Assessed Coursework 1

CID 01252821

Problem 1

(a) In PCA in order to find a principal component we have to solve the following equation

$$\Sigma c = \lambda c,$$

where Σ is the sample covariance matrix, $\Sigma \in R^{d\times d}$ if we have samples $x_i \in R^d$, $i = 1, .., n$, $c \in R^d$, $\lambda \in R$. Solving the equation will determine the eigenvector c of the matrix Σ, and hence determine a principal component.

Let's multiply the above equation by a scalar $k \in R_{\neq 0}$:

$$\Sigma(kc) = \lambda(kc).$$

This shows that if c is an eigenvector so is kc for any $k \in R_{\neq 0}$, and both are corresponding to the same eigenvalue λ. Hence, the princiapl components are unique up to a non-zero scalar multiple (in the case of distinct eigenvalues).

(b) Let X be a matrix with n observations in R^d: $X = [x_1, ..., x_n]^T$, and $\bar{x} = 0$.

Ordinary PCA:

The covariance matrix is

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T = \frac{1}{n}X^T X.$$

In ordinary PCA the eigenvector of Σ is solved from

$$\Sigma c = \lambda c,$$

where $\lambda \in R_{\neq 0}$.

Kernel PCA with Linear Kernel:

The kernel matrix K is defined as:

$$K = \begin{bmatrix} f(x_1)^T f(x_1) & f(x_1)^T f(x_2) & ... & f(x_1)^T f(x_n) \\ ... & ... & ... & ... \\ f(x_n)^T f(x_1) & f(x_n)^T f(x_2) & ... & f(x_n)^T f(x_n) \end{bmatrix} \in R^{n\times n},$$

where via the kernel trick $f(x)^T f(y) = k(x, y) = x^T y$, or $f(x) = x$, giving us

$$K = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & ... & x_1^T x_n \\ ... & ... & ... & ... \\ x_n^T x_1 & x_n^T x_2 & ... & x_n^T x_n \end{bmatrix} = XX^T.$$

In Kernel PCA we should solve the following equation:

$$K\alpha = n\lambda\alpha$$

where α are the coefficients of the eigenvectors in the feature space and $\lambda \in R_{>0}$ is an eigenvalue.

Let c' be a Kernel PCA eigenvector corresponding to an eigenvalue λ', then:

$$Kc' = \lambda' c'.$$

Multiplying both sides by X, and acknowledging $K = XX^T$:

$$XX^T Xc' = \lambda' Xc',$$

$$n\Sigma(Xc') = \lambda'(Xc').$$

Now, let $\nu = Xc'$:

$$\Sigma\nu = \frac{\lambda'}{n}\nu,$$

meaning ν is an eigenvector of Σ with eigenvalue $\frac{\lambda'}{n}$. Considering the eigenvalues from the kernel PCA were scaled by n, this shows that the eigenvectors of the kernel matrix K correspond to the eigenvectors of the covariance matrix Σ.

(c) Given $x, y \in R^d$ with components x_i and y_i, the square of their dot product is:

$$(x^T y)^2 = x^T y x^T y = \sum_{i=1}^{d}\sum_{j=1}^{d} x_i y_i x_j y_j.$$

The feature map f must be constructed in a way that $f(x)^T f(y)$ yields the above sum. The mapping can be defined as:

$$f(x) = [x_1x_1, x_1x_2, \ldots, x_1x_d, x_2x_1, x_2x_2, \ldots, x_2x_d, \ldots, x_dx_1, \ldots, x_dx_d]$$

The dimensionality p of $f(x)$ is d^2.

Showing that $f(x)^T f(y)$ equals $(x^T y)^2$ and confirming the needed property:

$$f(x)^T f(y) = \sum_{i=1}^{d}\sum_{j=1}^{d}(x_i x_j)(y_i y_j)$$

By the distributive law:

$$\sum_{i=1}^{d}\sum_{j=1}^{d} x_i y_i x_j y_j = \left(\sum_{i=1}^{d} x_i y_i\right)\left(\sum_{j=1}^{d} x_j y_j\right) = (x^T y)^2.$$

The dimensionality p of the feature space grows quadratically with the dimension d of the original space. As outlined earlier, $p = d^2$.

(d) If X_1 and X_2 are uncorrelated, then their covariance is zero, and by problem statement their variances are one, meaning they must be jointly normally distributed (having a rotational symmetry property) for the distribution to remain the same under an orthogonal transformation A. Therefore, to find such X_1 and X_2 where the distribution changes under transformation, we need X_1 and X_2 to not follow a joint normal distribution.

Let X_1 be a standard normal variable, $X_1 \sim N(0, 1)$, and let X_2 be a random variable that takes values $+1$ and -1, each with probability 0.5, and does not have a normal distribution.

Since X_1 is normally distributed and X_2 is not, their joint distribution is not multivariate normal. Now, let's consider a rotation matrix as our orthogonal matrix A:

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Multiplying A with X will result in a rotation of the vector X in the plane, resulting in a new random vector:

$$AX = A\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} X_1\cos(\theta) - X_2\sin(\theta) \\ X_1\sin(\theta) + X_2\cos(\theta) \end{bmatrix}$$

Because X_2 was not normally distributed, the result of this linear combination AX will also not be normally distributed. Therefore, the distribution of X and the distribution of AX will differ because the joint distribution of X_1 and X_2 is not multivariate normal, and hence it is not rotationally invariant.

