EM algorithm for handling missing data

Aude Sportisse & Vincent Vandewalle

03/11/2024

```
library(mvtnorm) #library for multivariate normal density
```

```
## Warning: package 'mvtnorm' was built under R version 4.3.3
```

```
library(ggplot2) #library to have nice graphics
```

Code an EM algorithm

We consider $X \sim \mathcal{N}(\mu, \Sigma)$, with

$$\mu = \begin{pmatrix} 5 \\ -1 \end{pmatrix} \text{ and } \Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}.$$

We want to introduce $r = 30\%$ of missing values in the variable X_2. We consider that the missing-data mechanism is MCAR.

Q1) Generate a bivariate normal set of sample size $n = 100$, with mean μ and covariance matrix Σ (use the package `mvtnorm`).

```
n = 100
mu = c(5,-1)
Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)
X = rmvnorm(n,mu,Sigma)
head(X)
```

```
##          [,1]       [,2]
## [1,] 4.595483 -1.4830984
## [2,] 3.045994 -1.9669335
## [3,] 4.325492 -1.8567179
## [4,] 3.854695 -1.9766984
## [5,] 5.714257  0.2671624
## [6,] 5.276211 -0.5705844
```

Q2) Introduce MCAR missing values in X_2.

```
missing_idx.mcar <- sample.int(n,0.3*n) #indexes of values which will be missing
XNA <- X
XNA[missing_idx.mcar,2] <- NA
head(XNA)
```

```
##          [,1]       [,2]
## [1,] 4.595483 -1.4830984
## [2,] 3.045994 -1.9669335
## [3,] 4.325492 -1.8567179
## [4,] 3.854695 -1.9766984
## [5,] 5.714257  0.2671624
## [6,] 5.276211 -0.5705844
```

The goal is now to estimate the parameters μ and Σ in presence of missing values in X_2 by using the EM algorithm.

Q3) Propose a simple initialization for the EM algorithm.

```
#we have to estimate mu and Sigma
hat_mu <- apply(XNA,2,mean,na.rm=TRUE) # mean of each column removing missing values
hat_mu
```

```
## [1]  4.892830 -1.091849
```

```
hat_Sigma <- cov(XNA,use="complete.obs") # covariance matrix lines with complete observations
# hat_Sigma <- var(XNA,na.rm=TRUE)
hat_Sigma
```

```
##          [,1]     [,2]
## [1,] 1.104141 0.644759
## [2,] 0.644759 1.058517
```

Q4) Write a function for the E-step and the M-step.

```
Estep=function(X, mu, Sigma, missing_idx)
{
  n=nrow(X)

  #all the elements in X1 are observed
  s1_vec = X[,1]
  s11_vec = X[,1]^2

  s2_vec = rep(0, n)
  s22_vec = rep(0, n)

  #for observed elements in X2
  #setdiff(1:n, missing_idx): observed elements
  s2_vec[setdiff(1:n, missing_idx)] = X[setdiff(1:n, missing_idx),2]
  s22_vec[setdiff(1:n, missing_idx)] = X[setdiff(1:n, missing_idx),2]^2

  #for missing elements in X2
  s2_vec[missing_idx] = mu[2]+(Sigma[1,2]/Sigma[1,1])*(X[missing_idx,1]-mu[1])
  s22_vec[missing_idx] = s2_vec[missing_idx]^2 + Sigma[2,2] - Sigma[1,2]^2/Sigma[1,1]


  s12_vec = s1_vec*s2_vec

  return(list(s1=sum(s1_vec), s2=sum(s2_vec), s11=sum(s11_vec), s22=sum(s22_vec), s12=sum(s12_vec)))
}

Mstep=function(X, s1, s2, s11, s22, s12)
{
  n=nrow(X)
  mu1=s1/n
  mu2=s2/n
  sigma1=s11/n-mu1^2
  sigma2=s22/n-mu2^2
  sigma12=s12/n-mu1*mu2
  mu=c(mu1,mu2)
  Sigma=matrix(c(sigma1, sigma12,sigma12,sigma2), nrow=2)
```

```
  return(structure(list(mu=mu, Sigma=Sigma)))
}
```

Q5) Use the EM algorithm for 50 iterations to estimate μ and Σ. Show the results.

```
for(i in 1:50)
{
  # E step
  E=Estep(XNA, hat_mu, hat_Sigma, missing_idx.mcar)
  s1=E$s1
  s11=E$s11
  s2=E$s2
  s22=E$s22
  s12=E$s12
  # M step
  M=Mstep(XNA, s1, s2, s11, s22, s12)
  hat_mu=M$mu
  hat_Sigma=M$Sigma
}
```

```
hat_mu
```

```
## [1]  4.892830 -1.083446
```

```
hat_Sigma
```

```
##           [,1]      [,2]
## [1,] 1.3601522 0.7942559
## [2,] 0.7942559 1.1360718
```

Additionnal questions

Q6) Vary n and the percentage of missing values.

```
simu <- function(n, p, mu = c(5,-1), Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)){
  X = rmvnorm(n,mu,Sigma)
  missing_idx.mcar <- sample.int(n,p*n) #indexes of values which will be missing
  XNA <- X
  XNA[missing_idx.mcar,2] <- NA
  return(XNA)
}

EM <- function(X, niter=50){
  missing_idx.mcar <- which(is.na(X[,2]))
  hat_mu <- apply(X,2,mean,na.rm=TRUE) # mean of each column removing missing values
  hat_Sigma <- cov(X,use="complete.obs") # covariance matrix lines with complete observations
  for(i in 1:niter)
  {
    # E step
    E=Estep(X, hat_mu, hat_Sigma, missing_idx.mcar)
    s1=E$s1
    s11=E$s11
    s2=E$s2
    s22=E$s22
    s12=E$s12
    # M step
    M=Mstep(X, s1, s2, s11, s22, s12)
```

```
    hat_mu=M$mu
    hat_Sigma=M$Sigma
  }
  return(list(hat_mu=hat_mu, hat_Sigma=hat_Sigma))
}

# RMSE : Root Mean Squared Error
# Compute the RMSE between the estimated parameters and the true parameters
RMSE <- function(hat_mu, hat_Sigma, mu, Sigma){
  return(sqrt(sum((hat_mu-mu)^2)+sum((hat_Sigma-Sigma)^2)))
}
```

```
# Illustration of the previous functions
mu = c(5,-1)
Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)
X = simu(n=100, p=0.3, mu=mu, Sigma=Sigma)
param = EM(X)
RMSE(param$hat_mu, param$hat_Sigma, mu, Sigma)
```

```
## [1] 0.1610184
```

