midterm_ref

Youn Kyeong Chang

March 12, 2019

Simulation

Acceptance/rejection method

proof.

$$P(X \leq x) = P(Y \leq x | U \leq \frac{f(Y)}{Mg(Y)} \tag{1}$$

$$= \frac{P(Y \leq x, U \leq \frac{f(Y)}{Mg(Y)})}{P(U \leq \frac{f(Y)}{Mg(Y)})} \tag{2}$$

$$= \frac{\int_{-\infty}^{x} \int_{0}^{\frac{f(y)}{Mg(y)}} 1 \cdot g(y) du dy}{\int_{-\infty}^{\infty} \int_{0}^{\frac{f(y)}{Mg(y)}} 1 \cdot g(y) du dy} \tag{3}$$

$$= \frac{\int_{-\infty}^{x} \frac{f(y)}{Mg(y)} g(y) dy}{\int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)} g(y) dy} \tag{4}$$

$$= \int_{-\infty}^{x} f(y) dy \tag{5}$$

$$= F(x) \tag{6}$$

Multiple linear regression simulation

```
sim.regression <- function(n.obs=10, coefficients=runif(10,-5,5), s.deviation=.1){

  n.var=length(coefficients)
  M=matrix(0,ncol=n.var,nrow=n.obs)

  beta=as.matrix(coefficients)

  for (i in 1:n.var){
    M[,i]=rnorm(n.obs,0,1)
  }

  y=M %*% beta + rnorm(n.obs,0,s.deviation)

  return (list(x=M,y=y,coeff=coefficients))

}

y<-c(18.73,14.52,17.43,14.54,13.44,24.39,13.34,22.71,12.68,19.32,30.16,
     27.09,25.40,26.05,33.49,35.62,26.07,36.78,34.95,43.67)
x1<-c(610,950,720,840,980,530,680,540,890,730,670,770,880,1000,760,590,
      910,650,810,500)
```

```r
x2<-c(1,1,3,2,1,1,3,3,2,2,1,3,3,2,2,2,3,3,1,2)
lm <- lm(y~ x1+x2)
summary(lm)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2805  -7.5169  -0.9231   7.2556  12.8209
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.85352   11.33229   3.782  0.00149 **
## x1          -0.02534    0.01293  -1.960  0.06662 .
## x2           0.33188    2.41657   0.137  0.89238
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.679 on 17 degrees of freedom
## Multiple R-squared:  0.1869, Adjusted R-squared:  0.09127
## F-statistic: 1.954 on 2 and 17 DF,  p-value: 0.1722
```

Problem 1

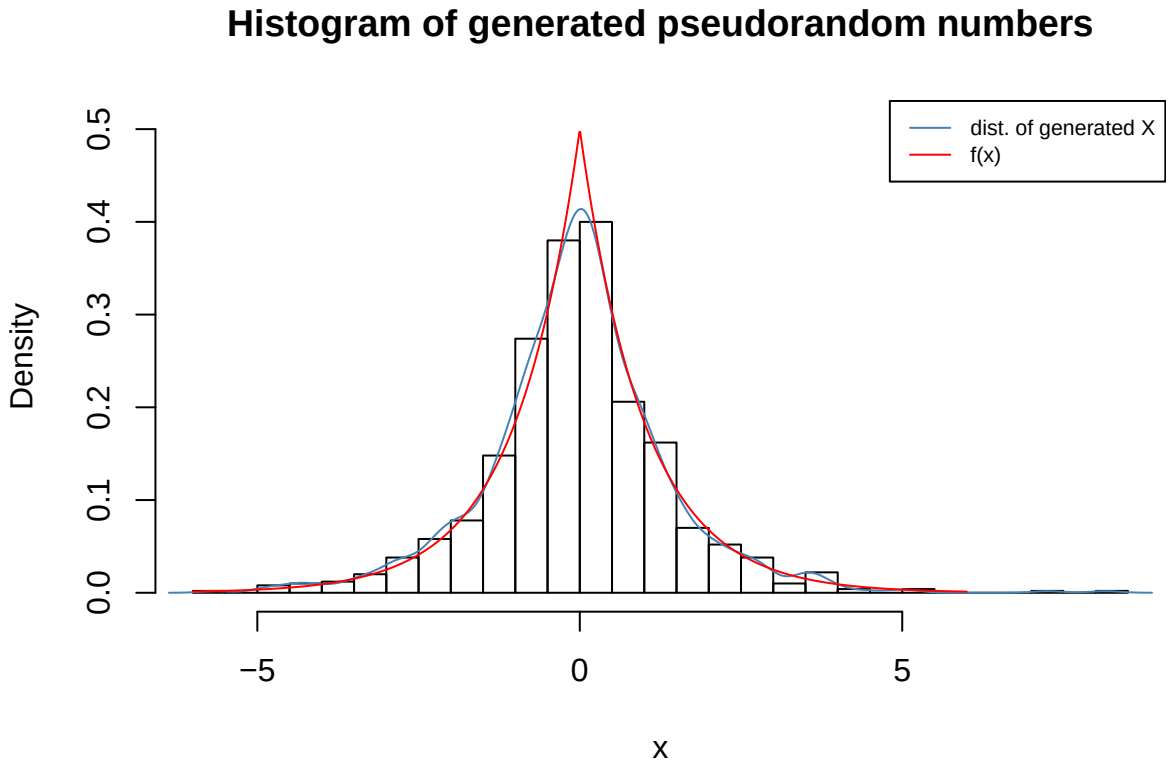
The standard Laplace distribution has density $f(x) = 0.5e^{-|x|}, x \in (-\infty, \infty)$. Please provide an algorithm that uses the inverse transformation method to generate a random sample from this distribution. Use the $U(0, 1)$ random number generator in **R**, *write a* **R**-function to implement the algorithm. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the standard Laplace distribution.)

Answer:

Since $F(x) = 0.5e^{x}, x \in (-\infty, 0), F(x) = 1 - 0.5e^{-x}, x \in [0, \infty)$, an algorithm using the inverse transformation method to generate a random sample from this distribution is as follows:

```r
set.seed(1001)
U <- runif(1000)
X <- (U < 0.5)*log(2*U) + (U >= 0.5)*(-log(2 - 2*U))

# Validation of algorithm
ffun <- function(x){
  0.5*exp(-abs(x))
}

xgrid <- seq(-6, 6, length.out = 1000)

hist(X, freq = F, breaks = 30, xlab = "x", ylab = "Density",
     ylim = c(0, 0.51), main = "Histogram of generated pseudorandom numbers")
lines(density(X), col = "steelblue", type = "l")
lines(xgrid, ffun(xgrid), col = "red", type = "l")
legend("topright", c("dist. of generated X","f(x)"),
```

```
        col = c("steelblue", "red"), lty = 1, cex = .7)
```



The histogram of generated pseudorandom number X with density functions show that generated pseudorandom number X seemingly follow the original function f(x).

Problem 2: inverse transformation method

Use the inverse transformation method to derive an algorithm for generating a Pareto random number with $U \sim U(0,1)$, where the Pareto random number has a probability density function

$$f(x;\alpha,\gamma) = \frac{\gamma\alpha^{\gamma}}{x^{\gamma+1}} I\{x \geq \alpha\}$$

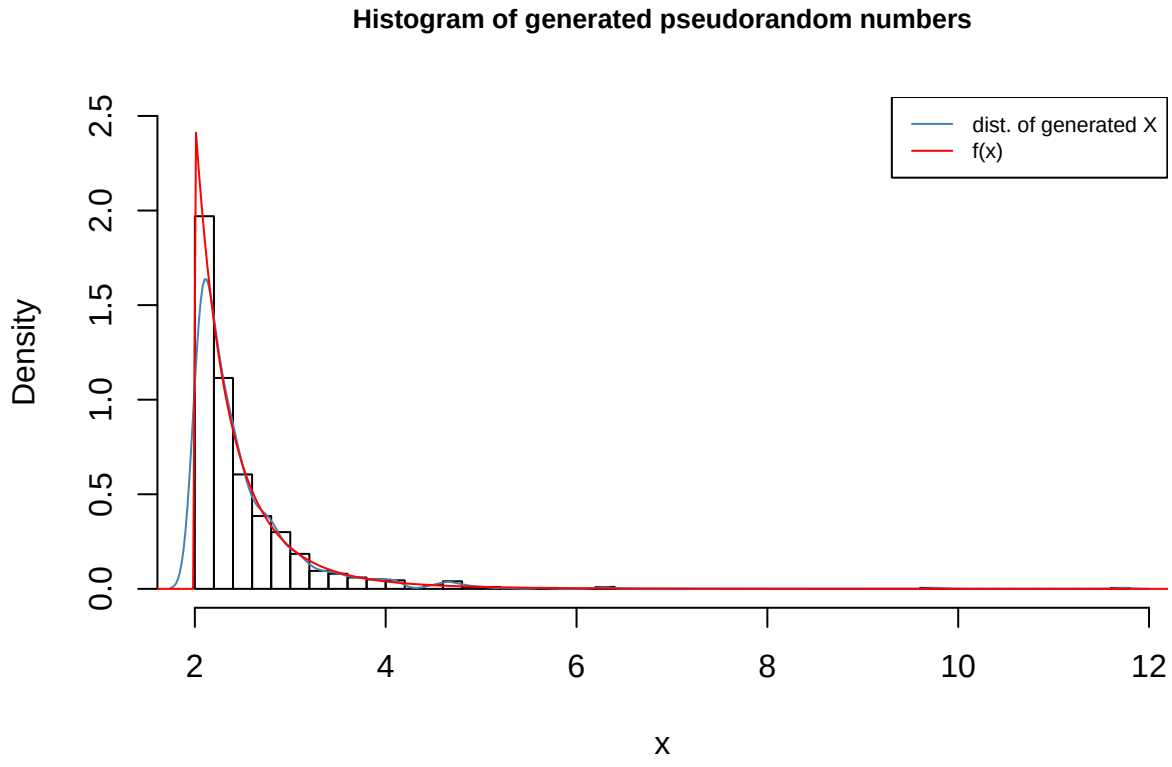
with two parameters $\alpha > 0$ and $\gamma > 0$. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