Note: The idea of using a rotation matrix was obtained from ChatGPT.

(e) Let $g_Y(y)$ be the joint pdf of Y, and $g_{Y_i}(y_i)$ are the marginal pdfs of the components of Y. The mutual Information $I(Y)$ is defined as:

$$I(Y) = \sum_{i=1}^{d} H(Y_i) - H(Y)$$

where $H(Y_i)$ is the entropy of Y_i and $H(Y)$ is the joint entropy of Y.

$$E\left[\frac{\prod_{i=1}^{d} g_{Y_i}(Y_i)}{g_Y(Y)}\right] = \int_{\mathbb{R}^d} g_Y(y)\frac{\prod_{i=1}^{d} g_{Y_i}(Y_i)}{g_Y(Y)}dy = \int_{\mathbb{R}^d}\prod_{i=1}^{d} g_{Y_i}(y_i)dy = 1$$

Applying Jensen's Inequality taking $f(x) = -\log(x)$:

$$E\left[-\log\left(\frac{\prod_{i=1}^{d} g_{Y_i}(Y_i)}{g_Y(Y)}\right)\right] \geq -\log\left(E\left[\frac{\prod_{i=1}^{d} g_{Y_i}(Y_i)}{g_Y(Y)}\right]\right) = \log 1 = 0,$$

or

$$E\left[\log\left(\frac{g_Y(Y)}{\prod_{i=1}^{d} g_{Y_i}(Y_i)}\right)\right] \geq 0,$$

The left-hand side of the inequality is actually the definition of the mutual information $I(Y)$ because:

$$I(Y) = \sum_{i=1}^{d} H(Y_i) - H(Y) = \int \cdots \int g_Y(y_1, \ldots, y_d) \log\left(\frac{g_Y(y_1, \ldots, y_d)}{g_{Y_1}(y_1) \cdots g_{Y_d}(y_d)}\right) dy_1 \cdots dy_d = E\left[\log\left(\frac{g_Y(Y_1, ..., Y_d)}{g_{Y_1}(Y_1) \cdots g_{Y_d}(Y_d)}\right)\right].$$

