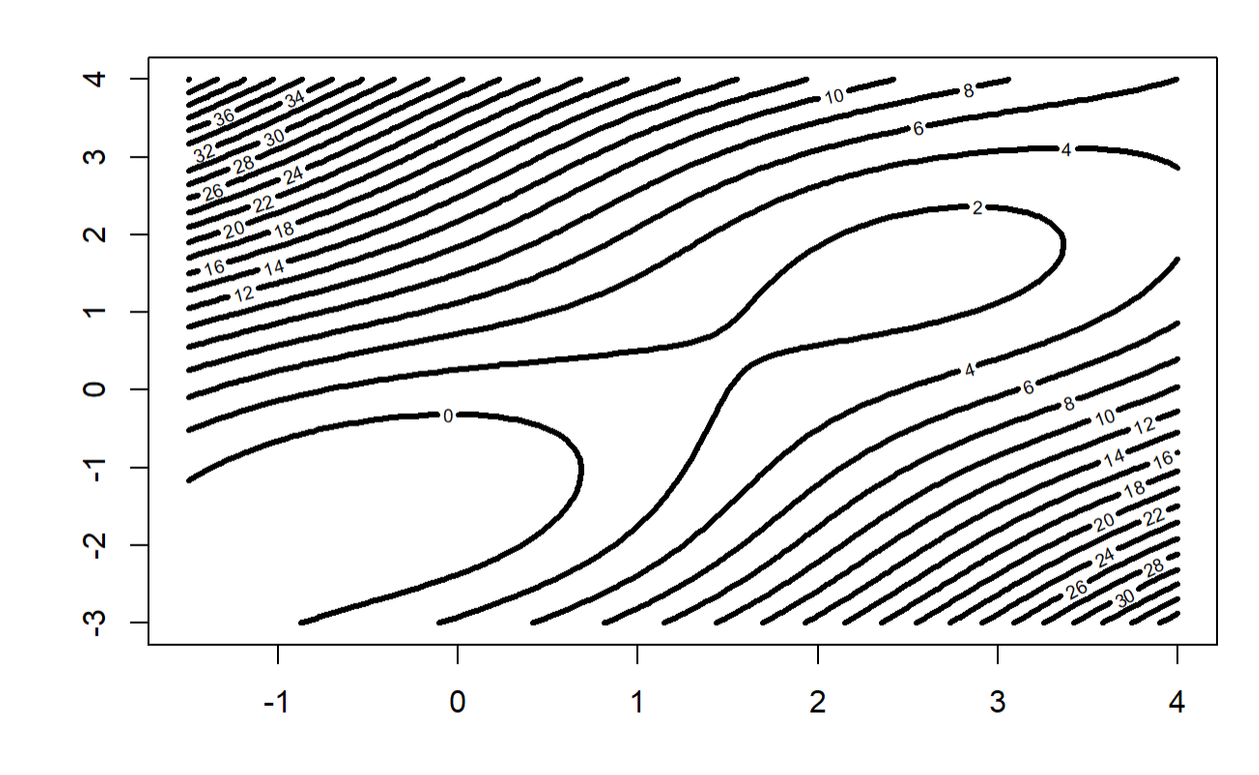
title: “Home assignment” author: “Edvin Magnusson” date: ‘2022-03-11’ output: html_document

Question 1

```
# Question 1a)

xgrid <- seq(-1.5,4, length=100)
ygrid <-seq(-3,4,length=100 )

fun_1 <- function(x,y){
  sin(x+y) + (x-y)^(2)-1.5*x+ 2.5*y+ 1
}



X_Y_Mat <- matrix(0,nrow = length(xgrid),ncol = length(ygrid))


for (i in 1:length(xgrid)){
  for(j in 1:length(ygrid)){
    X_Y_Mat[i,j]<- fun_1(x=xgrid[i],y=ygrid[j])
  }
}


contour(xgrid, ygrid, X_Y_Mat,  nlevels = 30,col = "black", lwd = 3)
```



```
# 1b)


# To find the minimum, we apply Newton's method to the gradient equation
# we need partial derivaies and the hessian matrix


f<- function(x){
  sin(x[1]+x[2]) + (x[1]-x[2])^(2)-1.5*x[1]+ 2.5*x[2]+ 1
}


# Setting up the function, gradient and the hessian matrix

f<- function(x){
  x1<- x[1]
  x2<- x[2]

  sin(x1+x2) + (x1-x2)^(2)-1.5*x1+ 2.5*x2+ 1
}

f(c(0,-1))
```

```
## [1] -1.341471
```

```
Gradient <- function(x){
  x1  <- x[1]
  x2  <- x[2]
  dx1 <- cos(x1+x2)+2*(x1-x2)-1.5
  dx2 <- cos(x2+x1)-2*(x1-x2)+2.5
  c(dx1,dx2)
}


Hessian<- function(x){
  x1   <- x[1]
  x2   <- x[2]
  hx11 <- 2-sin(1+x2)
  hx12<- -sin(x2+x1)-2
  hx22 <- 2-sin(x1+x2)
  matrix(c(hx11, hx12, hx12, hx22), ncol=2)
}



newton <- function(x0)
{
  xt  <- x0
  xt1 <- x0 + 2
  while(sum((xt-xt1)^2)>0.0001)
  {
    xt1 <- xt
    xt  <- xt1 - solve(Hessian(xt1)) %*% Gradient(xt1) # The newton formula
  }
  xt
}


newton(c(0,-2))
```

```
##            [,1]
## [1,] -0.5475879
## [2,] -1.5473507
```

```
newton(c(0,-1))
```

```
##            [,1]
## [1,] -0.5476804
## [2,] -1.5473867
```

```
f(newton(c(0,-2)))# With this value as starting point we get the correct minimum
```

```
## [1] -1.913223
```

```
f(newton(c(0,-1)))# We also get the right minimum with this starting value.
```

```
## [1] -1.913223
```

```
f(newton(c(3,3)))# However, with this starting value we do not reach the minimum
```

```
## [1] 1.913223
```

```
# The Newton method is sensitive to starting values. But we have a contour plot we can use as reference
# to select starting values.
# We can choose an optimal starting value at for example (0,-2)
# the miminized function seems to have a value of about -1.9132


# The stopping criteria is basically (xn+1-xn)< eps
# where epsilon is the predetermined tolerance level.
```

```
#1 c) Minimize boundaries
# Since the method is not specified in the question i will use the built in optimize function to check the bounda
ries.