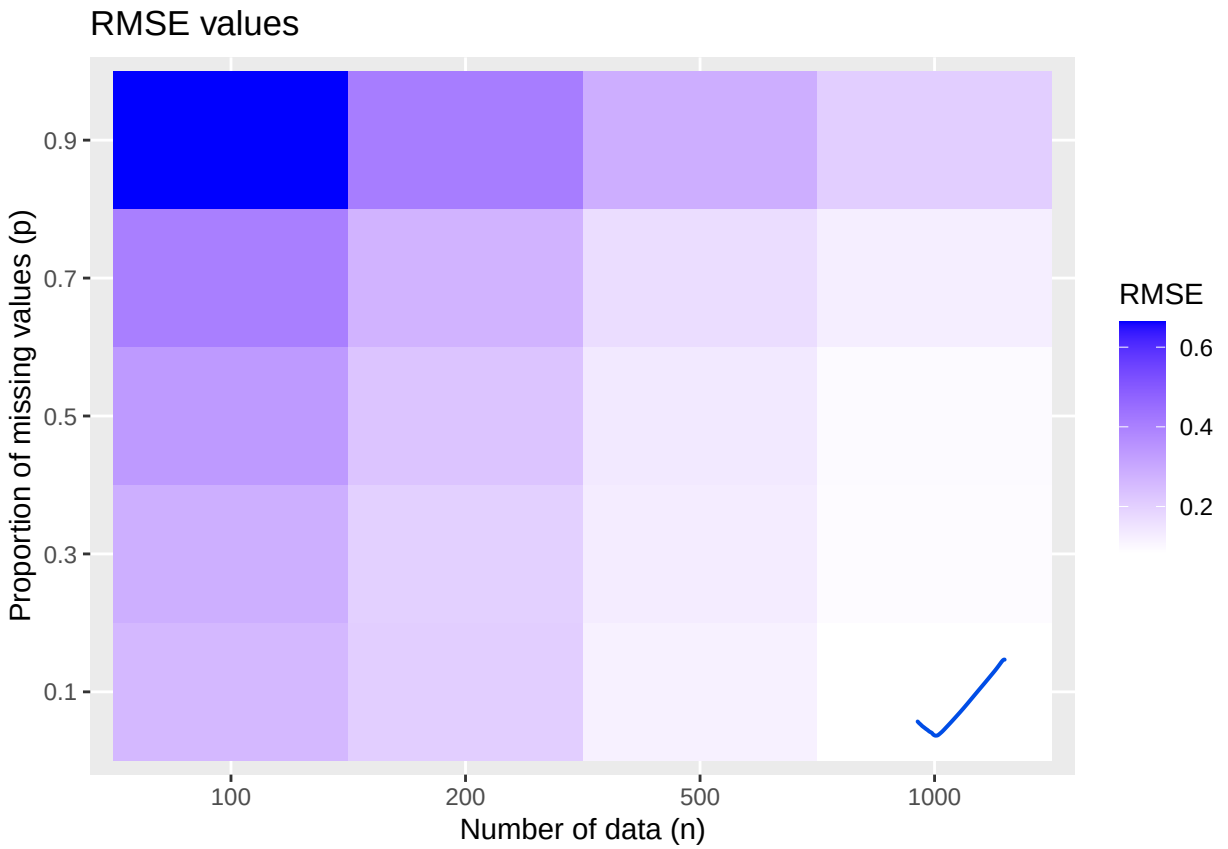
We now illustrate the RMSE for different values of n and p. For each value of n and p, we compute the RMSE over 100 simulations.

```
n = c(100, 200, 500, 1000)
p = c(0.1, 0.3, 0.5, 0.7, 0.9)
RMSE_values = matrix(0, nrow=length(n), ncol=length(p))
for(i in 1:length(n)){
  for(j in 1:length(p)){
    RMSE_values[i,j] = 0
    for(k in 1:100){
      X = simu(n=n[i], p=p[j], mu=mu, Sigma=Sigma)
      param = EM(X)
      RMSE_values[i,j] = RMSE_values[i,j] + RMSE(param$hat_mu, param$hat_Sigma, mu, Sigma)
    }
    RMSE_values[i,j] = RMSE_values[i,j]/100
  }
}
RMSE_values
```

```
##            [,1]       [,2]       [,3]      [,4]      [,5]
## [1,] 0.26165329 0.28517959 0.33764355 0.4041957 0.6626082
## [2,] 0.20716964 0.20213281 0.23093964 0.2729110 0.4109213
## [3,] 0.12103491 0.13288409 0.14045597 0.1692913 0.2852383
## [4,] 0.08602226 0.09510355 0.09802136 0.1282510 0.2061136
```

We now make a heatmap of the RMSE values.

```
df = data.frame(n=factor(rep(n, length(p))), p=factor(rep(p, each=length(n))), RMSE=as.vector(RMSE_valu
ggplot(df, aes(x=n, y=p, fill=RMSE)) + geom_tile() + scale_fill_gradient(low="white", high="blue") + lab
```



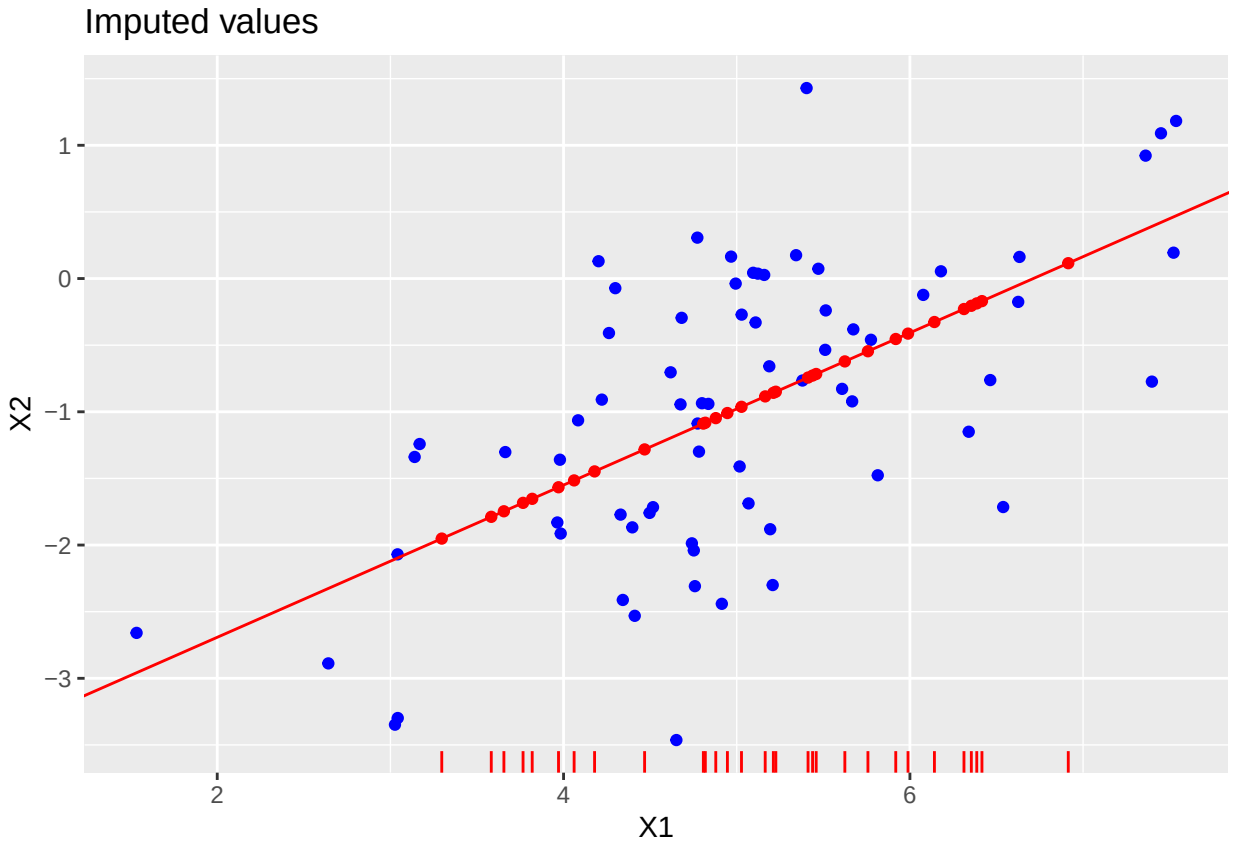
As expected, the RMSE increases with the proportion of missing values and decreases with the number of data.