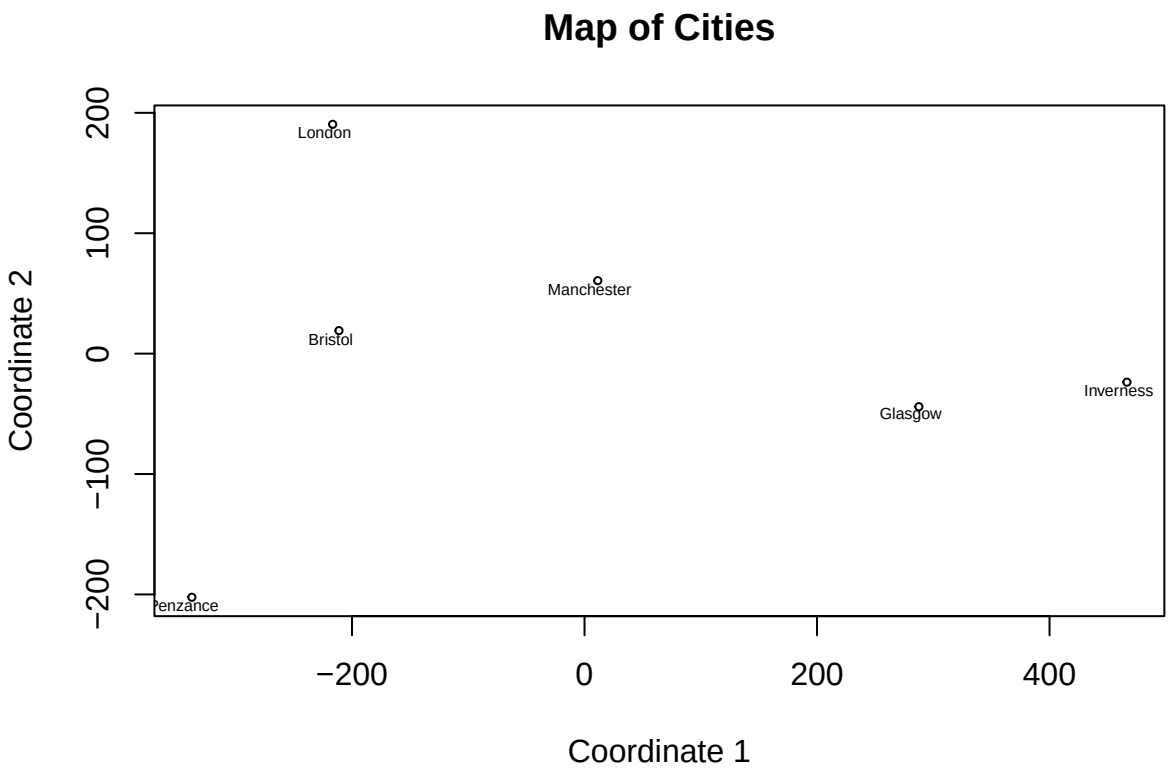
Hence,

$$I(Y) \geq 0.$$

Problem 2

Performing multidimensional scaling for creating a 2-dimensional map:

```r
# distance matrix based on the given distances
distances <- matrix(c(
  0, 262, 171, 556, 716, 411,   # London
  262, 0, 227, 295, 463, 437,   # Manchester
  171, 227, 0, 503, 679, 255,   # Bristol
  556, 295, 503, 0, 180, 645,   # Glasgow
  716, 463, 679, 180, 0, 824,   # Inverness
  411, 437, 255, 645, 824, 0    # Penzance
), nrow=6, byrow=TRUE)

# Multidimensional Scaling with k=2 for 2-dimensional map
fit <- cmdscale(distances, eig=TRUE, k=2)

coords <- fit$points
plot(
  coords, type="p", xlab="Coordinate 1", ylab="Coordinate 2", main="Map of Cities", cex=0.5
)
text(
  coords - 7,
  labels=c("London", "Manchester", "Bristol", "Glasgow", "Inverness", "Penzance"),
  cex=0.5
)
```



The map in terms of representing the pairwise distances between the cities as closely as possible in a 2D space, given the constraint of the flat map, is accurate, but geodesic distances would be needed to represent the distances closer to the real-world geographic layout of the cities.

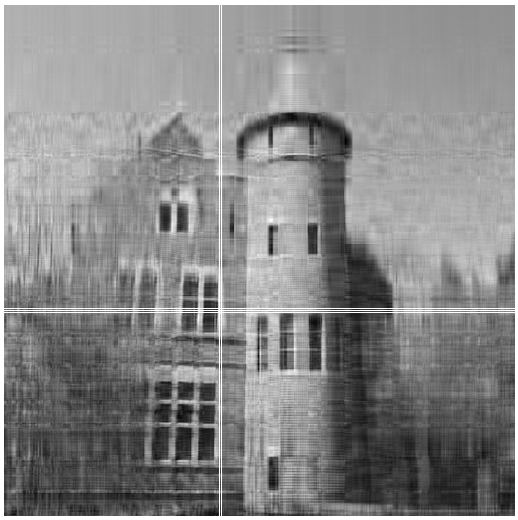
Problem 3

Applying PCA by treating columns as variables and rows as observations:

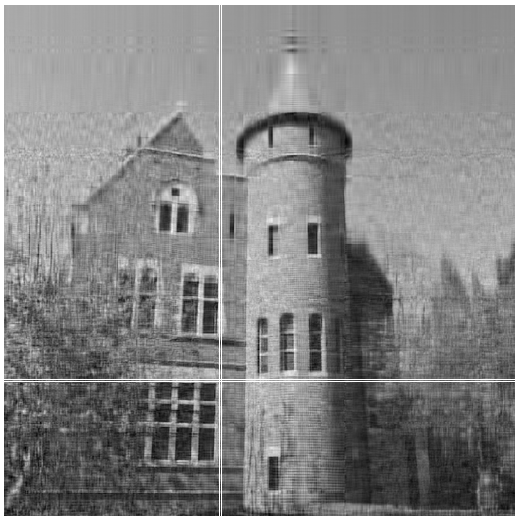
```
library(ggplot2)

# load data
image_data <- read.csv('problem3.csv', header = FALSE)
image_matrix <- as.matrix(image_data)

# apply PCA by treating columns as variables and rows as observations
pca_result <- prcomp(t(image_matrix), center = TRUE)

reconstruct_image <- function(pca, q) {
  pca_rotation = pca$rotation[, 1:q]
  projected <- pca$x[, 1:q]
  reconstructed <- pca$center + pca_rotation %*% t(projected)

  return(reconstructed)
}
```

```
# values of q as powers of 2
q_values <- c(16, 32, 64, 128, 256, 512)

for (q in q_values) {
  reconstructed_matrix <- reconstruct_image(pca_result, q)
  image(1:512, 1:512, reconstructed_matrix, col=grey(seq(0, 1, length = 256)), main = paste('q =', q), 
}
```

