Multivariate Statistical Analysis Problem set 3

Edoardo Bandoni 1116680 - Giorgia Cerase 1138847 -Lorenzo Sala 943481 - Stefano Sperti 947676 - Fran

Università di Torino

Old Faithful is a hydrothermal geyser in Yellowstone National Park in the state of Wyoming, U.S.A., and is a popular tourist attraction. Its name stems from the supposed regularity of its eruptions. The data set faithful (package dataset) comprises 272 observations, each of which represents a single eruption and contains two variables corresponding to eruptions, the duration in minutes of the eruption, and waiting, the time until the next eruption, also in minutes. Figure below shows a plot of the time to the next eruption versus the duration of the eruptions. It can be seen that the time to the next eruption varies considerably, although knowledge of the duration of the current eruption allows it to be predicted more accurately.

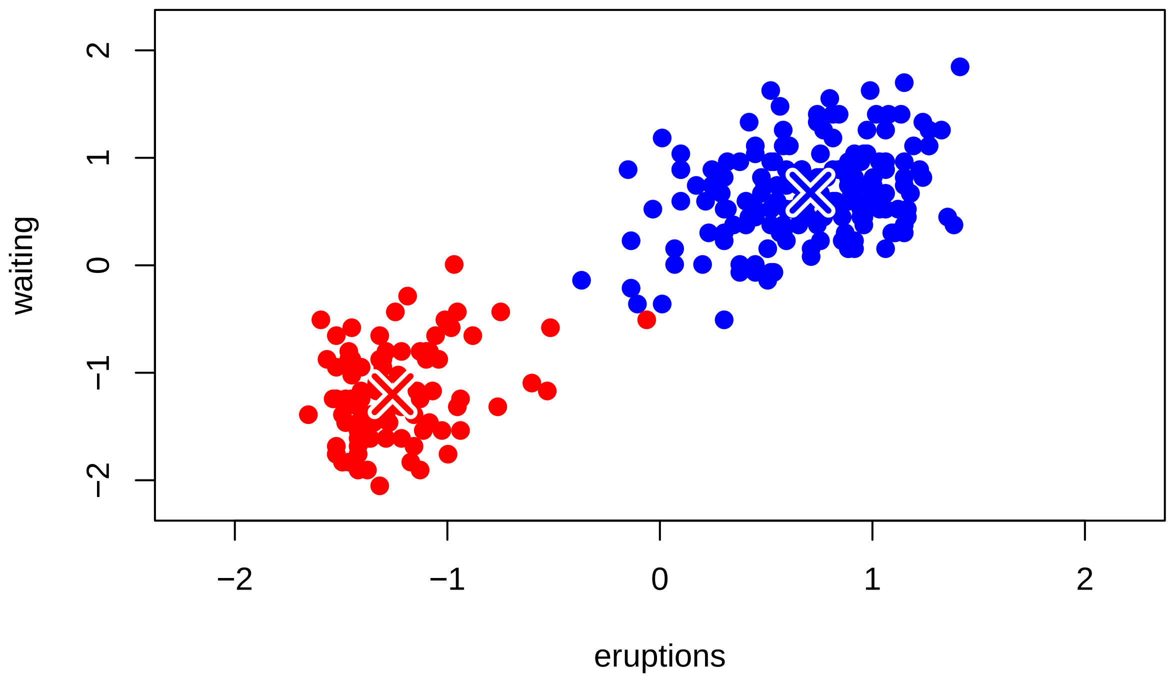
```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

Exercise 1

Obtain a 2-clusters solution via kmeans. Plot the observations with colors determined by cluster memberships as in panel (i) of Figure 9.1 of Bishop (2006).

```
km.data <- kmeans(data,center=2)
centroids <- km.data$centers
labels <- km.data$cluster
col.index <- c("blue","red")[labels]
plot(data,pch=16,col=col.index,cex=1.3,xlim=c(-2.2,2.2),ylim=c(-2.2,2.2))
points(centroids,cex=2.5,col="white",pch=4,lwd=7)
points(centroids,cex=2.5,col=c("blue","red"),pch=4,lwd=3)
```

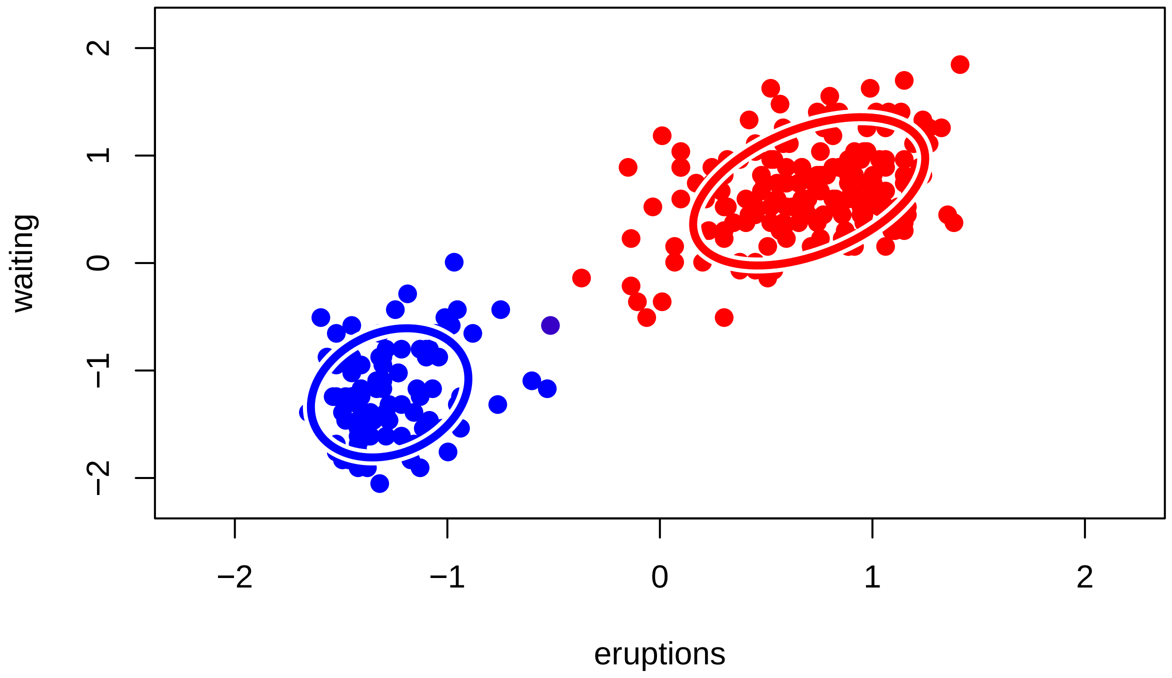


This code utilizes the K-means clustering algorithm to group `faithful` points into two distinct clusters. The `kmeans` function performs the clustering, assigning each data point to a cluster based on its similarity to the cluster's centroid (central point). K-means algorithm starts by randomly choosing k centroids (cluster centers) within the data space. It then assigns each data point to the closest centroid based on a distance measure (Euclidean distance). Once all points are assigned, the centroids are recalculated as the mean of their assigned points. These last two steps are repeated until the centroids no longer significantly change, indicating convergence.

Exercise 2

Consider model-based clustering with Gaussian mixtures. Use mclust:: Mclust to fit a mixture with two components, allowing for different covariance matrices. Reproduce the plot of panel (f) of Figure 9.8 of Bishop (2006), by adding contours of constant probability density for each component. Pay attention to the color-coding of the observations obtained as combination of blue and red inks with proportions equal to posterior probabilities.

