Applied Data Mining: Homework 2

Anshul Doshi

Anshul Doshi (ad3222)
STAT UN3106 Section 001
Assignment: 2

Problem 1: The Exponential Random Variable

i.) The likelihood function for an exponential:

$$\mathcal{L}(\lambda|x_1, x_2...x_n) = f(x_1|\lambda)f(x_2|\lambda)...f(x_n|\lambda)$$
$$\mathcal{L}(\lambda|x_1, x_2...x_n) = (\lambda e^{-\lambda x_1})(\lambda e^{-\lambda x_2})...(\lambda e^{-\lambda x_n})$$
$$\mathcal{L}(\lambda|x_1, x_2...x_n) = (\lambda^n)e^{-\lambda \sum_i^n x_i}$$

ii.) The log-likelihood function for an exponential:

$$log(\mathcal{L}(\lambda|x_1, x_2...x_n)) = (\lambda^n)e^{-\lambda \sum_i^n x_i}$$
$$\ell(\lambda|x_1, x_2...x_n) = log(\lambda^n) + log(e^{-\lambda \sum_i^n x_i})$$
$$\ell(\lambda|x_1, x_2...x_n) = nlog(\lambda) - \lambda \sum_i^n x_i$$

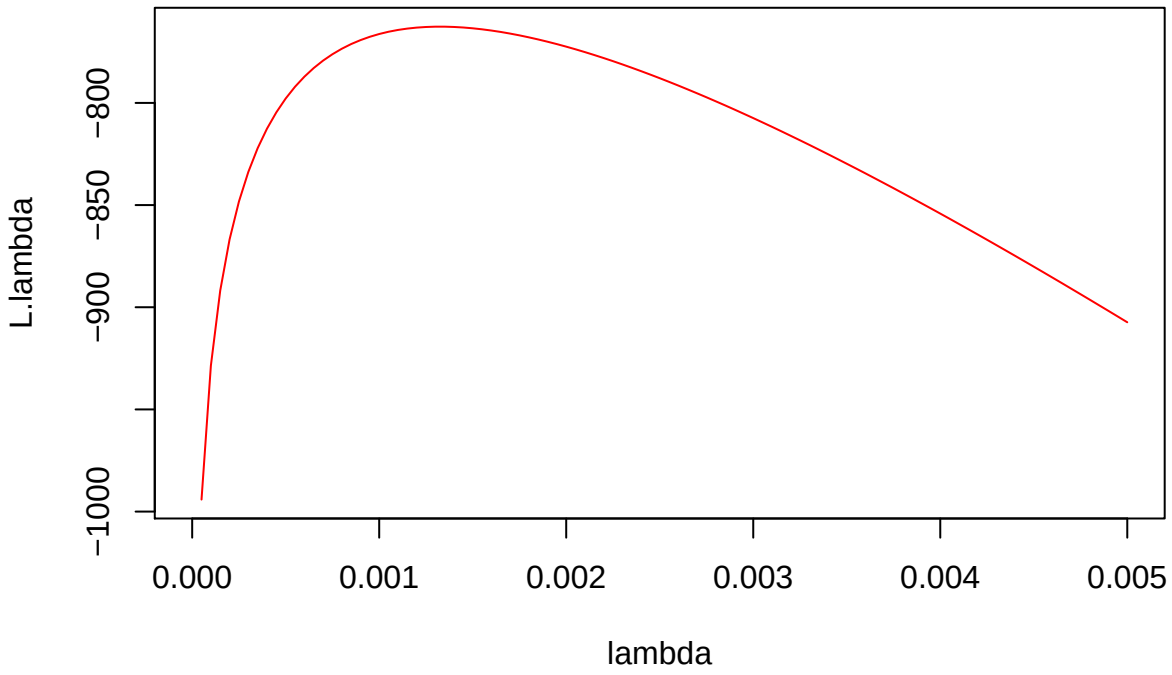
iii.) *lightbulbs.txt* and plotting log likelihood

```
#setwd("~/Desktop/Spring2017/Applied Data Mining/Homeworks/Homework2")
lightbulb <- read.delim("lightbulbs.txt", header=TRUE, sep="\n")
summary(lightbulb)
```

```
##        x
##  Min.   :   3.0
##  1st Qu.: 256.2
##  Median : 618.4
##  Mean   : 755.0
##  3rd Qu.: 969.6
##  Max.   :3646.7
```

```
#Define log-likelihood
lightbulb.likelihood <- function(lambda,data) {
  n = nrow(data)
  return(n*log(lambda) - lambda*sum(data))
}

# plot log-likelihood
lambda <- seq(from=0, to=.005,by=.005 / nrow(lightbulb))
L.lambda <- sapply(lambda, lightbulb.likelihood, data=lightbulb)
plot(lambda, L.lambda, type="l", col="red")
```



iv.) Find the MLE using nlm

```
#Define negative log-likelihood
neg.lightbulb.ll <- function(lambda, data) {
  n = nrow(data)
  return(lambda*sum(data) - n*log(lambda))
}

suppressWarnings(nlm(neg.lightbulb.ll,.005,data=lightbulb))
```

```
## $minimum
## [1] 762.6784
##
## $estimate
## [1] 0.001323916
##
## $gradient
## [1] 0.0155286
##
## $code
## [1] 2
##
## $iterations
## [1] 10
```

v.) Compare result to analytic MLE solution

Maximize log-likelihood by setting deriative to 0:

$$\ell(\lambda|x_1, x_2...x_n) = nlog(\lambda) - \lambda \sum_i^n x_i$$

$$\frac{d}{d\lambda}\ell(\lambda|x_1, x_2...x_n) = \frac{n}{\lambda} - \sum_i^n x_i = 0$$