func<- function(x){

  sin(x+4) + (x-4)^(2)-1.5*x+ 2.5*4+ 1
}

optimize(f=func,interval = c(-1.5,4)) # This is the minimum when y=4 and x is between -1.5 and 4.
```

```
## $minimum
## [1] 3.999955
##
## $objective
## [1] 5.989432
```

```
func2<- function(x){

  sin(x+(-3)) + (x-(-3))^(2)-1.5*x+ 2.5*(-3)+ 1
}

optimize(f=func2, interval = c(-1.5,4)) # This is the minimum when y=-3 and x is between -1.5 and 4
```

```
## $minimum
## [1] -1.499955
##
## $objective
## [1] -1.022412
```

```
func3 <- function(y){
  sin(-1.5+y)+(-1.5-y)^(2)-1.5*(-1.5)+2.5*(y)+1
}

optimize(f=func3,interval = c(-3,4)) # This is the minimun when x= -1.5 and y is between -3 and 4.
```

```
## $minimum
## [1] -2.380343
##
## $objective
## [1] -1.252489
```

```
func4 <- function(y){
  sin(4+y)+(4-y)^(2)-1.5*(4)+2.5*y+1
}

optimize(f=func4,interval = c(-3,4)) # This is the minimun when x= 4 and y is between -3 and 4.
```

```
## $minimum
## [1] 2.250287
##
## $objective
## [1] 3.654321
```

The output displays $minimum which represents where the mimimun is located and objective displays the value at this point. None of the values at the boundaries are lower than the result obtaind from the newton raphson algorithm. From these results we can draw the conclusion that the local minimun is obtained at x= -0.547 and y=-1.547 with the minimum value of -1.9132

Question 2

```
# 2) Simpsons 3/8 rule
#


Simpson <- function (f,a,b,n){
  h <- (b-a)/n
  x<- c()
  x[1]<-f(a)
  x[n+1]<-f(b)
  sum<- (f(a)+f(b))
  for (i in 1:(n-1)){
    x[i+1]<- f(a+i*h)
    if(i%%3==0){
      sum<- sum+2*f(a+i*h)

    }
    else{
      sum<- sum+3*f(a+i*h)
    }
    integral<- sum*h*3/8
  }
  return(integral)
}



# 2b)

Simpson(f=dnorm, a=-2, b=2, n=10)
```

```
## [1] 0.9471507
```

```
Simpson(f=dnorm,a=-2,b=2,n=20)
```

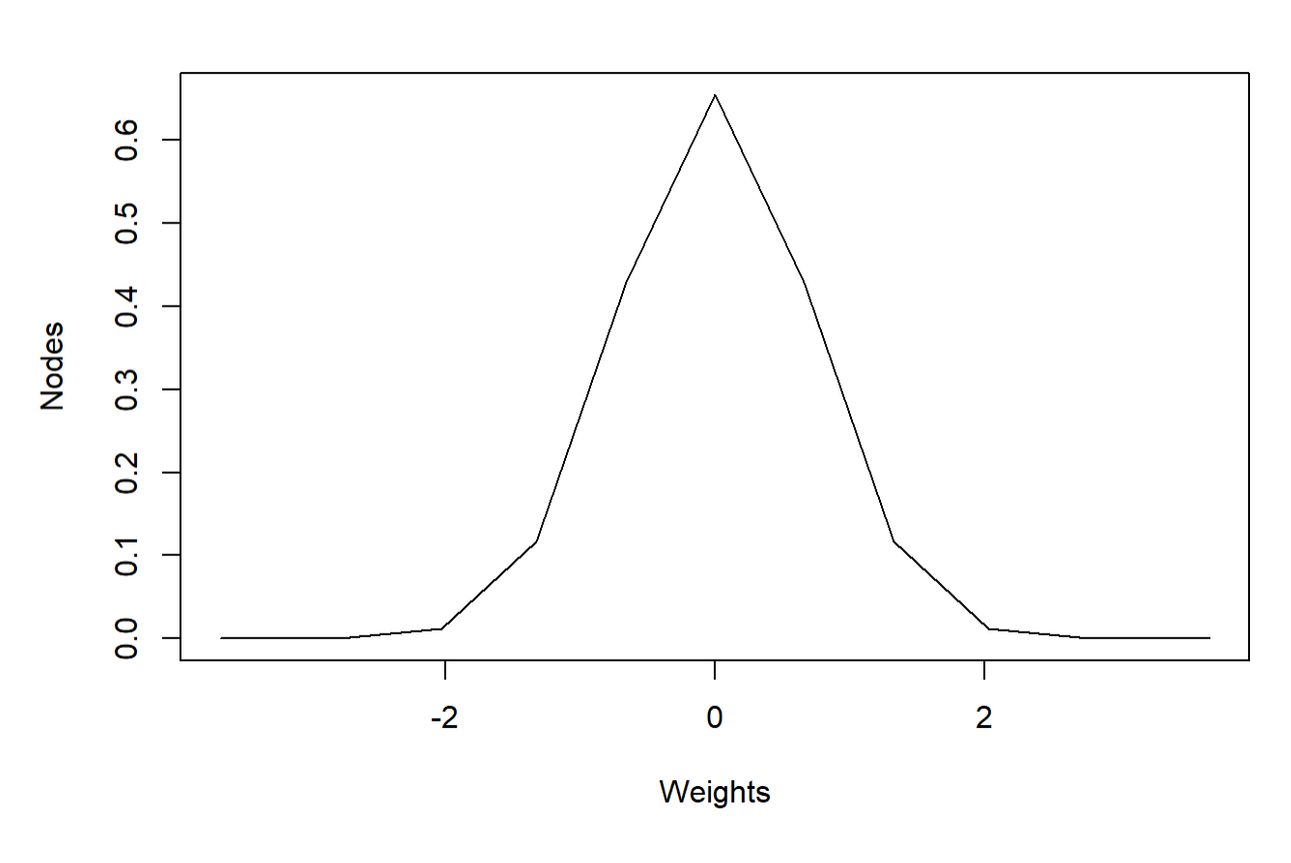
```
## [1] 0.9525818
```

```
pnorm(2)-pnorm(-2)
```

```
## [1] 0.9544997
```

When we increase the nodes the result gets closer to the true value of the normal distribution between -2 and 2. With 20 nodes we are correct within two decimals.