```r
model <- Mclust(data,G=2)
mean1 <- as.numeric(model$parameters$mean[,1])
mean2 <- as.numeric(model$parameters$mean[,2])
cov1 <- model$parameters$variance$sigma[,,1]
cov2 <- model$parameters$variance$sigma[,,2]
prob <- model$z
col.index2 <- rgb(red=prob[,1],green=0,blue=prob[,2],maxColorValue = 1)
plot(data,pch=16,cex=1.3,col=col.index2,xlim=c(-2.2,2.2),ylim=c(-2.2,2.2))
lines(ellipse(centre=mean1,cov1,level=0.68),lwd=8,col="white")
lines(ellipse(centre=mean1,cov1,level=0.68),lwd=4,col="red")
lines(ellipse(centre=mean2,cov2,level=0.68),lwd=8,col="white")
lines(ellipse(centre=mean2,cov2,level=0.68),lwd=4,col="blue")
```



In Figure 9.8 of Bishop (2006) represent a mixture of two Gaussians. The centers of these Gaussians are initialized using the same values as those for the K-means algorithm, and the precision matrices are initialized to be proportional to the unit matrix. Specifically, Bishop (2006) plot the result of the EM algorithm when have converged.

For the replication we utilized `Mclust`, which is a model-based clustering method based on parameterized finite Gaussian mixture models. Models are estimated using the EM algorithm initialized by hierarchical model-based agglomerative clustering. The correct model is selected by calculating the Bayesian Information Criterion (BIC). We specified the parameter `G = 2` to denote the number of mixture components (clusters) for which the BIC is calculated.

The Expectation-Maximization algorithm works as follows: In the Expectation Step, the probabilities that each data point belongs to each Gaussian component, based on the current parameter estimates (the responsibilities) are calculated. In the Maximization Step, the parameters of the Gaussian components (means, covariances, and mixing proportions) are updated using the responsibilities calculated in the previous step. This repeated process continues until convergence. In particular some controll are added such that the log likelihood increases with each iteration.

In the plot, we included the following command: `col.index2 <- rgb(red = prob[, 1], green = 0, blue = prob[, 2], maxColorValue = 1)`: This line creates a vector of colors based on the posterior probabilities extracted earlier. It assigns a color to each data point based on its posterior probability of belonging to the first and second components. The `rgb` function is used to specify the colors based on the red, green, and blue components, with the red component determined by `prob[, 1]` and the blue component determined by `prob[, 2]`, while the green component is fixed at 0. Our scatterplot reveals that only two central data points deviate from the dominant red and blue colors (likely due to their distance from the centroids). The remaining data points likely belong to the correct clusters with high probability, given the bimodal behavior exhibited by the dataset.

We also plotted the 0.68 countour plot (one standard deviation for Gaussian) for the two clusters.

Exercise 3

Many models of interest involve distributions which are not tractable from a computational point of view, in particular in Bayesian inference this is often the case for posterior distributions.

Variational inference is an approximation schema used to deal with such cases, with similarities with Monte Carlo methods, but with the difference that variational inference is a deterministic approach. Variational methods are based on the idea of optimizing a functional, which is basically a function of functions; often, to do this we need to restrict the range of functions on which the optimization is performed, which leads to an approximate result.

Variational methods are particularly useful in the context of Bayesian models, in which we have the observed variables denoted by X, the parameters (which are also random variables) denoted by θ, and latent variables denoted by U, for simplicity we denote $Z = (U, \theta)$; we wish to find a simpler distribution $q(Z)$, that we choose in a restricted family of distributions, and that approximates the posterior distribution $p(Z|X)$ which is usually intractable. The main idea for is to find the distribution $q(Z)$ that minimizes the Kullback-Leibler divergence with $p(Z|X)$, which is a measure of the difference between two distribution and it is defined as

$$\mathrm{KL}(q\,||\,p) = -\int q(Z) \ln\left(\frac{p(Z|X)}{q(Z)}\right) dZ$$

We notice that KL divergence it is not a proper distance but it is always non-negative. Then we make use of the following relation:

$$\ln p(X) = L(q) + \mathrm{KL}(q\,||\,p)$$

where

$$L(q) = \int q(Z) \ln\left(\frac{p(X,Z)}{q(Z)}\right) dZ$$

which is easily proved

$$\begin{aligned} L(q) + \mathrm{KL}(q\,||\,p) &= \int q(Z) \left(\ln\left(\frac{p(X,Z)}{q(Z)}\right) - \ln\left(\frac{p(Z|X)}{q(Z)}\right)\right) dZ \\ &= \int q(Z) \ln\left(p(X)\right) dZ \\ &= \ln p(X) \end{aligned}$$

Thus, since the KL divergence is always non-negative, minimizing it with respect to q is equivalent to maximize L with respect to q, with the advantage that to maximize L we do not need to deal with $p(Z|X)$ which is intractable. We therefore assume some restriction on q to ensures tractability while maintaining flexibility to approximate the true posterior well. In general models the only restriction that we impose is that

$$q(Z) = \prod_{i=1}^{N} q_i(Z_i).$$

In particular, in our model we are dealing with a Gaussian mixture model and we can assume a weaker restriction. We will restrict the maximization problem only to the distributions q which are such that the distribution of the parameters and the one of the latent variables factorize, that is