$$\lambda = \frac{n}{\sum_i^n x_i}$$

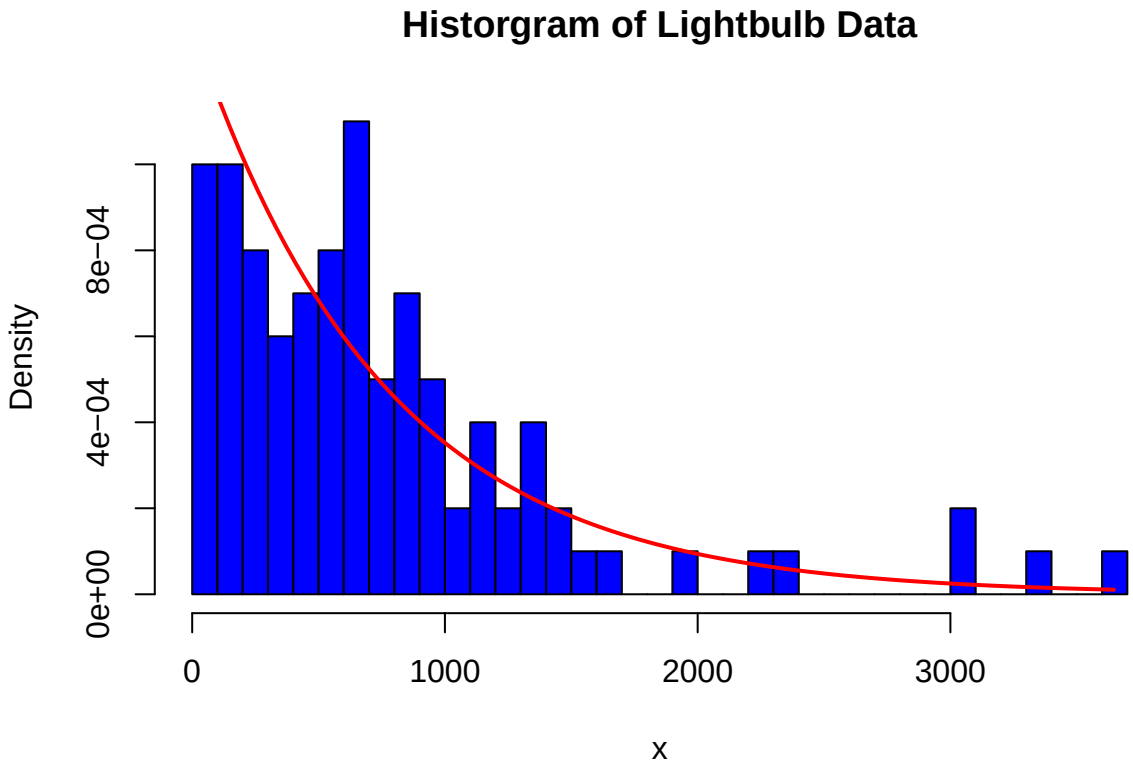
```
computed_mle <- suppressWarnings(nlm(neg.lightbulb.ll,.005, data=lightbulb)$estimate)
analytic_mle <- nrow(lightbulb) / sum(lightbulb)
if (abs(computed_mle - analytic_mle) < 1e5) {
  print(computed_mle)
  print(analytic_mle)
  print("Computed MLE extimate matches analytic solution")
} else {
  print(computed_mle)
  print(analytic_mle)
  print("MLEs do not match")
}
```

```
## [1] 0.001323916
## [1] 0.001324416
## [1] "Computed MLE extimate matches analytic solution"
```

vi.) Plotting *lightbulbs.txt* histogram with exponential pdf

```
exp_pdf <- function(x,lambda) {
  return(lambda*exp(-1*lambda*x))
}

hist(lightbulb$x, col='blue', xlab='x', main="Historgram of Lightbulb Data", breaks = 40, freq=FALSE)
xfit <- seq(min(lightbulb$x), max(lightbulb$x), length=nrow(lightbulb))
exp_est <- sapply(xfit,exp_pdf, lambda=computed_mle)
#summary(exp_est)
lines(xfit,exp_est,col='red',lwd=2)
```



Problem 2: Principle Component Analysis

i.) Generate observations of the form:

$$X_1 \sim Z_1$$
$$X_2 \sim X_1 + .001 \cdot Z_2$$
$$X_3 \sim 10 \cdot Z_3$$

```
set.seed(1)

x1 <- rnorm(200)
x2 <- x1 + .001*rnorm(200)
x3 <- 10*rnorm(200)
dataSet <- cbind(x1,x2,x3)
head(dataSet)
```

```
##              x1         x2        x3
## [1,] -0.6264538 -0.6260444 10.744410
## [2,]  0.1836433  0.1853322 18.956548
## [3,] -0.8356286 -0.8340420 -6.029973
## [4,]  1.5952808  1.5949499 -3.908678
## [5,]  0.3295078  0.3272225 -4.162220
## [6,] -0.8204684 -0.8179707 -3.756574
```

ii.) Run a principle component analysis - Without scaling

```
#Center without scaling the dataset
dataSet.C <- apply(dataSet, 2, scale, TRUE, FALSE)
head(dataSet.C)
```

```
##              x1         x2        x3
## [1,] -0.6619935 -0.6616247 11.162678
## [2,]  0.1481037  0.1497519 19.374816
## [3,] -0.8711683 -0.8696223 -5.611705
## [4,]  1.5597412  1.5593696 -3.490410
## [5,]  0.2939681  0.2916423 -3.743952
## [6,] -0.8560080 -0.8535510 -3.338306
```

```
#Run prcomp
pca <- prcomp(dataSet.C)
pca
```

```
## Standard deviations:
## [1] 10.700723042  1.310825135  0.000714302
##
## Rotation:
##            PC1          PC2           PC3
## x1 0.006008959  0.707088545 -7.070995e-01
## x2 0.006006478  0.707073974  7.071141e-01
## x3 0.999963907 -0.008496197  1.667073e-06
```

```
#Exract stdev
pca$sdev
```

```
## [1] 10.700723042  1.310825135  0.000714302
```

```
#extract loadings
pca$rotation
```

```
##            PC1          PC2           PC3
## x1 0.006008959  0.707088545 -7.070995e-01
## x2 0.006006478  0.707073974  7.071141e-01
## x3 0.999963907 -0.008496197  1.667073e-06
```

iii.) Run a principle component analysis - With scaling

```
#Center and scale the dataset
dataSet.C <- apply(dataSet, 2, scale)
head(dataSet.C)
```

```
##              x1         x2         x3
## [1,] -0.7125125 -0.7121317  1.0432075
## [2,]  0.1594060  0.1611836  1.8106725
## [3,] -0.9376502 -0.9360073 -0.5244416
## [4,]  1.6787706  1.6784084 -0.3261961
## [5,]  0.3164019  0.3139056 -0.3498909
## [6,] -0.9213331 -0.9187092 -0.3119812
```

```
#Run prcomp
pca <- prcomp(dataSet.C)
pca
```

```
## Standard deviations:
## [1] 1.4174642736 0.9953865791 0.0007688216
##
## Rotation:
##           PC1         PC2           PC3
## x1 0.70390435  0.06721418 -7.071074e-01
## x2 0.70390307  0.06724134  7.071061e-01
```

```
## x3 0.09507441 -0.99547017  1.919978e-05
```

```
#Exract stdev
pca$sdev
```

