Homework3

Jieqi Tu

10/7/2020

Question 1

```
# Define the function written by Dr. Demirtas

#############################################################
#EM algorithm for univariate normal MAR example
#Written by Hakan Demirtas
#############################################################
# Required arguments:
# y= incomplete univariate data, with missing values
# denoted by NA.
# start=starting values for parameters. This should be a
# list with two components: mu and sigma
#
# Optional arguments:
# maxits=maximum number of iterations
# eps=convergence criterion
#
# Output:
# This function returns a list with the following components:
# theta=final parameter estimates
# iter=how many iterations were performed
# cvgd=logical value indicating whether it converged or not.
# loglik= vector of length ``iter'' giving the observed-data
# log-likelihood at each iteration
#################################################################

univariate.normal.em.demirtas<-function(y, start, maxits=500, eps=0.000001){
  newmu<-start$mu ; newsigma2<-start$sigma2 ; n<-length(y)
  w<-!is.na(y) ; yobs<-y[w] ; nobs<-length(yobs) ; nmis<-n-nobs
  loglik<-numeric(maxits) ; iter<-0 ; cvgd<-F

  while ((iter<maxits)&(!cvgd)){
  iter<-iter+1 ; mu<-newmu ; sigma2<-newsigma2
  # EVALUATE OBSERVED-DATA-LIKELIHOOD
  ll<--(nobs/2)*log(sigma2)-1/(2*sigma2)*sum((yobs-mu)^2)
  loglik[iter]<-ll
  # E-STEP
  ET1<-sum(yobs)+nmis*mu
  ET2<-sum(yobs^2)+nmis*(sigma2+mu^2)
  # M-STEP
  newmu<-ET1/n
```

```
  newsigma2<-(ET2/n)-newmu^2
  # ASSESS CONVERGENCE
  cvgd<-(abs(newmu-mu)<=eps*abs(mu)) & (abs(newsigma2-sigma2)<=eps*abs(sigma2))}

  theta<-list(mu=newmu, sigma2=newsigma2)
  loglik<-loglik[1:iter]
  result<-list(theta=theta, iter=iter, cvgd=cvgd,loglik=loglik)
  result}
```

```
# Import data
data = c(50.1, 39.8, 46.4, NA, NA, 31.2, 50.0, 49.6, 48.7, 49.7, 53.4, 62.1, NA)

# Convergence criteria
eps = c(0.001, 0.0001, 0.00001, 0.000001)

# Calculate the sample mean and sample variance (only observed data)
mu = mean(data, na.rm = T); mu
```

```
## [1] 48.1
```

```
sigma2 = sum((data[!is.na(data)] - mu)^2)/length(data[!is.na(data)]); sigma2
```

```
## [1] 59.426
```

```
# Use the result to be the starting values
start1 = list(mu = 48.1, sigma2 = 59.426)

# Try resonable starting value
start2 = list(mu = 50, sigma2 = 60)

# Try some crazy starting values
start3 = list(mu = 0, sigma2 = 10)
start4 = list(mu = 1000, sigma2 = 1000)
start5 = list(mu = 10000, sigma2 = 21000)

# Create variables to store results
mu_result = numeric(20)
sigma2_result = numeric(20)
iteration = numeric(20)
log_like = numeric(20)

# Start iterations and try 4 different convergence criteria
for (i in 1:4) {
  # At each iteration, try different starting values
  EM1 = univariate.normal.em.demirtas(data, start1, eps = eps[i])
  mu_result[i] = EM1$theta$mu
  sigma2_result[i] = EM1$theta$sigma2
  iteration[i] = EM1$iter
  log_like[i] = EM1$loglik[EM1$iter]


  EM2 = univariate.normal.em.demirtas(data, start2, eps = eps[i])
  mu_result[i+4] = EM2$theta$mu
  sigma2_result[i+4] = EM2$theta$sigma2
  iteration[i+4] = EM2$iter
```

```
  log_like[i+4] = EM2$loglik[EM2$iter]


  EM3 = univariate.normal.em.demirtas(data, start3, eps = eps[i])
  mu_result[i+8] = EM3$theta$mu
  sigma2_result[i+8] = EM3$theta$sigma2
  iteration[i+8] = EM3$iter
  log_like[i+8] = EM3$loglik[EM3$iter]


  EM4 = univariate.normal.em.demirtas(data, start4, eps = eps[i])
  mu_result[i+12] = EM4$theta$mu
  sigma2_result[i+12] = EM4$theta$sigma2
  iteration[i+12] = EM4$iter
  log_like[i+12] = EM4$loglik[EM4$iter]


  EM5 = univariate.normal.em.demirtas(data, start5, eps = eps[i])
  mu_result[i+16] = EM5$theta$mu
  sigma2_result[i+16] = EM5$theta$sigma2
  iteration[i+16] = EM5$iter
  log_like[i+16] = EM5$loglik[EM5$iter]
}

# Sort the results
convergence_criteria = rep(eps, 5)
start_mu = c(rep(start1$mu, 4), rep(start2$mu, 4), rep(start3$mu, 4), rep(start4$mu, 4), rep(start5$mu,
start_sigma2 = c(rep(start1$sigma2, 4), rep(start2$sigma2, 4), rep(start3$sigma2, 4), rep(start4$sigma2
result_table = cbind(convergence_criteria, start_mu, start_sigma2, mu_result, sigma2_result, iteration,

result_table %>% knitr::kable()
```

convergence_criteria	start_mu	start_sigma2	mu_result	sigma2_result	iteration	log_like
1e-03	48.1	59.426	48.10000	59.42600	1	-25.42366
1e-04	48.1	59.426	48.10000	59.42600	1	-25.42366
1e-05	48.1	59.426	48.10000	59.42600	1	-25.42366
1e-06	48.1	59.426	48.10000	59.42600	1	-25.42366
1e-03	50.0	60.000	48.10539	59.43784	4	-25.42371
1e-04	50.0	60.000	48.10029	59.42663	6	-25.42366
1e-05	50.0	60.000	48.10007	59.42615	7	-25.42366
1e-06	50.0	60.000	48.10000	59.42601	9	-25.42366
1e-03	0.0	10.000	48.09991	59.43020	9	-25.42366
1e-04	0.0	10.000	48.09998	59.42697	10	-25.42366
1e-05	0.0	10.000	48.10000	59.42605	12	-25.42366
1e-06	0.0	10.000	48.10000	59.42601	13	-25.42366
1e-03	1000.0	1000.000	48.10001	59.43077	13	-25.42366
1e-04	1000.0	1000.000	48.10000	59.42710	14	-25.42366
1e-05	1000.0	1000.000	48.10000	59.42606	16	-25.42366
1e-06	1000.0	1000.000	48.10000	59.42601	17	-25.42366
1e-03	10000.0	21000.000	48.10000	59.43241	16	-25.42366
1e-04	10000.0	21000.000	48.10000	59.42748	17	-25.42366
1e-05	10000.0	21000.000	48.10000	59.42608	19	-25.42366
1e-06	10000.0	21000.000	48.10000	59.42600	21	-25.42366

From the result, we could see that, the estimated $\hat{\mu}$ are almost the same no matter what starting values and convergence criteria we choose. However, the value of $\hat{\sigma^2}$ are affected by the convergence criteria. When we choose the sample mean and sample variance of observed data as the starting values, the iteration is only 1. But other starting values lead to more iterations, especially higher precision of convergence criteria. If we set the convergence criteria more precise, and give enough iterations, we can reach very similar (even identical) parameter estimates through EM.