$$q(Z) = q(\theta)q(U)$$

(recall that U are the latent variables and θ are the parameters that i our case $\theta = \{\mu, \pi, \Lambda\}$). In particular we deal with Gaussian mixture model. In particular, let define$U = \{u_1, \dots, u_N\}$ be a $1 \times K$ binary vector with elements u_{nk} for $k = 1, \dots, K$. To define our model we define the joint distribution of all random variables:

$$p(X, Z, \pi, \mu, \Lambda) = p(X \mid Z, \mu, \Lambda)p(Z \mid \pi)p(\pi)p(\mu, \Lambda). \tag{1}$$

$$p(Z|\pi) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{u_{nk}}$$

conditional distribution of Z given mixing coefficients. Since we are a Gaussian model we define: $p(X|U, \mu, \Lambda)$ that is conditional distribution of X given Z, $\mu = \{\mu_k\}$ and $\Lambda = \{\Lambda_k\}$.

$$p(X|U, \mu, \Lambda) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Lambda_k^{-1})^{u_{nk}}$$

.Then, let define $p(\pi)$ the distribution of mixing coefficients where we choose a Dirichlet distribution over the mixing coefficients since the analysis is considerably simplified if we use conjugate prior distributions and at the end we introduce $p(\mu, \Lambda)$, a Gaussian-Wishart prior that governs the mean and precision of each component.

Therefore, In our model the maximization can be made with respect to only two factors separately. It can be proved that the solution to the problem are given by

$$q(U) = \frac{1}{K_1} \exp\{E_\theta[\ln p(X, U, \theta)]\}$$
$$q(\theta) = \frac{1}{K_2} \exp\{E_U[\ln p(X, U, \theta)]\}$$

where K_1, K_2 are the normalization constants. (These equations follow from 10.43 Bishop (2006)). Since one solution depends on the other, an iterative algorithm is used to solve the problem, so we introduce the variational EM algorithm. We will explain it in the particular case of usage for a mixture of Gaussian model, in which the latent variable U defines the affiliation to one between K different Gaussian components (clusters), and the parameters are:

- $\mu = (\mu_1, ..., \mu_K)$ the vector of means of the components;
- $\Lambda = (\Lambda_1, ..., \Lambda_K)$ the importance matrices of the components (the inverse of the covariances);
- $\pi = (\pi_1, ..., \pi_K)$ which are the mixing coefficients of the components (prior probabilities).

In particular, as previous define we have that Λ is Wishart distributed with parameters W_0, ν_0 and that $\mu_k|\Lambda_k \sim \mathcal{N}(m_0, (\beta_0\Lambda_k)^{-1})$ and π has a Dirichelet distribution with parameter α_0.

In this context we analyze the `R` function that we used to reproduce image 10.6, which is an implementation of the variational EM on a model like the one defined above. The function is called `vbgmm` and it is from the `TargetScore` package (documentation at https://rdrr.io/bioc/TargetScore/src/R/vbgmm.R); its main part consists in the initialization of the posterior probabilities of the latent variable, which are stored in the matrix $R = (r_{nk})_{N \times K}$, called responsibility matrix; then we have a cycle of variational maximizations (`vmax`) and variational expectations (`vexp`) steps, that finishes when one between the convergence or the maximum number of iteration is reached.

The function `initialization`, basically chooses K points in the dataset as means of the Gaussian components and then assigns each data point to a specific component, based on some measure of the distance from each mean. Therefore, the initialization is random, hence we cannot reproduce exactly the initial setting and the various iterations in figure 10.6: as we'll see later the only panel well reproduced is the one in which the convergence is attained.

The function `vmax` performs the variational maximization step, in which the parameters (given in input) are updated according to the following formulas:

$$\begin{aligned}
\alpha_k &= \alpha_0 + N_k \\
\beta_k &= N_k + \beta_0 \\
m_k &= \frac{1}{\beta_k}(\beta_0 m_0 + N_k \bar{x}_k) \\
W_k^{-1} &= W_0^{-1} + N_k S_k + \frac{\beta_0 N_k}{\beta_0 + N_k}(\bar{x}_k - m_0)(\bar{x}_k - m_0)^T \\
\nu_k &= \nu_0 + N_k
\end{aligned}$$

This are useful for update $q(\mu_k, \Lambda_k)$ that as previously define is a Gaussian-Wishart distibution and is given by

$$q(\mu_k, \Lambda_k) = \mathcal{N}(\mu_k | m_k, (\beta_k \Lambda_k)^{-1})\mathcal{W}(\Lambda_k | W_k, \nu_k)$$

In particular, N_k is the column sum of the responsibility matrix and represents the numerosity of the k-th component, and

$$\bar{x}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_{nk} x_n$$

$$S_k = \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}(x_n - \bar{x}_k)(x_n - \bar{x}_k)^T$$

are the maximum likelihood estimate, and they are basically a weighted sum of the ML estimates of the single components, weighted with the posterior probabilities r_{nk}. The computation of the latter is performed in the variational expectation step; we have that

$$E[z_{nk}] = r_{nk}$$

in particular it can be proven that

$$r_{nk} = \frac{1}{C}\rho_{nk} = \frac{1}{C}\exp\left\{E[\ln \pi_k] + \frac{1}{2}E[\ln|\Lambda_k|] - \frac{D}{2}\ln(2\pi) - \frac{1}{2}E_{\mu_k,\Lambda_k}[(x_n - \mu_k)^T \Lambda_k (x_n - \mu_k)]\right\}$$

where C is the normalization constant. Notice that the computation of these expectations depend on the parameters and their distributions, so we see the dependence between one step and the other.

So in order to reproduce the last panel of figure 10.6, in which the algorithm converged, we run the function with some fixed parameters:

```r
set.seed(220202)
n <- dim(data)[1]
p <- dim(data)[2]
k <- 6
alpha <- 0.0015
par <- list(alpha = rep(alpha,k),
              kappa = 1,
              m = colMeans(data),
              v = p+1,
              M = diag(1,p,p))
model <- vbgmm(data,init=k,prior=par)
```

```
## Running VB-GMM on a 272-by-2 data ...
```

```
## Converged in 94 steps.
```

Then we extrapolate the means of the components from the model output, and compute the sample covariances S_k (according to 10.53), which we'll use to plot the ellipses:

```r
mk <- model$mu#cluster means
Nk <- colSums(model$full.model$R)#cluster cardinality
covk <- array(dim=c(2,2,6))
R <- model$R
for (i in 1:k) {
  covk[,,i] <- t(R[,i]*as.matrix(data-mk[,i]))%*%as.matrix(data-mk[,i])/Nk[i]
}
```

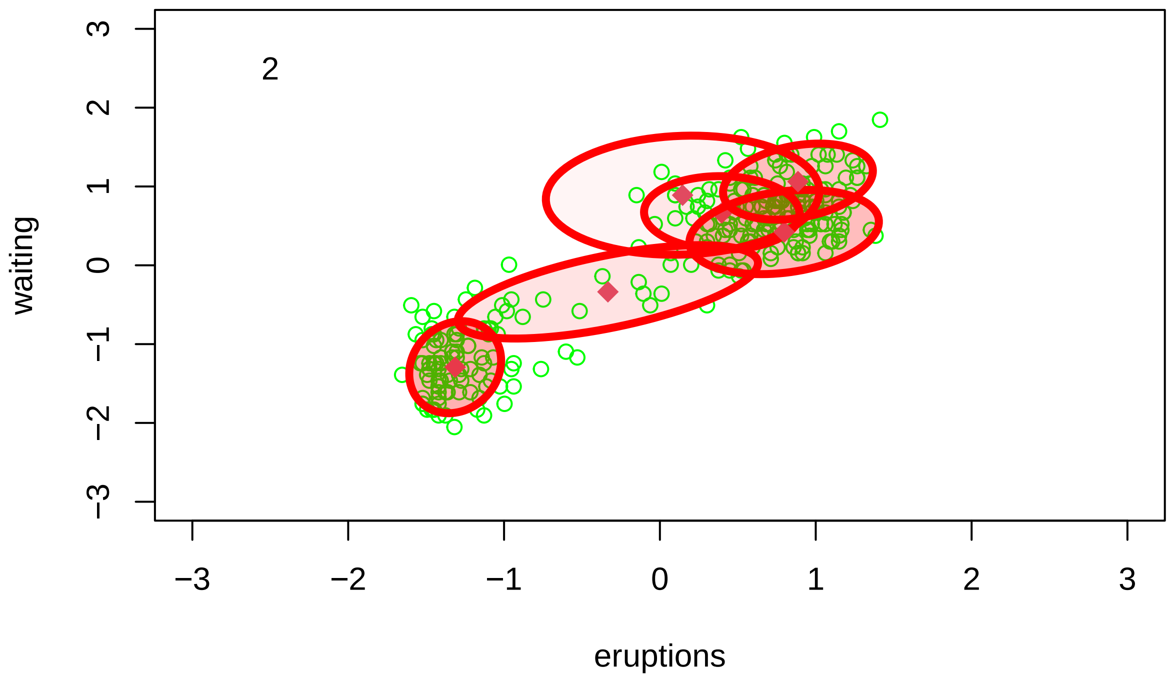
Finally we compute the expected mixing coefficients which will give us the transparency of the color of the ellipses:

```r
pik <- (model$full.model$alpha)/(k*alpha+n)#mixing coefficients
```

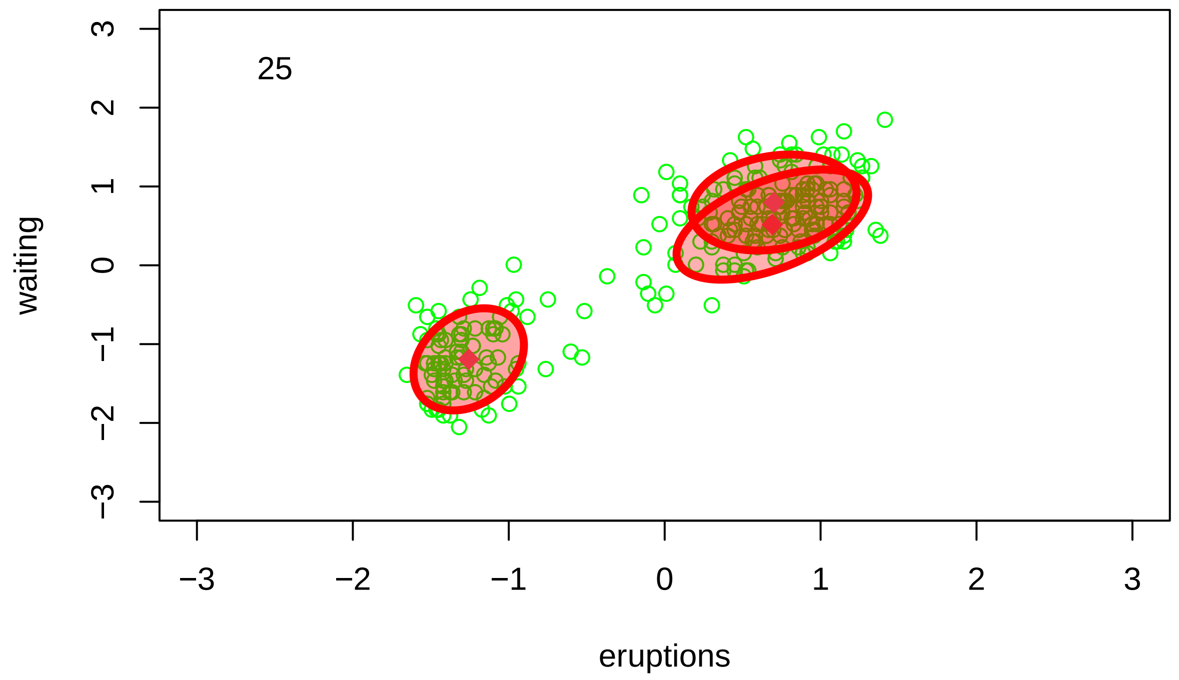
Our model converged in 94 steps, so we present four phases, respectively 2,25,50,94 iterations:

```r
for (j in c(2,25,50,100)) {
  set.seed(220202)
  model <- vbgmm(data,init=k,prior=par,maxiter = j)
  mk <- model$mu#cluster means
  Nk <- colSums(model$full.model$R)#cluster cardinality
  pik <- (model$full.model$alpha)/(k*alpha+n)
  covk <- array(dim=c(2,2,6))
  R <- model$R
  for (i in 1:k) {
    covk[,,i] <- t(R[,i]*as.matrix(data-mk[,i]))%*%as.matrix(data-mk[,i])/Nk[i]
  }
  plot(data,col="green",cex=1,xlim=c(-3,3),ylim=c(-3,3))
  text(-2.5,2.5,labels=j)
  for (i in 1:k) {
    if (pik[i]>10^{-5}) {
      points(x=mk[1,i],y=mk[2,i],pch=18,cex=1.5,col=2)
      polygon(ellipse(centre=mk[,i],covk[,,i],level=0.68,),lwd=4,col=rgb(1,0,0,pik[i]),border="red")
    }
  }
}
```