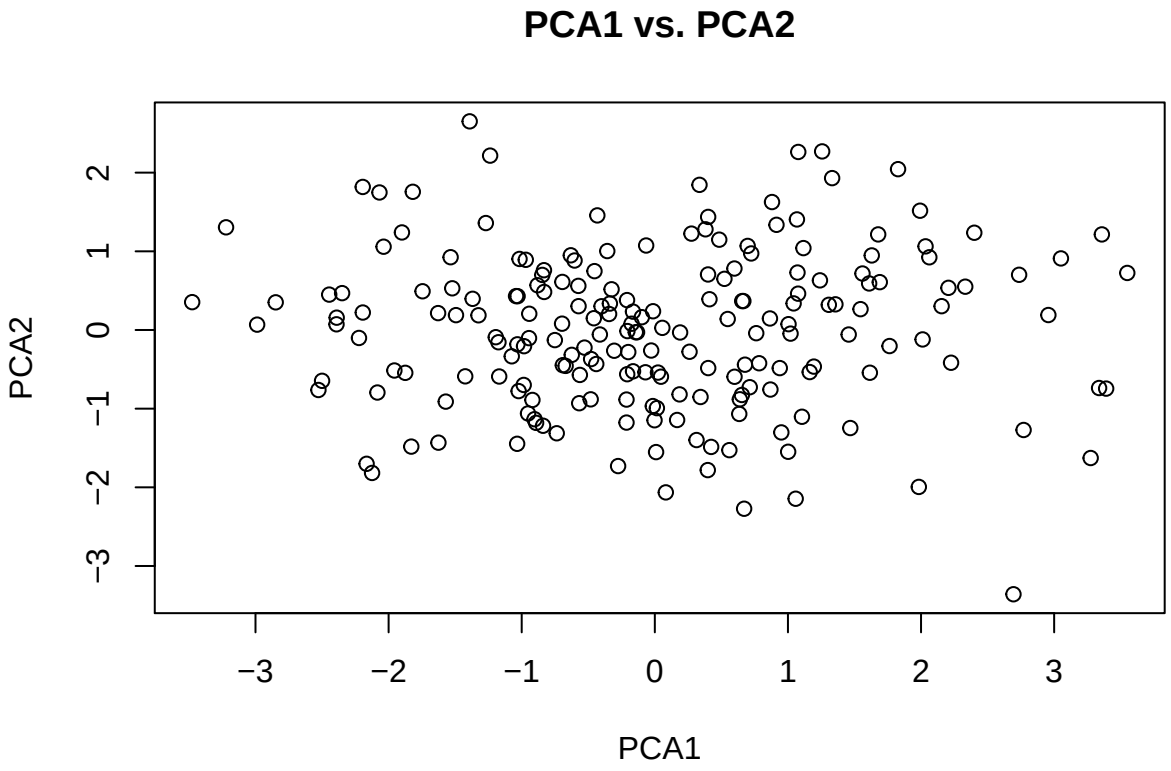
```
## [1] 1.4174642736 0.9953865791 0.0007688216
```

```
#extract loadings
pca$rotation
```

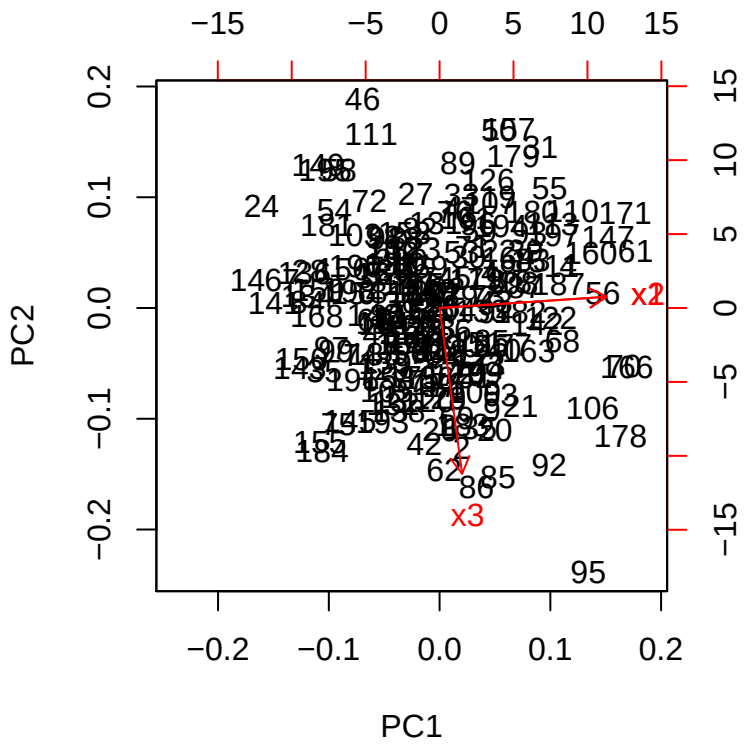
```
##           PC1         PC2           PC3
## x1 0.70390435  0.06721418 -7.071074e-01
## x2 0.70390307  0.06724134  7.071061e-01
## x3 0.09507441 -0.99547017  1.919978e-05
```

iv.) Interpret the first two components and compute the CPVE

```
# Plot PCA1 verses PCA2
plot(pca$x[,"PC1"],pca$x[,"PC2"], xlab="PCA1", ylab="PCA2", main="PCA1 vs. PCA2")
```



