2 Part II: Programming Problems

2.1 Task 1

2.1.1 Method

Firstly, we generated 1000 pairs of standard normal random numbers by the Box and Muller's algorithm:

1. Independently generate u_1 and u_2 from the uniform distribution, $U(0,1)$.
2. Set $x = \sqrt{-2\log(u_1)\cos(2\pi u_2)}$ and $x = \sqrt{-2\log(u_1)\sin(2\pi u_2)}$. Then, x and y are independent and normally distributed with zero mean and unit variance.

Secondly, the standard normal random numbers $\mathbf{Z}$ were adjusted for the given mean vector $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$ and correlation ρ such that

$$\mathbf{X} = \boldsymbol{\mu} + A\mathbf{Z},$$

where A is the Cholesky matrix, i.e.,

$$A = \begin{bmatrix} \sigma_1 & 0 \\ \rho\sigma_2 & \sigma_2\sqrt{1-\rho^2} \end{bmatrix},$$

in the two-dimensional case. Hence, $\mathbf{X}$ is $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ distributed with ρ-correlated variables. Lastly, the generated numbers were visualized in a scatter plot and the sample correlation was calculated.

2.1.2 Conlusions

As expected, our generated sample of two-dimensional Gaussian random numbers yields a scatter plot with a downward slope proving the negative correlation. In addition, the sample correlation is fairly close to -0.60 (as expected).

2.1.3 Source Code

Box and Muller's algorithm, as in first step.

```
# generate uniform random numbers by box-muller algorithm
bm = function(n) {
    u1 = runif(n)
    u2 = runif(n)
    x = sqrt(-2 * log(u1)) * cos(2 * pi * u2)
    y = sqrt(-2 * log(u1)) * sin(2 * pi * u2)
    return(rbind(x, y))
}
```

Two-dimensional Gaussian random numbers with arbitrary mean and covariance matrix, as in second step.

```
# generate multivariate gaussian numbers
mvn_num = function(sn_num, mu_vec, sig_vec, rho) {
    dim = dim(sn_num)[1]
    sig_mat = diag(1, dim, dim) * sig_vec
    cor_mat = matrix(1, dim, dim) * rho
    diag(cor_mat) = 1
    cov_mat = sig_mat %*% cor_mat %*% sig_mat
    A = t(chol(cov_mat))
    mvn_sample = mu_vec + A %*% sn_num
    return(mvn_sample)
}
```

Generation and visualization of $n = 1000$ two-dimensional Gaussian random numbers where $\boldsymbol{\mu} = (3, 1)^\intercal$, $\boldsymbol{\sigma} = (1, 2)$ and $\rho = -0.60$.

```
n = 1000
sn_sample = bm(n)
mu = c(3, 1)
sig = c(1, 2)
rho = -.6
mvn = mvn_num(sn_sample, mu, sig, rho)
sample_rho = cor(mvn[1,], mvn[2,])
```

```
plot(
    mvn[1,], mvn[2,],
    xlab=expression(x[1]), ylab=expression(x[2]),
    col=3,
    main=paste("Sample Correlation:", round(sample_rho, 3), sep=" ")
)
grid()
```