Q7) We have estimated the parameters μ and Σ, can we impute the missing values? Try it!

```
# Impute missing values based on the estimated parameters
impute <- function(X, hat_mu, hat_Sigma){
  missing_idx.mcar <- which(is.na(X[,2]))
  XNA <- X
  XNA[missing_idx.mcar,2] <- hat_mu[2]+(hat_Sigma[1,2]/hat_Sigma[1,1])*(XNA[missing_idx.mcar,1]-hat_mu[
  return(XNA)
}

mu = c(5,-1)
Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)
X = simu(n=100, p=0.3, mu=mu, Sigma=Sigma)
param = EM(X)

imputed_X <- impute(X, param$hat_mu, param$hat_Sigma)
missing_idx.mcar <- which(is.na(X[,2]))

ggplot() +
  geom_point(aes(x=X[,1], y=X[,2]), color="blue") +
  geom_point(aes(x=imputed_X[missing_idx.mcar,1], y=imputed_X[missing_idx.mcar,2]), color="red") +
  labs(title="Imputed values", x="X1", y="X2") +
  geom_rug(aes(x=X[missing_idx.mcar,1]), color="red") +
  geom_abline(intercept=param$hat_mu[2] - (param$hat_Sigma[1,2]/param$hat_Sigma[1,1])*param$hat_mu[1],
```

```
## Warning: Removed 30 rows containing missing values or values outside the scale range
```

(`geom_point()`).

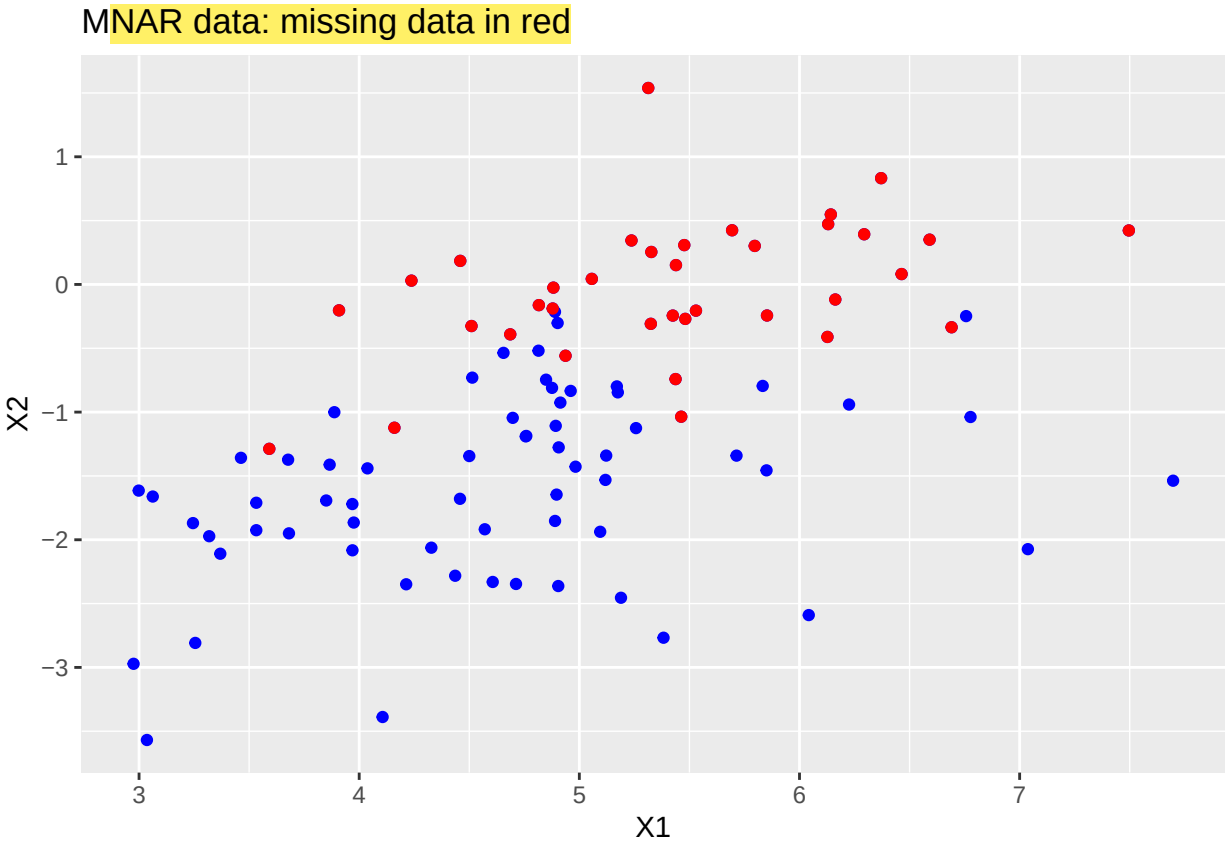


Q8) Do you think the algorithm will still work for MNAR data? If you have the time, try it!

No this algorithm is only accurate in the MCAR or MAR case. In the MNAR case, the missing values depend on the unobserved data an thus this needs to be modelled.

Simulation under the MNAR mechanism:

```
# w : the weight of the missingness mechanism related to X2
simuMNAR <- function(n, w = c(2,5), mu = c(5,-1), Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)){
  X = rmvnorm(n,mu,Sigma)
  p = 1/(1 + exp(-w[1] - w[2]*X[,2]))
  missing_idx.mnar <- which(rbinom(n, 1, p) == 1)
  XNA <- X
  XNA[missing_idx.mnar,2] <- NA
  print(ggplot() +
    geom_point(aes(x=X[,1], y=X[,2]), color="blue") +
    # geom_rug(aes(x=X[missing_idx.mnar,1], y=X[missing_idx.mnar,2]), color="red")+
    geom_point(aes(x=X[missing_idx.mnar,1], y=X[missing_idx.mnar,2]), color="red")+
    labs(title="MNAR data: missing data in red", x="X1", y="X2"))
  return(XNA)

}

X = simuMNAR(100)
```



We see that the missing values are depending on X2 even given X1. The EM algorithm will not work in this case.

$$P(M_{i2} = 1|X_{i1}, X_{i2}) = \frac{1}{1 + \exp(-w_1 - w_2 X_{i2})} = P(M_{i2} = 1|X_{i2})$$