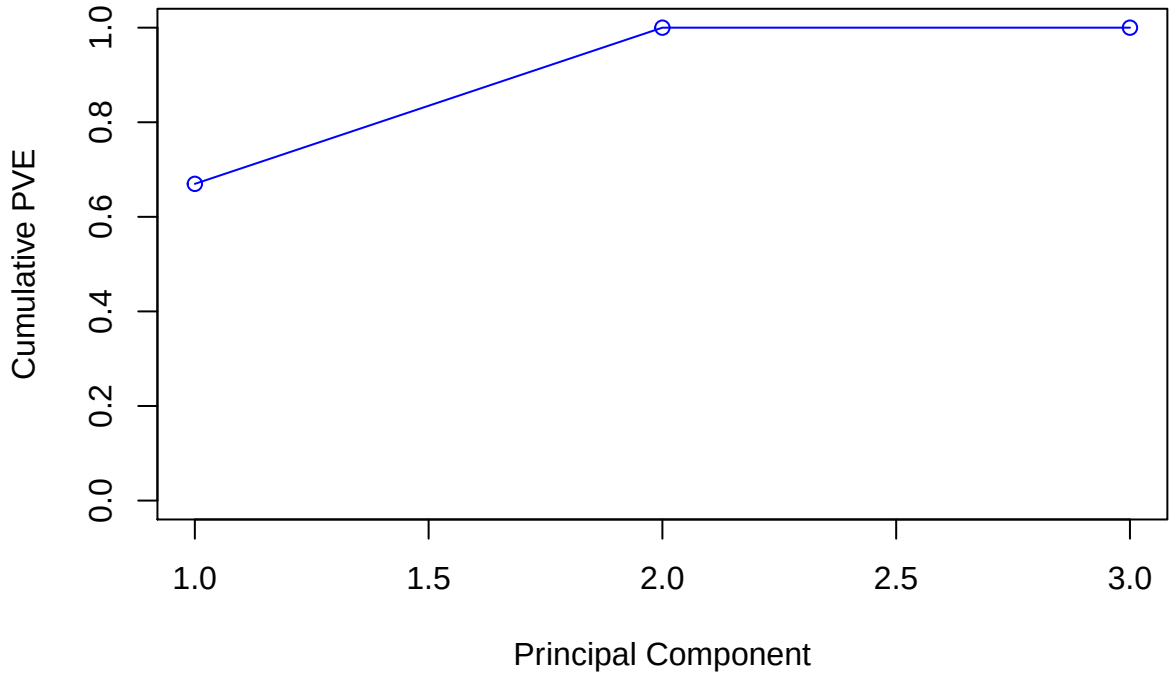

```
biplot(pca)
```




```
# Cumulative explained variance
CPVE <- cumsum((pca$sdev)^2)/sum((pca$sdev)^2)
CPVE
```

```
## [1] 0.6697350 0.9999998 1.0000000
```

```
plot(1:3,CPVE,type="l",col="blue",ylim=c(0,1),xlab="Principal Component",ylab="Cumulative PVE")
points(1:3,CPVE,col="blue")
```



v.) How many principle components needed?

We only need two principle componenets. Pc1 is the component with the most variation, pc2 is the second most and pc3 is the least. This is seen by our CPVE graph. By this graph, we see that the third pca can be essentially dropped as the change in variaance from pc2 to pc3 is very small and over 90% of the variance is explained by pc2. This makes sense as we have 3d data so pc1 projects is onto 2d, pc2 will get it to 1d and then pc3 won't be necessary.

Problem 3: Revisiting Yale Faces dataset

a.) Load the views and find the dimensions of the matrix

```
dir_list_1 = dir(path="CroppedYale/",all.files=FALSE)
pic_list = 1:length(dir_list_1)
view_list = c('P00A+000E+00', 'P00A+005E+10' , 'P00A+005E-10' , 'P00A+010E+00')

# preallocate an empty list
pic_data = vector("list",length(pic_list)*length(view_list))
# preallocate an empty list to store the pgm for debugging
pic_data_pgm = vector("list",length(pic_list)*length(view_list))

# preallocate matrix to store picture vectors as rows
this_filename = sprintf("CroppedYale/%s/%s_%s.pgm", dir_list_1[pic_list[1]] , dir_list_1[pic_list[1]] ,
this_face = read.pnm(file = this_filename)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
this_face_matrix = getChannels(this_face)
original_size = dim(this_face_matrix)
pic_vector_length = prod(original_size)
pic_mat = mat.or.vec(length(pic_list)*length(view_list),pic_vector_length)

#File list for part e
file_list <- list()

# outer loop through the pictures
suppressWarnings(for ( i in 1:length(pic_list) ){

  # inner loop over views
  for ( j in 1:length(view_list) ){

    # compile the correct file name
    # note that dir_list_1[pic_list[2]] should be "yaleB17" if pic_list[2] is B17
    this_filename = sprintf("CroppedYale/%s/%s_%s.pgm", dir_list_1[pic_list[i]] , dir_list_1[pic_list[i

    # you can print out each name to help debug the code
    # print(this_filename)
    # load the data
    this_face = read.pnm(file = this_filename)
    this_face_matrix = getChannels(this_face)

    # make the face into a vector and include in the data matrix
    pic_mat[(i-1)*length(view_list)+j,] =  as.vector(this_face_matrix)
    # if you would like to plot each picture, run the following:

    file_list[[(i-1)*length(view_list)+j]] = this_filename
```

```
    #Sys.sleep(0.5)
    #plot(this_face)
  }
})

pic_mat_size = dim(pic_mat)
dim(pic_mat)
```

```
## [1]   152 32256
```

b.) Compute the mean face, display it and save it

```
# Find the mean face vector
mean_face = colMeans(pic_mat)
# Now print it as a picture
mean_face_mat = mean_face
dim(mean_face_mat) = original_size
mean_face_pix = pixmapGrey(mean_face_mat)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
#png(filename='mean_face.png')
plot(mean_face_pix)
```