Answer

Since $F(x) = (1-(\alpha/x)^{\gamma}) * I\{x \geq \alpha\} * I\{\alpha > 0\} * I\{\gamma > 0\}$, an algorithm using the inverse transformation method to generate a random sample from this distribution is as follows:

```
set.seed(1001)

ranfun <- function(a, r){
  U <- runif(1000)
  return(a/((1 - U)^(1/r)))
```

```r
}

X <- ranfun(2, 5)

# Validation of algorithm
ffun <- function(x, a, r){
  (r*a^r)/(x^(r + 1))*(x >= a)
}

xgrid <- seq(0, 30, length = 1000)

hist(X, freq = F, breaks = 50, xlab = "x", ylab = "Density", ylim = c(0, 2.5),
     cex.main = 0.8, main = "Histogram of generated pseudorandom numbers")
lines(density(X), col = "steelblue", type = "l")
lines(xgrid, ffun(xgrid, 2, 5), col = "red", type = "l")
legend("topright", c("dist. of generated X","f(x)"),
        col = c("steelblue", "red"), lty = 1, cex = .7)
```



The histogram of generated pseudorandom number X with density functions show that generated pseudorandom number X seemingly follow the original function f(x).

Problem 3 : acceptance/rejection method

Support of g should cover the support of f

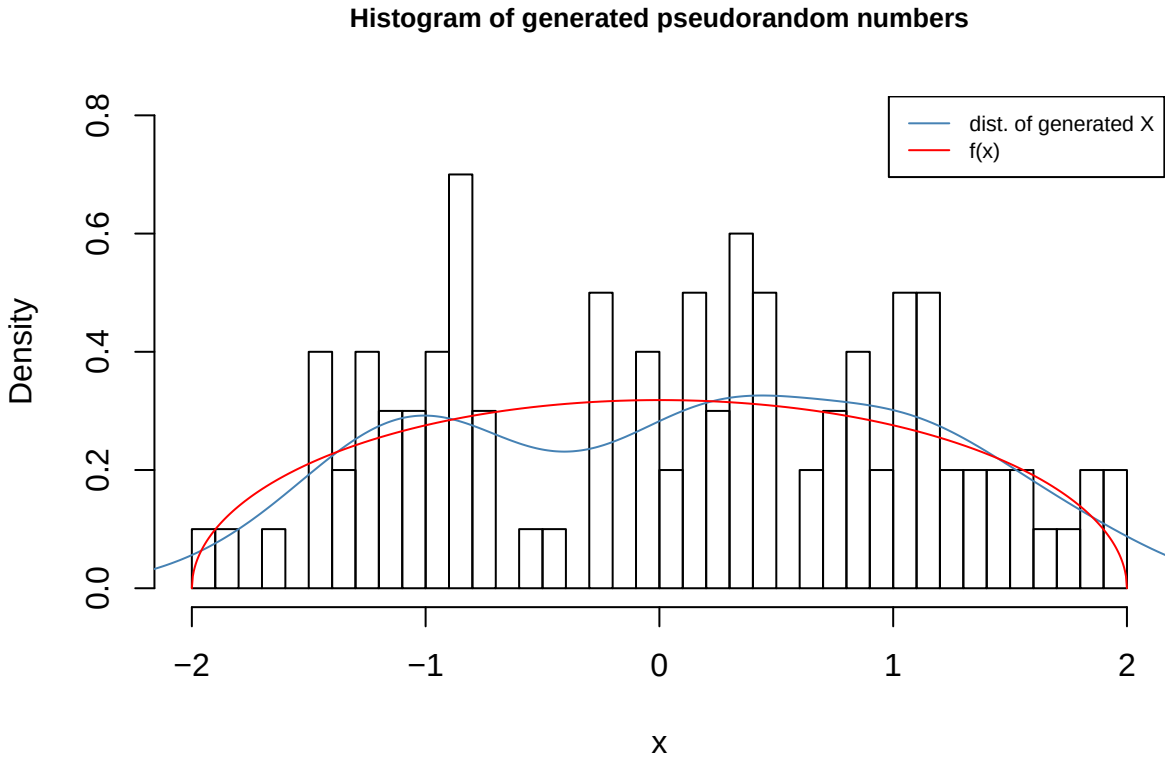
Construct an algorithm for using the acceptance/rejection method to generate 100 pseudorandom variable from the pdf

$$f(x) = \frac{2}{\pi\beta^2}\sqrt{\beta^2 - x^2},\ -\beta \le x \le \beta.$$

The simplest choice for $g(x)$ is the $U(-\beta, \beta)$ distribution but other choices are possible as well. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

Answer:

```
set.seed(1001)

gfun <- function(x, beta){
  1/(2*beta)*(x <= beta & x >= -beta)
}

ffun <- function(x, beta){
  (2/(pi*(beta)^2))*sqrt((beta)^2 - x^2)*(x <= beta & x >= -beta)
}

beta <- 2
M <- 4/pi + 0.01 # M = sup f(x)/g(x) > {2/(pi*beta)} /{1/(2*pi)} = 4/pi
x <- NULL
outer_count <- 0
count <- 0
while (count < 100) {
  outer_count <- outer_count + 1
  y <- runif(1, -beta, beta)
  u <- runif(1)
  if (u <= ffun(y, beta)/(M * gfun(y, beta))) {
    count <- count + 1
    x <- c(x, y)
  }
  else {
    count <- count
  }
}


# Validation of algorithm
xgrid <- seq(-2, 2, length = 1000)
hist(x, freq = F, breaks = 30, ylim = c(0, 0.8),
     main = "Histogram of generated pseudorandom numbers", cex.main = 0.8)
lines(density(x), col = "steelblue", type = "l")
lines(xgrid, ffun(x = xgrid, 2), col = "red", type = "l")
legend("topright", c("dist. of generated X","f(x)"),
        col = c("steelblue", "red"), lty = 1, cex = .7)
```



100 generated random numbers were accepted out of 139. Also, the histogram of generated pseudorandom number X with density functions show that generated pseudorandom number X seemingly follow the original function f(x).

Problem 4: Monte Carlo methods

Importance sampling, Control variate

support of p(x) should cover [a,b]

Develop two Monte Carlo methods for the estimation of $\theta = \int_0^1 e^{x^2} dx$ and implement in **R** .

Answer (1) Importance sampling:

$$g(x) = e^{x^2} \tag{7}$$

$$p(x) = 1, x \in (0,1) \tag{8}$$

$$\int_a^b g(x)dx = \int_a^b \frac{g(x)}{p(x)} p(x)dx \tag{9}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \frac{g(X_i)}{p(X_i)} \tag{10}$$

```
g <- function(x){
  exp(x^2)
```

```
}
p <- function(x){
  1
}

set.seed(1001)
U <- runif(1000)
X <- U
gp <- g(X)/p(X)
mean(gp)
```

```
## [1] 1.449202
```

```
var(gp)
```

```
## [1] 0.2220512
```

Answer (2) Control variate:

$$g(U) = e^{U^2} \tag{11}$$
$$f(U) = U \tag{12}$$
$$\theta = g(U) + c(f(U) - 1/2) \tag{13}$$
$$c = -\frac{Cov(g(U), f(U))}{var(f(U))} \tag{14}$$
$$Var(\theta) = Var(g(U)) - \frac{[(Cov(g(U), f(U))]^2}{var(f(U))} \tag{15}$$

```
g <- function(u)
  exp(u^2)

f <- function(u)
  u

set.seed(1001)
u <- runif(1000)
A <- g(u)
B <- f(u)

c <- -cov(A,B) / var(B)

theta <- g(u) + c(f(u) - 1/2)
mean(theta)
```

```
## [1] 1.439533
```

```
var(theta)
```

```
## [1] 0.5607103
```

Estimation of theta is 1.439533 with variance 0.5607103.

Problem 5: Control variate

Show that in estimating $\theta = E\sqrt{1-U^2}$ it is better to use U^2 rather than U as the control variate, where $U \sim U(0,1)$. To do this, use simulation to approximate the necessary covariances. In addition, implement your algorithms in **R**.