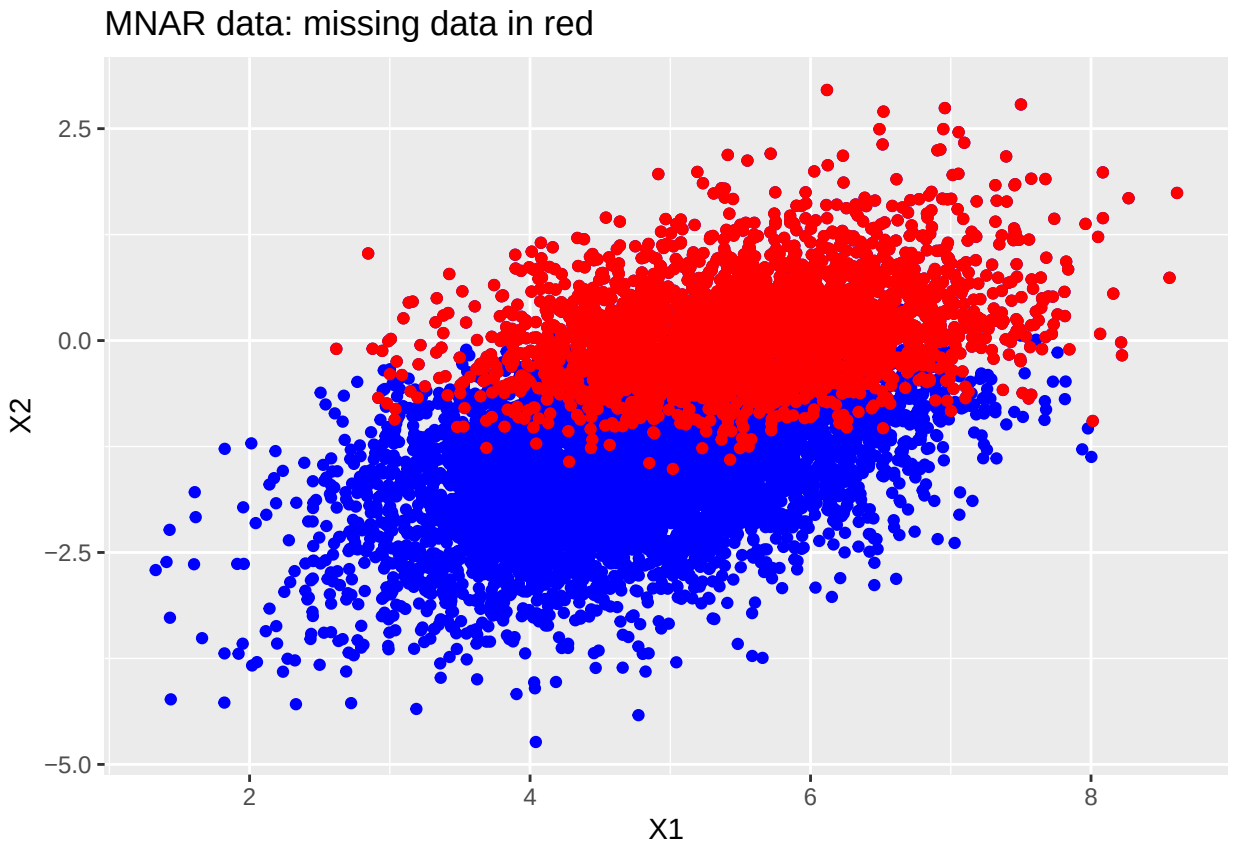
We have:

$$P(M_{i2} = 1|X_{i1}, X_{i2}) \neq P(M_{i2} = 1|X_{i1})$$

Thus we are in the MNAR case.

Let considered a large sample size and run the EM algorithm and see that it does not converge to the true parameters.

```
X = simuMNAR(10000)
```




```
param = EM(X)
param$hat_mu
```

```
## [1]  4.994170 -1.375134
```

```
param$hat_Sigma
```

```
##           [,1]      [,2]
## [1,] 0.9851705 0.3232579
## [2,] 0.3232579 0.5607643
```

Q9) How to stop the EM algorithm? (other than by giving a predefined number of steps)

We can stop the algorithm when the difference between the estimated parameters at two consecutive steps is small enough. Or more generally, we can stop the algorithm when the log-likelihood does not increase any more. Let notice that the log-likelihood should increase at each step of the algorithm.

```
ll <- function(X, mu, Sigma, missing_idx){
  n = nrow(X)
  loglik = 0
  for(i in 1:n){
    if(i %in% missing_idx){
      # Compute the distribution of X1
      loglik = loglik + dmvnorm(X[i,1], mu[1,drop=F], Sigma[1,1,drop=F], log=TRUE)
    }else{
      # Compute the distribution of X1 and X2
      loglik = loglik + dmvnorm(X[i,], mu, Sigma, log=TRUE)
    }
  }
  return(loglik)
```

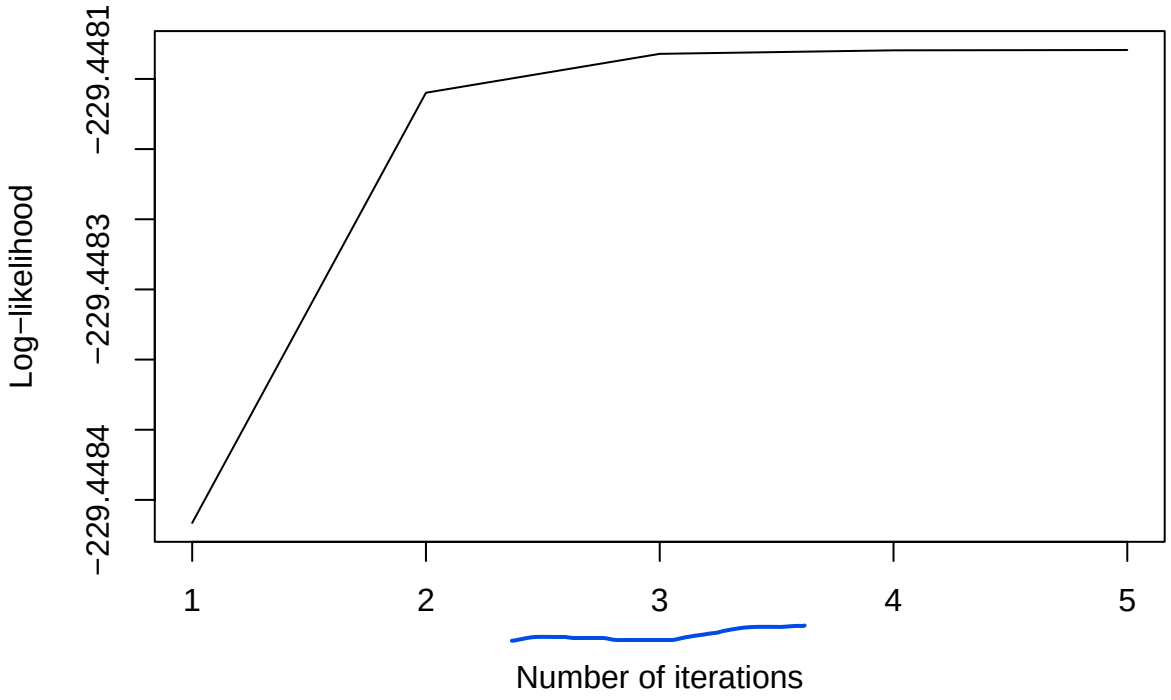
```
}

X = simu(n=100, p=0.3)
missing_idx <- which(is.na(X[,2]))
ll(X, mu, Sigma, missing_idx)
```

```
## [1] -237.346
```