```
# Question 3

x <- c(-3.66847,-2.78329,-2.02595,-1.32656,-0.656810,0,0.65681,1.32656,2.02595,2.78329,3.66847)# nodes
A<- c( 0.00000143956,0.000346819, 0.0119114, 0.117228, 0.429360, 0.654759, 0.429360,0.117228, # weights
       0.0119114, 0.000346819, 0.00000143956)



plot(x,A,type = "l",xlab = "Weights",ylab = "Nodes") # plot with lines between the points
```




```
f2<- function(x)dt(x,df=3) # t-distribution with 3 degrees of freedom

# t-distribution with 3 degress of freedom

# now computing the function f(x)* = f(x)/e^(-x^2)

f1 <- function(x){
  (2/(pi*sqrt(3)*(1+x^(2)/3)^2))/exp(-x^2)
}


summa1<-0
for (i in 1:11){
  summa1<- summa1+A[i]*f1(x[i])

}

print(summa1)  # The result is 0.976... Pretty close to 1.
```

```
## [1] 0.9761679
```

```
# Doubble exponential
f2<- function(x){
  ((1/2)*(exp(-abs(x))))/(exp(-x^2))
}

summa2<-0
for (i in 1:11){
  summa2<- summa2+A[i]*f2(x[i])

}

print(summa2) # 1.02 close to 1 but a little bit to high
```

```
## [1] 1.021386
```

```
# Normal distribution


f3<- function(x){
  (dnorm(x,mean = 0, sd=1))/(exp(-x^2))
}

summa3<-0
for (i in 1:11){
  summa3<- summa3+A[i]*f3(x[i])

}

print(summa3) # With mean= 0 we get 0.99999
```

```
## [1] 0.9999976
```

```
# With mean= 1

f4<- function(x){
  (dnorm(x,mean = 1, sd=1))/(exp(-x^2))
}


summa4<-0
for (i in 1:11){
  summa4<- summa4+A[i]*f4(x[i])

}

print(summa4) # we get 0,9997
```

```
## [1] 0.9997529
```

```
# with mean =2

f5<- function(x){
  (dnorm(x,mean = 2, sd=1))/(exp(-x^2))
}


summa5<-0
for (i in 1:11){
  summa5<- summa5+A[i]*f5(x[i])

}

print(summa5)  # With mean= 2 we get 0,9911
```

```
## [1] 0.9911869
```

```
f6<- function(x){
  (dnorm(x,mean = 3, sd=1))/(exp(-x^2))
}


summa6<-0
for (i in 1:11){
  summa6<- summa6+A[i]*f6(x[i])

}

print(summa6) # with mean= 3 we get 0.90
```

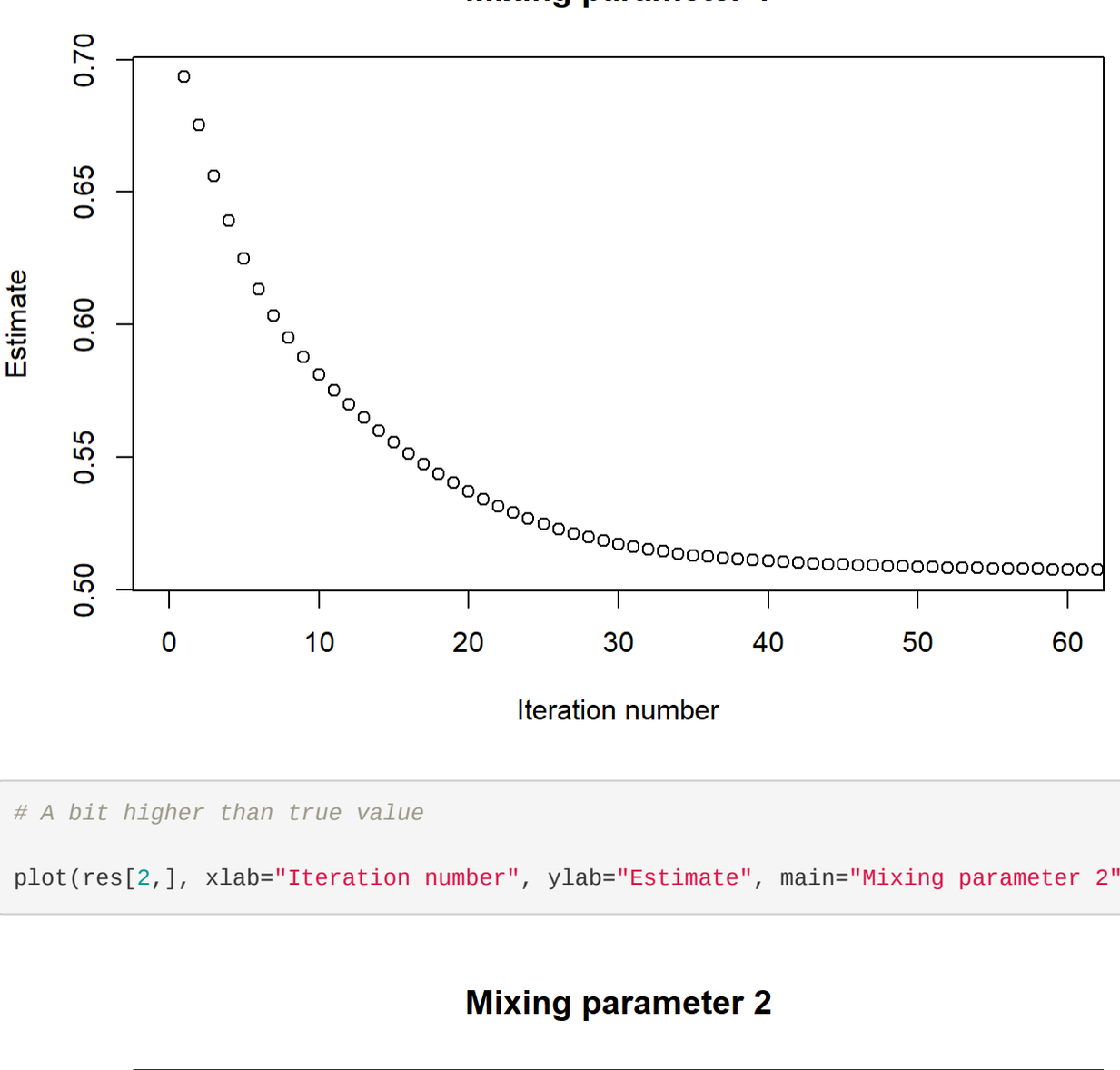
```
## [1] 0.9002919
```