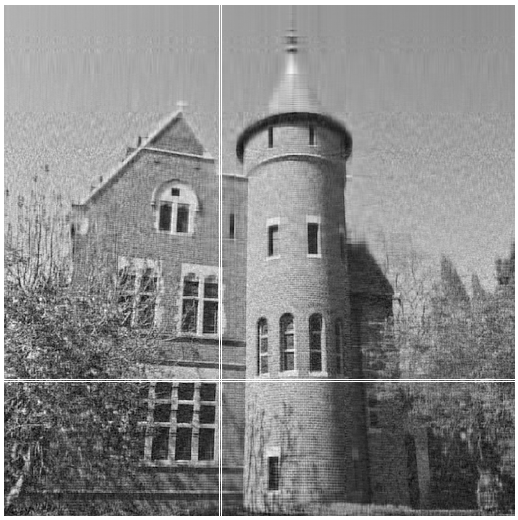
q = 16



q = 32



q = 64



q = 128

q = 256

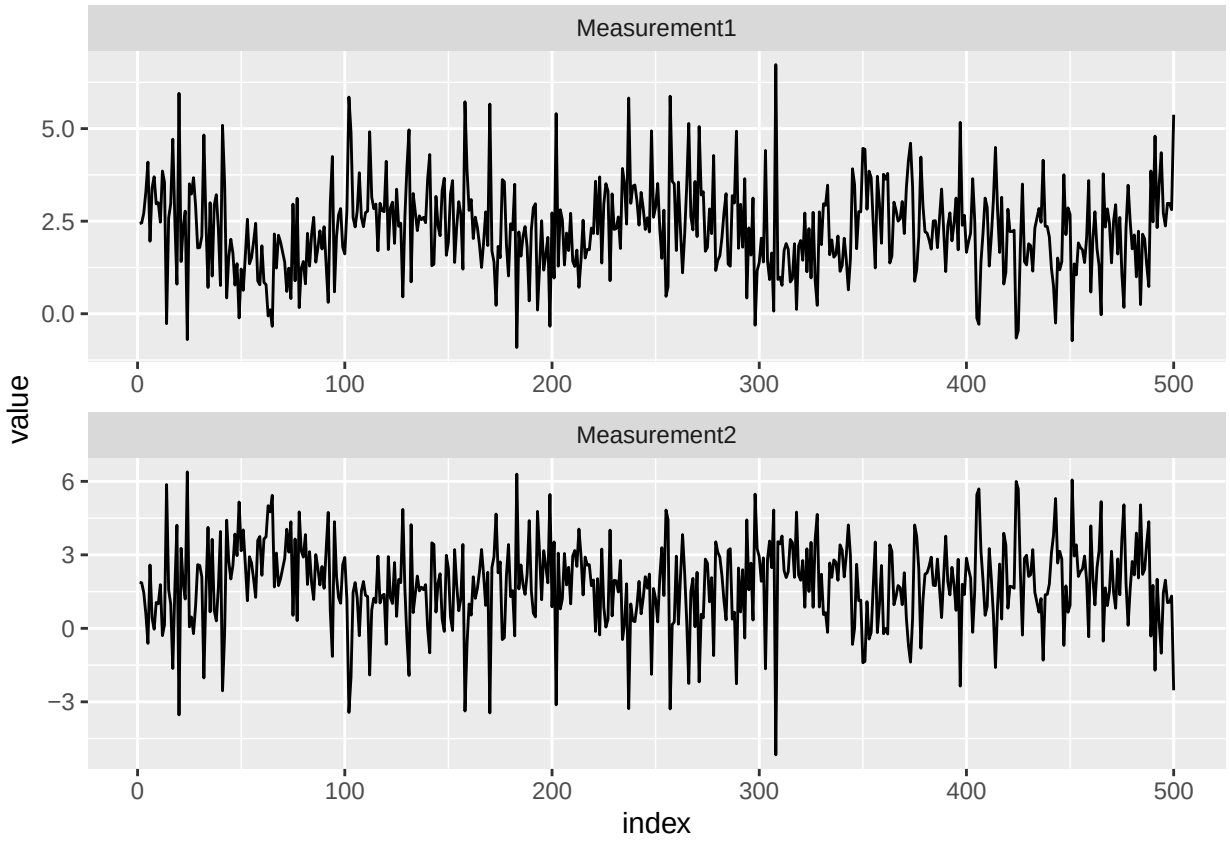
q = 512

```
image(1:512, 1:512, image_matrix, col=grey(seq(0, 1, length = 256)),
      main = 'Original', xlab = "", ylab = "", axes = FALSE, asp = 1)
```