Thus one can modify the EM algorithm as follows:

```
EM <- function(X, niter=50){
  missing_idx.mcar <- which(is.na(X[,2]))
  hat_mu <- apply(X,2,mean,na.rm=TRUE) # mean of each column removing missing values
  hat_Sigma <- cov(X,use="complete.obs") # covariance matrix lines with complete observations
  loglik = rep(0, niter)
  for(i in 1:niter)
  {
    # E step
    E=Estep(X, hat_mu, hat_Sigma, missing_idx.mcar)
    s1=E$s1
    s11=E$s11
    s2=E$s2
    s22=E$s22
    s12=E$s12
    # M step
    M=Mstep(X, s1, s2, s11, s22, s12)
    hat_mu=M$mu
    hat_Sigma=M$Sigma
    # Compute the log-likelihood
    loglik[i] = ll(X, hat_mu, hat_Sigma, missing_idx.mcar)
    # Stop the algorithm if the log-likelihood does not increase anymore
    if (i > 1 && loglik[i] - loglik[i-1] < 1e-6){
      break
    }
    loglik = loglik[1:i]
  }
  return(list(hat_mu=hat_mu, hat_Sigma=hat_Sigma, loglik=loglik))
}
```

```
X = simu(n=100, p=0.3)
param = EM(X)
plot(param$loglik, type="l", xlab="Number of iterations", ylab="Log-likelihood")
```



Thus monitoring the log-likelihood can be a good way to stop the algorithm.

Q10) Extend the code for considering missing values in both X_1 and X_2.

We need to modify the E-step in order to consider missing values for X_1, the M-step remains the same.

```
Estep_general = function(X, mu, Sigma)
{
  n=nrow(X)
  missing_idx1 = which(is.na(X[,1]))
  missing_idx2 = which(is.na(X[,2]))

  s1_vec = rep(0, n)
  s11_vec = rep(0, n)

  # for observed elements in X1 are observed
  s1_vec[setdiff(1:n, missing_idx1)] = X[setdiff(1:n, missing_idx1),1]
  s11_vec[setdiff(1:n, missing_idx1)] = X[setdiff(1:n, missing_idx1),1]^2

  #for missing elements in X1
  s1_vec[missing_idx1] = mu[1]+(Sigma[1,2]/Sigma[2,2])*(X[missing_idx1,2]-mu[2])
  s11_vec[missing_idx1] = s1_vec[missing_idx1]^2 + Sigma[1,1] - Sigma[1,2]^2/Sigma[2,2]


  s2_vec = rep(0, n)
  s22_vec = rep(0, n)

  #for observed elements in X2
  s2_vec[setdiff(1:n, missing_idx2)] = X[setdiff(1:n, missing_idx2),2]
  s22_vec[setdiff(1:n, missing_idx2)] = X[setdiff(1:n, missing_idx2),2]^2

  #for missing elements in X2
  s2_vec[missing_idx2] = mu[2]+(Sigma[1,2]/Sigma[1,1])*(X[missing_idx2,1]-mu[1])
```

```
  s22_vec[missing_idx2] = s2_vec[missing_idx2]^2 + Sigma[2,2] - Sigma[1,2]^2/Sigma[1,1]

  s12_vec = s1_vec*s2_vec

  return(list(s1=sum(s1_vec), s2=sum(s2_vec), s11=sum(s11_vec), s22=sum(s22_vec), s12=sum(s12_vec)))
}
```

```
EM_general <- function(X, niter=50){
  hat_mu <- apply(X,2,mean,na.rm=TRUE) # mean of each column removing missing values
  hat_Sigma <- cov(X,use="complete.obs") # covariance matrix lines with complete observations
  for(i in 1:niter)
  {
    # E step
    E=Estep_general(X, hat_mu, hat_Sigma)
    s1=E$s1
    s11=E$s11
    s2=E$s2
    s22=E$s22
    s12=E$s12
    # M step
    M=Mstep(X, s1, s2, s11, s22, s12)
    hat_mu=M$mu
    hat_Sigma=M$Sigma
  }
  return(list(hat_mu=hat_mu, hat_Sigma=hat_Sigma))
}
```

```
simu_general <- function(n, p, mu = c(5,-1), Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)){
  X = rmvnorm(n,mu,Sigma)
  missing_idx1 <- sample.int(n,p*n) #indexes of values which will be missing
  missing_idx2 <- sample.int(n,p*n) #indexes of values which will be missing
  XNA <- X
  XNA[missing_idx1,1] <- NA
  XNA[missing_idx2,2] <- NA
  XNA <- XNA[!apply(is.na(XNA),1,all),] # remove rows with all missing values
  return(XNA)
}
```

```
X = simu_general(100, 0.3)
param = EM_general(X)
param
```

```
## $hat_mu
## [1]  4.938057 -1.108594
##
## $hat_Sigma
##           [,1]      [,2]
## [1,] 0.9711838 0.4796578
## [2,] 0.4796578 0.8677308
```

One should also modify the log-likelihood function to consider missing values in both X_1 and X_2. But we let this as an exercise for the reader ;-)

Extra questions

Q12) Extend the algorithm for a multivariate normal distribution with p variables with a missing-data mechanism MAR.

We first consider the extension to the multivariate setting with p variables before considering the mixture setting.

Let first recall that in the multivariate normal setting the likelihood is given by:

$$\ell(\mu, \Sigma) = -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{n}(X_i - \mu)^T\Sigma^{-1}(X_i - \mu)$$

Which by using the trace can be written as:

$$\begin{aligned}
\ell(\mu, \Sigma) = & -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{n}\mathrm{tr}((X_i - \mu)^T\Sigma^{-1}(X_i - \mu)) \\
= & -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{n}\mathrm{tr}(\Sigma^{-1}(X_i - \mu)(X_i - \mu)^T) \\
= & -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\mathrm{tr}(\Sigma^{-1}\sum_{i=1}^{n}(X_i - \mu)(X_i - \mu)^T)
\end{aligned}$$

Moreover the MLE of μ is given by:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n}X_i$$

Thus the likelihood can be written as:

$$\ell(\hat{\mu}, \Sigma) = -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\mathrm{tr}(\Sigma^{-1}S)$$

where $S = \sum_{i=1}^{n}(X_i - \hat{\mu})(X_i - \hat{\mu})^T$.

The MLE of Σ is given by:

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu})(X_i - \hat{\mu})^T$$

Note also that the formula above also allows to impose constraints on the covariance matrix. For example, if we want to impose that the covariance matrix is diagonal, we can replace Σ by a diagonal matrix in the likelihood, or if we want to impose that the covariance matrix is sparse, we can replace Σ by a sparse matrix in the likelihood (see high dimensional Gaussian distribution).