```
# Seems like the gauss-hermite integration is less accurate when we increase the mean of the normal distribution.
```

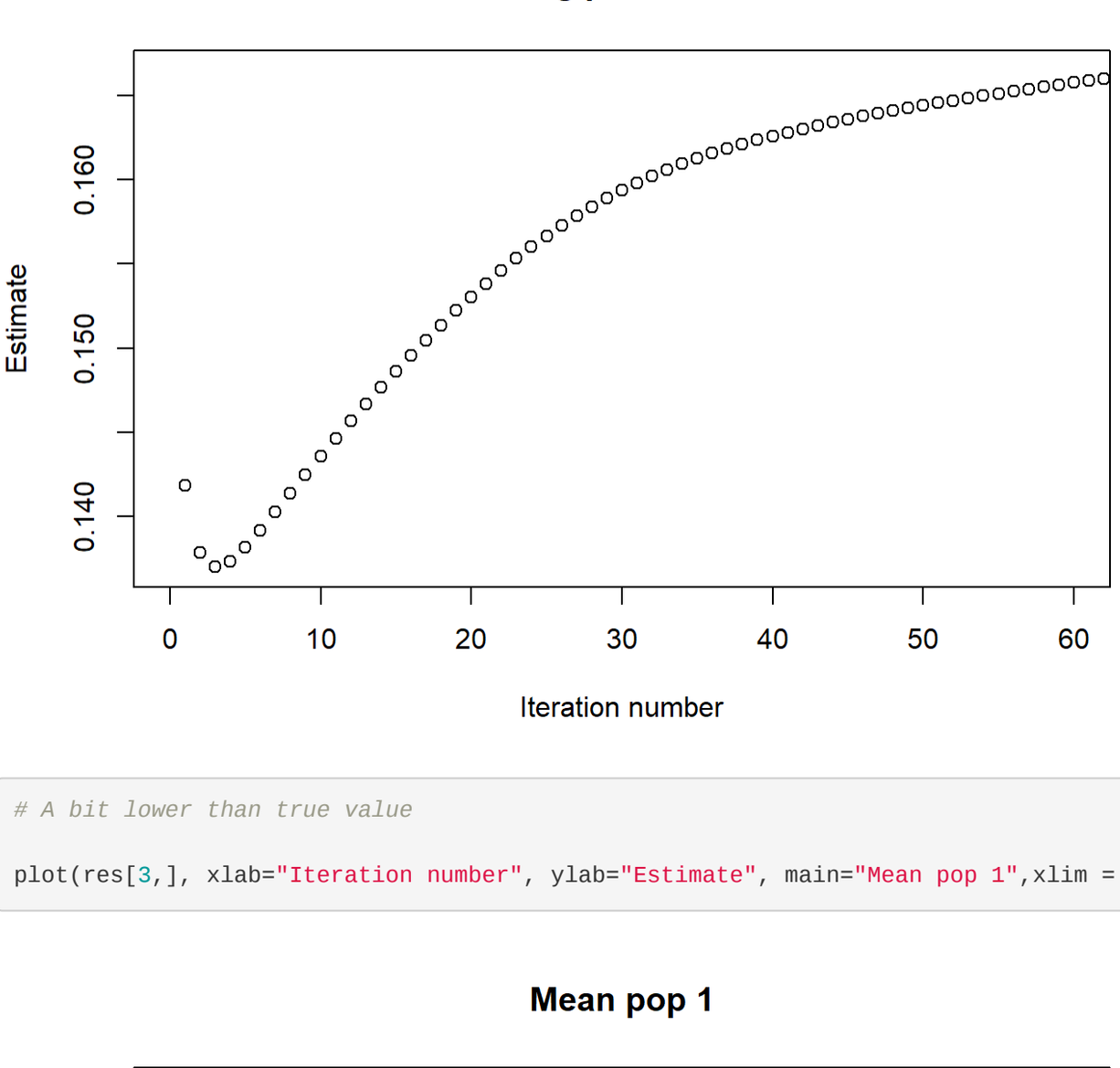
Question 4

```
# Question 4



emalg <- function(dat, start)
{
  n <- length(dat)
  pi <- rep(NA, n)
  pk <- rep(NA,n)                    # initialize vector for prob. to belong to group 1 # starting values
  p1 <- start[1] # starting value for mixing parameter  mu1 <- start[2] #
  p2 <- start[2]
  mu1 <- start[3]
  mu2 <- start[4]
  mu3 <- start[5]
  sigma1 <- start[6]
  sigma2 <- start[7]
  sigma3 <- start[8]                          # starting values for standard deviations (sd) sigma2 <- start[5]
  pv <- start # parameter vector