```
## Running VB-GMM on a 272-by-2 data ...
##
## Running VB-GMM on a 272-by-2 data ...
```

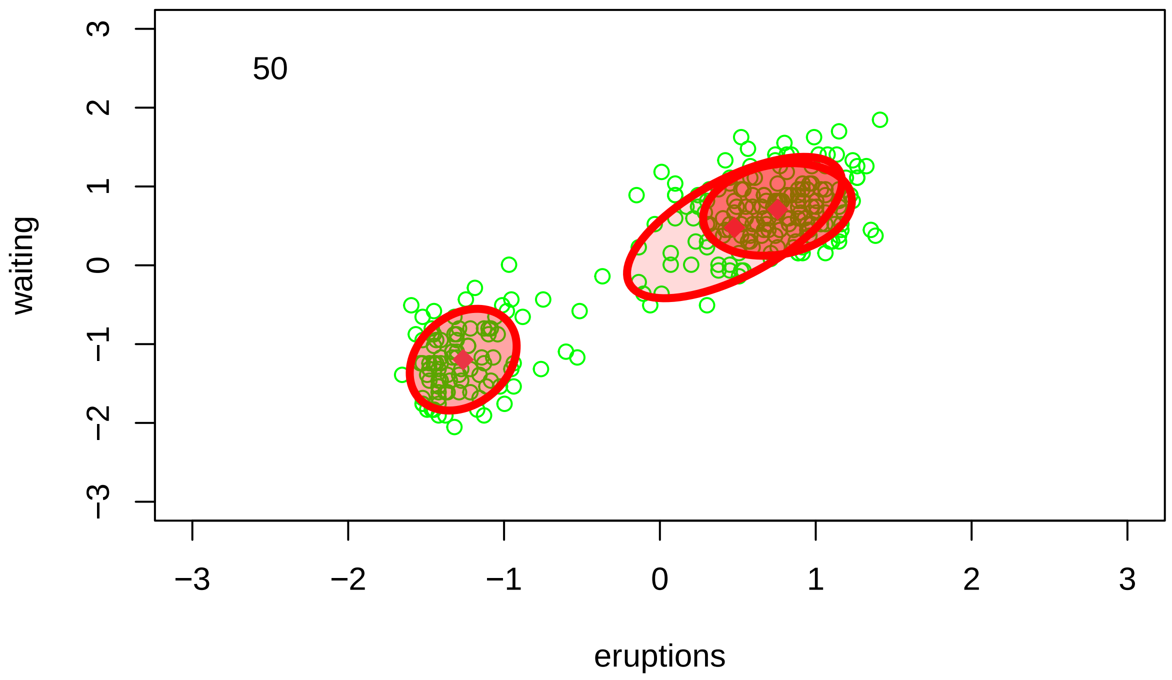


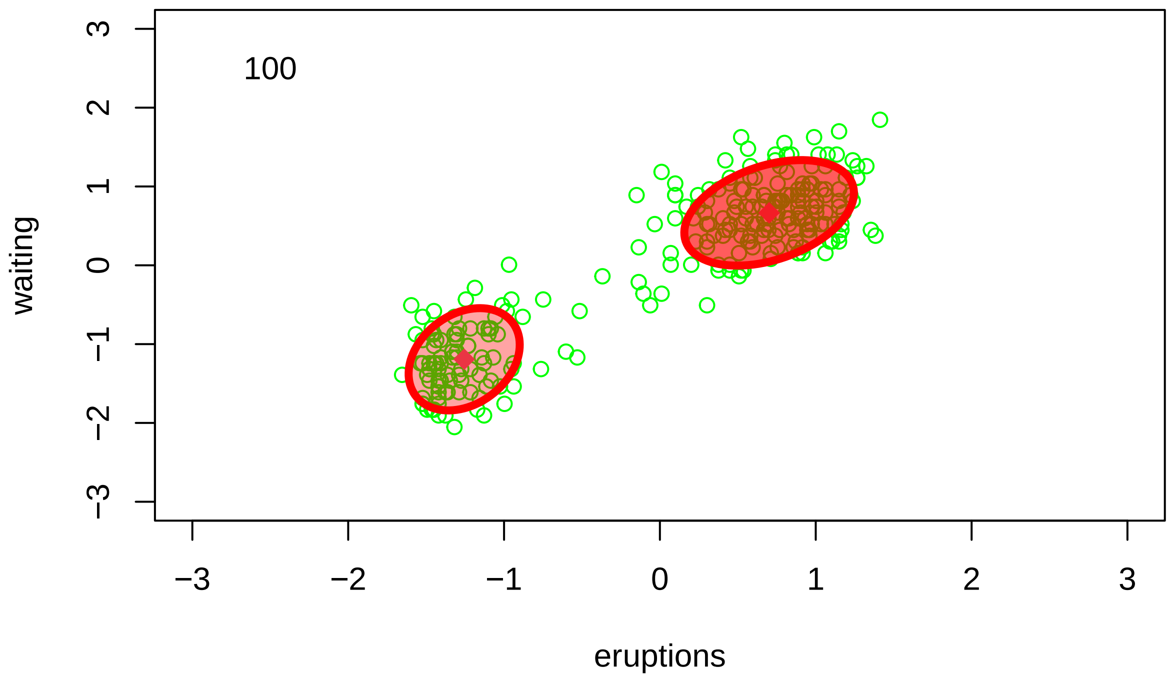
```
## Running VB-GMM on a 272-by-2 data ...
```



```
## Running VB-GMM on a 272-by-2 data ...