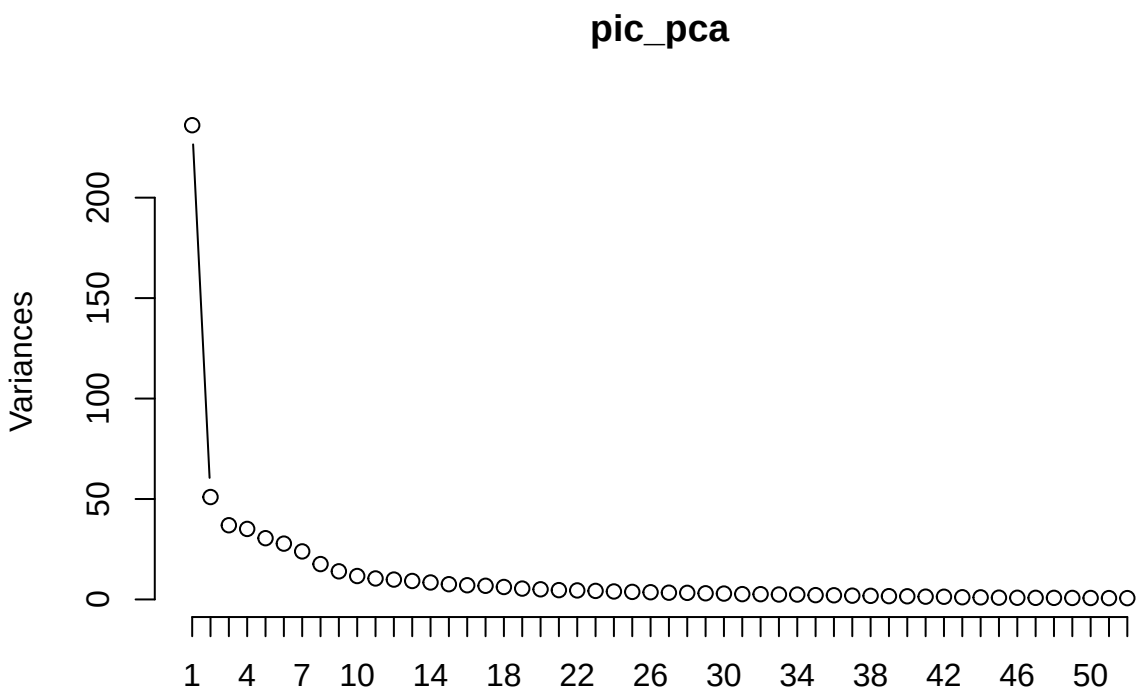
```
#dev.off()


# Subtract off the mean face
pic_mat_centered = mat.or.vec(pic_mat_size[1],pic_mat_size[2])
# I am using a loop, but one could use apply()
for (i in 1:pic_mat_size[1]){
  pic_mat_centered[i,] = pic_mat[i,] - mean_face
}
```

c.) Use prcomp() to find the principal components of your image matrix. Plot thenumber of components on the x-axis against the proportion of the variance explained on the y-axis

```
# run pca
pic_pca = prcomp(pic_mat_centered)
```

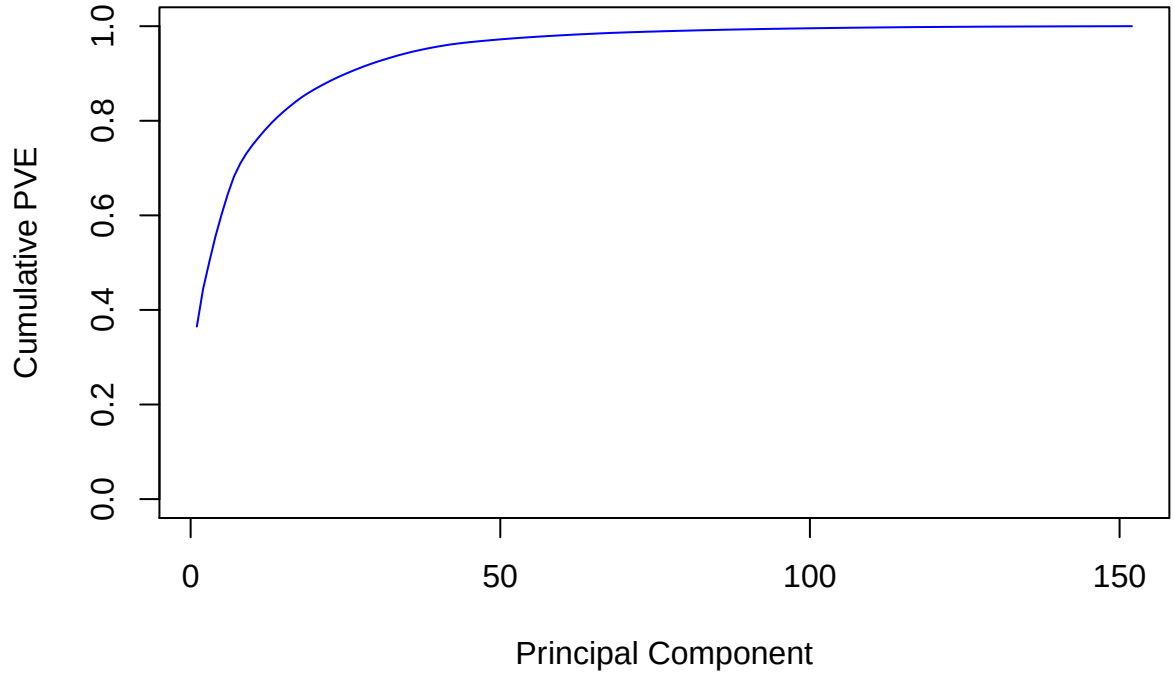
```r
#skree plot
screeplot(pic_pca, npcs=52,type="l")
```




```r
# Cumulative explained variance
CPVE <- cumsum((pic_pca$sdev)^2)/sum((pic_pca$sdev)^2)
CPVE
```

```
##   [1] 0.3649001 0.4437039 0.5007963 0.5550613 0.6022067 0.6451596 0.6821229
##   [8] 0.7093335 0.7309588 0.7489765 0.7651229 0.7804113 0.7946280 0.8076941
##  [15] 0.8194221 0.8303418 0.8408504 0.8504347 0.8587935 0.8665934 0.8737712
##  [22] 0.8807132 0.8872907 0.8934396 0.8992588 0.9047981 0.9100002 0.9150709
##  [29] 0.9198139 0.9243274 0.9284223 0.9324743 0.9362720 0.9400380 0.9433515
##  [36] 0.9465605 0.9494824 0.9522651 0.9547972 0.9572239 0.9593707 0.9614430
##  [43] 0.9631259 0.9647431 0.9661357 0.9674279 0.9686621 0.9698850 0.9710353
##  [50] 0.9721517 0.9731907 0.9742106 0.9751680 0.9760768 0.9769448 0.9777660
##  [57] 0.9785685 0.9793544 0.9801200 0.9808548 0.9815551 0.9822310 0.9828774
##  [64] 0.9835203 0.9841391 0.9847207 0.9852743 0.9858072 0.9863031 0.9867858
##  [71] 0.9872248 0.9876480 0.9880589 0.9884455 0.9888233 0.9891937 0.9895586
##  [78] 0.9899189 0.9902731 0.9906078 0.9909356 0.9912532 0.9915619 0.9918670
##  [85] 0.9921650 0.9924535 0.9927285 0.9929872 0.9932401 0.9934885 0.9937285
##  [92] 0.9939631 0.9941907 0.9944113 0.9946241 0.9948348 0.9950378 0.9952367
##  [99] 0.9954292 0.9956149 0.9957979 0.9959741 0.9961434 0.9963104 0.9964760
## [106] 0.9966342 0.9967876 0.9969351 0.9970800 0.9972179 0.9973519 0.9974842
## [113] 0.9976119 0.9977357 0.9978561 0.9979706 0.9980821 0.9981841 0.9982847
## [120] 0.9983806 0.9984666 0.9985507 0.9986331 0.9987133 0.9987891 0.9988621
## [127] 0.9989295 0.9989948 0.9990554 0.9991152 0.9991739 0.9992292 0.9992818
## [134] 0.9993335 0.9993837 0.9994328 0.9994807 0.9995274 0.9995736 0.9996169
## [141] 0.9996595 0.9997002 0.9997390 0.9997775 0.9998142 0.9998498 0.9998841
## [148] 0.9999150 0.9999447 0.9999732 1.0000000 1.0000000
```

```r
plot(1:152,CPVE,type="l",col="blue",ylim=c(0,1),xlab="Principal Component",ylab="Cumulative PVE")
```




```r
#points(1:152,CPVE,col="blue")
```