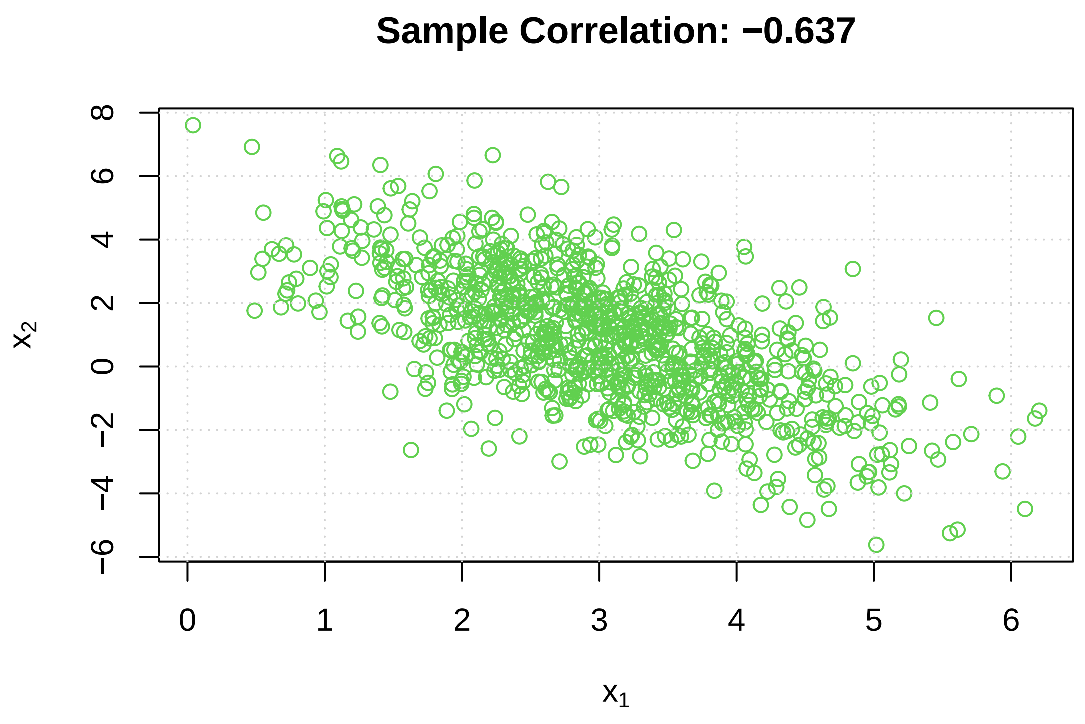


Figure 1: *Scatter plot of 1000 two-dimensional Gaussian random numbers with correlation -0.60. The sample correlation is, as expected, close to -0.60.*

2.2 Task 2

2.2.1 Probability Model

In this task, a linear discriminant analysis (LDA) is performed. Without going through the details, LDA, for two-class classification, approaches the problem by assuming that the conditional density functions $P(\boldsymbol{x}|y=0)$ and $P(\boldsymbol{x}|y=1)$ are normally distributed with mean and covariance matrix $(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, respectively. Further, LDA makes the assumption of homoscedasticity, i.e., $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}$. Then, by assuming equal densities, i.e., $f_1(\boldsymbol{x}) = f_2(\boldsymbol{x})$, the decision boundary is given by (omitting some steps)

$$(\boldsymbol{\mu_2} - \boldsymbol{\mu_1})\boldsymbol{\Sigma^{-1}}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{\mu_1} + \boldsymbol{\mu_2})\boldsymbol{\Sigma^{-1}}(\boldsymbol{\mu_2} - \boldsymbol{\mu_1}).$$

The distribution parameters are estimated through maximum likelihood estimation on each group of the training data, and the predictions on the test set are

$$\hat{y} = \arg\min_{y} \log P(\boldsymbol{x}|y)P(y).$$

Mathematically speaking, the input $\boldsymbol{x}$ being in class y is solely a function of this linear combination of the known observations. In layman's terms, we consider the class means as well as the within-class variance. A greater difference in means and a smaller within-group variance, yields more distinct classifications. Ideally, the projected classes have both faraway means and small variances.

2.2.2 Method

Firstly, train and test data was read from external files. Then, LDA was performed by fitting a corresponding probability model to the training data. Consequently, the fitted model was used to predict the cancer diagnosis in the test set. Lastly, the confusion matrix and test accuracy were calculated.

2.2.3 Conclusions

The confusion matrix showed that we incorrectly predicted three Benign cases and only one Malign, which resulted in an test accuracy of 96%. These results indicates that LDA is a feasible approach for this problem.

2.2.4 Source Code

Import needed libraries for LDA model and confusion matrix.

```r
library(MASS)
library(caret)
```

Read data.

```r
# read data
train = read.table(file="Train.txt", header=T, sep=",")
test = read.table(file="Test.txt", header=T, sep=",")

# convert to numeric, factor response
train$Diagnosis = factor(as.numeric(train$Diagnosis == "M"))
test$Diagnosis = factor(as.numeric(test$Diagnosis == "M"))
```

Fit model and perform predictions on the test set. Then, compute confusion matrix and test accuracy. We incorrectly predicted three Benign cases and only one Malign, which resulted in an accuracy of 96%.

```
# fit model and predict
mdl = lda(
    Diagnosis ~ .,
    data=train
)
pred = predict(mdl, test)$class

# create confusion matrix
cm = confusionMatrix(
    data=pred,
    reference=test$Diagnosis
)
cm$table
```

```
##           Reference
## Prediction  0  1
##          0 58  3
##          1  1 38
```

```
# compute accuracy
cm$overall['Accuracy']
```

```
## Accuracy
##     0.96
```

2.3 Task 3

2.3.1 Probability models

The task was firstly to simulate 1000 realizations from a two dimensional Gaussian distribution,

$$\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

with mean vector $\boldsymbol{\mu} = (2,3)^\top$, variances where $\sigma_1^2 = 1, \sigma._2^2 = 4$ and correlation $\rho = 0.7$. Based on this the following covariance matrix could be computed, $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 1.4 \\ 1.4 & 4 \end{bmatrix}$.