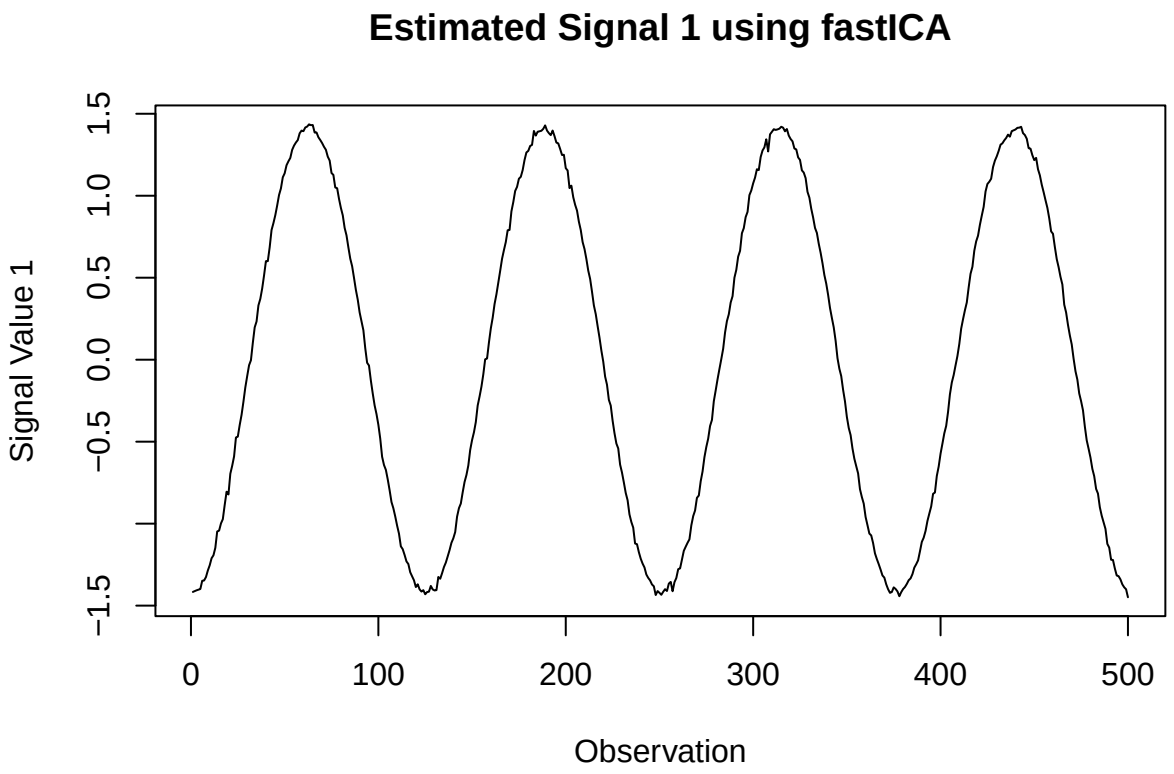
If we observe the powers of 2 as values for q, we can conclude that around 256 we begin to have a pretty clear reconstructed image.

Problem 4

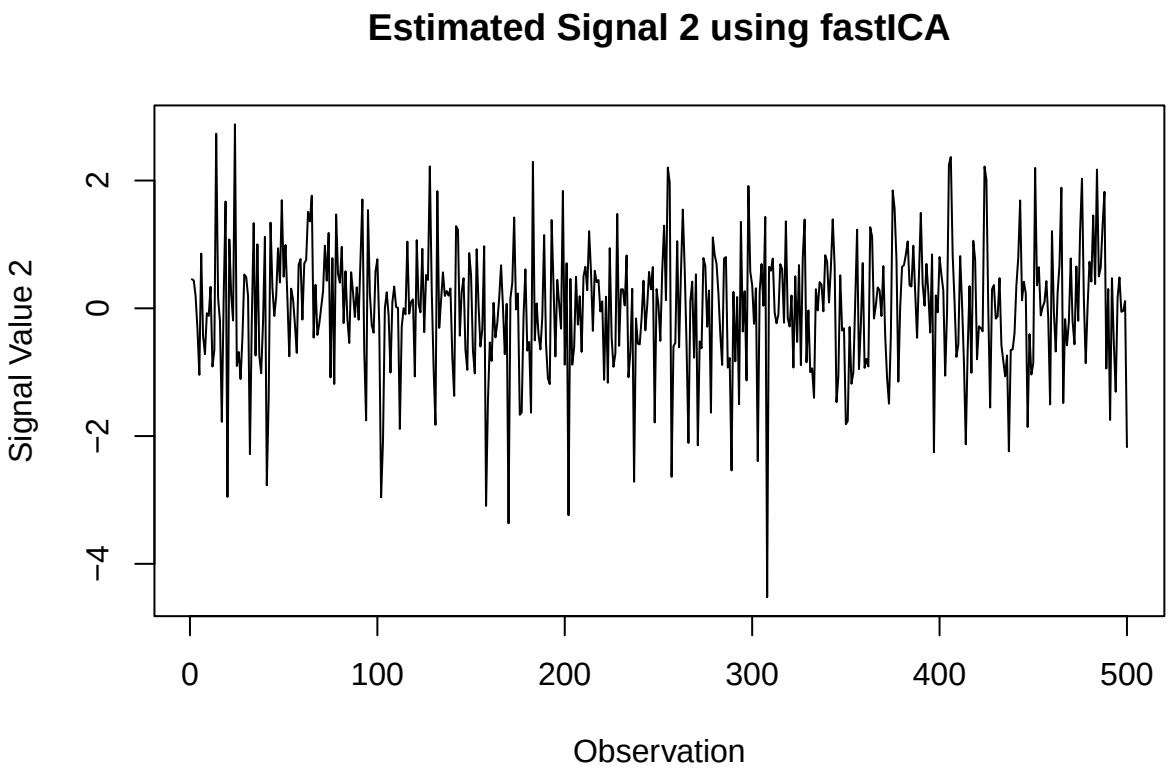
```
library(fastICA)
library(ggplot2)
library(tidyr)
library(dplyr)

# loading the data
data <- read.csv("problem4.csv")
data_matrix <- as.matrix(data)
data$index <- 1:nrow(data)
df_long <- gather(data, key = "variable", value = "value", -index)
ggplot(df_long, aes(x = index, y = value, group = variable)) +
    geom_line() +
    facet_wrap(~ variable, scales = "free", nrow = 2, ncol = 1)
```




```r
# Whitening
data_matrix_centered <- scale(data_matrix)
pca_result <- prcomp(data_matrix_centered)

# Use the PCA transformed data for ICA
ica_results <- fastICA(pca_result$x, n.comp = 2)

signal_estimate_1 <- ica_results$S[,1]
signal_estimate_2 <- ica_results$S[,2]

plot(
  signal_estimate_1, type = 'l', main = "Estimated Signal 1 using fastICA",
  xlab = "Observation", ylab = "Signal Value 1")
```




```
plot(
  signal_estimate_2, type = 'l', main = "Estimated Signal 2 using fastICA",
  xlab = "Observation", ylab = "Signal Value 2")
```

