HOME ASSIGNMENT 2

Multivariate Data Analysis

Simon Styrefors

Felix Karlsson

Oscar Lindquist

Oktober, 2025

Part I: Implement and practice Gaussian Process Regression and Principal Components Analysis in R.

Task 1: Gaussian Process Regression

Main task: Write a function to implement Gaussian Process Regression (GPR), and apply it to predict the values of y at `truth[-ID,1]`. The hyper-parameter of Gaussian noise, σ_f, can be fixed as 0.01. The hyper-parameter of the RBF kernel can be set as $1, 10, 100, 1000$ separately. Visualize the results with different values of the hyper-parameter in a plot.

Solution:

The RBF kernel is defined as

$$\kappa_{RBF}(s,t) = e^{-\sigma_k(t-s)^2},$$

where σ_k is a hyperparameter that controls the smoothness of the function.

```
load("HA2Ex1.RData")

# Hyper-parameter of Gaussian noise
sigma_f   <- 0.01
# Hyper-parameter of RBF kernel, change to 1, 10, 100, 1000
sigma_k <- 100

# RBF-kernel function
rbf_kernel <- function(s, sigma_k)
{
  D2 <- outer(s, s, function(a, b) (b - a)^2)
  K <- exp(-sigma_k * D2)

  return(K)
}
```

According to the definition of a Gaussian Process, the joint distribution of the observed values y and the values to be predicted y^* is multivariate Gaussian:

$$\begin{pmatrix} y \\ y^* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_{x,x} & \Sigma_{x,x^*} \\ \Sigma_{x^*,x} & \Sigma_{x^*,x^*} \end{pmatrix}\right).$$

```
# Build full kernel (covariance) matrix
X_values <- true_values[, 1]
cov_matrix <- rbf_kernel(X_values, sigma_k)

# Extract the different parts in the covariance matrix
cov_xx = cov_matrix [ID_obs,ID_obs]
cov_xstar_x = cov_matrix [-ID_obs,ID_obs]
cov_x_xstar = cov_matrix [ID_obs,-ID_obs] # not used
cov_xstar_xstar = cov_matrix [-ID_obs,-ID_obs] # not used
```

From this joint distribution, the predictive mean vector and covariance matrix can be derived. The predictive mean, i.e. the expected interpolation values given the observed data, is

$$\mu_{y^*} = \Sigma_{x^*,x} \left(\Sigma_{x,x} + \sigma_f^2 I\right)^{-1} y,$$

while the covariance matrix, which measures the uncertainty of the interpolation, is

$$\Sigma_{y^*} = \Sigma_{x^*,x^*} - \Sigma_{x^*,x} \left(\Sigma_{x,x} + \sigma_f^2 I\right)^{-1} \Sigma_{x,x^*}.$$

```
y_obs = as.matrix(true_values[ID_obs,2])
I = diag(length(ID_obs))
mu_star = cov_xstar_x%*%solve(cov_xx+sigma_f^2*I)%*%y_obs
mu_star
```

The predictions are plotted against the true values, and observed points are highlighted.

```
n_points = length(mu_star)
x = seq(0, 1, length.out = n_points)

# Plot mu_star against x-values
plot(x, mu_star, col="blue")
lines(true_values)
points(dat, col = "red", pch=3)
```

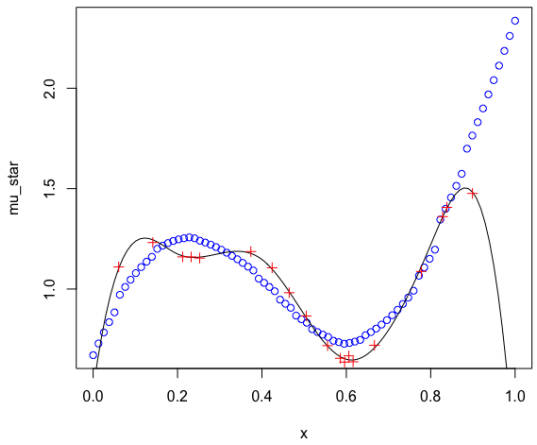

Figure 1: Plotted predictions against true values with the hyper parameter of RBF kernel = 1