Answer:

$$g(U) = \sqrt{1-U^2}, f_1(U) = U, f_2(U) = U^2 \tag{16}$$

$$\theta_1 = g(U) + c_1(f_1(U) - E(f_1(U))) \tag{17}$$

$$\theta_2 = g(U) + c_2(f_2(U) - E(f_2(U))) \tag{18}$$

$$c = -\frac{Cov(g(U), f(U))}{var(f(U))} \tag{19}$$

$$Var(\theta) = Var(g(U)) - \frac{[(Cov(g(U), f(U))]^2}{var(f(U))} \tag{20}$$

```
set.seed(1001)
g <- function(u)
  sqrt(1 - u^2)

f1 <- function(u)
  u
f2 <- function(u)
  u^2

u <- runif(10000)
A <- g(u)
B1 <- f1(u)
B2 <- f2(u)

cor(A, B1)
```

```
## [1] -0.921181
```

```
c1 <- -cov(A,B1) / var(B1)

cor(A, B2)
```

```
## [1] -0.9837891
```

```
c2 <- -cov(A,B2) / var(B2)

theta1 <- g(u) + c1*(f1(u) - 1/2)
theta2 <- g(u) + c2*(f2(u) - 1/3)

c(var(theta1), var(theta2))
```

```
## [1] 0.007483606 0.001589329
```

```
(var(theta1) - var(theta2))/var(theta1)
```

```
## [1] 0.7876252
```

Therefore, the approximate reduction in variance of estimating theta using U^2 rather than U as the control variate is 79%.

Problem 6

Obtain a Monte Carlo estimate of $\int_1^\infty \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ by importance sampling and evaluate its variance. Write a **R** *function to implement your procedure.*

Answer:

$$g(x) = \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} I(x > 1) \tag{21}$$

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{22}$$

```
g <- function(x){
  (x^2/sqrt(2*pi))*exp(-x^2/2)*(x > 1)
}
p <- function(x){
  (1/sqrt(2*pi))*exp(-x^2/2)
}

set.seed(1001)
X <- rnorm(1000)
gp <- g(X)/p(X)
mean(gp)
```

```
## [1] 0.3936859
```

```
var(gp)/1000
```

```
## [1] 0.001427865
```

Estimation of theta is 0.3936859 with variance 0.001427865.

EM examples

Example 1, Guassian mixture

$$X_i \sim \begin{cases} N(\mu_1, \sigma_1^2) & \text{with probability } 1-p, \\ N(\mu_2, \sigma_2^2) & \text{with probability } p. \end{cases}$$

The density of X_i *is thus*

$$f(x) = (1-p)f(x, \mu_1, \sigma_1) + pf(x, \mu_2, \sigma_2)$$

where $f(x, \mu_1, \sigma_1) = \frac{1}{\sigma}\phi((x-\mu_1)/\sigma_1)$ *and* $\phi(x)$ *is the standard normal density:*

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

Observed likelihood of $(x_1, ..., x_n)$

$$L_{obs} f(x) = \prod_{i=1}^{n} \{(1-p)f(x_i, \mu_1, \sigma_1) + pf(x_i, \mu_2, \sigma_2)\}$$

Suppose there exist another sequence of iid Bernoullis: $Z_i \sim Bin(1, p)$. *For each* i, *if* $Z_i = 0$, *then* X_i *is from the* $N(\mu_1, \sigma_1^2)$ *distribution; if* $Z_i = 1$, *then* X_i *is from* $N(\mu_2, \sigma_2^2)$. *The joint likelihood of* (x_i, z_i) *is* $\{(1-p)f(x_i, \mu_1, \sigma_2)\}^{1-z_i}\{pf(x_i, \mu_2, \sigma_2)\}^{z_i}$

The complete log-likelihood of $(X_i, Z_i)'s$ is a linear function of Z_i's

$$\ell(\mathbf{X}, \mathbf{Z}, \theta) = \sum_{i=1}^{n} \{Z_i \log p + (1 - Z_i) \log(1 - p) + Z_i \log f(x_i, \mu_2, \sigma_2) + (1 - Z_i) \log f(x_i, \mu_1, \sigma_1)\}$$

where $\theta = (p, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2)$.

E-step $E_Z(\ell(\mathbf{X}, \mathbf{Z}, \theta) \mid \mathbf{X}, \theta^{(t)})$. *Replacing* Z_i *by* $\delta_i^{(t)}$

$$\delta_i^{(t)} \widehat{=} E[Z_i | x_i, \theta^{(t)}] = P(Z_i = 1 \mid x_i, \theta^{(t)}) = \frac{p^{(t)} f(x_i, \mu_2^{(t)}, \sigma_2^{(t)})}{(1 - p^{(t)}) f(x_i, \mu_1^{(t)}, \sigma_1^{(t)}) + p^{(t)} f(x_i, \mu_2^{(t)}, \sigma_2^{(t)})}.$$

M-step $\theta^{(t+1)} = \arg\max \ell(\mathbf{X}, \delta^{(t)}, \theta)$.

$$\begin{aligned}
p^{(t+1)} &= \sum \delta_i^{(t)} / n \\
\mu_1^{(t+1)} &= \frac{1}{\sum_{i=1}^n (1 - \delta_i^{(t)})} \sum_{i=1}^{n} (1 - \delta_i^{(t)}) x_i; \\
\mu_2^{(t+1)} &= \frac{1}{\sum_{i=1}^n \delta_i^{(t)}} \sum_{i=1}^{n} \delta_i^{(t)} x_i; \\
\sigma_1^{2(t+1)} &= \frac{1}{\sum_{i=1}^n (1 - \delta_i^{(t)})} \sum_{i=1}^{n} \left[(1 - \delta_i^{(t)}) (x_i - \mu_1^{(t+1)})^2 \right]. \\
\sigma_2^{2(t+1)} &= \frac{1}{\sum_{i=1}^n \delta_i^{(t)}} \sum_{i=1}^{n} \left[\delta_i^{(t)} (x_i - \mu_2^{(t+1)})^2 \right].
\end{aligned}$$

```
# E-step evaluating conditional means E(Z_i | X_i , pars)
delta <- function(X, pars){
  phi1 <- dnorm(X, mean=pars$mu1, sd=pars$sigma1)
  phi2 <- dnorm(X, mean=pars$mu2, sd=pars$sigma2)
  return(pars$p * phi2 / ((1 - pars$p) * phi1 + pars$p * phi2))
}

# M-step - updating the parameters
mles <- function(Z, X) {
  n <- length(X)
  phat <- sum(Z) / n
  mu1hat <- sum((1 - Z) * X) / (n - sum(Z))
  mu2hat <- sum(Z * X) / sum(Z)
  sigmahat1 <- sqrt(sum((1 - Z) * (X - mu1hat)^2 ) / (n - sum(Z)))
  sigmahat2 <- sqrt(sum(Z * (X - mu2hat)^2) / sum(Z))
  return(list(p=phat, mu1=mu1hat, mu2=mu2hat, sigma1=sigmahat1,sigma2=sigmahat2))
}
EMmix <- function(X, start, nreps=10) {
  i <- 0
  Z <- delta(X, start)
  newpars <- start
  res <- c(0, t(as.matrix(newpars)))
```

```
  while(i < nreps) {
  # This should actually check for convergence
    i <- i + 1
    newpars <- mles(Z, X)
    Z <- delta(X, newpars)
    res <- rbind(res, c(i, t(as.matrix(newpars))))
  }
  return(res)
}
```

Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming.(272 observations on 2 variables)

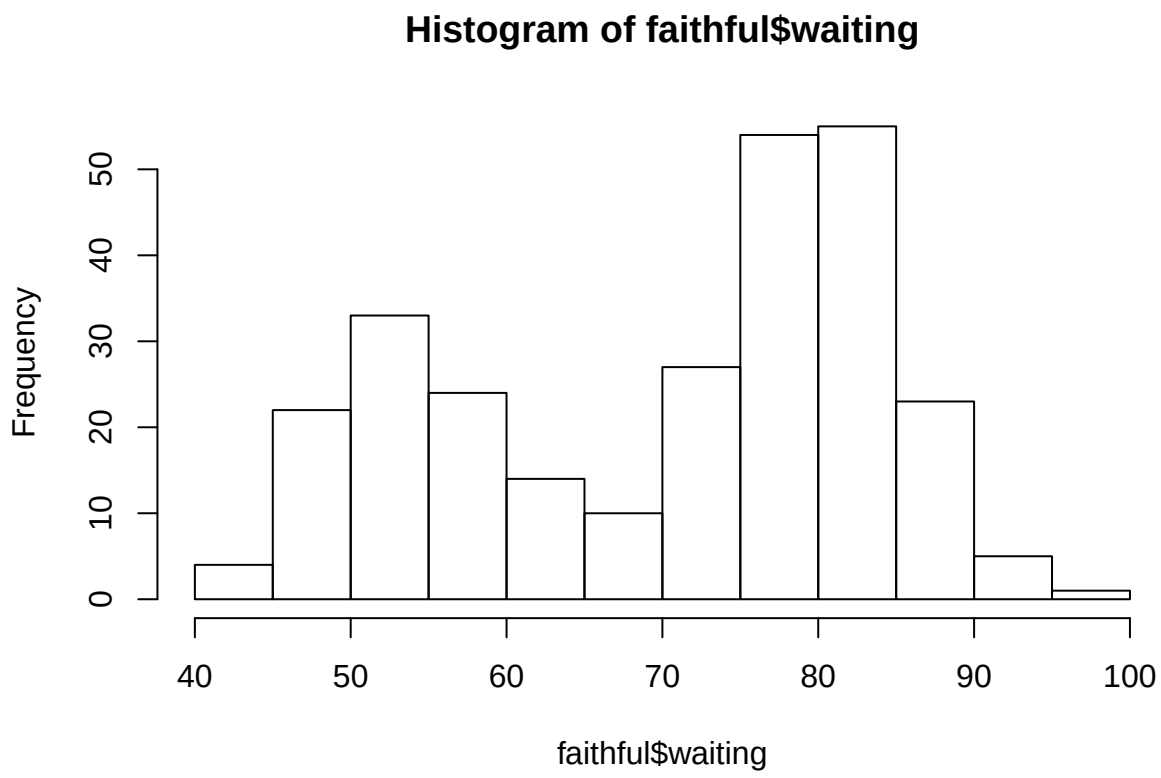
eruptions *numeric Eruption time in mins*