d.) Each principal component is a picture, which are called "eigenfaces." Display the first 9 eigenfaces in a 3-by-3 grid. What image components does each describe

```r
# initialize an empty matrix of faces data
eigenface_mat = vector()


# loop through first 9 eigenfaces
this_face_row = vector()


# outer loop through the pictures
for (i in 1:9){

  # Extract ith eigenface
  eigenface_temp <- pic_pca$rotation[,i]
  dim(eigenface_temp) <- original_size
  this_face_row = cbind(this_face_row,eigenface_temp)

  if ((i %% 3)==0){
    # make a new row
    eigenface_mat = rbind(eigenface_mat,this_face_row)
    # clear row vector
    this_face_row = vector()
  }
}
# Plot the first 9 eigen faces
eigenface_pgm = pixmapGrey(eigenface_mat)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
plot(eigenface_pgm)
```

e.) Use the eigenfaces to reconstruct yaleB05P00A+010E+00.pgm. Starting with the mean face, add in one eigenface at a time until you reach 24 eigenfaces. Save the results in a 5-by-5 grid. Do this again for 120 eigenfaces. How many do you think you need?

```
eigenface_mat = vector()

this_face_row = mean_face_mat
eigenface_sum = mean_face_mat

for(i in 2:25) {
  eigenface_temp <- pic_pca$rotation[,i-1]*pic_pca$x[20, i-1]
  dim(eigenface_temp) <- original_size
  eigenface_sum = eigenface_sum + eigenface_temp
  this_face_row = cbind(this_face_row,eigenface_sum)

  if ((i %% 5) == 0) {
    # make a new row
    eigenface_mat = rbind(eigenface_mat,this_face_row)
    # clear row vector
    this_face_row = vector()
  }
}

eigenface_pgm = pixmapGrey(eigenface_mat)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
#png(filename='eigen_face_24.png')
plot(eigenface_pgm)
```

```
eigenface_mat_2 = vector()

this_face_row <- mean_face_mat
eigenface_sum_tot <- mean_face_mat

for(i in 2:25) {
  for(j in (5*i-9):(5*i-5) ) {
    eigenface_sum_tot <- eigenface_sum_tot + pic_pca$rotation[,j]*pic_pca$x[20, j]
  }
  this_face_row = cbind(this_face_row,eigenface_sum_tot)

  if ((i %% 5) == 0) {
    # make a new row
    eigenface_mat_2 = rbind(eigenface_mat_2,this_face_row)
    # clear row vector
    this_face_row = vector()
  }
}

eigenface_pgm_2 = pixmapGrey(eigenface_mat_2)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
#png(filename='eigen_face_120.png')
plot(eigenface_pgm_2)
```

```
#dev.off()
```

```
summary(pic_pca)[6]
```

```
## $importance
##                              PC1     PC2      PC3      PC4      PC5
## Standard deviation     15.36422 7.13998 6.077322 5.924934 5.522595
## Proportion of Variance  0.36490 0.07880 0.057090 0.054270 0.047150
## Cumulative Proportion   0.36490 0.44370 0.500800 0.555060 0.602210
##                              PC6      PC7     PC8      PC9     PC10
## Standard deviation     5.271319 4.890002 4.19558 3.740289 3.414073
## Proportion of Variance 0.042950 0.036960 0.02721 0.021630 0.018020
## Cumulative Proportion  0.645160 0.682120 0.70933 0.730960 0.748980
##                             PC11     PC12     PC13     PC14     PC15
## Standard deviation     3.231925 3.144884 3.032657 2.907343 2.754444
## Proportion of Variance 0.016150 0.015290 0.014220 0.013070 0.011730
## Cumulative Proportion  0.765120 0.780410 0.794630 0.807690 0.819420
##                             PC16     PC17     PC18     PC19     PC20
## Standard deviation     2.657844 2.607331 2.490023 2.325382 2.246305
## Proportion of Variance 0.010920 0.010510 0.009580 0.008360 0.007800
## Cumulative Proportion  0.830340 0.840850 0.850430 0.858790 0.866590
##                             PC21     PC22     PC23     PC24     PC25
## Standard deviation     2.154858 2.119178 2.062777 1.994447 1.940237
## Proportion of Variance 0.007180 0.006940 0.006580 0.006150 0.005820
## Cumulative Proportion  0.873770 0.880710 0.887290 0.893440 0.899260
##                             PC26     PC27     PC28     PC29     PC30    PC31
## Standard deviation     1.89300 1.834482 1.811165 1.751663 1.708761 1.62758
## Proportion of Variance 0.00554 0.005200 0.005070 0.004740 0.004510 0.00409
## Cumulative Proportion  0.90480 0.910000 0.915070 0.919810 0.924330 0.92842
##                             PC32     PC33     PC34     PC35     PC36
## Standard deviation     1.619036 1.567414 1.560871 1.464075 1.440824
## Proportion of Variance 0.004050 0.003800 0.003770 0.003310 0.003210
## Cumulative Proportion  0.932470 0.936270 0.940040 0.943350 0.946560
##                             PC37     PC38     PC39     PC40     PC41
## Standard deviation     1.374857 1.341687 1.279866 1.252954 1.178464
## Proportion of Variance 0.002920 0.002780 0.002530 0.002430 0.002150
## Cumulative Proportion  0.949480 0.952270 0.954800 0.957220 0.959370
##                             PC42     PC43    PC44      PC45      PC46
```

```
## Standard deviation     1.157861 1.043402 1.02281 0.9491672 0.9143132
## Proportion of Variance 0.002070 0.001680 0.00162 0.0013900 0.0012900
## Cumulative Proportion  0.961440 0.963130 0.96474 0.9661400 0.9674300
##                            PC47    PC48      PC49      PC50      PC51
## Standard deviation     0.8935352 0.88944 0.8626282 0.8498411 0.8198501
## Proportion of Variance 0.0012300 0.00122 0.0011500 0.0011200 0.0010400
## Cumulative Proportion  0.9686600 0.96988 0.9710400 0.9721500 0.9731900
##                            PC52      PC53      PC54      PC55      PC56
## Standard deviation     0.8122652 0.7869895 0.7667599 0.7493496 0.7288899
## Proportion of Variance 0.0010200 0.0009600 0.0009100 0.0008700 0.0008200
## Cumulative Proportion  0.9742100 0.9751700 0.9760800 0.9769400 0.9777700
##                            PC57      PC58      PC59      PC60      PC61
## Standard deviation     0.7204997 0.7130499 0.7037565 0.6894312 0.6731078
## Proportion