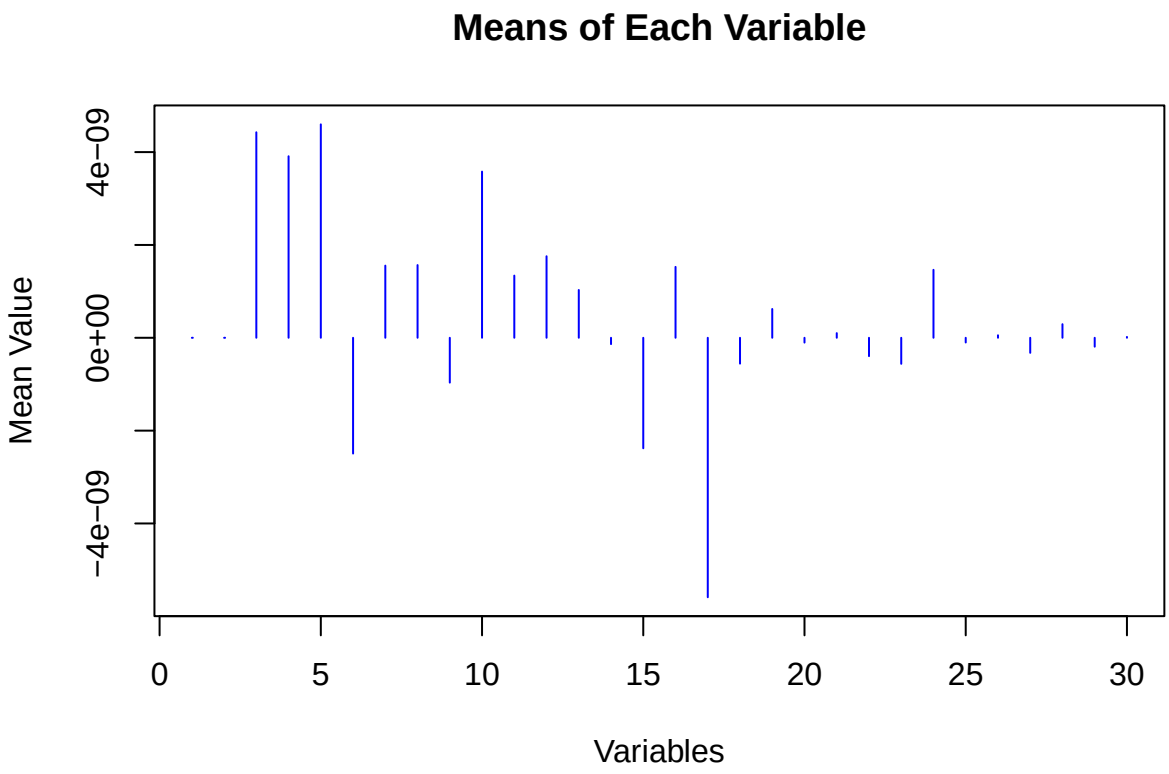


First, we are applying PCA in order to whiten the data so as not to have two Gaussian signals. After that we are applying ICA for 2 components. It is noticeable that on every run there might be differences in the obtained results. The result either shows a white noise, and a denoised periodic wave for Measurement 1, or produces analogical results for Measurement 2. This might be due to the random initialization FastICA performs in the random initial guess, or/combined with the ambiquity in the component ordering, because ICA does not have a fixed order for the independent components.