waiting *numeric Waiting time to next eruption (in mins)*

```
library(datasets)
data(faithful)
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```
hist(faithful$waiting)
```




```
res=EMmix(faithful$waiting, start=list(p=0.5, mu1=50, mu2=80, sigma1=15, sigma2=15), nreps=20)
res
```

```
##     [,1] [,2]      [,3]     [,4]     [,5]     [,6]
## res 0    0.5       50       80       15       15
##     1    0.6307318 59.18832 77.75205 11.25962 9.511798
##     2    0.6233299 57.79941 78.8118  10.26942 8.085988
##     3    0.6170174 56.58917 79.77796 8.746949 6.709331
##     4    0.6173725 55.75875 80.27929 7.354872 5.889559
##     5    0.622173  55.30386 80.36634 6.611671 5.665596
##     6    0.627412  55.04593 80.31024 6.287799 5.674619
##     7    0.6313336 54.88948 80.24466 6.123764 5.724151
##     8    0.6339918 54.79163 80.19484 6.029467 5.768029
##     9    0.6357493 54.72944 80.16025 5.972129 5.800265
##     10   0.6369042 54.68949 80.13691 5.936268 5.822627
##     11   0.6376621 54.66363 80.12136 5.913448 5.83777
##     12   0.6381597 54.6468  80.11105 5.898762 5.847902
##     13   0.6384865 54.63582 80.10424 5.889238 5.854636
##     14   0.6387013 54.62862 80.09974 5.883031 5.859095
##     15   0.6388425 54.6239  80.09678 5.878973 5.862041
##     16   0.6389354 54.6208  80.09483 5.876313 5.863984
##     17   0.6389964 54.61877 80.09354 5.874568 5.865265
##     18   0.6390366 54.61743 80.0927  5.873421 5.866109
##     19   0.6390631 54.61655 80.09214 5.872668 5.866664
##     20   0.6390805 54.61597 80.09177 5.872172 5.86703
```

Example 2 Zero-inflated Poisson

The following table shows the number of children of N widows entitled to support from certain pension fund.

```
library(knitr)
n.child = c(0:6)
n.widows = c(3062, 587, 284 ,103,33, 4 ,2 )
xx= as.data.frame(rbind(n.child,n.widows))
rownames(xx)=c("Number of Child", "Number of widows")
kable (xx, caption="")
```

	V1	V2	V3	V4	V5	V6	V7
Number of Child	0	1	2	3	4	5	6
Number of widows	3062	587	284	103	33	4	2

Poisson distribution is often used to model count data. But the observed data above is not consistent with poison distribution due to the large number of windows without kids. One way is to model the data as a mixture of two populations. With probability δ, $Y = 0$, and with probability $1 - \delta$, $Y \sim Poisson(\lambda)$. Construct an EM algorithm to estimate the (δ, λ), and implement into R.

The observed likelihood of Y_i is

$$pI\{Y_i = 0\} + (1-p)e^{-\lambda}\frac{\lambda^{Y_i}}{Y_i!}$$

Let z_i be the indicator whether Y_i is from 0 state or a Poission distribution

$$\mathbf{z}_i \sim \begin{cases} 1, with\ probabilityp \\ 0, with\ probability\ 1-p \end{cases}$$

The joint likelihood function is

$$\prod_{i=1}^{n} p^{z_i}\left[(1-p)e^{-\lambda}\frac{\lambda^{y_i}}{y_i!}\right]^{1-z_i}$$

The joint log-likehood is

$$\sum_{i=1}^{n} z_i \log p + (1-z_i)\left[\log(1-p) - \lambda + y_i\log(\lambda) - \log(y_i!)\right]$$

E-step:

$$\widehat{z}_i^{(t)} = E(z_i|Y_i) = P(z_i = 1|Y_i) = \begin{cases} \frac{\widehat{p}^{(t)}}{\widehat{p}^{(t)} + (1-\widehat{p}^{(t)})e^{-\widehat{\lambda}^{(t)}}}, Y_i = 0 \\ 0, Y_i > 0 \end{cases}$$

M-step:

$$\widehat{p}^{(t+1)} = \frac{\sum_i \widehat{z}_i^{(t)}}{n}$$

and

$$\widehat{\lambda}^{(t+1)} = \frac{\sum_i Y_i(1-\widehat{z}_i^{(t)})}{\sum_i 1-\widehat{z}_i^{(t)}}$$

```
Y <- c(rep(0,3062),rep(1,587),rep(2,284),rep(3,103),rep(4,33),rep(5,4),rep(6,2))
n <- length(Y)
Q <- function(Y,delta,lambda){
```

```
mid <- NULL
for (ii in 1:n){
if (Y[[ii]] == 0) mid[[ii]] <- delta / (delta + (1-delta)*exp(-lambda))
else mid[[ii]] <- 0
}
return(mid)
}

mles <- function(Y,Z){
delta <- sum(Z)/n
lambda <- sum(Y*(1-Z))/(n-sum(Z))
return(c(delta,lambda))
}
EMmix <- function(Y, delta, lambda, nreps=20) {
i <- 0
Z <- Q(Y, delta, lambda)
18
res <- c(0, delta, lambda)
while(i < nreps) {
i <- i + 1
para <- mles(Y,Z)
Z <- Q(Y, para[1], para[2])
res <- rbind(res, c(i,para[1],para[2]))
}
return(res)
}
delta <- 0.2
lambda <- 5
EMmix(Y, delta, lambda)
```

```
##     [,1]      [,2]     [,3]
## res    0 0.2000000 5.000000
##        1 0.7316907 1.488987
##        2 0.6939984 1.305579
##        3 0.6712037 1.215066
##        4 0.6560611 1.161570
##        5 0.6454941 1.126946
##        6 0.6378957 1.103299
##        7 0.6323234 1.086578
##        8 0.6281810 1.074472
##        9 0.6250715 1.065561
##       10 0.6227210 1.058922
##       11 0.6209349 1.053933
##       12 0.6195725 1.050159
##       13 0.6185301 1.047289
##       14 0.6177309 1.045099
##       15 0.6171171 1.043424
##       16 0.6166450 1.042139
##       17 0.6162816 1.041152
##       18 0.6160017 1.040393
##       19 0.6157859 1.039809
##       20 0.6156195 1.039359
```

Example 3 Lifetime data are often modeled as having an exponential distribution

$$f(y;\theta) = \frac{1}{\theta} e^{-y/\theta},\ y \geq 0.$$

Suppose it is of interest to estimate the mean lifetime θ of a population of lightbulbs. A first experiment is performed, giving data $X_1, \ldots, X_m$ of lifetimes. A second experiment of n bulbs is performed but in it, all bulbs are observed only once, at some fixed time t. For the second experiment, let E_i be the indicator function for the ith bulb, i.e., $E_i = 1$ if the ith bulb was still burning at time t, otherwise $E_i = 0$.

The observed data from both experiments is thus $(X_1, \ldots, X_m, E_1, \ldots, E_n)$ and the unobserved data is $Y_1, \ldots, Y_n$, the actual lifetimes of the bulbs in the second experiment. The log-likelihood function for the complete data is

$$\log L(\theta; \mathbf{X}, \mathbf{Y}) = -m\left(\log\theta + \bar{X}/\theta\right) - \sum_{i=1}^{n}\left(\log\theta + Y_i/\theta\right). \tag{23}$$

The expected value of Y_i still given the observed data at time t is

$$E[Y_i|X_1, \ldots, X_m, E_1, \ldots, E_n] = E[Y_i|E_i] = \begin{cases} \theta - \frac{te^{-t/\theta}}{1-e^{-t/\theta}}, & E_i = 0 \\ t + \theta, & E_i = 1. \end{cases} \tag{24}$$

Write the specific algorithm to obtain the maximum likelihood estimator for θ using the EM algorithm, and write an R function to execute your algorithm.