of Variance 0.0008000 0.0007900 0.0007700 0.0007300 0.0007000
## Cumulative Proportion  0.9785700 0.9793500 0.9801200 0.9808500 0.9815600
##                           PC62      PC63      PC64      PC65     PC66
## Standard deviation     0.661206 0.6466565 0.6449179 0.6327147 0.613382
## Proportion of Variance 0.000680 0.0006500 0.0006400 0.0006200 0.000580
## Cumulative Proportion  0.982230 0.9828800 0.9835200 0.9841400 0.984720
##                            PC67      PC68      PC69      PC70      PC71
## Standard deviation     0.5984479 0.5871486 0.5664066 0.5587929 0.5329305
## Proportion of Variance 0.0005500 0.0005300 0.0005000 0.0004800 0.0004400
## Cumulative Proportion  0.9852700 0.9858100 0.9863000 0.9867900 0.9872200
##                            PC72      PC73      PC74      PC75      PC76
## Standard deviation     0.5232126 0.5155826 0.5001075 0.4943225 0.4895193
## Proportion of Variance 0.0004200 0.0004100 0.0003900 0.0003800 0.0003700
## Cumulative Proportion  0.9876500 0.9880600 0.9884500 0.9888200 0.9891900
##                            PC77      PC78      PC79      PC80      PC81
## Standard deviation     0.4858992 0.4827721 0.4786886 0.4653311 0.4604431
## Proportion of Variance 0.0003600 0.0003600 0.0003500 0.0003300 0.0003300
## Cumulative Proportion  0.9895600 0.9899200 0.9902700 0.9906100 0.9909400
##                            PC82     PC83      PC84      PC85      PC86
## Standard deviation     0.4533044 0.446884 0.4442969 0.4390185 0.4320325
## Proportion of Variance 0.0003200 0.000310 0.0003100 0.0003000 0.0002900
## Cumulative Proportion  0.9912500 0.991560 0.9918700 0.9921600 0.9924500
##                            PC87      PC88      PC89      PC90      PC91
## Standard deviation     0.4217671 0.4091332 0.4044429 0.4008411 0.3940965
## Proportion of Variance 0.0002700 0.0002600 0.0002500 0.0002500 0.0002400
## Cumulative Proportion  0.9927300 0.9929900 0.9932400 0.9934900 0.9937300
##                            PC92      PC93      PC94      PC95      PC96
## Standard deviation     0.3895794 0.3836707 0.3777923 0.3709835 0.3692047
## Proportion of Variance 0.0002300 0.0002300 0.0002200 0.0002100 0.0002100
## Cumulative Proportion  0.9939600 0.9941900 0.9944100 0.9946200 0.9948300
##                            PC97      PC98      PC99     PC100     PC101
## Standard deviation     0.3624215 0.3587219 0.3528418 0.3466113 0.3440389
## Proportion of Variance 0.0002000 0.0002000 0.0001900 0.0001900 0.0001800
## Cumulative Proportion  0.9950400 0.9952400 0.9954300 0.9956100 0.9958000
##                           PC102   PC103     PC104     PC105     PC106
## Standard deviation     0.3376989 0.33095 0.3286491 0.3273363 0.3198184
## Proportion of Variance 0.0001800 0.00017 0.0001700 0.0001700 0.0001600
## Cumulative Proportion  0.9959700 0.99614 0.9963100 0.9964800 0.9966300
##                           PC107     PC108     PC109     PC110     PC111
## Standard deviation     0.3150668 0.3088896 0.3061521 0.2987606 0.2943448
## Proportion of Variance 0.0001500 0.0001500 0.0001400 0.0001400 0.0001300
```

```
## Cumulative Proportion  0.9967900 0.9969400 0.9970800 0.9972200 0.9973500
##                            PC112     PC113     PC114     PC115     PC116
## Standard deviation     0.2925393 0.2874705 0.2829922 0.2790658 0.2722245
## Proportion of Variance 0.0001300 0.0001300 0.0001200 0.0001200 0.0001100
## Cumulative Proportion  0.9974800 0.9976100 0.9977400 0.9978600 0.9979700
##                            PC117     PC118     PC119     PC120     PC121
## Standard deviation     0.2685836 0.2568183 0.2550781 0.2491506 0.2358039
## Proportion of Variance 0.0001100 0.0001000 0.0001000 0.0001000 0.0000900
## Cumulative Proportion  0.9980800 0.9981800 0.9982800 0.9983800 0.9984700
##                            PC122     PC123    PC124     PC125     PC126
## Standard deviation     0.2332621 0.2308772 0.227814 0.2214611 0.2172618
## Proportion of Variance 0.0000800 0.0000800 0.000080 0.0000800 0.0000700
## Cumulative Proportion  0.9985500 0.9986300 0.998710 0.9987900 0.9988600
##                            PC127     PC128     PC129     PC130     PC131
## Standard deviation     0.2087813 0.2054981 0.1980484 0.1967573 0.1947874
## Proportion of Variance 0.0000700 0.0000700 0.0000600 0.0000600 0.0000600
## Cumulative Proportion  0.9989300 0.9989900 0.9990600 0.9991200 0.9991700
##                           PC132     PC133     PC134     PC135     PC136
## Standard deviation     0.189132 0.1845547 0.1828193 0.1802311 0.1781294
## Proportion of Variance 0.000060 0.0000500 0.0000500 0.0000500 0.0000500
## Cumulative Proportion  0.999230 0.9992800 0.9993300 0.9993800 0.9994300
##                            PC137     PC138     PC139     PC140     PC141
## Standard deviation     0.1760513 0.1738488 0.1728453 0.1674738 0.1659593
## Proportion of Variance 0.0000500 0.0000500 0.0000500 0.0000400 0.0000400
## Cumulative Proportion  0.9994800 0.9995300 0.9995700 0.9996200 0.9996600
##                            PC142    PC143     PC144     PC145     PC146
## Standard deviation     0.1622199 0.158578 0.1576462 0.1540602 0.1518917
## Proportion of Variance 0.0000400 0.000040 0.0000400 0.0000400 0.0000400
## Cumulative Proportion  0.9997000 0.999740 0.9997800 0.9998100 0.9998500
##                            PC147     PC148     PC149     PC150     PC151
## Standard deviation     0.1489806 0.1413852 0.1384531 0.1358403 0.1317146
## Proportion of Variance 0.0000300 0.0000300 0.0000300 0.0000300 0.0000300
## Cumulative Proportion  0.9998800 0.9999200 0.9999400 0.9999700 1.0000000
##                              PC152
## Standard deviation     1.257978e-14
## Proportion of Variance 0.000000e+00
## Cumulative Proportion  1.000000e+00
```