## Converged in 94 steps.
```


50
waiting
eruptions
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3



As anticipated, the only figure perfectly resembling figure 10.6 is the last one, in which the algorithm converged; the other are different because the algorithm has a random initialization, and moreover we do not know the exact parameters with which it was initialized. However, we see that it proceeds in a similar fashion, starting with six different components and then moving to three and ending up with two. Notice that the components are actually always 6, but we plot them only when their mixing coefficient is distinguishable from 0, as suggested by the book.

We tried also with different parametrization of the prior distibution $alpha_0 = 0.002$, $m_0 = 0$, $beta_0 = 5$, $M = diag(6, p, p)$.

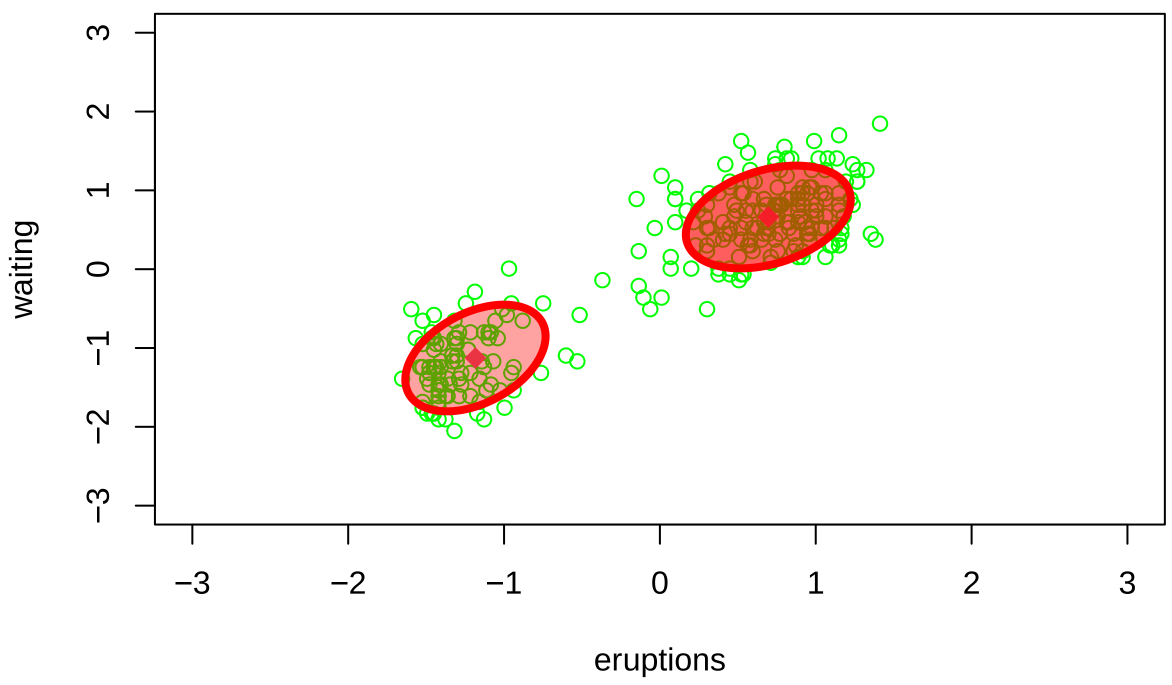
```
set.seed(220202)
n <- dim(data)[1]
p <- dim(data)[2]
k <- 6
alpha <- 0.002
par <- list(alpha = rep(alpha,k),
              kappa = 5,
              m = rep(0,p),
              v = p+1,
              M = diag(6,p,p))
model <- vbgmm(data,init=k,prior=par)
```

```
## Running VB-GMM on a 272-by-2 data ...
```

```
## Converged in 48 steps.
```

```
mk <- model$mu#cluster means
Nk <- colSums(model$full.model$R)#cluster cardinality
```

```r
covk <- array(dim=c(2,2,6))
R <- model$R
for (i in 1:k) {
  covk[,,i] <- t(R[,i]*as.matrix(data-mk[,i]))%*%as.matrix(data-mk[,i])/Nk[i]
}
pik <- (model$full.model$alpha)/(k*alpha+n)#mixing coefficients
  plot(data,col="green",cex=1,xlim=c(-3,3),ylim=c(-3,3))
  for (i in 1:k) {
    if (pik[i]>10^{-5}) {
      points(x=mk[1,i],y=mk[2,i],pch=18,cex=1.5,col=2)
      polygon(ellipse(centre=mk[,i],covk[,,i],level=0.68,),lwd=4,col=rgb(1,0,0,pik[i]),border="red")
    }
  }
```



However, we can see that convergence occurs in nearly half the steps. This is because selecting the right hyperparameters aids in the model's convergence.

Exercise 4

*Reproduce Figure 10.7 of Bishop (2006) about determining the number of components by using the variational lower bound. Code should be provided in a separate file. Make a comparison with the best model selected by BIC method with mixtures up to 6 components (use **mclust::mclustBIC**)*

In a Bayesian Gaussian mixture model, the goal is to approximate the true posterior distribution of the latent variables and model parameters using a simpler distribution. The variational lower bound (ELBO) is used as an objective function to optimize during the inference process. The ELBO is defined as

$$\mathcal{L} = \mathbb{E}_q[log(p(X, Z, \pi, \mu, \Lambda))] - \mathbb{E}_q[log(q(Z, \pi, \mu, \Lambda))]$$

```r
Nobs <- 1
data <- as.data.frame(scale(faithful))
n <- dim(data)[1]
p <- dim(data)[2]
Lower <- matrix(0, ncol = 6,nrow=Nobs)

for (k in 2:6) {
  for (j in 1:Nobs) {
    alpha <- 0.0015
    par <- list(alpha = rep(alpha,k),
                kappa = 1,
                m = colMeans(data),
                v = p+1,
                M = diag(1,p,p))
    model <- vbgmm(data,init=k,prior=par,maxiter=120)
    Lower[j,k] <- max(model$L)
  }
}