Then we can try another dataset with the same mean and variance and the same sample size. Let's see whether EM would give us the same estimates.

```
# Create a new dataset
set.seed(1029)
data_new = rnorm(13, mean = 48.1, sqrt(59.426))

# Randomly choose 3 data points to be missing
missing = sample(1:13, size = 3, replace = F); missing
```

```
## [1] 11  9  7
```

```
data_new[missing] = NA; data_new
```

```
##  [1] 37.36741 42.33496 48.58878 44.80481 52.97735 38.24142       NA 43.15490
##  [9]       NA 35.51063       NA 41.82736 37.33631
```

```
# Calculate the sample mean and sample variance from the new dataset
mu = mean(data_new, na.rm = T); mu
```

```
## [1] 42.21439
```

```
sigma2 = sum((data_new[!is.na(data_new)] - mu)^2)/length(data_new[!is.na(data_new)]); sigma2
```

```
## [1] 27.22471
```

```
# Use the new dataset to run EM method again
start_new = list(mu = mu, sigma2 = sigma2); start_new
```

```
## $mu
## [1] 42.21439
##
## $sigma2
## [1] 27.22471
```

```
start_new2 = list(mu = 0, sigma2 = 100); start_new2
```

```
## $mu
## [1] 0
##
## $sigma2
## [1] 100
```

```
univariate.normal.em.demirtas(data_new, start_new)
```

```
## $theta
## $theta$mu
## [1] 42.21439
##
## $theta$sigma2
## [1] 27.22471
##
##
```

```
## $iter
## [1] 1
##
## $cvgd
## [1] TRUE
##
## $loglik
## [1] -21.52063
```

```
univariate.normal.em.demirtas(data_new, start_new2)
```

```
## $theta
## $theta$mu
## [1] 42.21439
##
## $theta$sigma2
## [1] 27.22471
##
##
## $iter
## [1] 14
##
## $cvgd
## [1] TRUE
##
## $loglik
##  [1] -113.48983  -31.13003  -25.31123  -22.29828  -21.59610  -21.52544
##  [7]  -21.52089  -21.52064  -21.52063  -21.52063  -21.52063  -21.52063
## [13]  -21.52063  -21.52063
```

We could see that, if we try a sample that is derived from the population, the results could be different. So the EM method can provide estimates based on the sample instead of the true population. Also it is found that EM method will converge to the observed data sample mean and sample variance in univariate cases.

Let's try another dataset with higher portion of missing points. In this dataset, 6 points will be missing.

```
# Create a new dataset
set.seed(1029)
data_new = rnorm(13, mean = 48.1, sqrt(59.426))

# Randomly choose 3 data points to be missing
missing = sample(1:13, size = 6, replace = F); missing
```

```
## [1] 11  9  7 13  5  6
```

```
data_new[missing] = NA; data_new
```

```
##  [1] 37.36741 42.33496 48.58878 44.80481       NA       NA       NA 43.15490
##  [9]       NA 35.51063       NA 41.82736       NA
```

```
# Calculate the sample mean and sample variance from the new dataset
mu = mean(data_new, na.rm = T); mu
```

```
## [1] 41.94126
```

```
sigma2 = sum((data_new[!is.na(data_new)] - mu)^2)/length(data_new[!is.na(data_new)]); sigma2
```

```
## [1] 16.61478
```

```r
# Use the new dataset to run EM method again
start_new = list(mu = mu, sigma2 = sigma2); start_new
```

```
## $mu
## [1] 41.94126
##
## $sigma2
## [1] 16.61478
```

```r
start_new2 = list(mu = 0, sigma2 = 100); start_new2
```

```
## $mu
## [1] 0
##
## $sigma2
## [1] 100
```

```r
univariate.normal.em.demirtas(data_new, start_new)
```

```
## $theta
## $theta$mu
## [1] 41.94126
##
## $theta$sigma2
## [1] 16.61478
##
##
## $iter
## [1] 1
##
## $cvgd
## [1] TRUE
##
## $loglik
## [1] -13.33602
```

```r
univariate.normal.em.demirtas(data_new, start_new2)
```

```
## $theta
## $theta$mu
## [1] 41.94126
##
## $theta$sigma2
## [1] 16.61478
##
##
## $iter
## [1] 25
##
## $cvgd
## [1] TRUE
##
## $loglik
##  [1] -78.26705 -24.47890 -21.31414 -18.84094 -16.73016 -15.10718 -14.08306
##  [8] -13.58769 -13.40626 -13.35329 -13.33997 -13.33689 -13.33621 -13.33606
## [15] -13.33603 -13.33603 -13.33602 -13.33602 -13.33602 -13.33602 -13.33602
```

```
## [22] -13.33602 -13.33602 -13.33602 -13.33602
```

From the result, we could see that, if the missing portion becomes larger, we need more iterations to get the convergence. Here we can also see that EM method leads us to the sample mean and sample variance of observed data.

Let's see how stable the estimation of EM method gives us.

```
mu_result = vector()
sigma2_result = vector()
set.seed(1029)
for (i in 1:1000) {
  data_new = rnorm(13, 48.1, sqrt(59.426))
  mis = sample(1:13, 3, replace = F)
  data_new[mis] = NA; data_new

  mu = mean(data_new, na.rm = T)
  sigma2 = sum((data_new[!is.na(data_new)] - mu)^2)/length(data_new[!is.na(data_new)])
  start_new = list(mu = mu, sigma2 = sigma2)
  result = univariate.normal.em.demirtas(data_new, start_new)
  theta = result$theta
  mu_result[i] = theta$mu
  sigma2_result[i] = theta$sigma2
}

mean(mu_result)
```

```
## [1] 48.12695
```

```
sd(mu_result)
```

```
## [1] 2.384809
```

```
mean(sigma2_result)
```

```
## [1] 53.69598
```

```
sd(sigma2_result)
```

```
## [1] 25.7048
```

From the result, we could see that, in EM method, $\hat{\sigma^2}$ has much larger variance than $\hat{\mu}$. This means that the result for estimation of mean is more stable.

Question 2

Add two additional steps in the previous EM function and define the function to calculate elementwise convergence rate.

```
# Add two additional steps in the previous EM function
univariate.normal.em.demirtas_new = function(y, start, maxits=500, eps=0.000001){
  newmu<-start$mu ; newsigma2<-start$sigma2 ; n<-length(y)
  w<-!is.na(y) ; yobs<-y[w] ; nobs<-length(yobs) ; nmis<-n-nobs
  loglik<-numeric(maxits) ; iter<-0 ; cvgd<-F
  mut = c()
  sigma2t = c()
  cat(paste("Performing iterations of EM...","\n"))

  while ((iter<maxits)&(!cvgd)){
```

```
  iter<-iter+1 ; mu<-newmu ; sigma2<-newsigma2
  # EVALUATE OBSERVED-DATA-LIKELIHOOD
  ll<--(nobs/2)*log(sigma2)-1/(2*sigma2)*sum((yobs-mu)^2)
  loglik[iter]<-ll
  # E-STEP
  ET1<-sum(yobs)+nmis*mu
  ET2<-sum(yobs^2)+nmis*(sigma2+mu^2)
  # M-STEP
  newmu<-ET1/n
  newsigma2<-(ET2/n)-newmu^2

  # Record parameter theta_t
  mut[iter] = newmu
  sigma2t[iter] = newsigma2

  # ASSESS CONVERGENCE
  cvgd<-(abs(newmu-mu)<=eps*abs(mu)) & (abs(newsigma2-sigma2)<=eps*abs(sigma2))}

  # Adding two additional steps
  for (i in 1:2) {
    iter = iter + 1
    mu = newmu
    sigma2 = newsigma2
    # EVALUATE OBSERVED-DATA-LIKELIHOOD
    ll<--(nobs/2)*log(sigma2)-1/(2*sigma2)*sum((yobs-mu)^2)
    loglik[iter]<-ll
    # E-STEP
    ET1<-sum(yobs)+nmis*mu
    ET2<-sum(yobs^2)+nmis*(sigma2+mu^2)
    # M-STEP
    newmu<-ET1/n
    newsigma2<-(ET2/n)-newmu^2

    # Record parameter theta_t
    mut[iter] = newmu
    sigma2t[iter] = newsigma2
  }
  cat(paste("Done!","\n"))
  theta<-list(mu=mut, sigma2=sigma2t)
  loglik<-loglik[1:iter]
  result<-list(theta=theta, iter=iter, cvgd=cvgd,loglik=loglik)
  result}

# Calculate the elementwise rate of convergence
# Because we added two steps, t-1 will be the step that we get the convergence
rate_cal = function(theta) {
  rate = c()
  t = length(theta)
  for (i in 2:t-1) {
    rate[i] = (theta[i+1]-theta[i])/(theta[i]-theta[i-1])
  }
  return(rate)
}
```