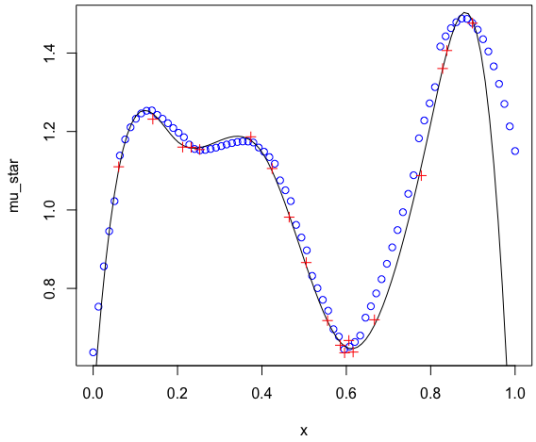

Figure 2: Plotted predictions against true values with the hyper parameter of RBF kernel = 10

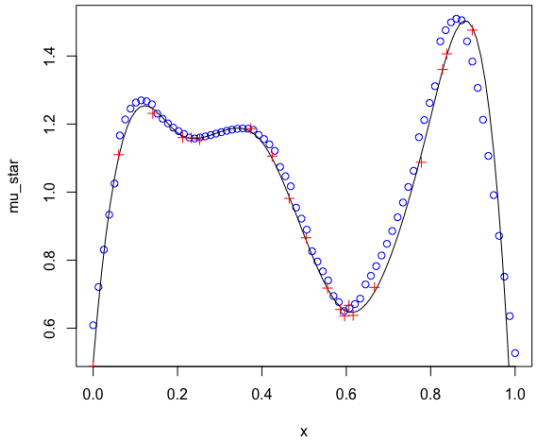

Figure 3: Plotted predictions against true values with the hyper parameter of RBF kernel = 100

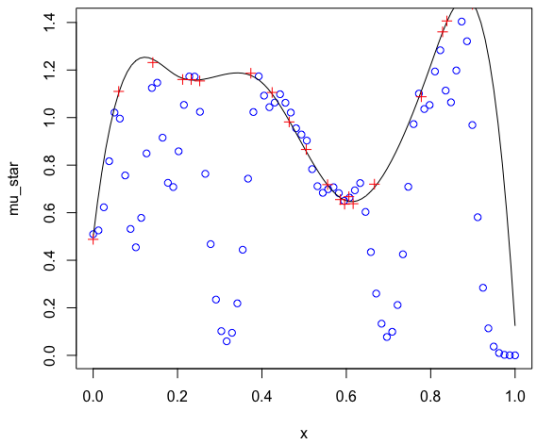

Figure 4: Plotted predictions against true values with the hyper parameter of RBF kernel = 1000

Results and Discussion

The experiments with different values of the RBF kernel hyperparameter σ_k show how model flexibility and smoothness are affected:

- For $\sigma_k = 1$ (Figure 1), the predictions are very smooth and fail to capture the variations of the true function. This indicates underfitting.
- For $\sigma_k = 10$ and $\sigma_k = 100$ (Figures 2 and 3), the predictions align well with the true function. The model captures the underlying structure without overfitting the noise.
- For $\sigma_k = 1000$ (Figure 4), the predictions fluctuate strongly around the true function and do not provide stable interpolation. This indicates overfitting, where the model becomes too sensitive to small variations in the data.

Task 2: PCA on handwritten digit data

Main task: Implement Principal Component Analysis (PCA) in R.

1. Read in the dataset, choose a digit, and display the first 24 images.
2. Perform PCA and visualize the first 4 and last 4 principal vectors (eigenvectors) as 16×16 images.
3. Reconstruct a chosen image with the first $q \in \{30, 60, 100, 150, 200\}$ principal components and report on the MSE.
4. Calculate how many principal components do you need if you want to keep 85% of information from the original dataset.
5. With the same idea as Task 2 in HA1, build a classifier based on the first two principal components (PCs) to classify the images of digits 5 and 6.