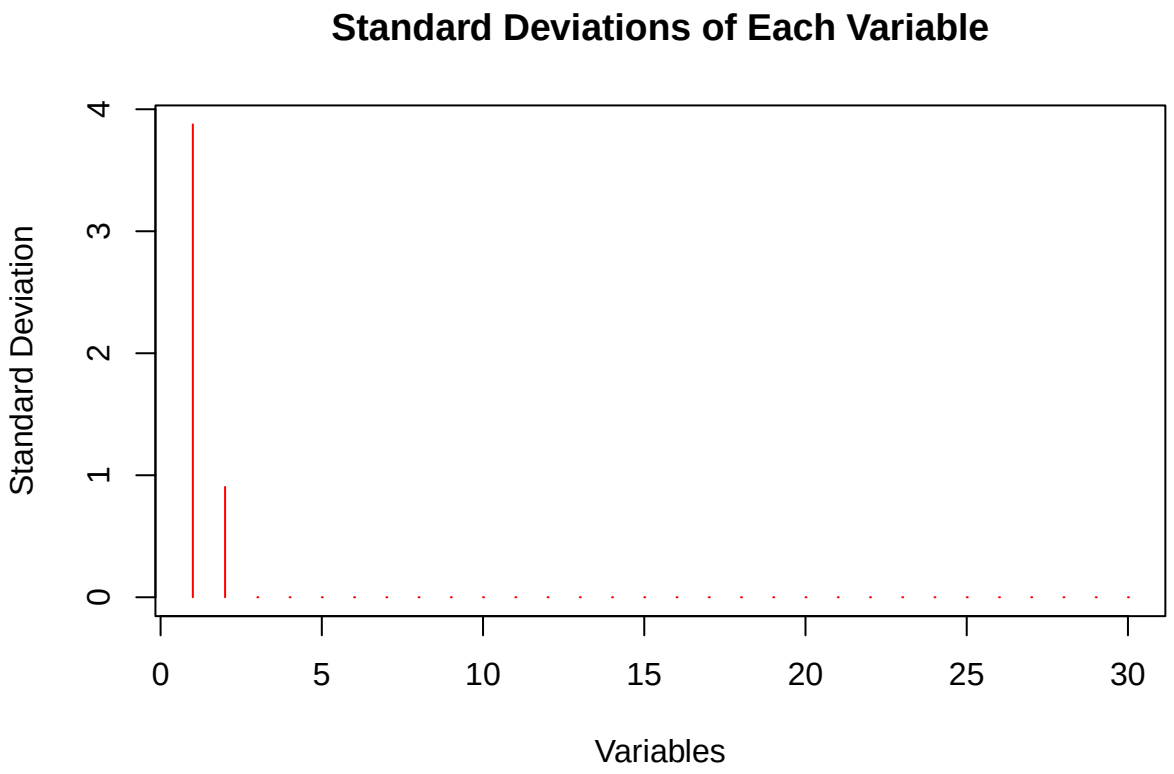
Problem 5

Applying PCA without centering (because the means of the variables are all close to 0) or scaling (because most of the variables have similar standard deviations) the data:

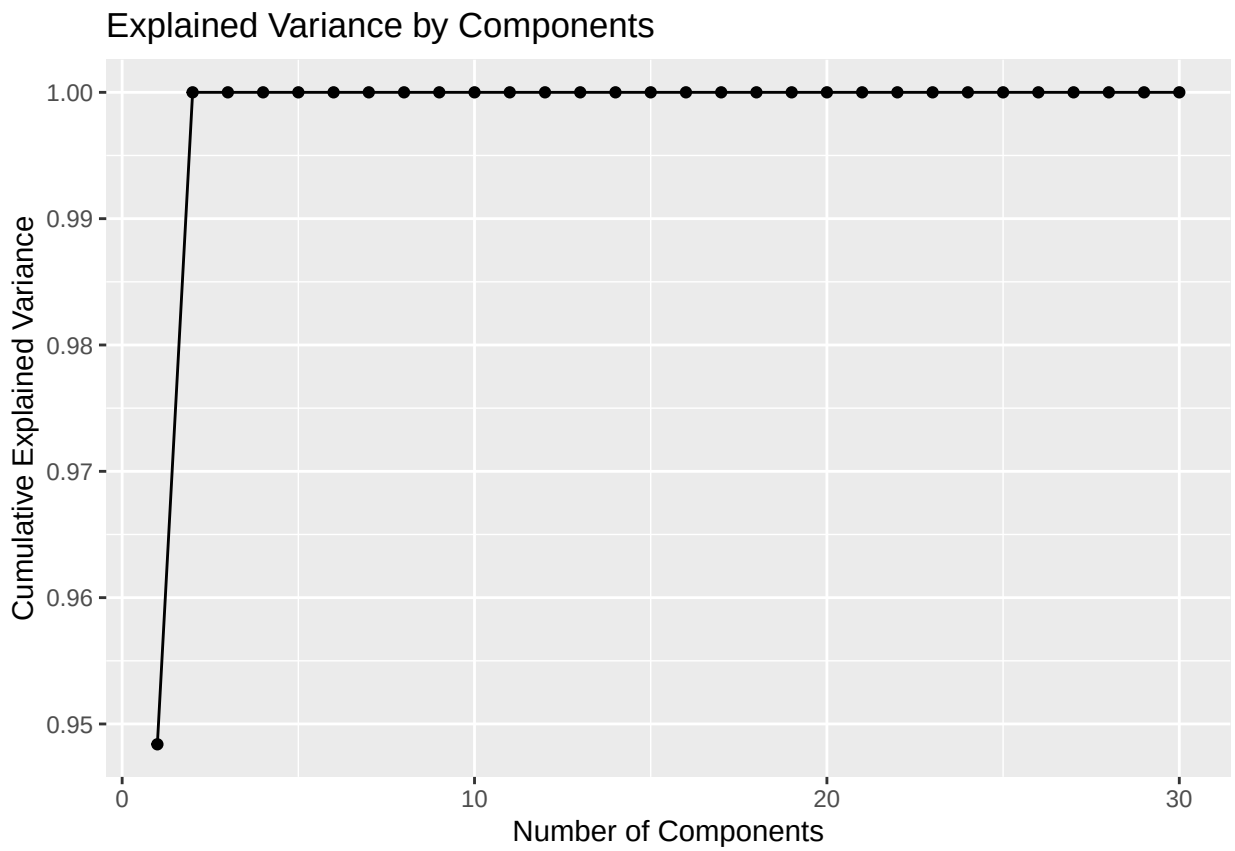
```
data <- read.csv("problem5.csv", header = FALSE)

means <- colMeans(data)
standard_deviations <- apply(data, 2, sd)

plot(
  means, type = 'h', main = 'Means of Each Variable',
  xlab = 'Variables', ylab = 'Mean Value', col = 'blue')
```




```
plot(
  standard_deviations, type = 'h', main = 'Standard Deviations of Each Variable',
  xlab = 'Variables', ylab = 'Standard Deviation', col = 'red')
```




```
pca_result <- prcomp(data, center = FALSE, scale. = FALSE)

explained_variance <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance <- cumsum(explained_variance)

ggplot(
  data = data.frame(Component = 1:length(cumulative_variance),
                    CumulativeVariance = cumulative_variance),
                    aes(x = Component, y = CumulativeVariance)) +
    geom_point() +
    geom_line() +
    xlab("Number of Components") +
    ylab("Cumulative Explained Variance") +
    ggtitle("Explained Variance by Components")
```