Then we create the dataset.

```
# Create the dataset, normally distributed with mean 50 and sd 10
set.seed(1029)
data_2 = rnorm(100, 50, 5)
mis = sample(1:100, size = 90, replace = F)
data_2[mis] = NA

# Calculate the sample mean and sample variance
mu = mean(data_2, na.rm = T); mu
```

```
## [1] 50.01021
```

```
sigma2 = sum((data_2[!is.na(data_2)] - mu)^2)/length(data_2[!is.na(data_2)]); sigma2
```

```
## [1] 27.95854
```

```
# Set starting values
start_value = list(mu = mu, sigma2 = 10); start_value
```

```
## $mu
## [1] 50.01021
##
## $sigma2
## [1] 10
```

```
# Run EM function
result_question2 = univariate.normal.em.demirtas_new(data_2, start_value)
```

```
## Performing iterations of EM...
## Done!
```

```
mu_result = result_question2$theta$mu
sigma2_result = result_question2$theta$sigma2

# Calculate the convergence rate for mu
rate_cal(mu_result)
```

```
##   [1]  NA NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
##  [19] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
##  [37] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
##  [55] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
##  [73] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
##  [91] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
```

```
# Calculate the convergence rate for sigma2
rate_cal(sigma2_result)
```

```
##   [1]  NA 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [19] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [37] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [55] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [73] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [91] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
```

Since the $\hat{\mu}$ converges at the very beginning, the denominator becomes 0, the rate is not available in thie case. Also, the elementwise rate of convergence for $\hat{\sigma^2}$ is always 0.9, which is also the proportion of missing in the sample (90 out of 100 are missing). Let's try another pair of starting values that are not convergent at beginning.

```r
start_value = list(mu = 0, sigma2 = 0)

# Run EM function
result_question2 = univariate.normal.em.demirtas_new(data_2, start_value)
```

```
## Performing iterations of EM...
## Done!
```

```r
mu_result = result_question2$theta$mu
sigma2_result = result_question2$theta$sigma2

# Calculate the convergence rate for mu
rate_cal(mu_result)
```

```
##   [1]  NA 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [19] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [37] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [55] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [73] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
##  [91] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
## [109] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
## [127] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
## [145] 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9
```

```r
# Calculate the convergence rate for sigma2
rate_cal(sigma2_result)
```

```
##   [1]         NA  0.6865994  0.6482455  0.5854261  0.4647531  0.1414251
##   [7] -3.4446714  1.9216312  1.3306348  1.1621412  1.0827096  1.0366893
##  [13]  1.0067999  0.9859236  0.9705918  0.9589119  0.9497633  0.9424403
##  [19]  0.9364762  0.9315499  0.9274332  0.9239596  0.9210045  0.9184729
##  [25]  0.9162912  0.9144014  0.9127571  0.9113209  0.9100623  0.9089559
##  [31]  0.9079809  0.9071197  0.9063574  0.9056816  0.9050813  0.9045475
##  [37]  0.9040722  0.9036485  0.9032704  0.9029327  0.9026308  0.9023608
##  [43]  0.9021192  0.9019028  0.9017089  0.9015351  0.9013792  0.9012394
##  [49]  0.9011139  0.9010013  0.9009002  0.9008093  0.9007278  0.9006545
##  [55]  0.9005886  0.9005294  0.9004762  0.9004283  0.9003853  0.9003466
##  [61]  0.9003118  0.9002806  0.9002524  0.9002271  0.9002044  0.9001839
##  [67]  0.9001655  0.9001489  0.9001340  0.9001206  0.9001085  0.9000976
##  [73]  0.9000879  0.9000791  0.9000711  0.9000640  0.9000576  0.9000519
##  [79]  0.9000467  0.9000420  0.9000378  0.9000340  0.9000306  0.9000276
##  [85]  0.9000248  0.9000223  0.9000201  0.9000181  0.9000163  0.9000146
##  [91]  0.9000132  0.9000119  0.9000107  0.9000096  0.9000086  0.9000078
##  [97]  0.9000070  0.9000063  0.9000057  0.9000051  0.9000046  0.9000041
## [103]  0.9000037  0.9000033  0.9000030  0.9000027  0.9000024  0.9000022
## [109]  0.9000020  0.9000018  0.9000016  0.9000014  0.9000013  0.9000012
## [115]  0.9000011  0.9000009  0.9000009  0.9000008  0.9000007  0.9000006
## [121]  0.9000006  0.9000005  0.9000005  0.9000004  0.9000004  0.9000003
## [127]  0.9000003  0.9000003  0.9000002  0.9000002  0.9000002  0.9000002
## [133]  0.9000002  0.9000001  0.9000001  0.9000001  0.9000001  0.9000001
## [139]  0.9000001  0.9000001  0.9000001  0.9000001  0.9000000  0.9000001
## [145]  0.9000000  0.9000001  0.9000000  0.9000000  0.9000000  0.9000000
## [151]  0.9000000  0.9000000  0.9000000  0.9000000
```

In this case, $\hat{\mu}$ has the elementwise rate of convergence at 0.9. Because the update of new $\hat{\sigma^2}$ is dependent on $\hat{\mu}$, the rate for $\hat{\sigma^2}$ is not 0.9 at the early phase of iterations. However, when $\hat{\mu}$ became stable, the rate for $\hat{\sigma^2}$

is 0.9.

According to the paper written by Schafer in 1997, in most cases, the elementwise rate of convergence would estimate the largest eigenvalue of the ratio between the missing information to the complete data in the complete data information. Also this value could be affected by the starting values.

Question 3

Import the dataset.

```
# Import dataset
mixture = read.table(file = "./mixture.dat")
data3 = t(mixture)
data3 = as.vector(data3)
```

Then define the functions provided.

```
############################################################
# EM algorithm for two-part mixture of exponentials
# Written by Hakan Demirtas
# startval should be a list with three components:
# pi=mixing proportion
# lambda1=reciprocal mean of the first component
# lambda1=reciprocal mean of the second component
############################################################
# The function below allows us that pi can be fixed
# to a specific value to perform constrained estimation.
em.exponential.mixture<-function(y, startval, maxits=10000, eps=0.00001, pi.fix){
  n<-length(y) ; newtheta<-startval
  if(!missing(pi.fix)) newtheta$pi<-pi.fix
  iter<-0
  converged<-F
  loglik<-numeric(maxits)
  while((!converged)&(iter<maxits)){
    iter<-iter+1
    theta<-newtheta

    # DO E-STEP
    pix<-theta$pi
    lambda1<-theta$lambda1
    lambda2<-theta$lambda2
    num<-pix*lambda1*exp(-lambda1*y)
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    T1<-sum(delta) ; T2<-sum(delta*y) ; T3<-sum(y)
    loglik[iter]<-sum(log(denom))

    # DO M-STEP
    if (missing(pi.fix)) pix<-T1/n
    lambda1<-T1/T2
    lambda2<-(n-T1)/(T3-T2)
    newtheta<-list(pix=pix, lambda1=lambda1, lambda2=lambda2)

    # ASSESS CONVERGENCE
    old<-c(theta$pi, theta$lambda1, theta$lambda2)
    new<-c(newtheta$pi, newtheta$lambda1, newtheta$lambda2)
```

```
    converged<-all(abs(old-new)<=eps*abs(old))}
    loglik<-loglik[1:iter]
    list(theta=newtheta,iter=iter, converged=converged, loglik=loglik)}
```

Then we can try some starting values. Here we want to fix the $\hat{\pi}$ at 0.45.

```
start1 = list(pi = 0.3, lambda1 = 1, lambda2 = 0.5)
start2 = list(pi = 0.7, lambda1 = 0.5, lambda2 = 1)
em.exponential.mixture(data3, start1, eps = 0.0000001)
```

```
## $theta
## $theta$pix
## [1] 0.3331261
##
## $theta$lambda1
## [1] 1.935756
##
## $theta$lambda2
## [1] 0.5072857
##
##
## $iter
## [1] 197
##
## $converged
## [1] TRUE
##
## $loglik
##   [1] -139.4670 -138.3940 -138.0684 -137.7830 -137.5710 -137.4368 -137.3629
##   [8] -137.3263 -137.3095 -137.3021 -137.2989 -137.2975 -137.2968 -137.2964
##  [15] -137.2962 -137.2960 -137.2958 -137.2957 -137.2956 -137.2955 -137.2954
##  [22] -137.2953 -137.2952 -137.2951 -137.2951 -137.2950 -137.2950 -137.2949
##  [29] -137.2949 -137.2949 -137.2948 -137.2948 -137.2948 -137.2947 -137.2947
##  [36] -137.2947 -137.2947 -137.2947 -137.2947 -137.2946 -137.2946 -137.2946
##  [43] -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946
##  [50] -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946
##  [57] -137.2946 -137.2946 -137.2946 -137.2946 -137.2945 -137.2945 -137.2945
##  [64] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [71] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [78] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [85] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [92] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [99] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [106] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [113] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [120] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [127] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [134] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [141] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [148] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [155] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [162] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [169] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [176] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
```

```
## [183] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [190] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [197] -137.2945
```

```
em.exponential.mixture(data3, start2, eps = 0.0000001)
```

```
## $theta
## $theta$pix
## [1] 0.6668735
##
## $theta$lambda1
## [1] 0.5072856
##
## $theta$lambda2
## [1] 1.935755
##
##
## $iter
## [1] 187
##
## $converged
## [1] TRUE
##
## $loglik
##   [1] -139.4670 -138.3940 -138.0684 -137.7830 -137.5710 -137.4368 -137.3629
##   [8] -137.3263 -137.3095 -137.3021 -137.2989 -137.2975 -137.2968 -137.2964
##  [15] -137.2962 -137.2960 -137.2958 -137.2957 -137.2956 -137.2955 -137.2954
##  [22] -137.2953 -137.2952 -137.2951 -137.2951 -137.2950 -137.2950 -137.2949
##  [29] -137.2949 -137.2949 -137.2948 -137.2948 -137.2948 -137.2947 -137.2947
##  [36] -137.2947 -137.2947 -137.2947 -137.2947 -137.2946 -137.2946 -137.2946
##  [43] -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946
##  [50] -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946 -137.2946
##  [57] -137.2946 -137.2946 -137.2946 -137.2946 -137.2945 -137.2945 -137.2945
##  [64] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [71] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [78] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [85] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [92] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
##  [99] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [106] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [113] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [120] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [127] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [134] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [141] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [148] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [155] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [162] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [169] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [176] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
## [183] -137.2945 -137.2945 -137.2945 -137.2945 -137.2945
```