  monit <- NULL # initialize matrix to monitor iterations#initialize convergence criterion just not to stop direc
tly
  eps <- 0.001
  cc <- eps + 100
  Q <- -100000
  while (cc > eps) # A CONDITION was ADDED HERE AND ALSO SOME LINES AT OTHER PLACES IN THE CODE
  {
    #Save previous Q value
    Q1 <- Q
    ### E step ###
    for (j in 1:n){
      pi1   <- p1*dnorm(dat[j], mean=mu1, sd=sigma1)
      pi2   <- p2*dnorm(dat[j], mean=mu2, sd=sigma2)
      pi3   <- (1-p1-p2)*dnorm(dat[j],mean = mu3, sd=sigma3)
      pi[j] <- pi1/(pi1+pi2+pi3)
      pk[j] <- pi2/(pi1+pi2+pi3)
    }

    ### M step ###
    p1      <- mean(pi)
    p2      <- mean(pk)
    mu1    <- sum(pi*dat)/(p1*n)
    mu2    <- sum(pk*dat)/(p2*n)
    mu3    <- sum((1-pi-pk)*dat)/((1-p1-p2)*n)
    sigma1 <- sqrt(sum(pi*(dat-mu1)*(dat-mu1)/(p1*n)))
    sigma2 <- sqrt(sum(pk*(dat-mu2)*(dat-mu2)/(p2*n)))
    sigma3 <- sqrt(sum((1-pi-pk)*(dat-mu3)*(dat-mu3)/((1-p1-p2)*n)))
    ######
    pv <- c(p1,p2, mu1, mu2,mu3, sigma1, sigma2,sigma3)
    Q <- t(pi) %*% (log(p1)+log(dnorm(dat, mean=mu1, sd=sigma1))) + t(pk) %*% (log(p2)+log(dnorm(dat, mean=mu2, s
d=sigma2)))+t(1-pi-pk) %*% (log(1-p1-p2)+log(dnorm(dat, mean=mu3, sd=sigma3)))
    monit <- cbind(monit, c(pv, Q))
    cc <- Q-Q1
  }
  monit
}
```

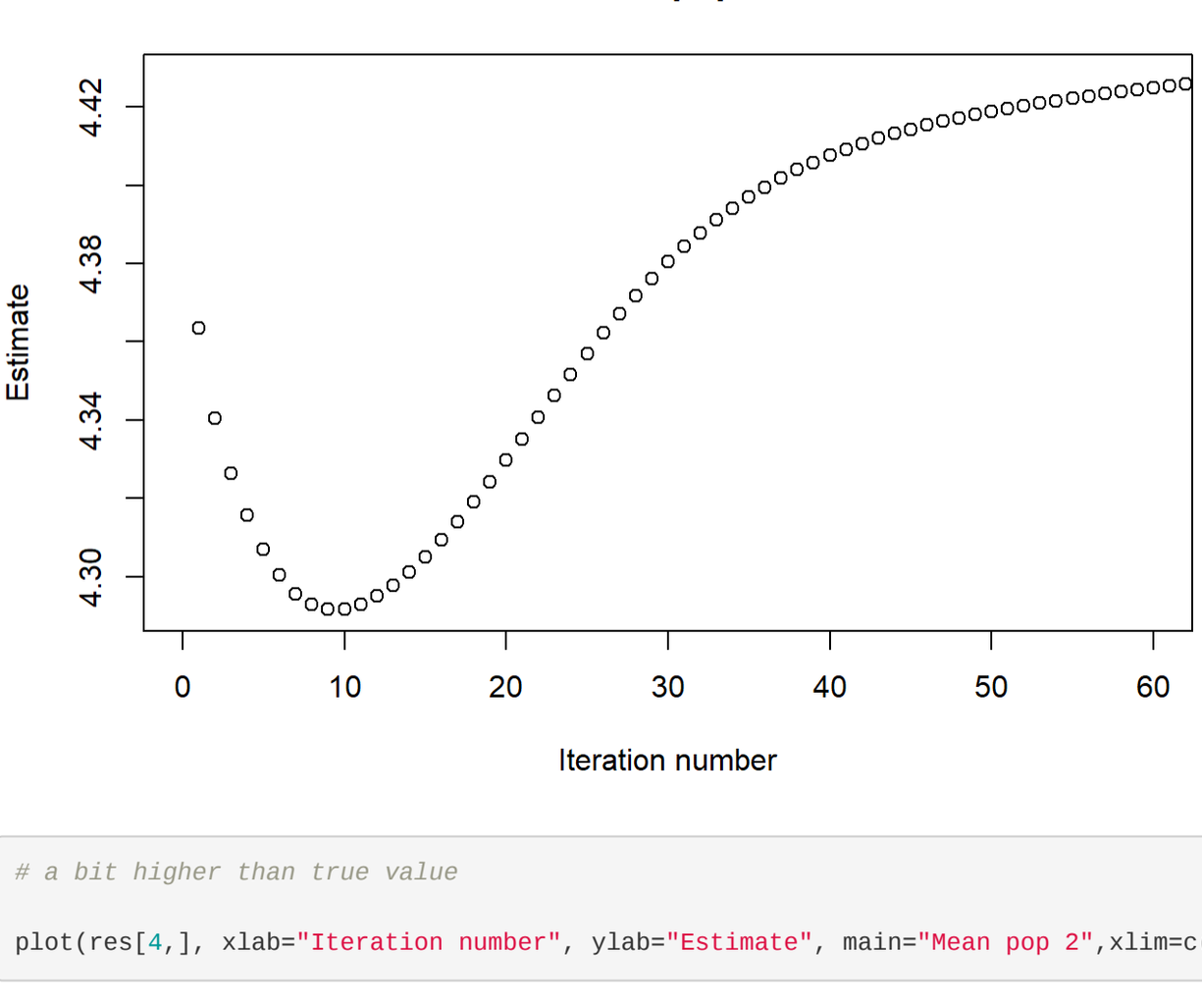
```
# 4b) Generate data

mu    <- c(2, 4, 5.5)
sigma <- c(1.2, 1, 0.8)
prob<- c(0.2, 0.3, 0.5)
n  <- 800


g <-sample(length(mu), n, replace=TRUE, p=prob)