Task 2.1: Data and visualization of the first 24 images

We begin by loading the dataset and selecting all images corresponding to the chosen digit 5. Each image is represented as a 16×16 grayscale pixel image with values between 0 (white) and 1 (black). We then display these first 24 handwritten samples of digit 5.

```
# Load data and subset chosen digit five
data <- read.table(gzfile("zip.train"))
data <- as.matrix(data)

five <- data[which(data[,1] == 5), 2:257]  # all "5" digits

colors  <- c('white','black')
cus_col <- colorRampPalette(colors=colors)

z <- matrix(five[8, 256:1], 16, 16, byrow = TRUE)[, 16:1]
image(t(z), col = cus_col(256), axes = FALSE)

#First 24 images
par(mfrow = c(4,6), mar=c(0.2,0.2,0.2,0.2))
for (i in 1:24) {
  z <- matrix(five[i, 256:1], 16, 16, byrow = TRUE)[, 16:1]
  image(t(z), col = cus_col(256), axes = FALSE)
}
par(mfrow=c(1,1))
```

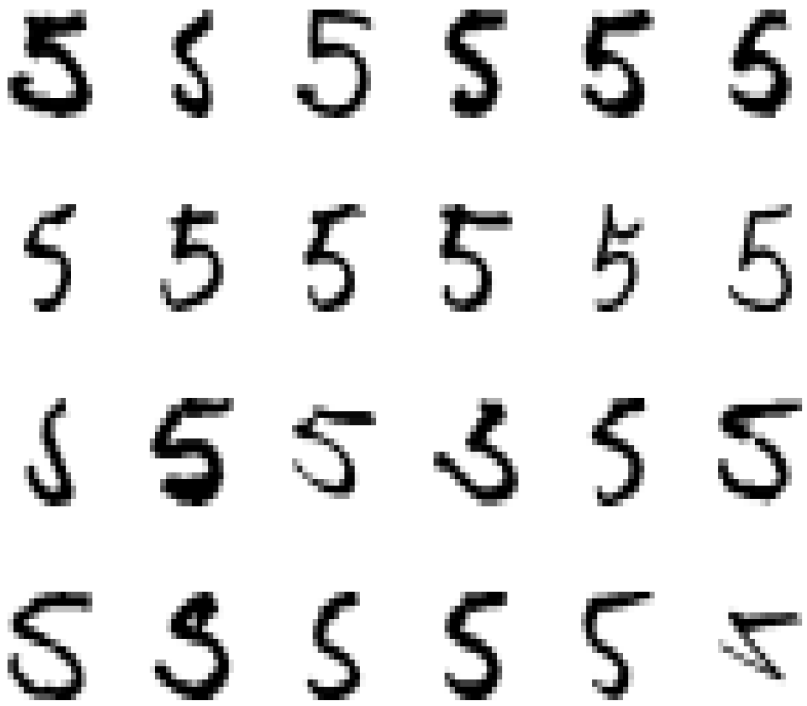

Figure 5: The first 24 samples of digit 5 from `zip.train`.

Task 2.2: PCA implementation and visualization of principal vectors

To identify the dominant directions of variation among the handwritten digits, we implement PCA. This is performed by first centering the data, then computing the sample covariance matrix.

$$S = \frac{1}{n-1} X^{*\top} X^*,$$

where the demeaned data matrix is

$$X^* = X - \mathbf{1}\bar{x}^\top.$$

The eigen decomposition of S is

$$S = Q\Lambda Q^\top,$$

where $Q = (q_1, q_2, \ldots, q_{256})$ contains the eigenvectors and Λ is the diagonal matrix of eigenvalues.
For the i-th observation and j-th component, the principal component scores becomes

$$z_{ij} = q_j^\top (x_i - \bar{x}), \quad i = 1, \ldots, n,\ j = 1, \ldots, 256.$$