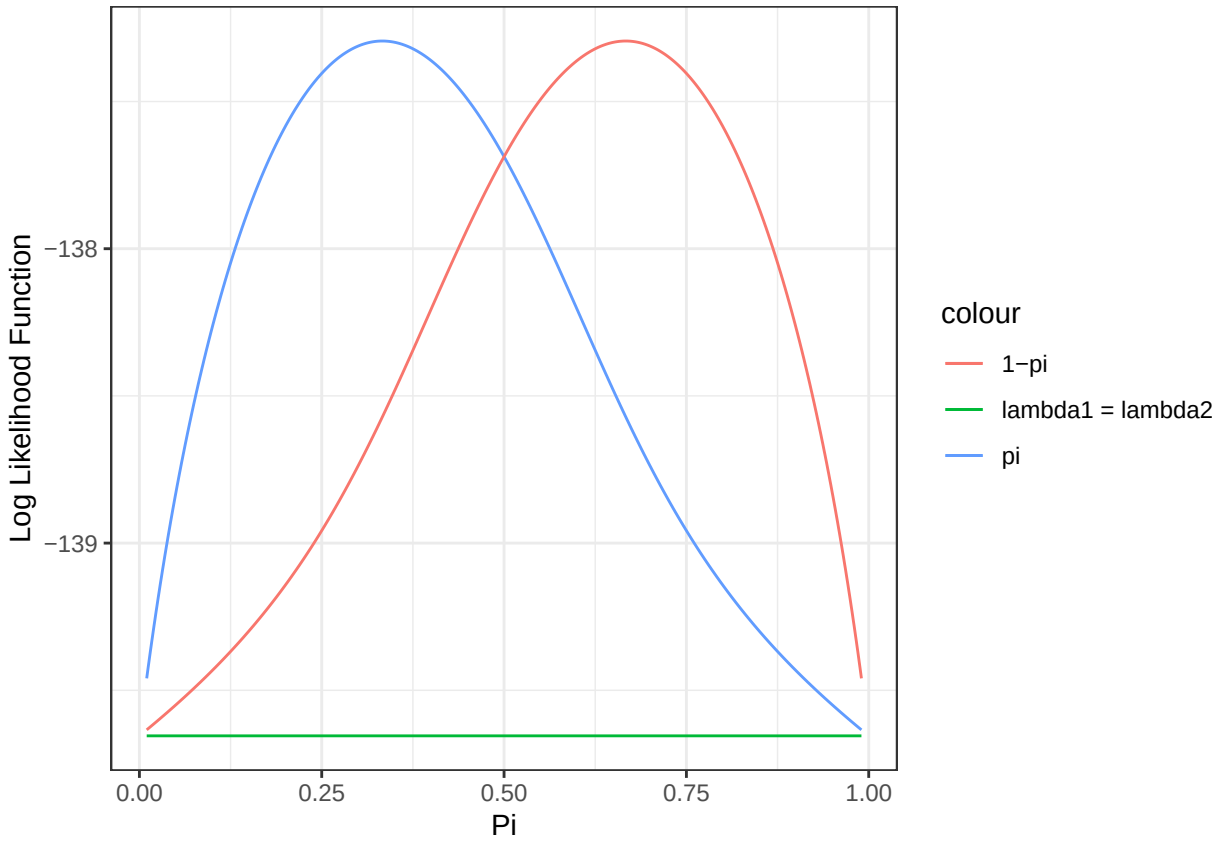
From the result, we could see that, if we interchange the π and $1-\pi$, and λ_1 and λ_2, the likelihood would be the same. Then we might not be able to know how the true distribution it is.

We can visualize the situation by plotting the log likelihood function.

```r
pi_range = seq(0.01, 0.99, 0.001)
start3 = list(pi = 0.5, lambda1 = 0.75, lambda2 = 0.75)

logL1 = numeric(length(pi_range))
logL2 = numeric(length(pi_range))
logL3 = numeric(length(pi_range))
for (i in 1:length(pi_range)) {
  result1 = em.exponential.mixture(data3, start1, eps = 0.000001, pi.fix = pi_range[i])
  result2 = em.exponential.mixture(data3, start2, eps = 0.000001, pi.fix = pi_range[i])
  result3 = em.exponential.mixture(data3, start3, eps = 0.000001, pi.fix = pi_range[i])
  conv_n1 = result1$iter
  conv_n2 = result2$iter
  conv_n3 = result3$iter
  logL1[i] = result1[[4]][conv_n1]
  logL2[i] = result2[[4]][conv_n2]
  logL3[i] = result3[[4]][conv_n3]
}

plot_table = cbind(pi_range, logL1, logL2, logL3) %>% as.data.frame()
plot_table %>%
  ggplot(aes(x = pi_range)) +
  geom_line(aes(y = logL1, colour = "pi")) +
  geom_line(aes(y = logL2, colour = "1-pi")) +
  geom_line(aes(y = logL3, colour = "lambda1 = lambda2")) +
  theme_bw() + labs(
    x = "Pi",
    y = "Log Likelihood Function"
  )
```



Here we could see that, $l(\pi, \lambda_1, \lambda_2|Y) = l(1-\pi, \lambda_2, \lambda_1|Y)$. However, they are not the same distribution. We might need other information to differ these two cases.

Then let's try to look at the confidence interval for π. Since π has the restriction that $0 < \pi < 1$, we should constrain π in a lower dimension parameter space H. To test whether π should be restricted or not, we would like to perform a likelihood ratio test:

$$H_0 : \pi \in \Theta_0$$

$$H_1 : \pi \in \Theta - \Theta_0$$

. The critical value for this test is:

$$2(l(\hat{\theta}, y_{obs}) - l(\tilde{\theta}, y_{obs})) \sim \chi^2_d$$

, since we only reduce one parameter in this case, so d = 1. If the critical value is larger than 3.84, then we will reject the null hypothesis, and conclude that π cannot be reduced. However, if the test is not rejected, we will be able to construct the confidence interval for π. The interval would be $l(\tilde{\theta}, y_{obs}) \leq l(\hat{\theta}, y_{obs}) + 1.92$

```
# Find the 95% confidence interval for pi using inverting likelihood ratio test
pi_CI = function(start, y) {
  pi.grid = seq(0.01, 0.99, 0.01)
  global = em.exponential.mixture(y, start, maxits = 10000, eps = 0.00001)
  n_iter_result = global$iter
  llmax = global[[4]][n_iter_result] # log likelihood for theta hat
  loglik = numeric(length(pi.grid)) # results vector for log likelihood of theta tilde

  # Calculate log likelihood function for each fixed pi value
  for (i in 1:length(pi.grid)) {
    temp = em.exponential.mixture(y, start, maxits = 10000, eps = 0.00001, pi.fix = pi.grid[i])
```

```
    n_iter = temp[[2]]
    loglik[i] = temp[[4]][n_iter]
  }

  # Calculate the 95% CI for pi
  range(pi.grid[loglik+1.92 >= llmax])
}

# CI for pi
start = list(pi = 0.5, lambda1 = 1, lambda2 = 0.5)
pi_CI(start, data3)
```

```
## [1] 0.03 0.82
```

```
# CI for 1-pi
start2 = list(pi = 0.5, lambda1 = 0.5, lambda2 = 1)
pi_CI(start2, data3)
```

```
## [1] 0.18 0.97
```

```
# CI for pi when lambda1 = lambda2
start3 = list(pi = 0.5, lambda1 = 1, lambda2 = 1)
pi_CI(start3, data3)
```

```
## [1] 0.01 0.99
```

From the 95% confidence interval, we could be able to do more concret inference. It is quite interesting that the sum of upper bound of π and lower bound of $1-\pi$ is 1. Similarly, the lower bound of π plus upper bound of $1-\pi$ is also 1. However, if we set $\lambda_1 = \lambda_2$, the density function becomes just one exponential distribution density function with only one parameter $\lambda = \lambda_1(\lambda_2)$.