The following data is collected from one of such experiments where $n = m = 20$ and $t = 8$

$$\mathbf{Y} = (4.0, 12.8, 2.9, 27.2, 2.9, 3.1, 11.2, 9.0, 8.1, 9.8, 13.7, 8.3, 1.2, 0.9, 8.0, 18.8, 2.6, 22.6, 1.7, 4.0)$$

$$\mathbf{E} = (1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0)$$

Please apply your algorithm to this data, and present your results.

```
# E-step evaluating conditional means E(Zi|Xi)
delta <- function(E, theta)
{  t=8
  Ey0 <- theta-t*exp(-t/theta)/(1-exp(-t/theta))
  Ey1 <- t+theta
  return ((E==0)*Ey0+(E==1)*Ey1)
}
# M-step updating the parameters
mles <- function(delta, X, E)
{
  m=length(X)
  n=length(E)
  thetahat <- 1/(m+n)*(sum(X)+sum(delta))
  return(thetahat)
}
Emix <- function(X, E, start, nreps=10)
{
  i=0
  Z=delta(E,start)
  newpars <- start
  res <- c(0,t(as.matrix(newpars)))
  error <- 1
```

```
  while(i < nreps & error > 1e-5) # should check for convergence
  {
i <- i+1
1
oldpars <- newpars
    newpars <- mles(Z,X,E)
    error <- abs(newpars-oldpars)
    Z <- delta(E,newpars)
    res <- rbind(res,c(i, t(as.matrix(newpars))))
}
  return(res)
}
# given data
n=20;m=20;t=8
X <- c(4.0, 12.8, 2.9, 27.2, 2.9, 3.1, 11.2, 9.0, 8.1, 9.8, 13.7, 8.3, 1.2, 0.9, 8.0, 18.8, 2.6, 22.6,
E <- c(1,0,0,0,0,1,1,1,1,1,1,1,1,0,0,1,0,1,0)
Emix(X, E, start=1, nreps=50)
```

```
##     [,1]      [,2]
## res    0  1.000000
##        1  7.219463
##        2  9.541028
##        3 10.271799
##        4 10.498730
##        5 10.568989
##        6 10.590723
##        7 10.597445
##        8 10.599523
##        9 10.600166
##       10 10.600365
##       11 10.600426
##       12 10.600445
##       13 10.600451
```

Example 4 The Fisher's genotype example

A two linked bi-allelic loci, A and B, with alleles A and a, and B and b, respectively. A is dominant over a and B is dominant over b. Since the two loci are linked, types AB and ab will appear with the same frequency $(1-r)/2$*, and types Ab and aB will appear with the same frequency* $r/2$*. So a genotype AABB will have the frequency* $(1-r)(1-r)/4$ *and a genotype AaBB will have the frequency* $r(1-r)/4\ldots$

Due to the dominate feature, there are 4 classes of phenotypes, A$_$B, A$_$bb, aaB *and aabb. Let* $\psi = (1-r)(1-r)$*, one can derive that the joint distribution of the 4 phenotypes* $\mathbf{y} = \{y_1, y_2, y_3, y_4\}$ *from a random sample with n subject is multinomial*

$$Multinomial[n, \frac{2+\psi}{4}, \frac{1-\psi}{4}, \frac{1-\psi}{4}, \frac{\psi}{4}]$$

Question – How to estimate ψ*?*

MLE

$$L(\mathbf{y}, \psi) = \frac{n!}{y_1!y_2!y_3!y_4!}(1/2+\psi/4)^{y_1}(1/4-\psi/4)^{(y_2+y_3)}(\psi/4)^{y_4}$$

$$\log L(\mathbf{y}, \psi) = y_1 \log(2+\psi) + (y_2+y_3)\log(1-\psi) + y_4 \log(\psi)$$

$$\frac{\partial L(\mathbf{y},\psi)}{\partial \psi} = \frac{y_1}{2+\psi} + \frac{y_2+y_3}{1-\psi} + \frac{4}{\psi}$$

Suppose $y_1 = y_{11} + y_{12}$, *where* $y_{11} \sim B(n, 1/2)$ *and* $y_{12} \sim B(n, \psi/4)$. *Then the complete log likelihood of* $\{y_{11}, y_{12}, y_2, y_3, y_4\}$ *is*

$$\log L_c(\psi) = (y_{12}+y_4)\log(\psi) + (y_2+y_3)\log(1-\psi)$$

In the t-th E step, we need to estimate $E[y_{12}|\mathbf{y}, \psi^{(t)}]$. *Since*

$$y_{11} \sim B(y_1, \frac{0.5}{0.5+\psi^{(t)}/4})$$

$$y_{12}^{(t)} = E[y_{12}|\mathbf{y}, \psi^{(t)}] = y_1 - \frac{0.5 y_1}{0.5+\psi^{(t)}/4}$$

In the M step, we need to maximize $(y_{12}^{(t)}+y_4)\log(\psi) + (y_2+y_3)\log(1-\psi)$, *which is equivalent to solve the following simple linear function*

$$\frac{y_{12}^{(t)}+y_4}{\psi} - \frac{y_2+y_3}{1-\psi} = 0$$

$$\psi^{(t)} = \frac{y_{12}^{(t)}+y_4}{y_{12}^{(t)}+y_2+y_3+y_4} = \frac{y_{12}^{(t)}+y_4}{n-y_{11}^{(t)}}$$

Question: When $\mathbf{y} = (125, 18, 20, 34)$, *what is* ψ?

```
p1=125/197
p2=18/197
p3=20/197
p4=34/197

my_prob <- c(p1,p2,p3,p4)
number_of_experiments <- 10
number_of_samples <- 197

experiments <- rmultinom(n=number_of_experiments, size=number_of_samples, prob=my_prob)
experiments
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  116  125  110  116  132  134  117  122  126   128
## [2,]   17   26   15   22   13   17   21   16   21    17
## [3,]   25   17   26   23   15   19   16   30   21    23
## [4,]   39   29   46   36   37   27   43   29   29    29
```

To illustrate the EM algorithm, we represent Rao's data as incomplete data from a five-category multinomial population where the cell probabilities are (p1,p2,p3,p4,p5), the idea being to split the first of the original four categories into two categories. The expectation step estimates the sufficient statistics of the complete data, given the observed data. In this case, (n3, n4, n5) are known to be (18,20,34) so that the only sufficient statistics that have to be estimated are z1 and z2, where z1+z2 =n1 = 125. Estimating z1 and z2, using the current estimate of p or theta leads to E-step computation

```
estep <- function(theta,z2){
z2 = 125*(0.25*theta/(0.5+0.25*theta))
# z1 = 125*(0.5/(0.5+0.25*theta))
return(z2)
```

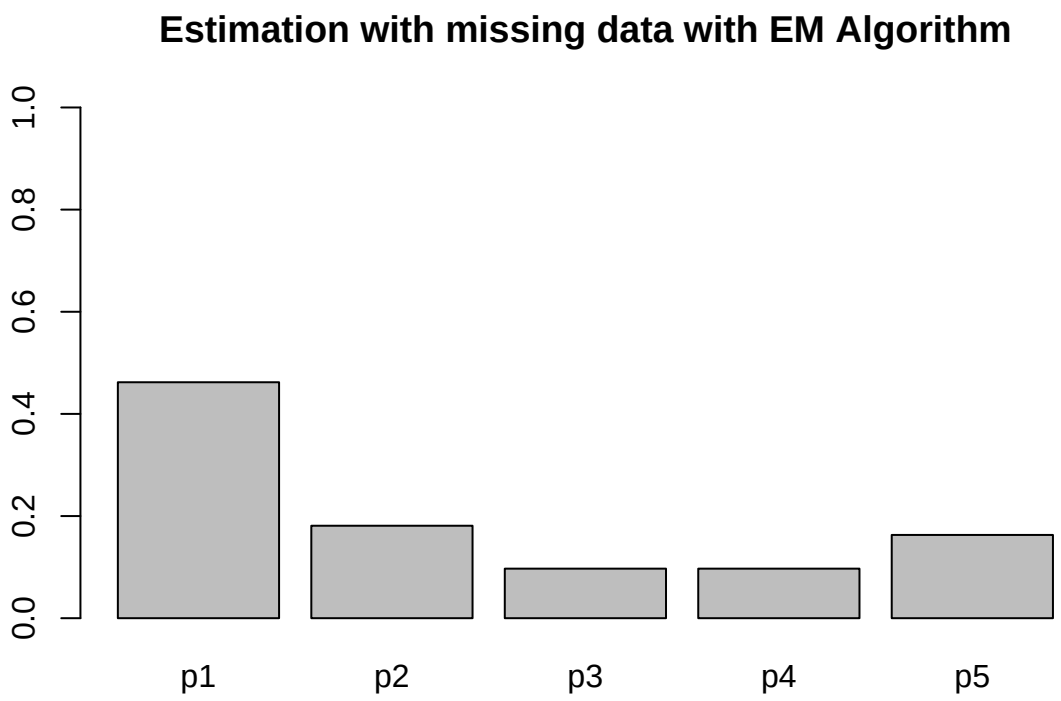
```
}

mstep <- function(theta,z2){
theta <- (z2+34)/(z2+34+18+20)
return(theta)
}

# set initial value for iteration

cur_theta = 0.5
maxit = 100
maxit=1000
tol=1e-6

# start iteration

for(i in 1:maxit){
  new_z2 <- estep(cur_theta,cur_z2)
  new_theta <- mstep(cur_theta,new_z2)

  # Stop iteration if the difference between the current and new estimates is less than a tolerance lev
  if( all(abs(cur_theta - new_theta) < tol) ){ flag <- 1; break}


  # Otherwise continue iteration
  cur_theta <- new_theta
  cur_z2 <- new_z2
}
if(!flag) warning("Didn't converge\n")

final_theta = cur_theta
paste0("Final Theta = ", format(round(cur_theta, 4), nsmall = 4))
```

```
## [1] "Final Theta = 0.6268"
```

```
p1=0.5
p2=0.25*final_theta
p3=0.25-p2
p4=p3
p5=p2

my_prob <- c(p1,p2,p3,p4,p5)
number_of_experiments <- 10
number_of_samples <- 1000

experiments <- rmultinom(n=number_of_experiments, size=number_of_samples, prob=my_prob)
experiments
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  462  474  480  464  469  496  514  511  486   488
## [2,]  181  168  147  169  172  164  141  151  171   154
## [3,]   97   93   87  107   99  106   89   80   87   103
## [4,]   97   88  119  102  108   96   96  105   97    77
## [5,]  163  177  167  158  152  138  160  153  159   178
```

```r
df=data.frame(experiments)/number_of_samples
x<-df[,1]
names(x)<- c("p1","p2","p3","p4","p5")

barplot(x,ylim=c(0,1),main="Estimation with missing data with EM Algorithm")
```