All scores gives us the new data matrix

$$Z = X^* Q,$$

```r
X <- five
p <- ncol(X)

mu     <- colMeans(X)
X_star <- sweep(X, 2, mu, "-")

# Sample covariance
S <- crossprod(X_star) / (nrow(X_star) - 1)

# Eigen-decomp.
eig  <- eigen(S, symmetric = TRUE)
vals <- eig$values
Q    <- eig$vectors  # eigenvectors (principal directions)

plot_pvec <- function(v, idx) {
  z <- matrix(v[256:1], 16, 16, byrow = TRUE)[, 16:1]
  image(t(z), col = cus_col(256), axes = FALSE)
  title(main = paste0("PC ", idx))
}

# Plotting first 4 and last 4
par(mfrow = c(2,4), mar=c(0.5,0.5,1.2,0.5))
for (j in 1:4) plot_pvec(Q[, j], j)
for (j in (p-3):p) plot_pvec(Q[, j], j)
par(mfrow=c(1,1))
```

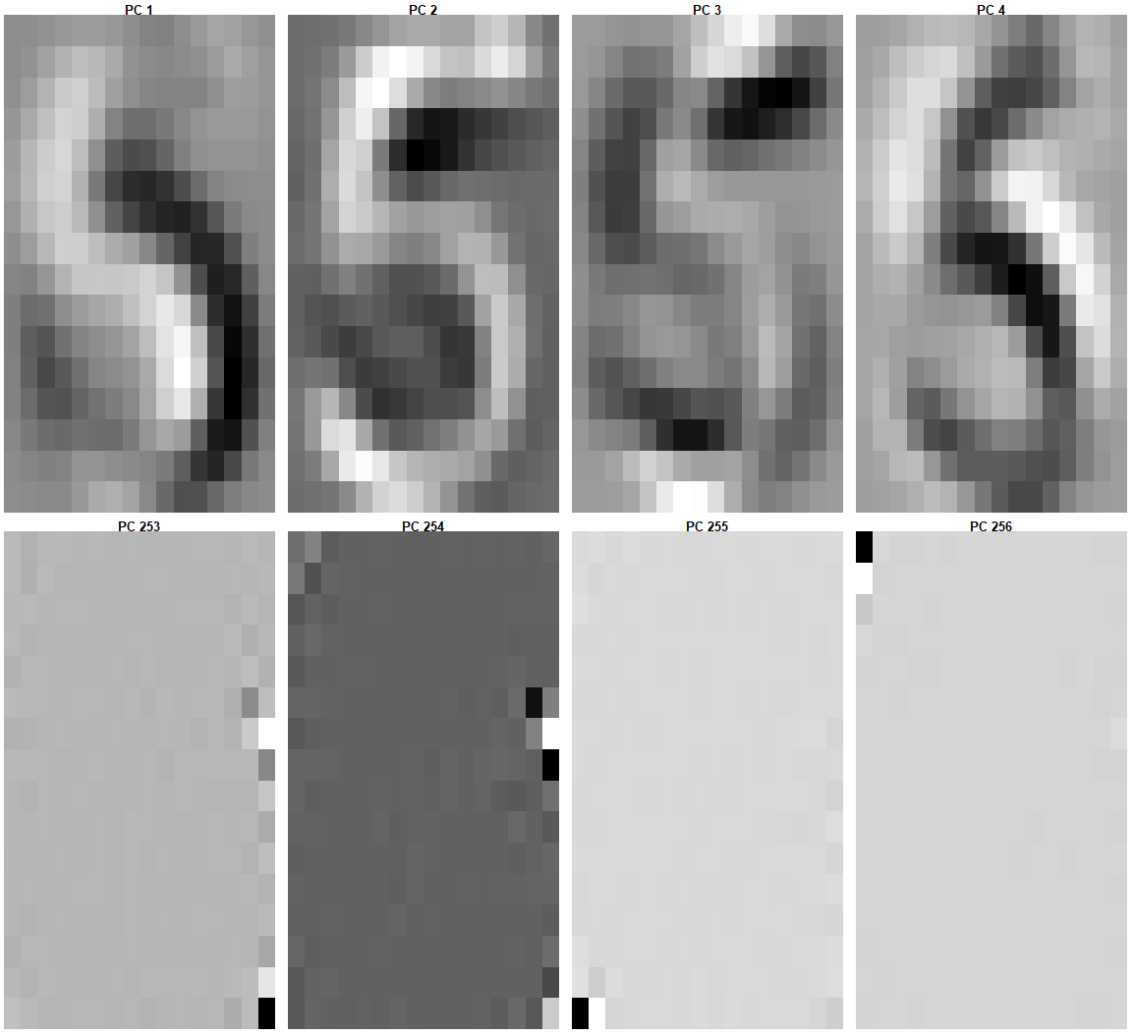

Figure 6: The first 4 (largest variance) and last 4 (smallest variance) principal vectors visualized as 16×16 images.

The first principal vectors capture the overall "stroke" structure of the digit, while the last ones mainly represent noise and fine-tuning variations with little variance explained.

Task 2.3: Image reconstruction and MSE

In PCA, each centered image $x_i - \mu$ can be expanded in the eigenbasis as

$$x_i = \mu + \sum_{j=1}^{p} z_{ij} q_j, \qquad z_{ij} = q_j^\top (x_i - \mu),$$

where q_j is the j-th eigenvector and z_{ij} its corresponding principal component score.
If we only keep the first q components, we obtain the rank-q approximation

$$\hat{x}_i^{(q)} = \mu + \sum_{j=1}^{q} z_{ij} q_j.$$

The reconstruction error for image i is measured by the per-pixel mean squared error

$$\mathrm{MSE}^{(q)} = \frac{\|x_i - \hat{x}_i^{(q)}\|^2}{p}, \qquad p = 256.$$

```
# Scores for all observations
Y <- X_star %*% Q  # n x p