x <-rnorm(n, mean = mu[g], sd= sigma[g])


# 4c)






res <- emalg(x,c(0.7, 0.1,5, 4, 5.5,1.2, 1, 0.8))


plot(res[1,], xlab="Iteration number", ylab="Estimate", main="Mixing parameter 1",xlim = c(0,60))
```



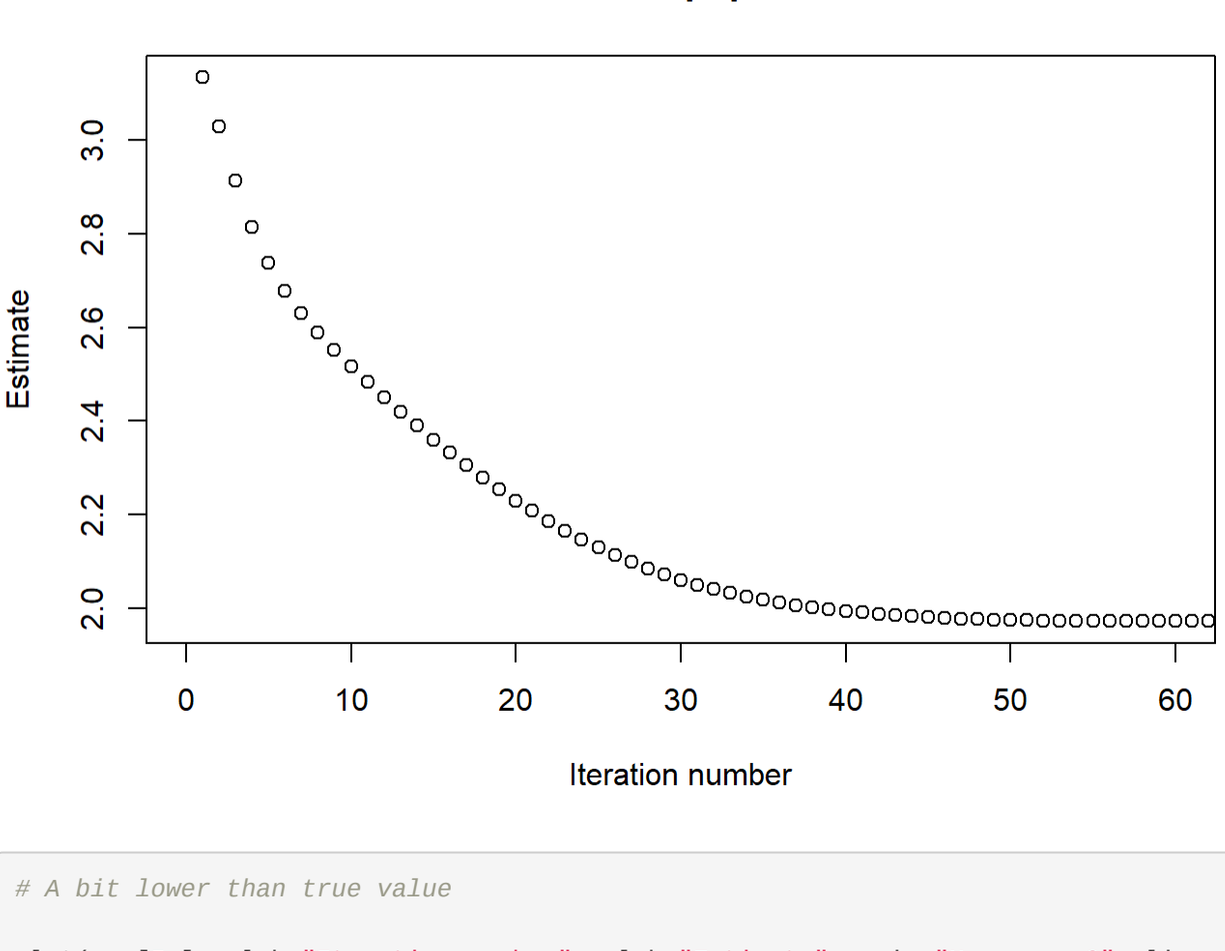

```
# A bit higher than true value

plot(res[2,], xlab="Iteration number", ylab="Estimate", main="Mixing parameter 2",xlim=c(0,60))
```



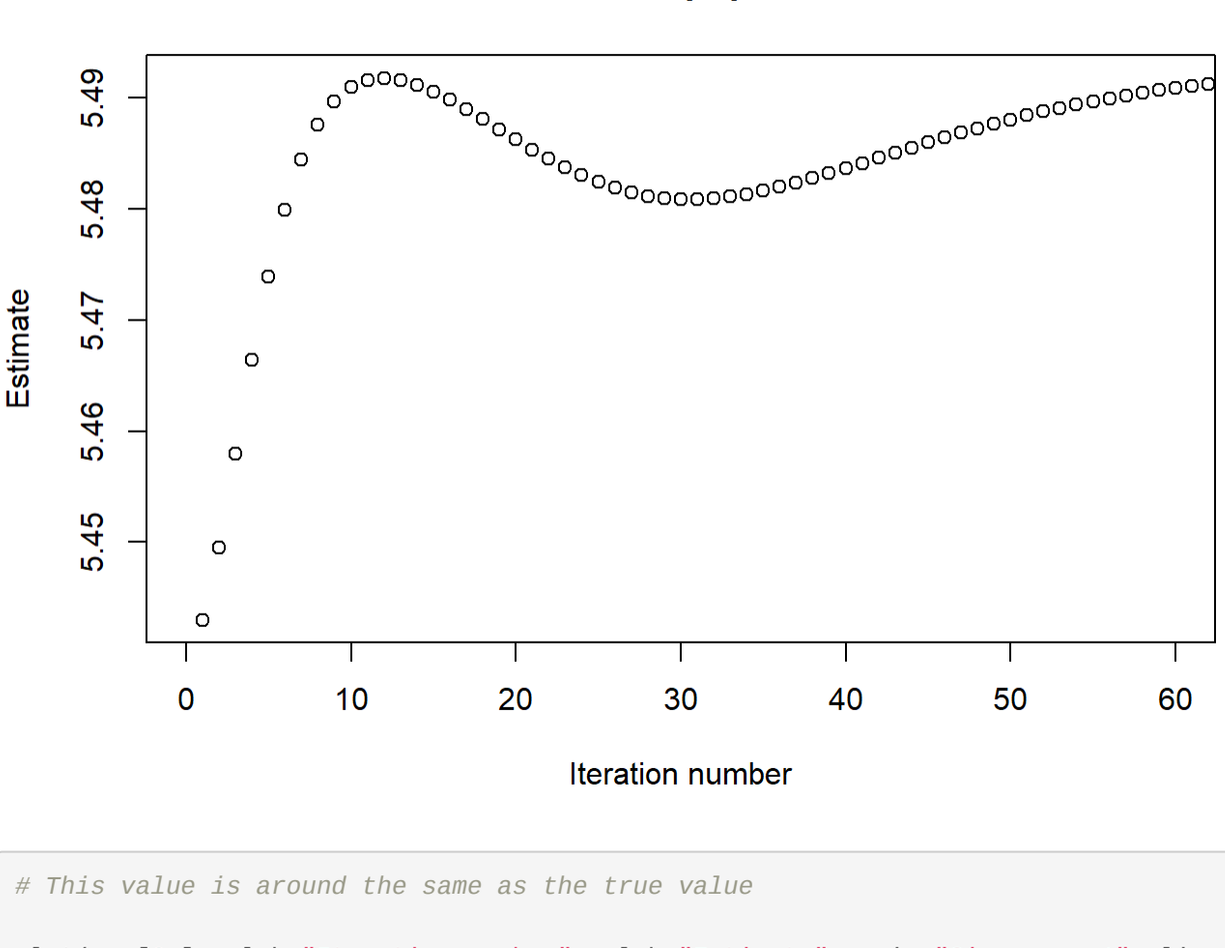

```
# A bit lower than true value

plot(res[3,], xlab="Iteration number", ylab="Estimate", main="Mean pop 1",xlim = c(0,60))
```



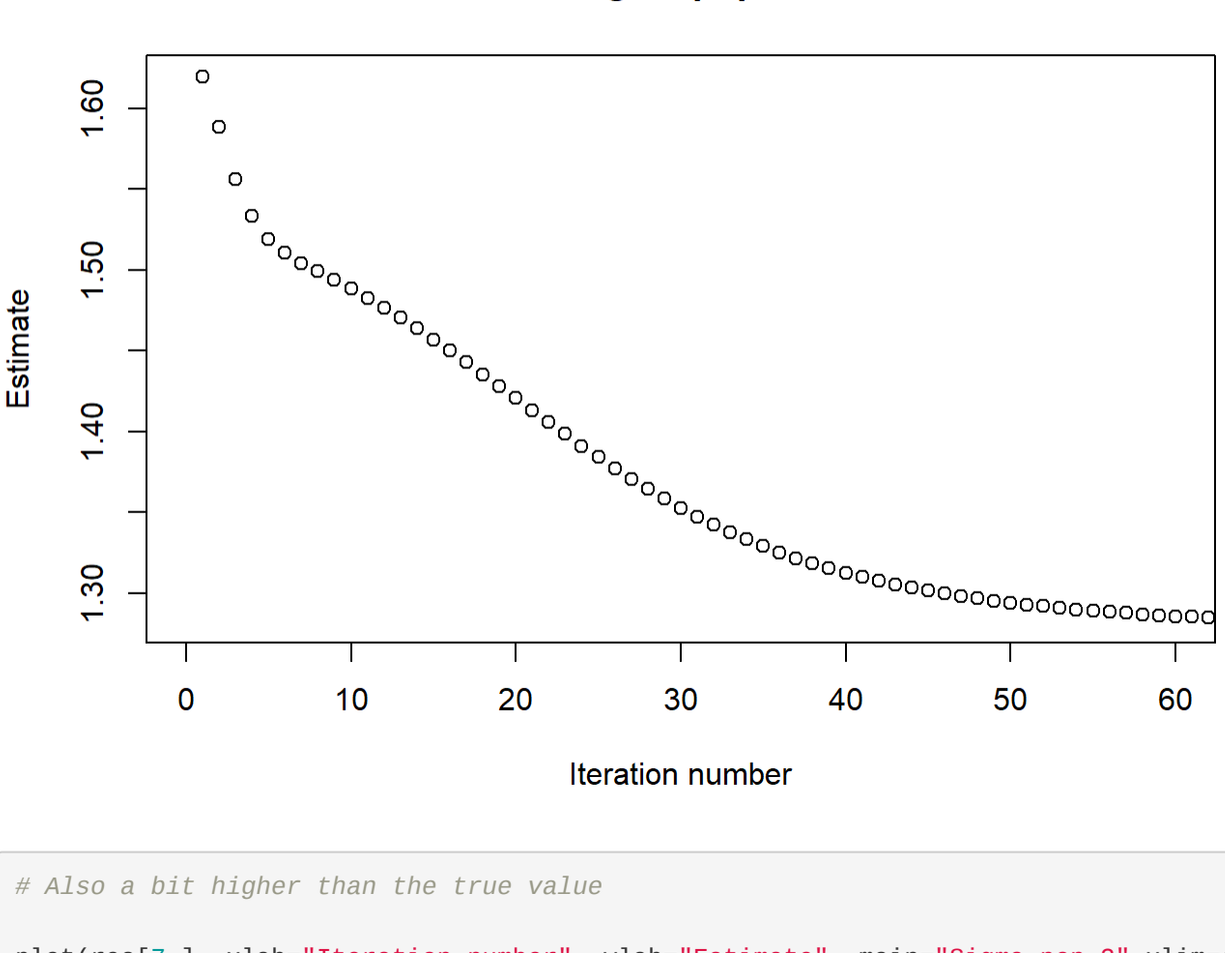

```
# a bit higher than true value

plot(res[4,], xlab="Iteration number", ylab="Estimate", main="Mean pop 2",xlim=c(0,60))