The task was then to simulate 1000 realizations from a two dimensional Gaussian Mixed Model(GMM) which was defined by a latent variable z_i belonging to a Bernoulli distribution with parameter $p = 0.6$ for $i = 1, \ldots, 1000$. The conditional distribution given the value of z_i is

$$\mathbf{X}_i \mid z_i = 1 \sim \mathcal{N}_2\left(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) \text{ and } \mathbf{X}_i \mid z_i = 0 \sim \mathcal{N}_2\left(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right)$$

where $\boldsymbol{\mu}_1 = (2,3)^\top$ and $\boldsymbol{\mu}_2 = (3,2)^\top$. Regarding $\boldsymbol{\Sigma_1}$, we have that $\sigma_1 = 0.2$, $\sigma_2 = 0.6$ and correlation $\rho = 0.7$. With respect to $\boldsymbol{\Sigma_2}$, we have that $\sigma_1 = 0.2$, $\sigma_2 = 0.6$ and correlation $\rho = 0.7$. This gives us the following two covariance matrixes $\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.04 & 0.06 \\ 0.06 & 0.36 \end{bmatrix}$, $\boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.16 & 0.06 \\ 0.06 & 0.09 \end{bmatrix}$.

2.3.2 Method

Subtask 3.1 The simulation of the 1000 realizations from a two dimensional Gaussian distribution was done through the built-in function `rmvnorm` in R. After the realizations where computed these where plotted in a scatter plot.

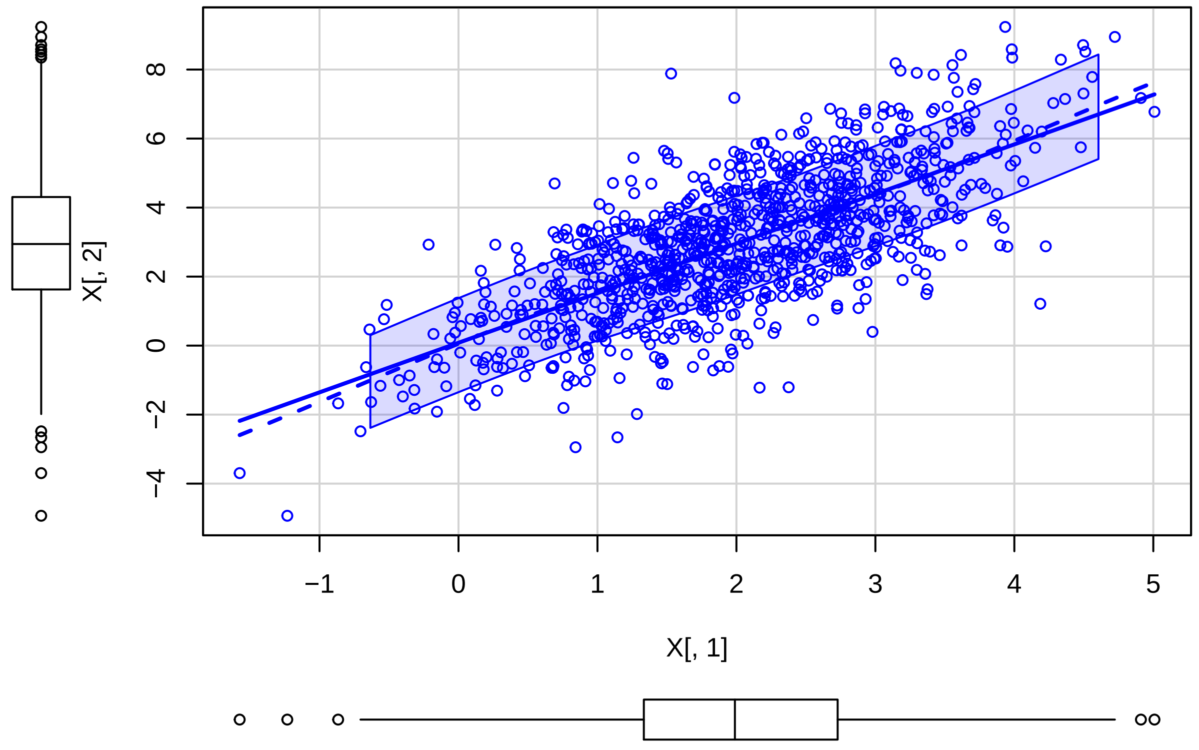
The initial parameters are defined and the 1000 realizations are simulated.

```r
n = 1000
mu = c(2,3)
cov_mat = cbind(c(1, 1.4),c(1.4,4))

X = rmvnorm(n,mu,cov_mat)
```

A scatter plot is made of all the realizations in order to check the correlation.

```r
scatterplot(X[,1],X[,2])
```



Subtask 3.2 To begin the simulation of the two dimensional Gaussian distributed conditional variables, 1000 Bernoulli distributed random variables where generated as the z_i:s mentioned in section 3.1. These 1000 Bernoulli distributed variables where then examined one by one in order to control whether they had the value 0 or 1. Based on this the $\mathbf{X}_i$:s was simulated using the built-in function `rmvnorm` in R.