# We choose digit 5
sample_id <- 5
x_obs     <- X[sample_id, ]

# Reconstruct with first q PCs
reconstruct_with_q <- function(sample_id, q, mean_vec, Y, Q) {
  q       <- min(q, ncol(Q))
  eig_q   <- Q[, 1:q, drop = FALSE]
  x_star  <- Y[sample_id, 1:q, drop = FALSE] %*% t(eig_q)
  as.vector(mean_vec + x_star)
}

q_vals   <- c(30, 60, 100, 150, 200)
recons   <- lapply(q_vals, function(k) reconstruct_with_q(sample_id, k, mu, Y,
    Q))
mse_vals <- sapply(recons, function(xhat) mean((x_obs - xhat)^2))

# We plot original + five reconstructions
plot_digit <- function(img_vec, main = "") {
  z <- matrix(img_vec[256:1], 16, 16, byrow = TRUE)[, 16:1]
  image(t(z), col = cus_col(256), axes = FALSE, main = main)
}

par(mfrow = c(2,3), mar = c(0.5,0.5,2,0.5))
plot_digit(x_obs, main = "Original")
for (i in seq_along(q_vals)) {
  plot_digit(recons[[i]], main = paste0("q␣=␣", q_vals[i]))
}
par(mfrow=c(1,1))

# Table of MSEs
data.frame(q = q_vals, MSE = mse_vals)
```

Figure 7: Original image and reconstructions using $q \in \{30, 60, 100, 150, 200\}$ principal components.

q	MSE
30	0.0486
60	0.0254
100	0.0085
150	0.0026
200	0.0004

Table 1: MSE for reconstructions with different numbers of principal components. (Values from `msevalues` in R.)

We observe that the reconstruction quality improves as the number of retained PCs increases. With only $q = 30$ components the overall shape of the digit is visible, however some details are blurry. At $q = 100$ the reconstruction is already close to the original, and by $q = 200$ the MSE is nearly zero. This shows us the trade-off between dimensionality reduction and approximation accuracy.