Question 4

From the question, we know that:

$$f(y_i;\theta) = \pi f_1(y_i;\lambda_1) + (1-\pi) f_2(y_i;\lambda_2)$$

. Then we have:

$$\frac{\partial f}{\partial \pi} = f_1 - f_2$$
$$\frac{\partial f}{\partial \lambda_1} = \pi \frac{\partial f_1}{\partial \lambda_1}$$
$$\frac{\partial f}{\partial \lambda_2} = (1-\pi) \frac{\partial f_2}{\partial \lambda_2}$$

Here we also have

$$\pi^* = \frac{\pi f_1}{f}$$

. So we have:

$$1 - \pi^* = (1-\pi)\frac{\pi f_2}{f}$$

.

(a)

$$\begin{aligned}\frac{\partial \pi^*}{\partial \pi} &= \frac{f_1 f - \pi f_1 \frac{\partial f}{\partial \pi}}{f^2} \\ &= \frac{f_1}{f} - \pi \frac{f_1}{f} \frac{f_1 - f_2}{f} \\ &= \frac{\pi^*}{\pi} - \pi \frac{\pi^*}{\pi} (\frac{\pi^*}{\pi} - \frac{1-\pi^*}{1-\pi}) \\ &= \frac{\pi^*(1-\pi^*)}{\pi(1-\pi)}\end{aligned}$$

$$\begin{aligned}\frac{\partial \pi^*}{\partial \lambda_1} &= \frac{\frac{\pi \partial f_1}{\partial \lambda_1} f - \frac{\partial f}{\partial \lambda_1} f1}{f^2} \\ &= \frac{f_1}{f} - \pi \frac{f_1}{f} \frac{f_1 - f_2}{f} \\ &= \frac{\pi}{f}(1-\pi) \frac{f_2}{f} \frac{\partial f_1}{\partial \lambda_1} \\ &= \frac{\pi^*}{f_1}(1-\pi^*) \frac{\partial f_1}{\partial \lambda_1} \\ &= \pi^*(1-\pi^*) \frac{\partial}{\partial \lambda_1} log f_1\end{aligned}$$

$$\begin{aligned}\frac{\partial \pi^*}{\partial \lambda_2} &= -[\frac{\frac{(1-\pi)\partial f_2}{\partial \lambda_2} f - \frac{\partial f}{\partial \lambda_2}(1-\pi) f2}{f^2}] \\ &= -[(1-\pi) \frac{f - (1-\pi) f_2}{f^2} \frac{\partial f_2}{\partial \lambda_2}] \\ &= -[\frac{1-\pi}{f} \pi \frac{f_1}{f} \frac{\partial f_2}{\partial \lambda_2}] \\ &= -[\frac{1-\pi}{f} \pi^* \frac{\partial f_2}{\partial \lambda_2}] \\ &= -\pi^*(1-\pi^*) \frac{\partial}{\partial \lambda_2} log f_2\end{aligned}$$

(b)

$$\begin{aligned}\frac{\partial l_i}{\partial \pi} &= \frac{\partial log f}{\partial \pi} \\ &= \frac{1}{f} \frac{\partial f}{\partial \pi} \\ &= \frac{f_1 - f_2}{f} \\ &= \frac{\pi^*}{\pi} - \frac{1-\pi^*}{1-\pi} \\ &= \frac{\pi^* - \pi}{\pi(1-\pi)}\end{aligned}$$

$$\begin{aligned}\frac{\partial l_i}{\partial \lambda_1} &= \frac{\partial log f}{\partial \lambda_1} \\ &= \frac{1}{f}\frac{\partial f}{\partial \lambda_1} \\ &= \frac{1}{f}\pi\frac{\partial f_1}{\partial \lambda_1} \\ &= \frac{\pi^*}{f_1}\frac{\partial f_1}{\partial \lambda_1} \\ &= \pi^*\frac{\partial log f_1}{\partial \lambda_1}\end{aligned}$$

$$\begin{aligned}\frac{\partial l_i}{\partial \lambda_2} &= \frac{\partial log f}{\partial \lambda_2} \\ &= \frac{1}{f}(1-\pi)\frac{\partial f_2}{\partial \lambda_2} \\ &= \frac{1-\pi^*}{f_2}\frac{\partial f_2}{\partial \lambda_2} \\ &= (1-\pi^*)\frac{\partial log f_2}{\partial \lambda_2}\end{aligned}$$

(c)

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \pi^2} &= \frac{\partial \frac{\pi^*(1-\pi^*)}{\pi(1-\pi)}}{\partial \pi} \\ &= \frac{(\frac{\partial \pi^*}{\partial \pi}-1)\pi(1-\pi)-(\pi^*-\pi)(1-2\pi)}{\pi^2(1-\pi)^2} \\ &= \frac{-[\pi^2-2\pi\pi^*+(\pi^*)^2]}{\pi^2(1-\pi)^2} \\ &= -\frac{(\pi^*-\pi)^2}{\pi^2(1-\pi)^2}\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_1 \partial \pi} &= \frac{\partial l_i}{\partial \pi}\frac{\partial log f_1}{\lambda_1} \\ &= -\frac{\pi^*(1-pi^*)}{\pi(1-\pi)}\frac{\partial}{\partial \lambda_2}log f_2\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_1 \partial \pi} &= \frac{\partial \frac{\partial l_i}{\partial \pi}}{\partial \lambda_1} \\ &= \frac{\pi^*(1-pi^*)}{\pi(1-\pi)}\frac{\partial}{\partial \lambda_1}log f_1\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_2 \partial \pi} &= \frac{\partial \frac{\partial l_i}{\partial \pi}}{\partial \lambda_2} \\ &= -\frac{\pi^*(1-pi^*)}{\pi(1-\pi)}\frac{\partial}{\partial \lambda_2}log f_2\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_1^2} &= \frac{\partial \frac{\partial l_i}{\partial \lambda_1}}{\partial \lambda_1} \\ &= \frac{\partial^2 log f_1}{\partial \lambda_1^2}\pi^* + \frac{\partial \pi^*}{\partial \lambda_1}\frac{\partial log f_1}{\partial \lambda_1} \\ &= \pi^* \frac{\partial^2}{\partial \lambda_2^2} log f_1(y_i;\lambda_1) + \pi^*(1-\pi^*)[\frac{\partial}{\partial \lambda_1} log f_1(y_i;\lambda_1)]^2\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_2^2} &= \frac{\partial \frac{\partial l_i}{\partial \lambda_2}}{\partial \lambda_2} \\ &= \frac{\partial^2 log f_2}{\partial \lambda_2^2}(1-\pi^*) + \frac{\partial (1-\pi^*)}{\partial \lambda_2}\frac{\partial log f_2}{\partial \lambda_2} \\ &= (1-\pi^*)\frac{\partial^2}{\partial \lambda_2^2} log f_2(y_i;\lambda_2) + \pi^*(1-\pi^*)[\frac{\partial}{\partial \lambda_2} log f_2(y_i;\lambda_2)]^2\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 l_i}{\partial \lambda_1 \partial \lambda_2} &= \frac{\partial \frac{\partial l_i}{\partial \lambda_1}}{\partial \lambda_2} \\ &= \frac{\partial log f_1}{\partial \lambda_1}\frac{\partial \pi^*}{\partial \lambda_2} \\ &= -(1-\pi^*)\pi^*[\frac{\partial log f_1}{\partial \lambda_1}][\frac{\partial log f_2}{\partial \lambda_2}]\end{aligned}$$

Question 5

```
# Import dataset
data5 = read.table(file = "mixture10000.dat") %>% t() %>% as.vector()
```