```



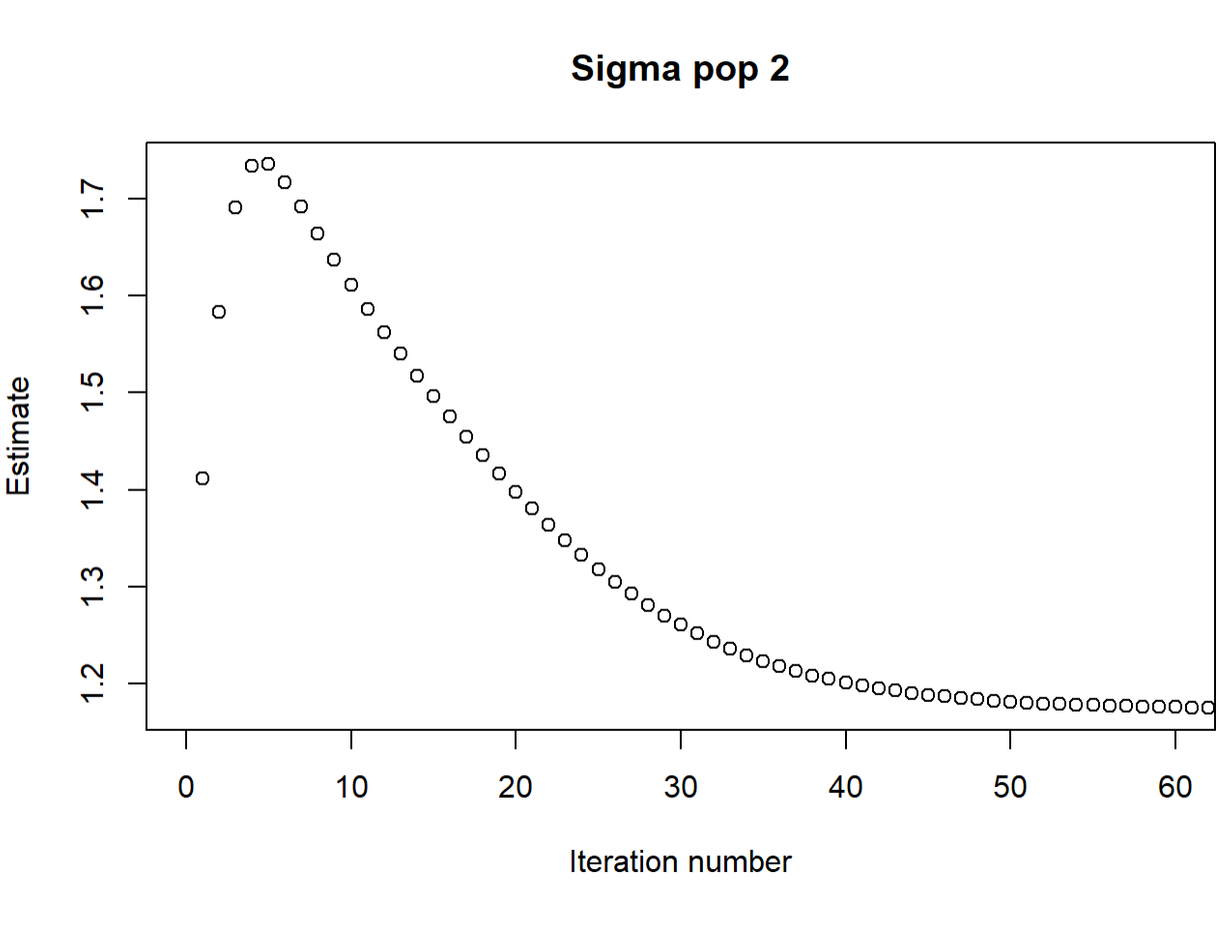

```
# A bit lower than true value

plot(res[5,], xlab="Iteration number", ylab="Estimate", main="Mean pop 3",xlim=c(0,60))
```



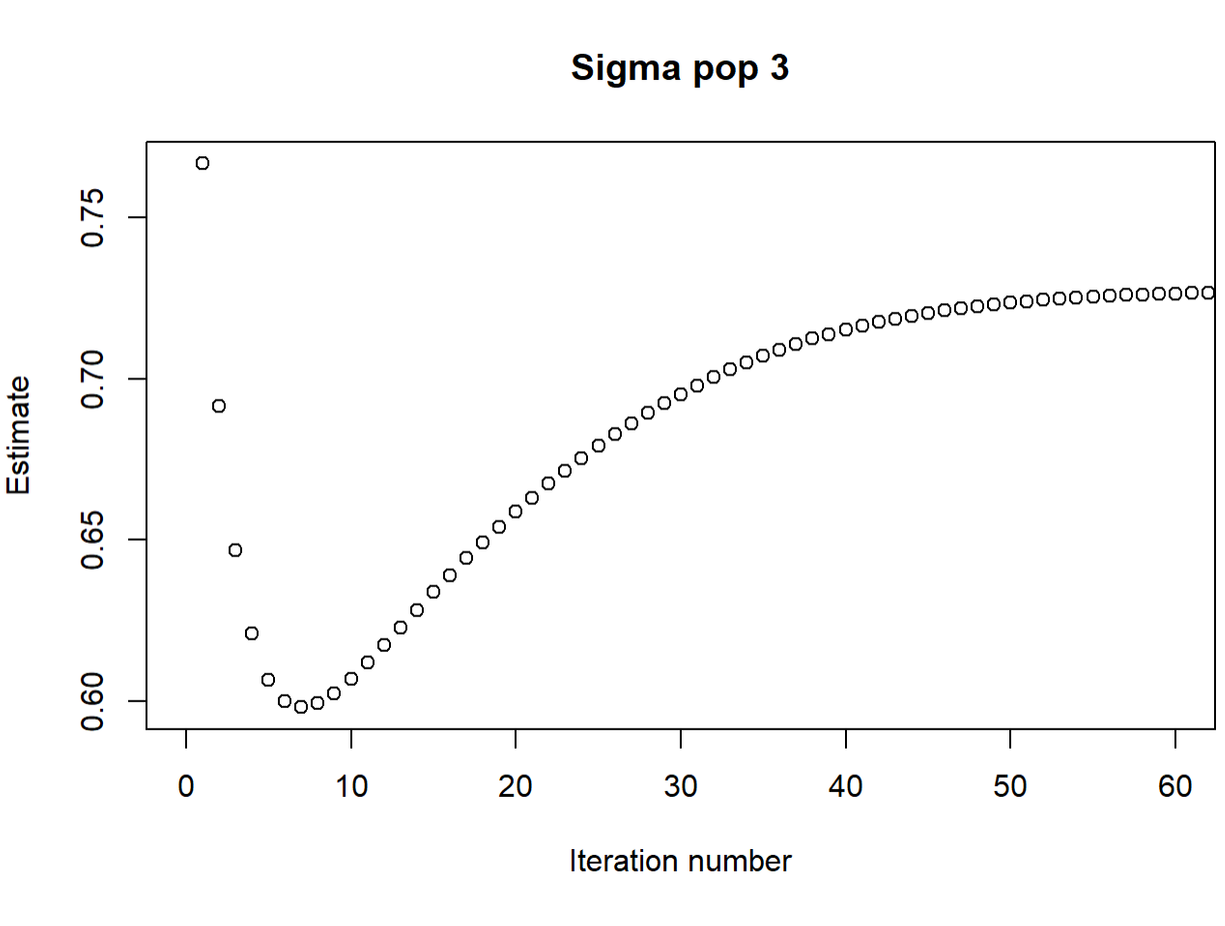

```
# This value is around the same as the true value

plot(res[6,], xlab="Iteration number", ylab="Estimate", main="Sigma pop 1",xlim=c(0,60))
```




```
# Also a bit higher than the true value

plot(res[7,], xlab="Iteration number", ylab="Estimate", main="Sigma pop 2",xlim=c(0,60))
```




```
# A bit higher than the original value

plot(res[8,], xlab="Iteration number", ylab="Estimate", main="Sigma pop 3",xlim=c(0,60))
```




```
# A bit lower than true value
```