Task 2.4

We now ask how many principal components are needed to retain at least 85% of the variance in the dataset. Formally, we seek the smallest q such that

$$\frac{\sum_{i=1}^{q} \lambda_i}{\sum_{i=1}^{p} \lambda_i} \geq 85\%,$$

where λ_i are the eigenvalues of the sample covariance matrix.

```
explained <- vals / sum(vals)   # sum(vals) is total variance
cum_explained <- cumsum(explained)

eighty_five <- which(cum_explained >= 0.85)[1]

cat("Number of principal components needed for 85% variance:",
    eighty_five, "\n")
```

```
Number of principal components needed for 85% variance: 34
```

Thus, only 34 PCs (out of 256) are enough to preserve 85 percent of the information.

Task 2.5

The goal of this task is to build a classifier that distinguishes between digits 5 and 6, using the first two principal components (PCs) of the digit images. The same method as Task 2 in HA1 was applied, using Quadratic Discriminant Analysis (QDA). The following steps was used.

1. Create a new working dataset (digits 5 and 6 only)

We begin by extracting only the digit images corresponding to classes 5 and 6 from the dataset.

```
data <- read.table(gzfile("zip.train"))
data <- as.matrix(data)
subset <- data[data[,1] %in% c(5,6), ]
labels <- subset[,1]
X      <- subset[, 2:257]
```

2. Split the dataset into training (80%) and testing (20%) sets

The dataset is randomly split into training and test subsets. A fixed random seed was used for reproducibility of the results.

```
set.seed(123)
n  <- nrow(X)
id <- sample.int(n, floor(0.8*n))
X_train <- X[id, ];   y_train <- labels[id]
X_test  <- X[-id, ];  y_test  <- labels[-id]
```

3. Perform PCA on the training set and project both sets

Principal Component Analysis is performed on the training set only. The eigenvectors of the training covariance matrix are used to project both the training and test sets. In this step, only the first two eigenvectors are kept.

```
mu_train        <- colMeans(X_train)
X_star     <- sweep(X_train, 2, mu_train, "-")
S_train         <- crossprod(X_star) / (nrow(X_star) - 1)
eig_train       <- eigen(S_train, symmetric = TRUE)
Q           <- eig_train$vectors[, 1:2, drop = FALSE]

PC_train     <- X_star %*% Q
PC_test      <- sweep(X_test, 2, mu_train, "-") %*% Q
```

4. Build the classifier using training PCs

From the training set, we estimate the class priors, means, and covariance matrices for each class (digit 5 and digit 6).

```
class_5 <- 5; class_6 <- 6
idx_5 <- which(y_train == class_5)
idx_6 <- which(y_train == class_6)
n_5 <- length(idx_5); n_6 <- length(idx_6); n_tr <- nrow(PC_train)

pi_5 <- n_5 / n_tr
```

```r
pi_6 <- n_6 / n_tr

mu_5 <- colMeans(PC_train[idx_5, , drop = FALSE])
mu_6 <- colMeans(PC_train[idx_6, , drop = FALSE])

Sigma_5 <- cov(PC_train[idx_5, , drop = FALSE])
Sigma_6 <- cov(PC_train[idx_6, , drop = FALSE])
```

The QDA log-score function is then implemented the same way as in HA1:

```r
qda_logscore <- function(X, mu_vec, Sigma, prior) {
  d <- ncol(X)
  log_const <- -0.5 * d * log(2*pi)
  cholS  <- chol(Sigma)
  Xc     <- sweep(X, 2, mu_vec, "-")
  z      <- t(backsolve(cholS, t(Xc), transpose = TRUE))
  maha   <- rowSums(z^2)
  logdet <- sum(log(diag(cholS)))
  ll     <- log_const - logdet - 0.5 * maha
  log(prior) + ll
}

qda_predict <- function(Xnew) {
  s_5 <- qda_logscore(Xnew, mu_5, Sigma_5, pi_5)
  s_6 <- qda_logscore(Xnew, mu_6, Sigma_6, pi_6)
  ifelse(s_6 > s_5, class_6, class_5)
}
```