Example 5 ABO blood type

Consider the ABO blood type data, where you have $N_{obs} = (N_A, N_B, N_{AB}, N_O) = (26, 27, 42, 7)$.

- *Design an EM algorithm to estimate the allele frequencies,* P_A, P_B *and* P_O*; and}*

The relationship between phenotype and genotype in ABO blood type data is determined by the following table.

Bloodtype	*A*	*B*	*AB*	*O*
Genotype	*A/A, A/O*	*B/B, B/O*	*A/B*	*O/O*

While complete data for this case would be the number of people with each genotype, denoted by $N = (N_{A/A}, N_{A/O}, N_{B/B}, N_{B/O}, N_{A/B}, N_{O/O})$*, we only observed the number of people with each phenotype, say* $N_{obs} = (N_A, N_B, N_{AB}, N_O)$.

Note that the goal is to estimate the frequencies of alleles A, B, and O, denoted by p_A*,* p_B*, and* p_O*, respectively. According to the Hardy-Weinberg law, the genotype frequencies are*

$$Prob(Genotype = A/A) = p_A^2$$

$$Prob(Genotype = A/O) = 2p_A p_O$$

$$Prob(Genotype = B/B) = p_B^2$$
$$Prob(Genotype = B/O) = 2p_Bp_O$$
$$Prob(Genotype = A/B) = 2p_Ap_B$$
$$Prob(Genotype = O/O) = p_O^2$$

Furthermore, genotype counts $N = (N_{A/A}, N_{A/O}, N_{B/B}, N_{B/O}, N_{A/B}, N_{O/O})$ *are jointly multinomially distributed with log-likelihood function as shown below.*

$$\begin{aligned}\log L(p|N) &= N_{A/A}\log(p_A^2) + N_{A/O}\log(2p_Ap_O) + N_{B/B}\log(p_B^2) + N_{B/O}\log(2p_Bp_O)\\ &+ N_{A/B}\log(2p_Ap_B) + N_{O/O}\log(p_O^2)\\ &+ \log\Big(\frac{n!}{N_{A/A}!N_{A/O}!N_{B/B}!N_{B/O}!N_{A/B}!N_{O/O}!}\Big)\end{aligned}$$

where $n = N_{A/A} + N_{A/O} + N_{B/B} + N_{B/O} + N_{A/B} + N_{O/O}$.

*** E-step ***

Note $N_{A/A} + N_{A/O} = N_A$ *and* $N_{B/B} + N_{B/O} = N_B$. *Thus the conditional distribution of* $N_{A/A}|N_A$ *and* $N_{B/B}|N_B$ *are*

$$N_{A/A}|N_A \sim Bin\left(N_A, \frac{p_A^2}{p_A^2 + 2p_Ap_O}\right)$$

and

$$N_{B/B}|N_B \sim Bin\left(N_B, \frac{p_B^2}{p_B^2 + 2p_Bp_O}\right)$$

respectively.

Therefore, the expectations in the k-th iteration can be easily calculated as follows.

$$N_{A/A}^{(k)} = E(N_{A/A}|N_{obs}, p^{(k)}) = N_A \times \frac{{p_A^{(k)}}^2}{{p_A^{(k)}}^2 + 2p_A^{(k)}p_O^{(k)}}$$

$$N_{A/O}^{(k)} = E(N_{A/O}|N_{obs}, p^{(k)}) = N_A \times \frac{2p_A^{(k)}p_O^{(k)}}{{p_A^{(k)}}^2 + 2p_A^{(k)}p_O^{(k)}}$$

$$N_{B/B}^{(k)} = E(N_{B/B}|N_{obs}, p^{(k)}) = N_B \times \frac{{p_B^{(k)}}^2}{{p_B^{(k)}}^2 + 2p_B^{(k)}p_O^{(k)}}$$

$$N_{B/O}^{(k)} = E(N_{B/O}|N_{obs}, p^{(k)}) = N_B \times \frac{2p_B^{(k)}p_O^{(k)}}{{p_B^{(k)}}^2 + 2p_B^{(k)}p_O^{(k)}}.$$

Moreover, it is obvious that

$$E(N_{A/B}|N_{obs}, p^{(k)}) = N_{A/B}$$

and

$$E(N_{O/O}|N_{obs}, p^{(k)}) = N_{O/O}.$$

M-step

Now consider maximizing $Q(p|p^{(k)})$ under the restriction $p_A + p_B + p_O = 1$. Introduce Lagrange multiplier λ and maximize

$$Q_L(p, \lambda|p^{(k)}) = Q(p|p^{(k)}) + \lambda(p_A + p_B + p_O - 1)$$

with respect to $p = (p_A, p_B, p_O)$ and λ.

$$\frac{\partial Q_L(p,\lambda|p^{(k)})}{\partial p_A} = \frac{2N_{A/A}^{(k)}}{p_A} + \frac{N_{A/O}^{(k)}}{p_A} + \frac{N_{A/B}^{(k)}}{p_A} + \lambda \quad (25)$$

$$\frac{\partial Q_L(p,\lambda|p^{(k)})}{\partial p_B} = \frac{2N_{B/B}^{(k)}}{p_B} + \frac{N_{B/O}^{(k)}}{p_B} + \frac{N_{A/B}^{(k)}}{p_B} + \lambda \quad (26)$$

$$\frac{\partial Q_L(p,\lambda|p^{(k)})}{\partial p_O} = \frac{N_{A/O}^{(k)}}{p_O} + \frac{N_{B/O}^{(k)}}{p_O} + \frac{2N_{O/O}^{(k)}}{p_O} + \lambda \quad (27)$$

$$\frac{\partial Q_L(p,\lambda|p^{(k)})}{\partial p_\lambda} = p_A + p_B + p_C - 1 \quad (28)$$

Since $N_{A/A}^{(k)} + N_{A/O}^{(k)} + N_{B/B}^{(k)} + N_{B/O}^{(k)} + N_{A/B}^{(k)} + N_{O/O}^{(k)} = n$, from the above four functions, we get $\lambda = -2n$. By plugging $\lambda = -2n$ in and setting (1), (2), and (3) to be zero, update (p_A, p_B, p_O) as follows. $p_A^{(k+1)} = \frac{2N_{A/A}^{(k)} + N_{A/O}^{(k)} + N_{A/B}^{(k)}}{2n}$ $p_B^{(k+1)} = \frac{2N_{B/B}^{(k)} + N_{B/O}^{(k)} + N_{A/B}^{(k)}}{2n}$ $p_O^{(k+1)} = \frac{2N_{O/O}^{(k)} + N_{A/O}^{(k)} + N_{B/O}^{(k)}}{2n}$