plot(jitter(rep(2,Nobs)),Lower[,2], ylim = c(-1.615,-1.603), xlim = c(2,6), pch=3,
     col="red", xlab="Number of Components", ylab="Variational Lower Bound")
points(jitter(rep(3,Nobs)),Lower[,3], pch=3, col="red")
points(jitter(rep(4,Nobs)),Lower[,4], pch=3, col="red")
points(jitter(rep(5,Nobs)),Lower[,5], pch=3, col="red")
points(jitter(rep(6,Nobs)),Lower[,6], pch=3, col="red")
```

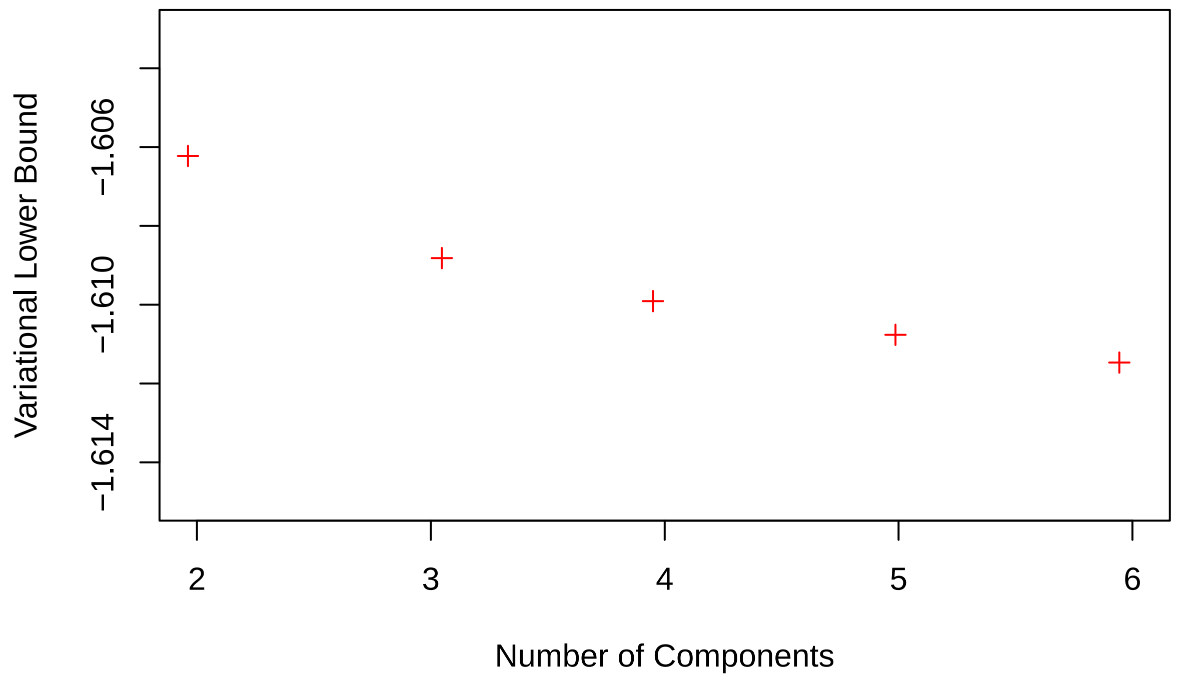


Figure 10.7 of Bishop (2006) represents the plot of the variational lower bound versus the number of components in the Gaussian mixture model, for the Old Faithful data, showing a peak at $K = 2$ components. For each value of K, the model is trained from 100 different random starts and the result, shown as '+' symbols plotted with small random horizontal perturbations so that they can be distinguished. Note that the solution for $K = 2$ find suboptimal local maxima, but that this happens infrequently.

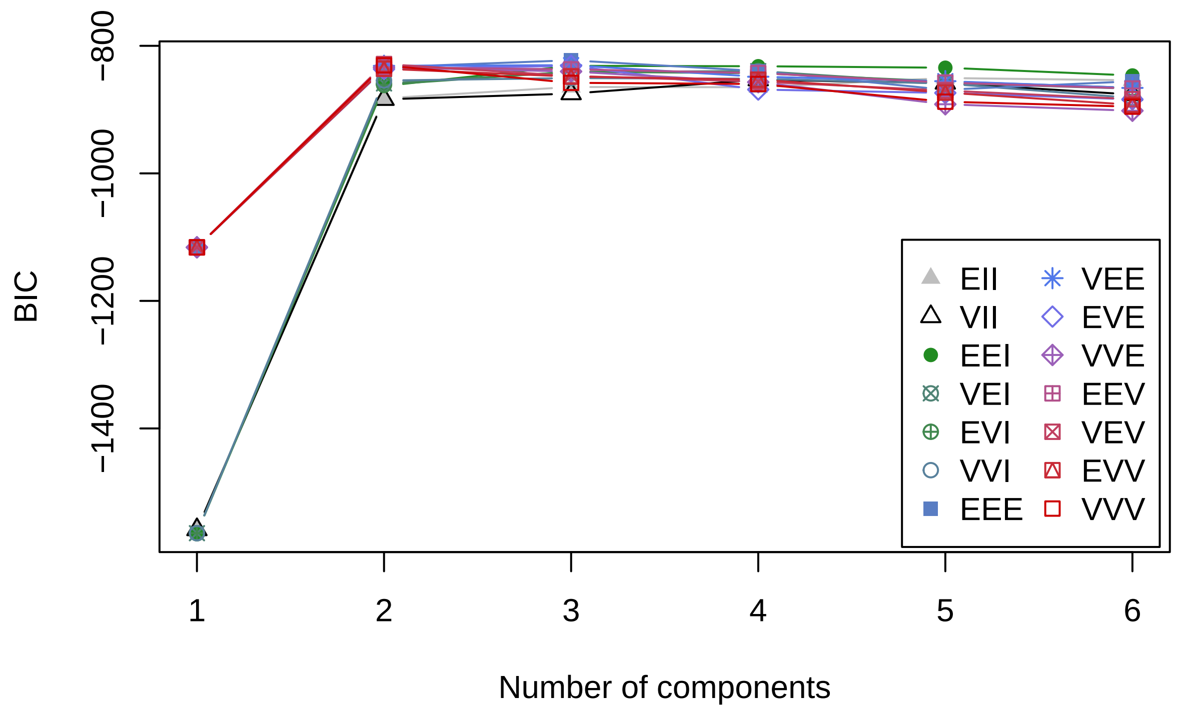
To replicate the results, we adapt the code from Exercise 3 and compute the Variational Bayesian Gaussian Mixture Model (VB-GMM) multiple times. The '`vbgmm(data, init=k, prior=par, maxiter=120)` function returns a data frame describing the VB-GMM, which includes the variable `L` representing the Evidence Lower Bound (ELBO). For each value of the number of clusters `K` we compute the variational lower bound 100 times and store the values in a matrix called `Lower`. At this point, we plot the values of the ELBO, specifying the '+' symbols by `pch=3` and we use `jitter()` to create small random horizontal perturbations so that the values are distinguishable.

It is noteworthy to observe that the peak in $K = 2$ has a precise meaning, i.e. that it is the number of clusters that maximizes the log likelihood and hence the model is well approximated by a Variational Bayesian Gaussian Mixture Model with two clusters.

```
model <- mclustBIC(data,G=seq(from=1,to=6,by=1))
model
```

```
## Bayesian Information Criterion (BIC):
##           EII        VII        EEI        VEI        EVI        VVI        EEE
## 1 -1558.6188 -1558.6188 -1564.2246 -1564.2246 -1564.2246 -1564.2246 -1116.0123
## 2  -882.3289  -883.8998  -862.9904  -858.9966  -861.0073  -854.4547  -833.6097
## 3  -864.6004  -874.9929  -831.3747  -841.0229  -840.5377  -850.5895  -822.6927
## 4  -864.9560  -853.1671  -832.0085  -839.6101  -842.9552  -851.6255  -839.5390
## 5  -850.4965  -858.4101  -834.3480  -858.5191  -855.9145  -859.1833  -868.9658
## 6  -854.0485  -876.3864  -846.4979  -867.1726  -865.9458  -881.6239  -855.6515
##          VEE        EVE        VVE        EEV        VEV        EVV        VVV
## 1 -1116.0123 -1116.0123 -1116.0123 -1116.0123 -1116.0123 -1116.0123 -1116.0123
## 2  -831.3617  -836.5169  -833.9578  -833.5272  -828.5690  -835.9876  -830.5817
## 3  -830.3818  -831.2903  -840.2493  -836.9204  -847.2879  -848.2006  -858.0580
## 4  -848.4854  -868.5792  -856.9392  -842.6627  -856.1226  -853.0365  -859.8410
## 5  -855.4642  -873.8728  -891.6495  -857.6422  -870.1658  -873.2136  -887.6064
## 6  -865.9795  -883.9228  -901.7853  -866.8989  -883.5923  -892.4555  -895.3787
##
## Top 3 models based on the BIC criterion:
##     EEE,3     VEV,2     VEE,3
## -822.6927 -828.5690 -830.3818
```