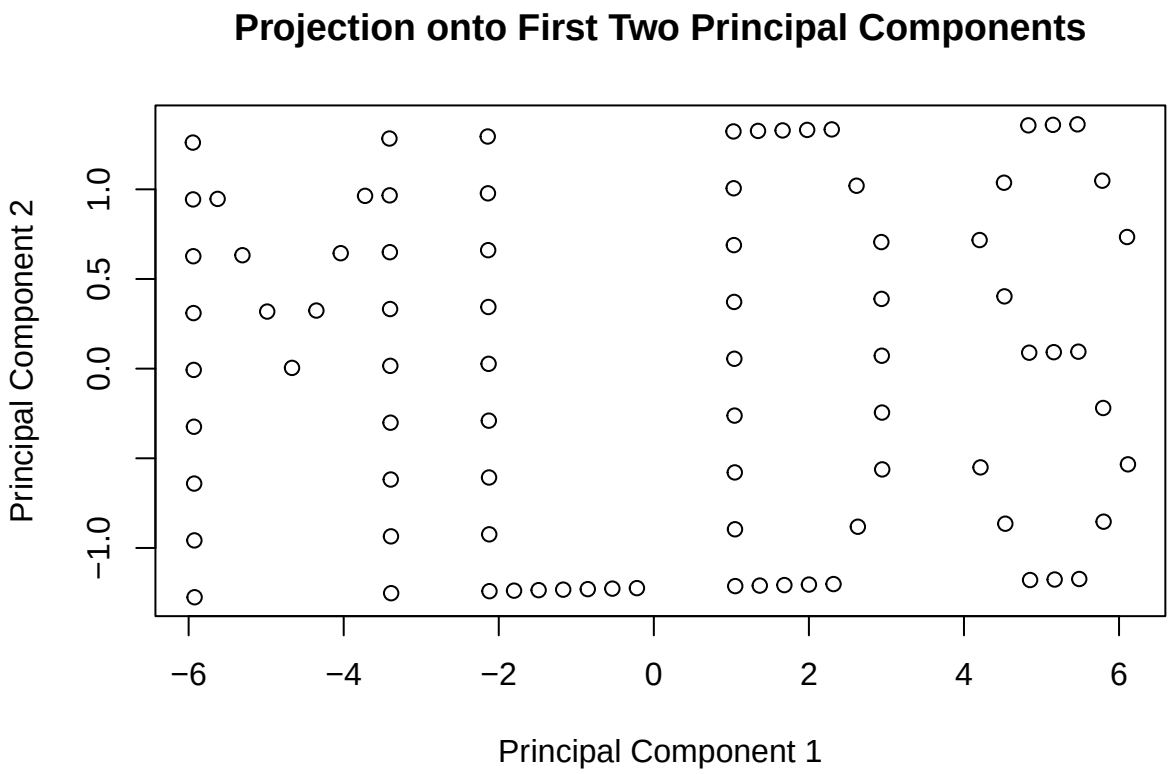
Graphically we can observe that the intrinsic dimension is 2, which is also confirmed by a calculation:

```
intrinsic_dimensions <- which(cumulative_variance >= 0.95)[1]
print(paste('Number of components that explain 95% of variance:', intrinsic_dimensions))
```

```
## [1] "Number of components that explain 95% of variance: 2"
```

Plotting the PCs (with negative signs so we mirror the results horizontally and vertically for better visualisation) will give us a plot desiphering the password, uncovering the password to be `MLDS`:

```
plot(
  -pca_result$x[,1], -pca_result$x[,2],
  xlab = "Principal Component 1",
  ylab = "Principal Component 2",
  main = "Projection onto First Two Principal Components")
```


Projection onto First Two Principal Components
Principal Component 2
1.0
0.5
0.0
−1.0
−6
−4
−2
0
2
4
6
Principal Component 1