The inital parameters are defined and the 1000 Bernoulli distributed random variables are simulated.

```r
X = matrix(0,1000,2)
mu1 = c(2,3)
mu2 = c(3,2)

cov_mat1 = cbind(c(0.04, 0.06),c(0.06,0.36))
cov_mat2 = cbind(c(0.16, 0.06),c(0.06,0.09))
```

```
p = 0.6
n = 1000
B = rbern(n,p)
```

A loop is created over all the Bernoulli variables in order to simulate the two dimensional Gaussian distributed conditional variables.

```
for (i in 1:1000) {
  if (B[i]==1) {
    x = rmvnorm(1,mu1,cov_mat1)
    X[i,1] = x[1]
    X[i,2] = x[2]
  }
  else if (B[i]==0) {
    x = rmvnorm(1,mu2,cov_mat2)
    X[i,1] = x[1]
    X[i,2] = x[2]
  }
}
```

Subtask 3.3 To finish off, the task was to fit a GMM based on the 1000 simulated variables from sub task 3.2. This was simply done by the built-in function `mvnormalmixEM` in R. By fitting a GMM one also estimates the parameters p, μ_1 and μ_2. These can then be compared to the true values which where presented in section 3.1.

The GMM is fitted using the variables obtained from subtask 3.2.

```
#3.3
mdl = mvnormalmixEM(X)
```

```
## number of iterations= 29
```

The the estimates from the given GMM are presented below.

Table 1: *The estimates for μ_1,μ_2 and p extracted from the GMM fitted by `mvnormalmixEM` in R.*

Estimates	μ_1	μ_2	p
$\mathbf{X}_i \mid z_i = 1$	1.99	3.01	0.59
$\mathbf{X}_i \mid z_i = 0$	3.00	2.00	0.41

2.3.3 Conclusions

Subtask 3.1 was easily computed and the results turned out to be just as one would have assumed. A trivial way to verify this is to look at the scatter plot and notice that the correlation between the variables looks to be round about 0.7. Since this was defined in the task description this is of course very trivial but still ensures that our results are relevant.

Since subtask 3.2 did not constitute any real output it is difficult to examine and discuss the results. Subtask 3.3 does in many ways control that the simulation done in subtask 3.2 is correct and hence acts as a conclusion of the results in subtask 3.2 as well.

In subtask 3.3, the GMM does a great job of estimating the different parameters as can be clearly stated by the results obtained in Table 1. The estimates sometimes differ by 0.01 which although it is a difference still should be considered a very good estimate. One should also mention that these results conclude that the results from subtask 3.2 seems to be correct as well.

2.4 Task 4

2.4.1 Probability models

Bayes formula:

$$\operatorname{Pr}(y \mid \mathbf{x})=\frac{\operatorname{Pr}(\mathbf{x} \mid y) \operatorname{Pr}(y)}{\operatorname{Pr}(x)} \tag{1}$$

Law of total probability:

$$Pr(A)=P\left(A \mid B_{X}\right) \cdot Pr\left(B_{X}\right)+Pr\left(A \mid B_{Y}\right) \cdot Pr\left(B_{Y}\right) \tag{2}$$

Binomial distribution:

$$\operatorname{Pr}(y=n)=\left(\begin{array}{c} N \\ n \end{array}\right) \pi^{n}(1-\pi)^{N-n} \tag{3}$$

2.4.2 Method

Suppose the unknown probabilities of getting head for the two different coins A and B are denoted q_A and q_B respectively. Further, assume that coin A is chosen with a probability q while coin B is chosen with probability $1-q$.