(a)

Apply the EM algorithm to the dataset.

```
start1 = list(pi = 0.5, lambda1 = 0.5, lambda2 = 2)
start2 = list(pi = 0.5, lambda1 = 1, lambda2 = 0.5)

# Revise function so that the result won't show the log likelihood
em.exponential.mixture_new<-function(y, startval, maxits=10000, eps=0.00001, pi.fix){
  n<-length(y) ; newtheta<-startval
  if(!missing(pi.fix)) newtheta$pi<-pi.fix
  iter<-0
  converged<-F
  loglik<-numeric(maxits)
  while((!converged)&(iter<maxits)){
    iter<-iter+1
    theta<-newtheta

    # DO E-STEP
    pix<-theta$pi
    lambda1<-theta$lambda1
    lambda2<-theta$lambda2
    num<-pix*lambda1*exp(-lambda1*y)
```

```
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    T1<-sum(delta) ; T2<-sum(delta*y) ; T3<-sum(y)
    loglik[iter]<-sum(log(denom))

    # DO M-STEP
    if (missing(pi.fix)) pix<-T1/n
    lambda1<-T1/T2
    lambda2<-(n-T1)/(T3-T2)
    newtheta<-list(pix=pix, lambda1=lambda1, lambda2=lambda2)

    # ASSESS CONVERGENCE
    old<-c(theta$pi, theta$lambda1, theta$lambda2)
    new<-c(newtheta$pi, newtheta$lambda1, newtheta$lambda2)
    converged<-all(abs(old-new)<=eps*abs(old))}
    loglik<-loglik[1:iter]
    list(theta=newtheta,iter=iter, converged=converged)}

em.exponential.mixture_new(data5, start1)
```

```
## $theta
## $theta$pix
## [1] 0.4868499
##
## $theta$lambda1
## [1] 0.5008038
##
## $theta$lambda2
## [1] 0.9753358
##
##
## $iter
## [1] 2098
##
## $converged
## [1] TRUE
```

```
em.exponential.mixture_new(data5, start2)
```

```
## $theta
## $theta$pix
## [1] 0.5130051
##
## $theta$lambda1
## [1] 0.9754489
##
## $theta$lambda2
## [1] 0.500845
##
##
## $iter
## [1] 977
##
## $converged
```

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

As the result shows, although we have a much larger sample size in this data, the estimates are still confusing.

(b)

Calculate the first derivative of the log-likelihood function for this dataset.

```r
# Add codes to previous function to calculate first derivatives at mode
em.exponential.mixture_first_der <-function(y, startval, maxits=10000, eps=0.00001, pi.fix){
  n<-length(y) ; newtheta<-startval
  if(!missing(pi.fix)) newtheta$pi<-pi.fix
  iter<-0
  converged<-F
  loglik<-numeric(maxits)
  while((!converged)&(iter<maxits)){
    iter<-iter+1
    theta<-newtheta

    # DO E-STEP
    pix<-theta$pi
    lambda1<-theta$lambda1
    lambda2<-theta$lambda2
    num<-pix*lambda1*exp(-lambda1*y)
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    T1<-sum(delta) ; T2<-sum(delta*y) ; T3<-sum(y)
    loglik[iter]<-sum(log(denom))

    # DO M-STEP
    if (missing(pi.fix)) pix<-T1/n
    lambda1<-T1/T2
    lambda2<-(n-T1)/(T3-T2)
    num<-pix*lambda1*exp(-lambda1*y)
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    newtheta<-list(pix=pix, lambda1=lambda1, lambda2=lambda2)

    # ASSESS CONVERGENCE
    old<-c(theta$pi, theta$lambda1, theta$lambda2)
    new<-c(newtheta$pi, newtheta$lambda1, newtheta$lambda2)
    converged<-all(abs(old-new)<=eps*abs(old))}
    loglik<-loglik[1:iter]
    list(theta=newtheta,iter=iter, converged=converged)

    # Calculate first derivative
    lambda1 = newtheta$lambda1
    lambda2 = newtheta$lambda2
    pi = newtheta$pi
    pi.first_der = sum((delta-newtheta$pi)/(newtheta$pi*(1-newtheta$pi)))
    lambda1.first.der = sum(delta * ( 1/newtheta$lambda1 - y))
    lambda2.first.der = sum((1 - delta) * (1/lambda2 - y))

    list(theta=newtheta,iter=iter, converged=converged,
         first.der = c(pi = pi.first_der, lambda1 = lambda1.first.der, lambda2 = lambda2.first.der))
```

```
}
```

We first look at the situation when $\lambda_1 < \lambda_2$:

```
em.exponential.mixture_first_der(data5, start1)
```

```
## $theta
## $theta$pix
## [1] 0.4868499
##
## $theta$lambda1
## [1] 0.5008038
##
## $theta$lambda2
## [1] 0.9753358
##
##
## $iter
## [1] 2098
##
## $converged
## [1] TRUE
##
## $first.der
##          pi     lambda1     lambda2
## -0.19427094 -0.02673784 -0.02041318
```

Then we want to also look at the situation when $\lambda_1 > \lambda_2$:

```
em.exponential.mixture_first_der(data5, start2)
```

```
## $theta
## $theta$pix
## [1] 0.5130051
##
## $theta$lambda1
## [1] 0.9754489
##
## $theta$lambda2
## [1] 0.500845
##
##
## $iter
## [1] 977
##
## $converged
## [1] TRUE
##
## $first.der
##          pi     lambda1     lambda2
##  0.20473496 -0.02151008 -0.02817907
```

From the result we could see that, at mode, although the direvatives are not exactly zero, they are very small.

(c)

Calculate the 3×3 matrix of second derivatives at the mode and obtain asymptotic standard errors for each of the estimated parameters.

Firstly we need to add codes to the previous function to calculate second derivatives for parameters.

```
em.exponential.mixture_sec_der <-function(y, startval, maxits=10000, eps=0.00001, pi.fix){
  n<-length(y) ; newtheta<-startval
  if(!missing(pi.fix)) newtheta$pi<-pi.fix
  iter<-0
  converged<-F
  loglik<-numeric(maxits)

  # Create a matrix to store values of second derivatives
  sec_mat = diag(length(startval))
  while((!converged)&(iter<maxits)){
    iter<-iter+1
    theta<-newtheta

    # DO E-STEP
    pix<-theta$pi
    lambda1<-theta$lambda1
    lambda2<-theta$lambda2
    num<-pix*lambda1*exp(-lambda1*y)
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    T1<-sum(delta) ; T2<-sum(delta*y) ; T3<-sum(y)
    loglik[iter]<-sum(log(denom))

    # DO M-STEP
    if (missing(pi.fix)) pix<-T1/n
    lambda1<-T1/T2
    lambda2<-(n-T1)/(T3-T2)
    num<-pix*lambda1*exp(-lambda1*y)
    denom<-num+(1-pix)*lambda2*exp(-lambda2*y)
    delta<-num/denom # parameter for the Bernoulli distribution
    newtheta<-list(pix=pix, lambda1=lambda1, lambda2=lambda2)

    # ASSESS CONVERGENCE
    old<-c(theta$pi, theta$lambda1, theta$lambda2)
    new<-c(newtheta$pi, newtheta$lambda1, newtheta$lambda2)
    converged<-all(abs(old-new)<=eps*abs(old))}
    loglik<-loglik[1:iter]
    list(theta=newtheta,iter=iter, converged=converged)

    # Calculate first derivative
    lambda1 = newtheta$lambda1
    lambda2 = newtheta$lambda2
    pi = newtheta$pi
    pi.first_der = sum((delta-newtheta$pi)/(newtheta$pi*(1-newtheta$pi)))
    lambda1.first.der = sum(delta * ( 1/newtheta$lambda1 - y))
    lambda2.first.der = sum((1 - delta) * (1/lambda2 - y))

    # Calculate second derivative
    sec_mat[1,1] = sum(-(delta - pi)^2/(pi^2*(1-pi)^2))
    sec_mat[1,2] = sum( delta * (1 - delta)/(pi * (1 - pi)) * (1/lambda1 - y))
```

```
    sec_mat[1,3] = sum( -delta * (1 - delta)/(pi * (1 - pi)) * (1/lambda2 - y))
    sec_mat[2,2] = sum( delta * (-1/lambda1^2) + delta * (1 - delta) * (1/lambda1 - y)^2)
    sec_mat[3,3] = sum( (1-delta)* (-1/lambda2^2) + delta * (1 - delta)*(1/lambda2 - y)^2)
    sec_mat[2,3] = sum( -delta*(1 - delta) * (1/lambda1 - y)*(1/lambda2 - y))
    sec_mat[2,1] = sec_mat[1,2]
    sec_mat[3,1] = sec_mat[1,3]
    sec_mat[3,2] = sec_mat[2,3]

    # Calculate the standard errors
    se_pi = sqrt(-1/sec_mat[1,1])
    se_lambda1 = sqrt(-1/sec_mat[2,2])
    se_lambda2 = sqrt(-1/sec_mat[3,3])

    # Calculate 95% CI for ML estimates
    CI_pi = pi + c(-1.96*se_pi, 1.96*se_pi)
    CI_lambda1 = lambda1 + c(-1.96*se_lambda1, 1.96*se_lambda1)
    CI_lambda2 = lambda2 + c(-1.96*se_lambda2, 1.96*se_lambda2)

    list(theta=newtheta,iter=iter, converged=converged,
         first.der = c(pi = pi.first_der, lambda1 = lambda1.first.der, lambda2 = lambda2.first.der),
         sec.der_matrix = sec_mat,
         SE = c(pi = se_pi, lambda1 = se_lambda1, lambda2 = se_lambda2),
         CI = list(pi = CI_pi, lambda1 = CI_lambda1, lambda2 = CI_lambda2))
}
```