In the case of missing values we need to consider the expectation of the complete likelihood with respect to the missing values given the observed values and the current values of the parameters.

$$Q(\theta, \theta^{(r)}) = E_{X^m|X^o,\theta^{(r)}}[\ell(\mu, \Sigma)]$$

$$Q(\theta, \theta^{(r)}) = -\frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\mathrm{tr}(\Sigma^{-1}E[\sum_{i=1}^{n}(X_i - \mu)(X_i - \mu)^T|X_i^o, \theta^{(r)}])$$

Where it can be shown that:

$$E[\sum_{i=1}^{n}(X_i - \mu)(X_i - \mu)^T|X_i^o, \theta^{(r)}] = \sum_{i=1}^{n}E[(X_i - \mu)(X_i - \mu)^T|X_i^o, \theta^{(r)}]$$

Moreover, we have:

$$E[(X_i - \mu)(X_i - \mu)^T | X_i^o, \theta^{(r)}] = (\hat{X}_i^{(r)} - \mu)(\hat{X}_i^{(r)} - \mu)^T + \tilde{\Sigma}_i^{(r)}$$

Where $\hat{X}_i^{(r)}$ is the imputed value of X_i at the r-th iteration and $\hat{\Sigma}_i^{(r)}$ is the covariance matrix of the imputed value of X_i at the r-th iteration :

$$\hat{X}_i^{m(r)} = \mu_i^{m(r)} + \Sigma_i^{mo(r)} {\Sigma_i^{o(r)}}^{-1} (X_i^o - \mu_i^{o(r)})$$

With μ_i^m the expectation of the missing values of X_i and μ_i^o the expectation of the observed values of X_i. Σ_i^{mo} is the covariance matrix between the missing values and the observed values of X_i, Σ_i^o is the covariance matrix of the observed values of X_i.

And the covariance matrix of the imputed value of X_i at the r-th iteration is given by:

$$\tilde{\Sigma}_i^{m(r)} = \Sigma_i^{m(r)} - \Sigma_i^{mo(r)} {\Sigma_i^{o(r)}}^{-1} \Sigma_i^{mo(r)}$$

Thus $\tilde{\Sigma}_i^{(r)}$ it composed with the values $\tilde{\Sigma}_i^{m(r)}$ for the indexes of missing values and the values O for the indexes of the observed values and for the intersection of the observed and missing values.

The M-step thus gives for μ:

$$\hat{\mu}^{(r+1)} = \frac{1}{n} \sum_{i=1}^{n} \hat{X}_i^{(r)}$$

Taking the expression of the expectation of the complete likelihood with respect to the missing values given the observed values and the current values of the parameters, evaluated in $\hat{\mu}^{(r)}$ we get:

$$Q(\hat{\mu}^{(r+1)}, \Sigma; \mu^{(r)}, \Sigma^{(r)}) = -\frac{n}{2} \log(|\Sigma|) - \frac{1}{2} \text{tr}(\Sigma^{-1} S^{(r)})$$

With $S^{(r)} = \sum_{i=1}^{n} \left((\hat{X}_i^{(r)} - \hat{\mu}^{(r)})(\hat{X}_i^{(r)} - \hat{\mu}^{(r)})^T + \hat{\Sigma}_i^{(r)} \right)$.

The M-step for Σ gives:

$$\hat{\Sigma}^{(r+1)} = S^{(r)}$$

Note the formulation above also allows to impose constraints on the covariance matrix. For example, if we want to impose that the covariance matrix is diagonal, we can replace Σ by a diagonal matrix in the likelihood, or if we want to impose that the covariance matrix is sparse, we can replace Σ by a sparse matrix (see high dimensional Gaussian distribution).

Thus the bivariate setting can be extended to the multivariate setting by using the formulas above.

```
Estep_mv <- function(X,mu,Sigma){
  n = nrow(X)
  p = ncol(X)
  X_hat = X
  naX = is.na(X)
  Sigma_tilde = matrix(0, nrow=p, ncol=p)
  for (i in 1:n){
    if (any(naX[i,])){
      # Compute the imputed value
      X_hat[i,naX[i,]] = mu[naX[i,]] +
        Sigma[naX[i,],!naX[i,]]%*%solve(Sigma[!naX[i,],!naX[i,]])%*%(X[i,!naX[i,]] - mu[!naX[i,]])
      # Correspond to the sum of tilde Sigma_i
      Sigma_tilde[naX[i,],naX[i,]] = Sigma_tilde[naX[i,],naX[i,]] +
        Sigma[naX[i,],naX[i,]] -
```

```
        Sigma[naX[i,],!naX[i,]]%*%solve(Sigma[!naX[i,],!naX[i,]])%*%Sigma[!naX[i,],naX[i,]]
    }
  }
  return(list(X_hat=X_hat, Sigma_tilde=Sigma_tilde))
}

Mstep_mv <- function(X_hat, Sigma_tilde){
  n = nrow(X_hat)
  p = ncol(X_hat)
  mu = colMeans(X_hat, na.rm=TRUE)
  Sigma = matrix(0, nrow=p, ncol=p)
  for (i in 1:n){
    Sigma = Sigma + (X_hat[i,] - mu)%*%t(X_hat[i,] - mu)
  }
  Sigma = (Sigma + Sigma_tilde)/n
  return(list(mu=mu, Sigma=Sigma))
}

EM_mv <- function(X, niter=50){
  p = ncol(X)
  hat_mu = apply(X,2,mean,na.rm=TRUE) # mean of each column removing missing values
  hat_Sigma = cov(X,use="complete.obs") # covariance matrix lines with complete observations
  for(i in 1:niter)
  {
    # E step
    E=Estep_mv(X, hat_mu, hat_Sigma)
    X_hat = E$X_hat
    Sigma_tilde = E$Sigma_tilde
    # M step
    M=Mstep_mv(X_hat, Sigma_tilde)
    hat_mu=M$mu
    hat_Sigma=M$Sigma
  }
  return(list(hat_mu=hat_mu, hat_Sigma=hat_Sigma))
}
```

```
mu = c(5,-1)
Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)
X = simu_general(1000, 0.3, mu = mu, Sigma = Sigma)
EM_general(X)
```

```
## $hat_mu
## [1]  4.9810984 -0.9804875
##
## $hat_Sigma
##           [,1]      [,2]
## [1,] 0.9819866 0.4075889
## [2,] 0.4075889 0.9295002
```

```
EM_mv(X)
```

```
## $hat_mu
## [1]  4.9810984 -0.9804875
##
## $hat_Sigma
```

```
##           [,1]      [,2]
## [1,] 0.9819866 0.4075889
## [2,] 0.4075889 0.9295002
```

Both codes give the same results!