Repeat E-step and M-step until convergence.

```
#EM iteration
# Arguments:
# N=(Na,Nb,Nab,No)
# p=(pa,pb,po)

estep <- function(N,p) {

#E-step

Naa <- N[1] * p[1]^2/(p[1]^2 + 2 * p[1] * p[3])
Nao <- N[1] * 2 * p[1] * p[3]/(p[1]^2 + 2 * p[1] * p[3])
Nbb <- N[2] * p[2]^2 / (p[2]^2 + 2 * p[2] * p[3])
Nbo <- N[2] * 2 * p[2] * p[3]/(p[2]^2 + 2 * p[2] * p[3])
Nab <- N[3]
Noo <- N[4]
N_alle <- c(Naa, Nao, Nbb, Nbo, Nab, Noo)
return(N_alle)
}

#M-step
mstep <- function(N_alle, p){
  n <- sum(N_alle)
  p[1] = (2*N_alle[1] + N_alle[2] + N_alle[5])/(2*n)
  p[2] = (2*N_alle[3] + N_alle[4] + N_alle[5])/(2*n)
  p[3] = (2*N_alle[6] + N_alle[2] + N_alle[4])/(2*n)

return(p)
}

# EM mix
```

```r
emmix <- function(obs, start, nreps = 10){
  i <- 0
  prob <- start
  N <- obs
  N_alle <- estep(obs, start)
  res <- c(i, t(as.matrix(prob)))
  while (i < nreps) {
    i <- i + 1
    prob <- mstep(N_alle, prob)
    N_alle <- estep(N, prob)
    res <- rbind(res, c(i, t(as.matrix(prob))))
  }
  return(res)
}

#observed data
N_obs <- c(26, 27, 42, 7)

#Starting value
p_ini <- c(1,1,1)/3

emmix(N_obs, p_ini)
```

```
##     [,1]      [,2]      [,3]      [,4]
## res    0 0.3333333 0.3333333 0.3333333
##        1 0.3758170 0.3823529 0.2418301
##        2 0.3890628 0.3966704 0.2142668
##        3 0.3939824 0.4018565 0.2041611
##        4 0.3959198 0.4038836 0.2001965
##        5 0.3967009 0.4046990 0.1986002
##        6 0.3970187 0.4050306 0.1979507
##        7 0.3971486 0.4051661 0.1976853
##        8 0.3972018 0.4052215 0.1975767
##        9 0.3972236 0.4052442 0.1975322
##       10 0.3972325 0.4052535 0.1975140
```

Problem 1: A classical clustering algorithm–K-means

Clustering is an important statstical learning method that automatically group data by similar features. Let $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \in \mathbb{R}^p$ *be a collection of* p *dimensional data points. The* K*-means algorithm partitions data into* k *clusters (*k *is predetermined). We denote* $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k\}$ *as the centers of the* k *(unknown) clusters, and denote* $\mathbf{r}_i = (r_{i,1}, ..., r_{i,k}) \in \mathbb{R}^k$ *as the "hard" cluster assignment of* $\mathbf{x}_i$*. The cluster assignment* $\mathbf{r}_i$ *takes form* $(0, 0, ..., 0, 1, 0, 0)$ *with* $r_{i,j} = I\{\mathbf{x}_i$ *assigned to cluster* $j\}$*. (Assign* $\mathbf{x}_i$ *to one and only one of the* k *clusters)*

k*-means essentially finds cluster centers and cluster assignments that minimize the objective function* $J(\mathbf{r}, \boldsymbol{\mu}) = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{i,j} \|\mathbf{x}_i - \mu_k\|^2$ *The k-means algorithm follows the following steps.*

1. *Standardize the data*
2. *Randomly initialize* k *cluster centers* $\{\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, ..., \boldsymbol{\mu}_k^{(0)}\}$.
3. *Repeat the following two steps iteratively until converge.*

- ***Find optimal cluster assignment fixing cluster centers.*** *Minimizing* $J(\mathbf{r}, \boldsymbol{\mu})$ *over* $\mathbf{r}$ *yields* $r_{i,j}^{(v+1)} = I\{j = \arg\min_j \|\mathbf{x}_i - \mu_j^{(v)}\|\}$ *That is, assign* $\mathbf{x}_i$ *to cluster* j *with minimal distance* $\|\mathbf{x}_i - \mu_j^{(v)}\|$,

where $\mu_j^{(v)}$ *is the cluster center in the* v*-th iteration.*

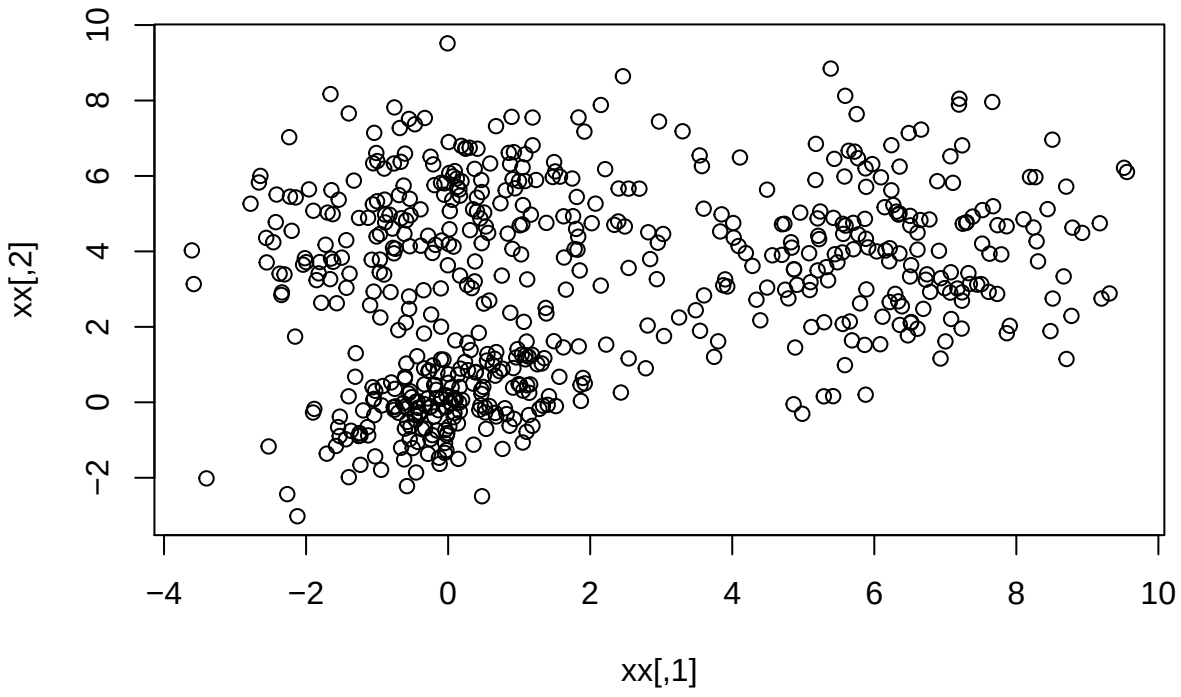
- ***Calcuate cluster centers using the cluster assignment in the last step.*** *Minimizing* $J(\mathbf{r}, \boldsymbol{\mu})$ *over* $\boldsymbol{\mu}$ *yields* $\mu_j^{(v+1)} = \frac{\sum_{i=1}^{n} \mathbf{x}_i r_{i,j}^{(v+1)}}{\sum_{i=1}^{n} r_{i,j}^{(v+1)}}$ *That is the sample mean of* $\mathbf{x}_i$ *that were assigned to the cluster* j *in the last step.*

*** Your jobs ***

1.1 Implement the k-means algorithm into $\mathbb{R}$*.*

1.2. Using the following R codes to generate a mixture of 3 bivariate Guassian distribution(different means/variances), and apply the k-means algorithm to particition the sample into 3 clusters. Comparing the resluting clusters with the orginal ones, how effective is the k-means algorithm?

```
library(MASS)
set.seed(123123)
Sigma <- matrix(c(1,0.5,0.5,1),2,2)
x1 = mvrnorm(n = 200, mu=c(0,0), Sigma)
Sigma <- matrix(c(2,0.5,0.5,2),2,2)
x2 = mvrnorm(n = 200, mu=c(0,5), Sigma)
Sigma <- matrix(c(3,0.5,0.5,3),2,2)
x3 = mvrnorm(n = 200, mu=c(6,4), Sigma)
xx = rbind(x1,x2,x3)
plot(xx)
```




```
## Standardize the data

center <- function(x) {

  for (i in 1:ncol(x)) {
    x[,i] <- (x[,i] - mean(x[,i]))/sd(x[,i])
  }

  return(x)
}

## Randomly initialize k cluster centers.
init_draw <- function(x, k) {
  p <- ncol(x)
  # p * k
  mu_0 <- matrix(rnorm(p*k), nrow = p, ncol = k)
  return(mu_0)
}

update_mu <- function(old_mu, x) {
  # old_mu: p * k matrix of previous centers
  k <- ncol(old_mu)
  n <- nrow(x)

  # n * k matrix of assignments
  r <- matrix(0, nrow = n, ncol = k)
```

```
  distance <- numeric(k)
  for (i in 1:n) {
    for (j in 1:k) {
     distance[j] <- sqrt(sum((x[i,] - old_mu[, j])^2))
    }
    min_k <- which(distance == min(distance))[1]
    r[i, min_k] <- 1
  }

  new_mu <- matrix(NA, nrow = nrow(old_mu), ncol = ncol(old_mu))

  for (j in 1:k) {
    denom <- sum(r[ ,j])

    numer <- 0
    for (i in 1:n) {
      numer <- numer + x[i,]*r[i,j]
    }
    new_mu[,j] <- numer/denom
  }

  return(new_mu)
}

my_kmeans <- function(x, k, tol) {
  x <- center(x)
  mu_0 <- init_draw(x, k)
  old_mu <- mu_0
  new_mu <- update_mu(old_mu, x)
  while (sum(abs(new_mu - old_mu)) > tol){
    old_mu <- new_mu
    new_mu <- update_mu(old_mu, x)
  }
  return(new_mu)
}

my_kmeans(xx, 3, 1e-05)
```

```
##           [,1]      [,2]       [,3]
## [1,] 1.3461711 -0.573453 -0.6389998
## [2,] 0.4131056 -1.106338  0.8188457
```

