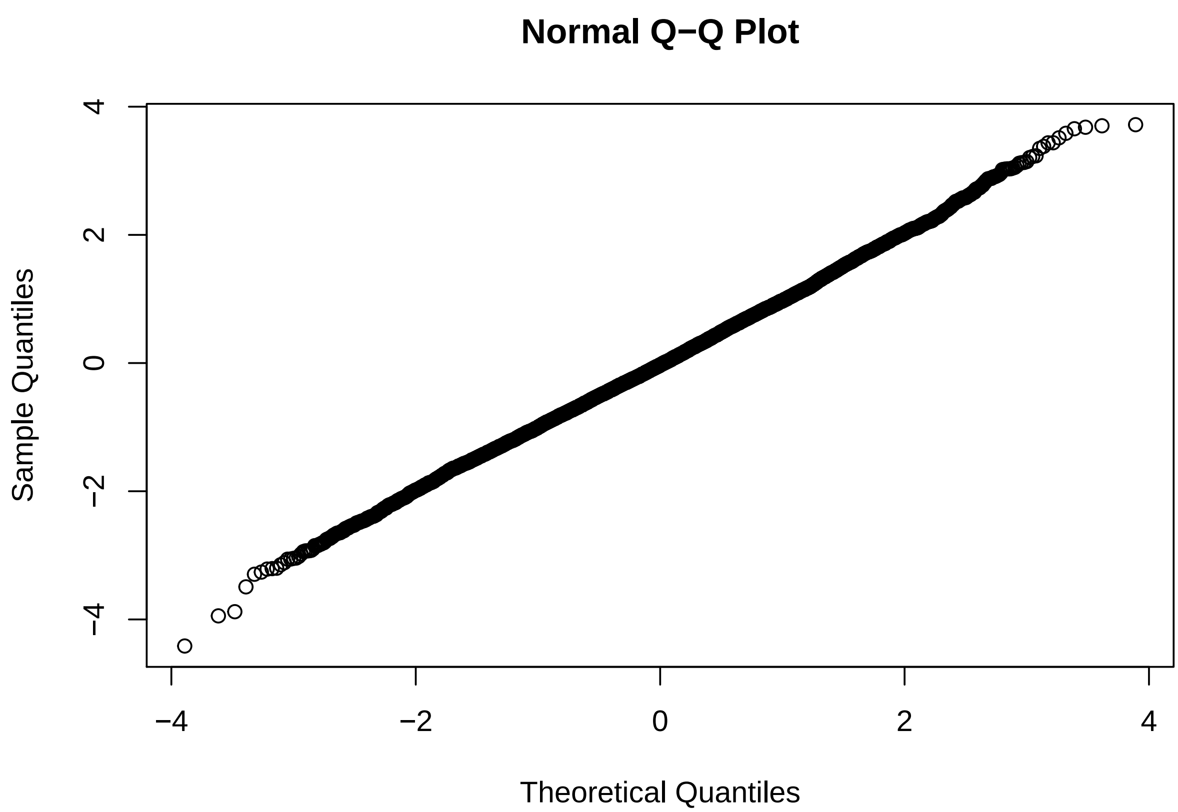
c)

```r
sample_std_normal <- function(n){
  x <- runif(n)
  y <- runif(n)
  df <- data.frame(x,y)

  return(apply(df,1, function(x){sqrt((-2)*log(x[1]))*cos(2*pi*x[2])}))

}

qqnorm(sample_std_normal(10000))
```




```r
#ks.test(rnorm(100000),sample_std_normal(100000))
# the generator is okay

# mu =0 , sigma =1 , n=1000
pool01 <- data.frame(rep(1,1000))
apply(pool01,1 , function(x){W_01 <- exp(0+1*(sample_std_normal(1000)))
                              return(prod(W_01)^(1/1000))}) %>%mean()
```

```
## [1] 1.001
```

```r
# mu =1 , sigma =1 , n=1000
pool11 <- data.frame(rep(1,500))
apply(pool11,1 , function(x){W_11 <- exp(1+1*(sample_std_normal(500)))
                            return(prod(W_11)^(1/500))}) %>%mean()
```

```
## [1] 2.716
```

```r
# mu =2 , sigma =2 , n=1000
pool22 <- data.frame(rep(1,300))
apply(pool22,1 , function(x){W_21 <- exp(2+sample_std_normal(300)+sample_std_normal(300))
                            return(prod(W_21)^(1/300))}) %>%mean()
```

```
## [1] 7.381
```

d)

all three methods have relatively close results to the theortical value that we are expecting. However, the laplace appromixation has the shortest run time, while the other two consumed a long time to run a result. Also, for the Monte Carlo method, we cannot set n larget enought when simulating expacted value of the geometric mean, as the product of x_is would exceed the maxim tolarance of the program.