Let now consider an example with more than two variables

```
# Add simulation function for simulating data with missing values in the general setting
simu_mv <- function(n, pmiss, mu = c(5,-1), Sigma = matrix(c(1, 0.5, 0.5, 1), ncol=2, nrow=2)){
  p = length(mu)
  X = rmvnorm(n,mu,Sigma)
  missing_idx = matrix(rbinom(n*p,1,pmiss), nrow=n, ncol=p)
  XNA <- X
  XNA[missing_idx == 1] <- NA
  XNA <- XNA[!apply(is.na(XNA),1,all),] # remove rows with all missing values
  return(XNA)
}

X = simu_mv(100, 0.3, mu = rep(0,5), Sigma = diag(5))
EM_mv(X)
```

```
## $hat_mu
## [1]  0.04395015  0.03406186 -0.10216834  0.10987493 -0.07820521
##
## $hat_Sigma
##              [,1]          [,2]        [,3]        [,4]          [,5]
## [1,]  0.997166396 -0.0079580168 -0.07870267 -0.08753408 -0.0942577457
## [2,] -0.007958017  1.1170143491  0.05392089 -0.33689834  0.0004589817
## [3,] -0.078702668  0.0539208875  1.08365960  0.01322605 -0.1892469719
## [4,] -0.087534084 -0.3368983374  0.01322605  1.09107433 -0.0421263401
## [5,] -0.094257746  0.0004589817 -0.18924697 -0.04212634  0.9209620969
```

Q11) Implement the EM algorithm for a mixture of two bivariate normal distributions.

See course with Charles!

Q13) Extend the algorithm for a multivariate normal distribution with p variables with a missing-data mechanism MAR and with sparse covariance matrix structure.

Just need to impose the sparsity of the covariance matrix in the M-step.

```
# d is the dimension of the subspace (d < p)
Mstep_mv_sparse <- function(X_hat, Sigma_tilde, d){
  n = nrow(X_hat)
  p = ncol(X_hat)
  mu = colMeans(X_hat, na.rm=TRUE)
  Sigma = matrix(0, nrow=p, ncol=p)
  for (i in 1:n){
    Sigma = Sigma + (X_hat[i,] - mu)%*%t(X_hat[i,] - mu)
  }
  Sigma = (Sigma +  + Sigma_tilde)/n
  # impose the sparsity of the covariance matrix
  decomp = eigen(Sigma)
  decomp$values[(d+1):p] = mean(decomp$values[(d+1):p]) # equalize the lowest eigenvalues
  # reconstruct the covariance matrix
  Sigma = decomp$vectors%*%diag(decomp$values)%*%t(decomp$vectors)
  return(list(mu=mu, Sigma=Sigma))
```

```
}
```

Including this in the EM algorithm we get:

```
EM_mv_sparse <- function(X, d=1, niter=50){
  p = ncol(X)
  hat_mu = apply(X,2,mean,na.rm=TRUE) # mean of each column removing missing values
  hat_Sigma = cov(X,use="complete.obs") # covariance matrix lines with complete observations
  for(i in 1:niter)
  {
    # E step
    E=Estep_mv(X, hat_mu, hat_Sigma)
    X_hat = E$X_hat
    Sigma_tilde = E$Sigma_tilde
    # M step
    M=Mstep_mv_sparse(X_hat, Sigma_tilde, d=d)
    hat_mu=M$mu
    hat_Sigma=M$Sigma
  }
  return(list(hat_mu=hat_mu, hat_Sigma=hat_Sigma))
}
```

```
X = simu_mv(100, 0.3, mu = rep(0,5), Sigma = diag(5))
param = EM_mv_sparse(X, d=2)
```

One can also check the sparsity of the estimated covariance matrix:

```
eigen(param$hat_Sigma)
```

```
## eigen() decomposition
## $values
## [1] 1.2479251 1.2154300 0.7811938 0.7811938 0.7811938
##
## $vectors
##              [,1]       [,2]        [,3]       [,4]        [,5]
## [1,]  0.220220347  0.1537827  0.82095067  0.4650706 -0.19391539
## [2,] -0.187886850 -0.2140414  0.47470759 -0.3663905  0.74786063
## [3,]  0.804207558 -0.5248007 -0.14139629  0.1273796  0.20399991
## [4,] -0.519073675 -0.6670269 -0.04464189  0.5315879 -0.03254338
## [5,]  0.003775456  0.4584719 -0.28054748  0.5921622  0.60035511
```

Where the first two eigenvalues are free, and the last three are equal.

```
# The estimated covariance matrix is:
param$hat_Sigma
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  0.81409817 -0.03360497  0.04761430 -0.09789507  0.03100391
## [2,] -0.03360497  0.81756408 -0.02174575  0.10751547 -0.04294354
## [3,]  0.04761430 -0.02174575  1.20264767 -0.04282662 -0.10306290
## [4,] -0.09789507  0.10751547 -0.04282662  1.10015120 -0.13370981
## [5,]  0.03100391 -0.04294354 -0.10306290 -0.13370981  0.87247539
```

```
# Where it can be recomputed by the following formula:
decomp = eigen(param$hat_Sigma)
decomp$vectors[,1:2] %*% diag(decomp$values[1:2] - decomp$values[3]) %*% t(decomp$vectors[,1:2]) +
  decomp$values[3]*diag(5)
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  0.81409817 -0.03360497  0.04761430 -0.09789507  0.03100391
## [2,] -0.03360497  0.81756408 -0.02174575  0.10751547 -0.04294354
## [3,]  0.04761430 -0.02174575  1.20264767 -0.04282662 -0.10306290
## [4,] -0.09789507  0.10751547 -0.04282662  1.10015120 -0.13370981
## [5,]  0.03100391 -0.04294354 -0.10306290 -0.13370981  0.87247539
```

```
# U %*% D %*% t(U) + lambda*I (with U a low rank matrix, and D a diagonal matrix)
```

This can be very usefull especially in high dimensional setting!

Q14) Implement the EM algorithm for a mixture of two multivariate normal distributions with p variables.

See course with Charles!

Q15) Implement the EM algorithm for a mixture of two multivariate normal distributions with p variables and with sparse covariance matrix structure.

See course with Charles!

Q16) Implement the EM algorithm for a mixture of K multivariate normal distributions with p variables and with missing values.

This is the most general case, just need to add the mixutre setting in the above algorithm, just need to compute the class membership probabilities in the E-step (just based on observed variables), and to update the previous of formulas taking into account the weights given by $P(Z_i = k|X_i^o, \theta^{(r)})$ in the M-step.