Now we check the result for $\lambda_1 < \lambda_2$:

```
start1 = list(pi = 0.5, lambda1 = 0.5, lambda2 = 2)
start2 = list(pi = 0.5, lambda1 = 2, lambda2 = 0.5)
em.exponential.mixture_sec_der(data5, start1)
```

```
## $theta
## $theta$pix
## [1] 0.4868499
##
## $theta$lambda1
## [1] 0.5008038
##
## $theta$lambda2
## [1] 0.9753358
##
##
## $iter
## [1] 2098
##
## $converged
## [1] TRUE
##
## $first.der
##          pi     lambda1     lambda2
## -0.19427094 -0.02673784 -0.02041318
##
## $sec.der_matrix
##           [,1]       [,2]      [,3]
```

```
## [1,] -3725.364   6486.476  2324.365
## [2,]  6486.476 -14944.909 -2892.112
## [3,]  2324.365  -2892.112 -1938.110
##
## $SE
##          pi     lambda1     lambda2
## 0.016383838 0.008180001 0.022714899
##
## $CI
## $CI$pi
## [1] 0.4547376 0.5189622
##
## $CI$lambda1
## [1] 0.4847710 0.5168366
##
## $CI$lambda2
## [1] 0.9308146 1.0198570
```

Then we check the result for $\lambda_1 > \lambda_2$:

```
em.exponential.mixture_sec_der(data5, start2)
```

```
## $theta
## $theta$pix
## [1] 0.5130055
##
## $theta$lambda1
## [1] 0.9754486
##
## $theta$lambda2
## [1] 0.5008449
##
##
## $iter
## [1] 2069
##
## $converged
## [1] TRUE
##
## $first.der
##          pi     lambda1     lambda2
##  0.20470964 -0.02150742 -0.02817558
##
## $sec.der_matrix
##           [,1]      [,2]       [,3]
## [1,] -3725.244 -2323.364  -6487.082
## [2,] -2323.364 -1936.561  -2891.157
## [3,] -6487.082 -2891.157 -14948.340
##
## $SE
##          pi     lambda1     lambda2
## 0.016384101 0.022723983 0.008179062
##
## $CI
## $CI$pi
```

```
## [1] 0.4808926 0.5451183
##
## $CI$lambda1
## [1] 0.9309096 1.0199876
##
## $CI$lambda2
## [1] 0.4848139 0.5168759
```

From the result, we could see that, using the information matrix, we could get a more concret and a relatively narrow confidence interval. The CI's are symmetric when we switch the starting values.

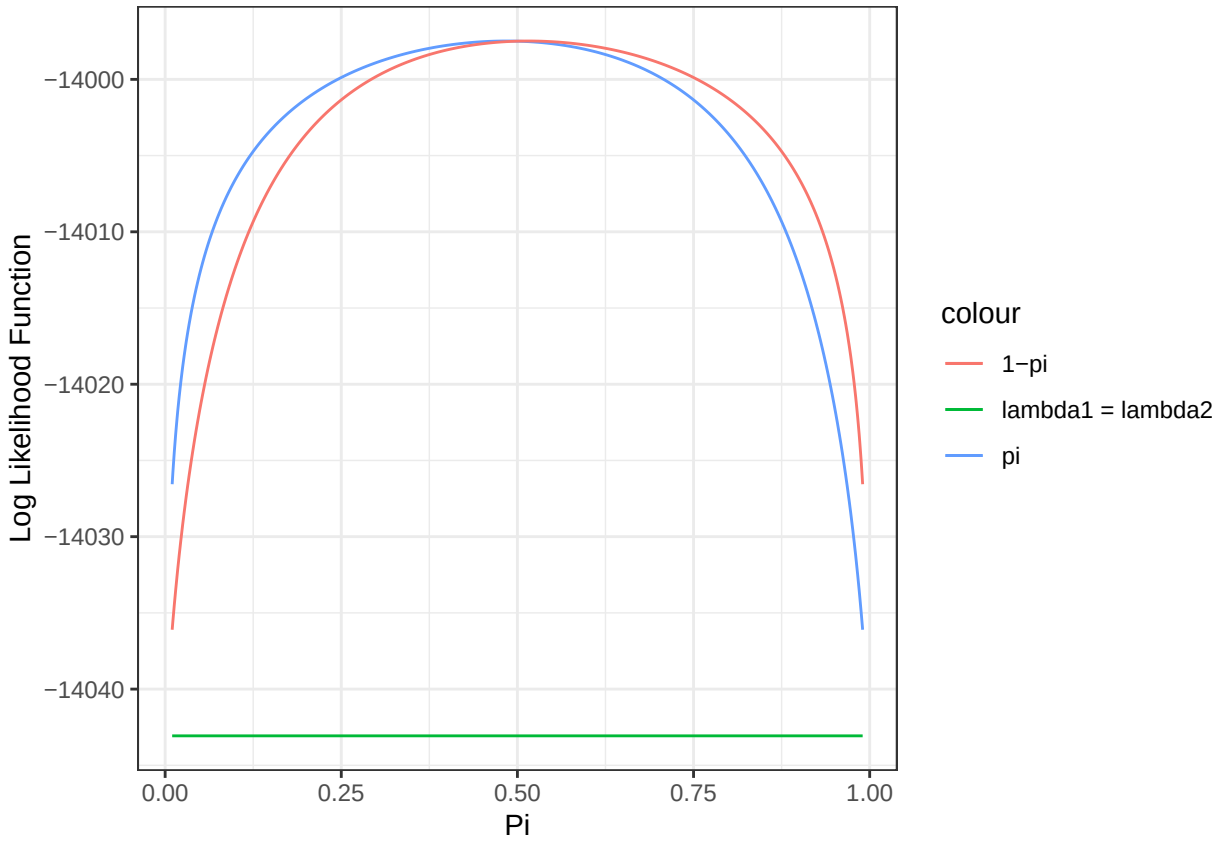
(d)

First let's plot the log likelihood function

```
pi_range = seq(0.01, 0.99, 0.001)
start3 = list(pi = 0.5, lambda1 = 0.75, lambda2 = 0.75)

logL1 = numeric(length(pi_range))
logL2 = numeric(length(pi_range))
logL3 = numeric(length(pi_range))
for (i in 1:length(pi_range)) {
  result1 = em.exponential.mixture(data5, start1, eps = 0.000001, pi.fix = pi_range[i])
  result2 = em.exponential.mixture(data5, start2, eps = 0.000001, pi.fix = pi_range[i])
  result3 = em.exponential.mixture(data5, start3, eps = 0.000001, pi.fix = pi_range[i])
  conv_n1 = result1$iter
  conv_n2 = result2$iter
  conv_n3 = result3$iter
  logL1[i] = result1[[4]][conv_n1]
  logL2[i] = result2[[4]][conv_n2]
  logL3[i] = result3[[4]][conv_n3]
}

plot_table = cbind(pi_range, logL1, logL2, logL3) %>% as.data.frame()
plot_table %>%
  ggplot(aes(x = pi_range)) +
  geom_line(aes(y = logL1, colour = "pi")) +
  geom_line(aes(y = logL2, colour = "1-pi")) +
  geom_line(aes(y = logL3, colour = "lambda1 = lambda2")) +
  theme_bw() + labs(
    x = "Pi",
    y = "Log Likelihood Function"
  )
```



From the plot we could see that, the log likelihood function are quite similar. The mode for π is approximately 0.5 for both situations.