Problem 2: Clustering based on Gussian Mixture

Assuming that $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \in \mathbb{R}^p$ *are i.i.d. random vectors following a mixture mulitvariate normal distributions with* k *hidden groups.*

$$\mathbf{x}_i \sim \begin{cases} N(\boldsymbol{\mu}_1, \Sigma_1), \textit{with probability } p_1 \\ N(\boldsymbol{\mu}_2, \Sigma_2), \textit{with probability } p_2 \\ \quad \vdots \quad , \quad \vdots \\ N(\boldsymbol{\mu}_k, \Sigma_k), \textit{with probability } p_k \end{cases}$$

$\sum_{j=1}^{k} p_j = 1$

2.1 What is the likelihood of $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$*?*

2.2 Let $\mathbf{r}_i = (r_{i,1}, ..., r_{i,k}) \in \mathbb{R}^k$ *as the cluster indicator of* $\mathbf{x}_i$*, which takes form* $(0, 0, ..., 0, 1, 0, 0)$ *with* $r_{i,j} = I\{\mathbf{x}_i$ *belongs to cluster* $j\}$*. The cluster indicator* $\mathbf{r}_i$ *is a latent variable that cannot be observed. What is complete likelihood of* $(\mathbf{x}_i, \mathbf{r}_i)$*.*

2.3 Derive an EM algorithm to estimate the parameter μ*'s,* $\mathbf{\Sigma}$*'s and* p_j*'s. Clearly write out E-step and M-step in each iteration.*

2.4 Design a clustering algorithm based on the mixture Gaussian. Comparing the two clustering algorithms (EM vs k-means), what are the differences?

2.5 Applying your Mixture Guassian EM-based clustering algoirthm to the same data that you generated in Problem 1. Which clustering method does a better job in grouping the sample?

2.5 If we make a more restrictive assumption such that

$$\mathbf{x}_i \sim \begin{cases} N(\boldsymbol{\mu}_1, \sigma^2 I), & \text{with probability } p_1 \\ N(\boldsymbol{\mu}_2, \sigma^2 I), & \text{with probability } p_2 \\ \quad \vdots \quad , \quad \vdots & \\ N(\boldsymbol{\mu}_k, \sigma^2 I), & \text{with probability } p_k \end{cases}$$

How would you adapt the E-M clustering algorithm under this restricive setting? How is

If we let $\sigma^2 \to 0$, how would the E-M clustering algorithm look like?

2.6 Now genenrate a random sample following bivariate sknewed normal distribution. Apply both k-means and Gaussian mixture EM to clustering the generated data, which method is more effective in this case?

```
library(sn)
```

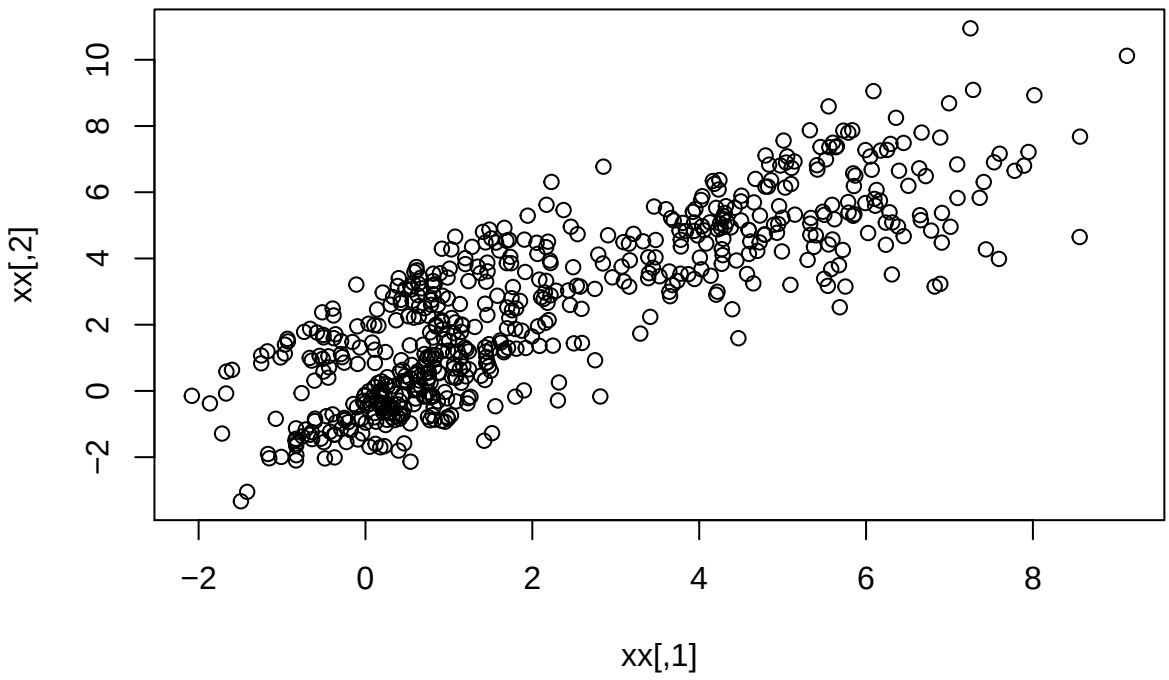
```
## Warning: package 'sn' was built under R version 3.5.2
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'sn'
```

```
## The following object is masked from 'package:stats':
##
##     sd
```

```
set.seed(666666)
Omega <- matrix(c(1,0.5,0.5,1),2,2)
x1 = rmsn(n = 200, xi=c(0,0), Omega=Omega, alpha = c(9,-6))
Omega <- matrix(c(2,0.5,0.5,2),2,2)
x2 = rmsn(n = 200, xi=c(0,3), Omega=Omega, alpha = c(9,-6))
Omega <- matrix(c(3,0.5,0.5,3),2,2)
x3 = rmsn(n = 200, xi=c(4,6), Omega=Omega, alpha = c(9,-6))
xx = rbind((x1),(x2),(x3))
plot(xx)
```



2.1 Likelihood of {x1, x2„„„}

$\prod_{i=1}^{n}(p_1 f(x_i, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + ... + p_k f(x_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))$

2.2 Likelihood of complete data

$\prod_{i=1}^{n}\prod_{j=1}^{k} p_j^{r_{i,j}} f(x_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)^{r_{i,j}}$

2.2.1 Log-likelihood of complete data

$\sum_{i=1}^{n}\sum_{j=1}^{k} r_{i,j}\{lnp_j + lnf(x_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\}$

2.3 Derive an EM algorithm to estimate the parameter μ, Σ and p. Clearly write out E-step and M-step in each iteration.

Let $Z(r_{i,j})$ be

$$Z(r_{i,j}) = E(r_{i,j}) = \frac{p_j f(x_i|\mu_j, \Sigma_j)}{\sum_{k=1}^{k} p_k f(x_i|\mu_k, \Sigma_k)}$$

Then, E-step is:

$E_r[lnf(X,r)|p,\boldsymbol{\mu},\boldsymbol{\Sigma})] = \sum_{i=1}^{n}\sum_{j=1}^{k} Z(r_{i,j})\{lnp_j + lnf(x_i|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)\}$

M-step is:

$$\mu_j = \frac{1}{n_j}\sum_{i=1}^{n} Z(r_{i,j})x_i \tag{29}$$

$$\Sigma_j = \frac{1}{n_j}\sum_{i=1}^{n} Z(r_{i,j})(x_i-\mu_j)(x_i-\mu_j)^T \tag{30}$$

$$p_j = \frac{n_j}{n} \tag{31}$$

```
# obs: x <- n x p
# p <- k x 1
# mu <- k x 1
# sigma <- k x k
# phi = c(f(x, mu_1, sigma_1), ,,, f(x, mu_k, sigma_k))
# N = c(n1, n2,,, nk)

# n <- nrow(x)
# p <- ncol(x)
# k <- ncol(mu)
# phi <- rep(NA, k)
#
#  for (j in 1:k){
#    phi[j] <- dnorm(x, mu = mu[j], sd = sqrt(sigma[j, j]))
#  }

# # E[r_i,j]
# estep <- function(p, phi){
#   for(j in 1:k){
#    delta <- (p[j] * phi[j])/(sum(p[j] * phi[j]))
#   }
#    return(delta)
#  }
#
# mstep <- function(pars, x, p, phi) {
#   k <- ncol(pars)
#   n <- nrow(x)
#
#   # n * k matrix of assignments
#   r <- matrix(0, nrow = n, ncol = k)
#
#   distance <- numeric(k)
#   for (i in 1:n) {
#     for (j in 1:k) {
#      distance[j] <- sqrt(sum((x[i,] - old_mu[, j])^2))
#     }
#     min_k <- which(distance == min(distance))[1]
#     r[i, min_k] <- 1
#   }
#
#   new_mu <- matrix(NA, nrow = nrow(old_mu), ncol = ncol(old_mu))
```

```
#
#   for (j in 1:k) {
#     denom <- sum(r[ ,j])
#
#     numer <- 0
#     for (i in 1:n) {
#       numer <- numer + x[i,]*r[i,j]
#     }
#     new_mu[,j] <- numer/denom
#   }
#
#   return(new_mu)
# }
```