At about 25, 90% of the variance is expained so we can use about 25 eigenfaces.

f.) Remove the pictures of subject 01 from your image matrix (there should be fourpictures of him) and recenter the data. Rerunprcomp()to get new principal components. Use these to reconstruct yaleB01P00A+010E+00.pgm.

```
# remove first four rows
orig_subject1 <- pic_mat[1:4, ]

new_pic_mat = pic_mat[-(1:4), ]
new_pic_mat_size = dim(new_pic_mat)

# center the new matrix and run prcomp()
#new_pic_mat_centered <- apply(new_pic_mat, 2, scale, TRUE, FALSE)

new_mean_face = colMeans(new_pic_mat)
new_mean_face_mat = new_mean_face
```

```
dim(new_mean_face_mat) = original_size
# Subtract off the mean face
new_pic_mat_centered = mat.or.vec(new_pic_mat_size[1],new_pic_mat_size[2])
# I am using a loop, but one could use apply()
for (i in 1:new_pic_mat_size[1]){
  new_pic_mat_centered[i,] = new_pic_mat[i,] - new_mean_face
}

new_pic_pca = prcomp(new_pic_mat_centered)

# project the remaining image onto pcs

#center the query pic
orig_subject1_size <- dim(orig_subject1)
orig_mean_face <- colMeans(orig_subject1)
orig_mean_face_mat = orig_mean_face
dim(orig_mean_face_mat) <- original_size
orig_subject1_centered <- mat.or.vec(orig_subject1_size[1],orig_subject1_size[2])
for (i in 1:4) {
  orig_subject1_centered[i,] = orig_subject1_centered[i,] - new_mean_face
}

eigenface_score_sum <- new_mean_face_mat

for (i in 1:48) {
  score_query <- sum(new_pic_pca$rotation[, i]*orig_subject1_centered)
  eigenface_temp <- new_pic_pca$rotation[, i]*score_query
  dim(eigenface_temp) <- original_size
  eigenface_score_sum = eigenface_score_sum + eigenface_temp
}

dim(orig_subject1_centered)
```

```
## [1]     4 32256
```

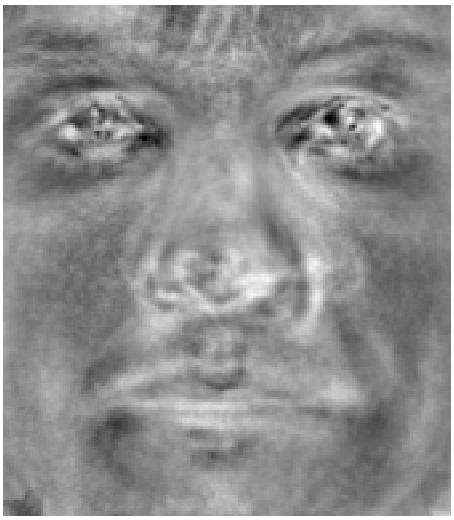
```
# orig_pgm <- pixmapGrey(orig_mean_face_mat)
# plot(orig_pgm)

reconstructed_image <- pixmapGrey(eigenface_score_sum)
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
#png(filename='reconstructed_subject01.png')
plot(reconstructed_image, main='Reconstructed Image')
```

Reconstructed Image



```
#dev.off()

first_face <- read.pnm(file="CroppedYale/yaleB01/yaleB01_P00A+010E+00.pgm")
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

```
plot(first_face, main='Original Image')
```

Original Image

The reconstructed image doesn't really look like the original image. This is because, as we shouldn't have access to the actual data, we start at the mean face of the modified matrix and continuously add pc components and their respective score (which we calculated from the query pic we were trying to construct).

With the original subject 01 picture not being there, the best we can do is with a linear combination of the eigenfaces of the rest of the subjects. What we get is the best amalgamtion of these eigenfaces to get as close to subject 01 as possible but in the end it doesn't quite look like the original image.