Then let's try to look at the confidence interval for π. Since π has the restriction that $0 < \pi < 1$, we should constrain π in a lower dimension parameter space H. To test whether π should be restricted or not, we would like to perform a likelihood ratio test:

$$H_0 : \pi \in \Theta_0$$

$$H_1 : \pi \in \Theta - \Theta_0$$

. The critical value for this test is:

$$2(l(\hat{\theta}, y_{obs}) - l(\tilde{\theta}, y_{obs})) \sim \chi^2_d$$

, since we only reduce one parameter in this case, so d = 1. If the critical value is larger than 3.84, then we will reject the null hypothesis, and conclude that π cannot be reduced. However, if the test is not rejected, we will be able to construct the confidence interval for π. The interval would be $l(\tilde{\theta}, y_{obs}) \leq l(\hat{\theta}, y_{obs}) + 1.92$

```
# Find the 95% confidence interval for pi using inverting likelihood ratio test
# CI for pi
start = list(pi = 0.5, lambda1 = 1, lambda2 = 0.5)
pi_CI(start, data5)
```

```
## [1] 0.32 0.72
```

```
# CI for 1-pi
start2 = list(pi = 0.5, lambda1 = 0.5, lambda2 = 1)
pi_CI(start2, data5)
```

```
## [1] 0.28 0.68
```

```r
# CI for pi when lambda1 = lambda2
start3 = list(pi = 0.5, lambda1 = 1, lambda2 = 1)
pi_CI(start3, data5)
```

```
## [1] 0.01 0.99
```

From the result, we could see that, with larger sample size, inverting likelihood ratio test can narrow the CI for MLE. However, the ranges are still larger than using information matrix directly.

Question 6

Firstly we want to run the EM algorithm to estimate $\theta_1, \theta_2, \theta_3, \theta_4$. So here we need to define the EM function for this question.

```r
# Define the function for question 6
EM_q6 = function(start, x, maxits = 500, eps = 0.000001, indep = F, homo = F) {

  newtheta11 = start$theta11
  newtheta12 = start$theta12
  newtheta21 = start$theta21
  newtheta22 = start$theta22

  x11 = x[1,1]; x12 = x[1,2]; x21 = x[2,1]; x22 = x[2,2]
  x1B = x[1,3]; x2B = x[2,3]; xC1 = x[3,1]; xC2 = x[3,2]

  n = sum(x) - x[3,3]

  iter = 0
  loglik = numeric(maxits)
  cvgd = F

  while ((iter<maxits)&(!cvgd)) {

    iter = iter+1
    theta11 = newtheta11
    theta12 = newtheta12
    theta21 = newtheta21
    theta22 = newtheta22

    theta_1 = theta11 + theta21
    theta_2 = theta12 + theta22
    theta1_ = theta11 + theta12
    theta2_ = theta21 + theta22

    # Evalueate ovserved data likelihood
    la = x11*log(theta11) + x12*log(theta12) + x21*log(theta21) + x22*log(theta22)
    lb = x1B*log(theta1_) + x2B*log(theta2_)
    lc = xC1*log(theta_1) + xC2*log(theta_2)
    ll = la + lb + lc
    loglik[iter] = ll

    # E-step
    ET11 = x11 + x1B*(theta11/theta1_) + xC1*(theta11/theta_1)
    ET12 = x12 + x1B*(theta12/theta1_) + xC2*(theta12/theta_2)
    ET21 = x21 + x2B*(theta21/theta2_) + xC1*(theta21/theta_1)
```

```
    ET22 = x22 + x2B*(theta22/theta2_) + xC2*(theta22/theta_2)

    # M-step
    if(!indep & !homo) {
      newtheta11 = (ET11)/n
      newtheta12 = (ET12)/n
      newtheta21 = (ET21)/n
      newtheta22 = (ET22)/n
    }

    if(indep) {
      newtheta11 = (ET11+ET12)*(ET11+ET21)/n^2
      newtheta12 = (ET11+ET12)*(ET12+ET22)/n^2
      newtheta21 = (ET21+ET22)*(ET11+ET21)/n^2
      newtheta22 = (ET21+ET22)*(ET12+ET22)/n^2
    }

    if(homo) {
      newtheta11 = (ET11)/n
      newtheta12 = (ET12+ET21)/(2*n)
      newtheta21 = (ET21+ET12)/(2*n)
      newtheta22 = (ET22)/n
    }

    # Assess convergence
    cvgd = (abs(newtheta11-theta11)) <= eps*abs(theta11) &
           (abs(newtheta12-theta12)) <= eps*abs(theta12) &
           (abs(newtheta21-theta21)) <= eps*abs(theta21) &
           (abs(newtheta22-theta22)) <= eps*abs(theta22)
  }

  theta = list(theta11 = newtheta11, theta12 = newtheta12,
               theta21 = newtheta21, theta22 = newtheta22)
  loglik = loglik[1:iter]
  result = list(theta = theta, iter = iter, cvgd = cvgd, loglik = loglik)

  return(result)
}

# Import data
x = matrix(c(392, 55, 33,
             76, 38, 9,
             31, 7, 115), byrow = T, ncol = 3)
```

Set the starting values of theta to be 0.25 and look at the result.

```
theta = list(theta11 = 0.25, theta12 = 0.25, theta21 = 0.25, theta22 = 0.25)
EM_q6(theta, x)
```

```
## $theta
## $theta$theta11
## [1] 0.6971233
##
## $theta$theta12
```

```
## [1] 0.09863044
##
## $theta$theta21
## [1] 0.135783
##
## $theta$theta22
## [1] 0.06846318
##
##
## $iter
## [1] 8
##
## $cvgd
## [1] TRUE
##
## $loglik
## [1] -833.1629 -564.4080 -562.5178 -562.5036 -562.5034 -562.5034 -562.5034
## [8] -562.5034
```

Then we want to test the independence and homogeneity assumptions through the likelihood ratio test. If the columns and rows are independent, we will have:

$$\tilde{\theta_{ij}} = \frac{x_{i+}x_{+j}}{n^2} = \frac{(x_{i1} + x_{i2})(x_{1j} + x_{2j})}{n^2}$$

Let's see the result for independent data.

```
EM_q6(theta, x, indep = T)
```

```
## $theta
## $theta$theta11
## [1] 0.6631284
##
## $theta$theta12
## [1] 0.1328915
##
## $theta$theta21
## [1] 0.1699267
##
## $theta$theta22
## [1] 0.03405344
##
##
## $iter
## [1] 7
##
## $cvgd
## [1] TRUE
##
## $loglik
## [1] -833.1629 -576.7041 -575.2007 -575.1943 -575.1943 -575.1943 -575.1943
```

Here we state the hypothesis:

$$H_0 : \theta_{ij} \in \Theta_0$$

$$H_1 : \theta_{ij} \in \Theta - \Theta_0$$

$$-2[l(\hat{\theta}|X) - l(\tilde{\theta}|X)] = 2 * (-562.5034 + 575.1943) = 25.3818 > 3.84$$

Hence we reject the null hypothesis at 0.05 significant level, and conclude that rows and columns are not independent.

Check homogeneity in this case. Here we state the hypothesis:

$$H_0 : \theta_{12} = \theta_{21}$$

$$H_1 : \theta_{12} \neq \theta_{21}$$

Let's get the log likelihood under homogeneous assumption:

```
EM_q6(theta, x, homo = T)
```

```
## $theta
## $theta$theta11
## [1] 0.6970112
##
## $theta$theta12
## [1] 0.1172513
##
## $theta$theta21
## [1] 0.1172513
##
## $theta$theta22
## [1] 0.06848612
##
##
## $iter
## [1] 8
##
## $cvgd
## [1] TRUE
##
## $loglik
## [1] -833.1629 -566.0587 -564.2647 -564.2518 -564.2516 -564.2516 -564.2516
## [8] -564.2516
```

```
1-pchisq(3.4964, 1)
```

```
## [1] 0.06150239
```

The critical value would be:

$$-2[l(\hat{\theta}|X) - l(\tilde{\theta}|X)] = 2 * (-562.5034 + 564.2516) = 3.4964 < 3.84$$

The p-value of this critical value is 0.06. So here we do not have enough evidence to reject null and conclude that the assumption of homogeneity holds.