5. Apply classifier to testing set PCs and calculate accuracy

The QDA classifier is applied to the test data and its accuracy is evaluated.

```r
pred_qda <- qda_predict(PC_test)
acc_qda <- mean(pred_qda == y_test)
cat(sprintf("Accuracy: %.4f\n", acc_qda))
```

Results: The classifier achieved an accuracy of:

```
Accuracy: 0.9713
```

Can you achieve such accuracy if only use two variables from the original dataset to build the classifier?

No, if you only use two variables from the original dataset (in this case two pixels), you won't reach the same accuracy. The two first principal components combine information from all 256 pixels to capture the overall shape differences between 5 and 6. Two single pixels only give a very small piece of information, so the classifier will lose a lot of power and the accuracy will be much lower.

However, if the difference between two classes was concentrated in just a few pixels, then choosing those pixels as variables could already give a high accuracy. But for handwritten digits like 5 and 6, the differences are spread out across the whole image. No single pixel, or even two pixels, have enough of that information.

Part II: Theoretical problems

Exercise 8.3

Problem Description

Let

$$\Sigma = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

Determine the principal components Y_1, Y_2, and Y_3. What can you say about the eigenvectors (and principal components) associated with eigenvalues that are not distinct?

Solution

To get the principal components, we first find our eigenvalues and eigenvectors by the following equation.

$$\det(\Sigma - \lambda I) = 0$$

This gives us:

$$\lambda_1 = 2$$

$$\lambda_2 = 4$$

$$\lambda_3 = 4$$

For λ_1 the eigenvector is $v_1 = (1,0,0)^T$, for λ_2 the eigenvector is $v_2 = (0,1,0)^T$ and λ_3 is $v_3 = (0,0,1)^T$. The principal components are defined after eigenvectors and the following equation.

$$Y_i = v_i^T X.$$

Therefore, the solution gives

$$Y_1 = X_1, \quad Y_2 = X_2, \quad Y_3 = X_3$$

with variances $Var(Y_1) = 2$ and $Var(Y_2) = Var(Y_3) = 4$.
Since $\lambda_2 = \lambda_3 = 4$, the eigenvalue is not distinct, which means that the choice of v_2, v_3 is not unique. Hence, the corresponding principal components are not unique either. This implies that any orthogonal linear combinations of X_2 and X_3 are valid principal components and has equal variance explained.

Exercise 8.4

Problem Description

Calculate the proportion of the total population variance explained by each principal component when the covariance matrix is

$$\Sigma = \begin{pmatrix} \sigma^2 & \sigma^2\rho & 0 \\ \sigma^2\rho & \sigma^2 & \sigma^2\rho \\ 0 & \sigma^2\rho & \sigma^2 \end{pmatrix},$$

where $-\frac{1}{\sqrt{2}} < \rho < \frac{1}{\sqrt{2}}$.

Solution

First step is to find the trace of the matrix, which is the sum of the diagonal elements in our matrix. This gives us $Ttr(\sum) = 3\sigma^2$ which equals to total variance. Next step is to find the eigenvalues of the matrix.

$$\lambda_1 = \sigma^2(1+\sqrt{2}p)$$
$$\lambda_2 = \sigma^2(1-\sqrt{2}p)$$
$$\lambda_3 = \sigma^2$$

The proportion of variance explained by each principal component can be calculated through eigenvalues divided of total variance that we have calculated above. This gives the following proportion of total variance explained. All these proportions explain how much of the total variance is explained in each principal component.

$$Y_1 = \frac{1+\sqrt{2}p}{3}$$
$$Y_2 = \frac{1-\sqrt{2}p}{3}$$
$$Y_3 = \frac{1}{3}$$