For more info see for intance : https://inria.hal.science/hal-00921023/document

```
# Initialization of the parameters
init <- function(X, K){
  n = nrow(X)
  p = ncol(X)
  Z = sample(1:K, n, replace=TRUE) # Sample the class membership
  mu = matrix(0, nrow=K, ncol=p)
  Sigma = array(0, dim=c(p,p,K))
  for (k in 1:K){
    mu[k,] = apply(X[Z==k,],2, mean, na.rm=TRUE)
    Sigma[,,k] = cov(X[Z==k,],use="complete.obs")
  }
  pi = rep(1/K, K)
  return(list(mu=mu, Sigma=Sigma, pi=pi))
}

param = init(X, 2)

# Compute the posterior probabilities based on observed variables
posterior <- function(X, mu, Sigma, pi){
  n = nrow(X)
  p = ncol(X)
  K = nrow(mu)
  naX = is.na(X)
  post = array(0, dim=c(n,K))
  for (i in 1:n){
    for (k in 1:K){
      post[i,k] = pi[k]*dmvnorm(X[i,!naX[i,]], mu[k,!naX[i,],drop=F], Sigma[,,k][!naX[i,],!naX[i,], drop
    }
    post[i,] = post[i,]/sum(post[i,])
  }
```

```
  return(post)
}

W = posterior(X, param$mu, param$Sigma, param$pi)
head(W)
```

```
##            [,1]      [,2]
## [1,] 0.86748535 0.1325147
## [2,] 0.66717808 0.3328219
## [3,] 0.57958577 0.4204142
## [4,] 0.12509996 0.8749000
## [5,] 0.08079577 0.9192042
## [6,] 0.27987499 0.7201250
```

```
# The Estep need to be updated to take into account the posterior probabilities
# Which can be performed for each class separately
# Thus the only novelty is to take the wt parameter into account !
# At this stage this does not change the code for X_hat, but the weight need to be considered to comput
Estep_mv_wt <- function(X,mu,Sigma, wt){
  n = nrow(X)
  p = ncol(X)
  X_hat = X
  naX = is.na(X)
  Sigma_tilde = matrix(0, nrow=p, ncol=p)
  for (i in 1:n){
    if (any(naX[i,])){
      # Compute the imputed value
      X_hat[i,naX[i,]] = mu[naX[i,]] +
        Sigma[naX[i,],!naX[i,]]%*%solve(Sigma[!naX[i,],!naX[i,]])%*%(X[i,!naX[i,]] - mu[!naX[i,]])
      # Correspond to the sum of tilde Sigma_i
      Sigma_tilde[naX[i,],naX[i,]] = Sigma_tilde[naX[i,],naX[i,]] +
        wt[i]*(Sigma[naX[i,],naX[i,]] -
        Sigma[naX[i,],!naX[i,]]%*%solve(Sigma[!naX[i,],!naX[i,]])%*%Sigma[!naX[i,],naX[i,]])
    }
  }
  return(list(X_hat=X_hat, Sigma_tilde=Sigma_tilde))
}

E = Estep_mv_wt(X,param$mu[1,],param$Sigma[,,1], W[,1])
head(E$X_hat)
```

```
##            [,1]       [,2]         [,3]       [,4]       [,5]
## [1,]  1.1392626  0.9936780 -0.240182717  1.4141072 -1.0098690
## [2,]  0.1661802 -0.4050718 -0.004927810 -0.5053836  0.0507264
## [3,] -0.4736925 -0.9104473 -0.002608625 -0.3801582  0.3505617
## [4,] -0.8727693 -0.4268292  0.788186594  0.6062437 -1.1039402
## [5,] -0.8218194  0.1297373 -0.218508557  0.6315036 -0.4833007
## [6,]  0.1326967 -0.1639325  0.438780786  0.1932645 -0.5116892
```

```
E$Sigma_tilde
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,]  2.6881384 -0.1234765 -0.1485312 -0.9282016 -0.6324729
## [2,] -0.1234765 13.4634953 -0.4635119  3.7137216  1.2338542
## [3,] -0.1485312 -0.4635119  6.8757021  3.6155519 -1.3220116
```

```
## [4,] -0.9282016  3.7137216  3.6155519 17.0853017 -3.2498068
## [5,] -0.6324729  1.2338542 -1.3220116 -3.2498068 19.1775640
```

```
# And now one consider the Mstep with the weights
Mstep_mv_wt <- function(X_hat, Sigma_tilde, wt){
  n = nrow(X_hat)
  p = ncol(X_hat)
  mu = ((wt %*% X_hat)/sum(wt))[1,] # take into account the weights
  Sigma = matrix(0, nrow=p, ncol=p)
  for (i in 1:n){
    Sigma = Sigma + wt[i]*(X_hat[i,] - mu)%*%t(X_hat[i,] - mu)
  }
  Sigma = (Sigma + Sigma_tilde)/sum(wt)
  return(list(mu=mu, Sigma=Sigma))
}

Mstep_mv_wt(X_hat = E$X_hat, Sigma_tilde = E$Sigma_tilde, wt = W[,1])
```

```
## $mu
## [1]  0.027597089 -0.042575899 -0.052598557 -0.235656319  0.009905128
##
## $Sigma
##             [,1]        [,2]       [,3]       [,4]        [,5]
## [1,]  0.37917358  0.07490729 -0.1261016 -0.1460644 -0.21657228
## [2,]  0.07490729  1.13486881 -0.2423048  0.4826438  0.06588085
## [3,] -0.12610157 -0.24230475  0.9759246  0.4640930 -0.13385969
## [4,] -0.14606437  0.48264384  0.4640930  1.6934392 -0.46702328
## [5,] -0.21657228  0.06588085 -0.1338597 -0.4670233  1.46414829
```

```
# Thus putting all together we get (for the EM algorithm for a mixture of Gaussians with missing data)
EM_mix_miss <- function(X, K, niter=50){
  p = ncol(X)
  param = init(X, K)
  for(i in 1:niter)
  {
    # E step (for the weights)
    W = posterior(X, param$mu, param$Sigma, param$pi)
    for (k in 1:K){
      # E step related to the k-th class / considering missing values
      E=Estep_mv_wt(X, param$mu[k,], param$Sigma[,,k], W[,k])
      X_hat = E$X_hat
      Sigma_tilde = E$Sigma_tilde
      # M step for class k
      M=Mstep_mv_wt(X_hat, Sigma_tilde, W[,1])
      param$mu[k,] = M$mu
      param$Sigma[,,k] = M$Sigma
    }
    param$pi = colMeans(W) # update the class proportions
  }
  return(param)
}

res = EM_mix_miss(X,2)
res
```

```
## $mu
```

```
##             [,1]       [,2]        [,3]       [,4]        [,5]
## [1,] -0.09991766 0.05653901 -0.04259937 -0.1815604 -0.06724485
## [2,] -0.06642673 0.09466487 -0.06625733 -0.1844637 -0.07816955
##
## $Sigma
## , , 1
##
##             [,1]        [,2]       [,3]        [,4]        [,5]
## [1,]  0.53730058 -0.16981787  0.2300992  0.04442009  0.09206139
## [2,] -0.16981787  0.52479899 -0.0461210 -0.02478357  0.38535499
## [3,]  0.23009920 -0.04612100  0.8640980 -0.14088133 -0.03163810
## [4,]  0.04442009 -0.02478357 -0.1408813  0.65783510  0.21213503
## [5,]  0.09206139  0.38535499 -0.0316381  0.21213503  0.46038208
##
## , , 2
##
##             [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  0.95940302 -0.09856722  0.18355533  0.05851020  0.07216052
## [2,] -0.09856722  1.31560028 -0.05321179 -0.04984263  0.26147772
## [3,]  0.18355533 -0.05321179  1.49435085 -0.15208818 -0.03206373
## [4,]  0.05851020 -0.04984263 -0.15208818  1.26249601  0.19977439
## [5,]  0.07216052  0.26147772 -0.03206373  0.19977439  0.83304195
##
##
## $pi
## [1] 0.3179166 0.6820834
```

Remember that the EM algorithm is a local optimization algorithm, thus it is important to run it several times with different initializations and to keep the best solution. But to able to do one need to compute the log-likelihood of the data given the parameters and to keep the solution related to the highest log-likelihood.