Firstly, initial guesses of q, q_A and q_B are randomly simulated from the uniform distribution. The expectation step is then performed using both equation (1) and (2). Equation 1 (Bayes formula) is used for evaluating the probability that the coin picked is A given head and the current guess of q as follows:

$$\operatorname{Pr}(A \mid \mathbf{x})=\frac{\operatorname{Pr}(\mathbf{x} \mid A) \operatorname{Pr}(A)}{\operatorname{Pr}(\mathbf{x})}$$

where the probability of getting head, $\operatorname{Pr}(\mathbf{x})$, is evaluated using equation 2 (Law of total probability) as follows:

$$\operatorname{Pr}(\mathbf{x})=\operatorname{Pr}(\mathbf{x} \mid A) \cdot \operatorname{Pr}(A)+\operatorname{Pr}(\mathbf{x} \mid B) \cdot \operatorname{Pr}(B)$$

The above probabilities are evaluated using equation 3 (Binomial distribution) as follows:

$$\begin{aligned} &\operatorname{Pr}(\mathbf{x} \mid A) \operatorname{Pr}(A)=q_{A}^{x_{i}}\left(1-q_{A}\right)^{50-x_{i}} q \\ &\operatorname{Pr}(\mathbf{x})=q_{A}^{x_{i}}\left(1-q_{A}\right)^{50-x_{i}} q+q_{B}^{x_{i}}\left(1-q_{B}\right)^{50-x_{i}}(1-q) \end{aligned}$$

Since we have $\left(\begin{array}{c} N \\ n \end{array}\right)$ in both numerator and denominator of equation 1 it is excluded in the above probabilities.

Thus, for each trial we get the following expression for the probability that the coin picked is A given head:

$$\mu_{i}=\frac{q_{A}^{x_{i}}\left(1-q_{A}\right)^{N-x_{i}} q}{q_{A}^{x_{i}}\left(1-q_{A}\right)^{N-x_{i}} q+q_{B}^{x_{i}}\left(1-q_{B}\right)^{N-x_{i}}(1-q)}$$

The probability of choosing coin A is then updated as the mean of the above vector of probabilities as follows:

$$q = \frac{1}{n}\sum_{i=1}^{n}\mu_i$$

While the probabilities of getting heads are updated as follows:

$$q_A = \frac{\sum_i^N \mu_i x_i}{\sum_i^N \mu_i x_i + \sum_i^N \mu_i(N - x_i)}$$
$$q_B = \frac{\sum_i^N (1-\mu_i) x_i}{\sum_i^N (1-\mu_i) x_i + \sum_i^N (1-\mu_i)(N - x_i)}$$

2.4.3 Conclusion

As n increases, q, q_A and q_B converge to 0.6, 0.7 and 0.4, respectively. This is because the more times we repeat the procedure the more information we will gain about the distribution of probability between coin A and B as well as about the skewness of the different coins.

2.4.4 Source Code

```
defaultW <- getOption("warn")
options(warn = -1)

#Data generation
set.seed(2021)
n = 30
N = 50
p = 0.6
p_a = 0.4
p_b = 0.7
z = rbinom ( n , 1 , p )
x = rbinom ( n , N, p_a )
x1 = rbinom ( n , N, p_b )
x[ z != 1] = x1[ z != 1 ]
x
```

```
##  [1] 14 35 40 24 32 39 40 18 37 40 17 34 39 26 40 23 24 37 33 23 20 36 15 20 19
## [26] 38 28 32 35 36
```

```
rm( z , x1 )
```

```
#EM Algorithm as a function
EM_algo = function(q,q_a,q_b){
  iter = 10
  for (i in 1:iter){

    q_est = q*(q_a^x)*((1-q_a)^(N-x))/ (q*(q_a^x)*((1-q_a)^(N-x)) +
                                         (1-q)*(q_b^x)*((1-q_b)^(N-x)))
    q = mean(q_est)

    q_a = q_est%*%x / ((q_est%*%x)+q_est%*%(N-x))
    q_b = (1-q_est)%*%x / ((1-q_est)%*%x+(1-q_est)%*%(N-x))

  }
  print(paste0("Num. iterations: ", iter))
  print(paste0('EM-estiamte of p: ', format(q, digits=3)))
  print('------------------------')
  print(paste0('EM-estiamte of p_a: ', format(q_a, digits=3)))
  print('------------------------')
  print(paste0('EM-estiamte of p_b: ', format(q_b, digits=3)))

}
```

```
#Randomly select initial guesses
q = runif(1)
q_a = runif(1)
q_b = runif(1)
```

```
#Call function and print output
EM_algo(q,q_a,q_b)
```

```
## [1] "Num. iterations: 10"
## [1] "EM-estiamte of p: 0.423"
## [1] "------------------------"
## [1] "EM-estiamte of p_a: 0.414"
## [1] "------------------------"
## [1] "EM-estiamte of p_b: 0.73"
```

```
options(warn = defaultW)
```