```r
plot(model)
```



When using the `mclustBIC` function on the dataset with the range of clusters specified as 1 to 6, the Bayesian Information Criterion (BIC) indicates that the top three models, ranked in order, are as follows: EEE with 3 clusters, VEV with 2 clusters, and VEE with 3 clusters.

In particular `mclustBIC` provide the lower bound of the EM algorithm for normal mixture models with a variety of covariance structures, and functions for simulation from these models.

- **EII:** equal volume, round shape (spherical covariance)
- **VII:** varying volume, round shape (spherical covariance)
- **EEI:** equal volume, equal shape, axis parallel orientation (diagonal covariance)
- **VEI:** varying volume, equal shape, axis parallel orientation (diagonal covariance)

- **EVI:** equal volume, varying shape, axis parallel orientation (diagonal covariance)
- **VVI:** varying volume, varying shape, equal orientation (diagonal covariance)
- **EEE:** equal volume, equal shape, equal orientation (ellipsoidal covariance)
- **EEV:** equal volume, equal shape, varying orientation (ellipsoidal covariance)
- **VEV:** varying volume, equal shape, varying orientation (ellipsoidal covariance)
- **VVV:** varying volume, varying shape, varying orientation (ellipsoidal covariance)
- **VEE:** Varying volume, Equal shape, Equal orientation.
- **EVE:** Equal volume, Varying shape, Equal orientation.
- **VVE:** Varying volume, Varying shape, Equal orientation.
- **EVV:** Equal volume, Varying shape, Varying orientation.

Interestingly, this outcome differs from a previous analysis where the best number of clusters, determined through computing the ELBO, was found to be $K = 2$. Nevertheless, recent studies Zhang2024 suggest that ELBO tends to achieve better approximation accuracy than BIC with similar computational costs when estimating model evidence. This result is reasonable, considering that from a visual representation of the data, they appear to cluster around two distinct points.

References

Zhang, Y., & Yang, Y. (2024). "Bayesian model selection via mean-field variational approximation." Journal of the Royal Statistical Society Series B: Statistical Methodology. doi: 10.1093/jrsssb/qkad164.

Appendix

Targetscore package

```
# Variational Bayesian Gaussian mixture model (VB-GMM)
# X: N x D data matrix
# init: k (1 x 1) or label (1 x n, 1<=label(i)<=k) or center (d x k)
# Reference: Pattern Recognition and Machine Learning by Christopher M. Bishop (P.474)
# Part of the implementation is based on the Matlab code vbgm.m from Michael Chen
# Matlab code: http://www.mathworks.com/matlabcentral/fileexchange/35362-variational-bayesian-inference
# Author: Yue Li (yueli@cs.toronto.edu)

# Main function vbgmm running four separate functions, namely,
# intialization: initialization of responsibility
# vexp: Variational-Expectation
# vmax: V-Maximimization
# vbound: V-(lower)-bound evaluation
require(pracma)
require(Matrix)


############ bsxfun {pracma} with single expansion (real Matlab style) ############
# expandByRow applies only when x is a square matrix
bsxfun.se <- function(func, x, y, expandByRow=TRUE) {

    if(length(y) == 1) return(arrayfun(func, x, y)) else
        stopifnot(nrow(x) == length(y) || ncol(x) == length(y))

    expandCol <- nrow(x) == length(y)
    expandRow <- ncol(x) == length(y)

    if(expandCol & expandRow & expandByRow) expandCol <- FALSE
    if(expandCol & expandRow & !expandByRow) expandRow <- FALSE

    # repeat row (if dim2expand = 1, then length(y) = ncol(x))
    if(expandRow) y.repmat <- matrix(rep(as.numeric(y), each=nrow(x)), nrow=nrow(x))

    # repeat col (if dim2expand = 2, then length(y) = nrow(x))
    if(expandCol) y.repmat <- matrix(rep(as.numeric(y), ncol(x)), ncol=ncol(x))

    bsxfun(func, x, y.repmat)
}


############ Compute log(sum(exp(x), dim)) ############
# Compute log(sum(exp(x),dim)) while avoiding numerical underflow
logsumexp <- function(x, margin=1) {

    stopifnot(is.matrix(x))

    # subtract the largest in each column
    y <- apply(x, margin, max)
    x <- bsxfun.se("-", x, y)

    s <- y + log(apply(exp(x), margin, sum))

    i <- which(!is.finite(s))

    if(length(i) > 0) s[i] <- y[i]

    s
}


############ Logarithmic Multivariate Gamma function ############
# Compute logarithm multivariate Gamma function.
# Gamma_p(x) = pi^(p(p-1)/4) prod_(j=1)^p Gamma(x+(1-j)/2)
# log Gamma_p(x) = p(p-1)/4 log pi + sum_(j=1)^p log Gamma(x+(1-j)/2)
logmvgamma <- function(x, d) {

    s <- size(x)

    x <- matrix(as.numeric(x), nrow=1)

    x <- bsxfun.se("+", kronecker(matrix(1,d,1), x), (1 - matrix(1:d))/2)

    y <- d*(d-1)/4*log(pi) + colSums(lgamma(x))

    y <- matrix(as.numeric(y), nrow=s[1], ncol=s[2])

    y
}

############ Sort the model  ############
# Sort model paramters in increasing order of averaged means
# of d variables
sort_components <- function(model) {

    idx <- order(apply(model$m, 2, mean))

    model$m <- model$m[, idx]

    model$R <- model$R[, idx]

    model$logR <- model$logR[, idx]

    model$alpha <- model$alpha[idx]

    model$kappa <- model$kappa[idx]

    model$v <- model$v[idx]

    model$M <- model$M[,,idx]

    model
```

```r
}


############ Vector dot product ############
# handle single row matrix by multiplying each value
# but not sum them up
dot.ext <- function(x,y,mydim) {

    if(missing(mydim)) dot(x,y) else {

        if(1 %in% size(x) & mydim == 1) x * y else dot(x,y)
    }
}



############ Main function of VB-GMM ############
# mirprior: miRNA-specific prior

vbgmm <- function(data, init=2, prior, tol=1e-20, maxiter=2e3, mirprior=TRUE,
    expectedTargetFreq=0.01, verbose=FALSE) {

    data <- as.matrix(data)

    n <- nrow(data)
    d <- ncol(data)

    X <- t(data) # Work with D by N for convenience

    message(sprintf("Running VB-GMM on a %d-by-%d data ...\n", n, d))

    if(missing(prior)) {

        # miRNA-specific prior with alpha for target-component
        # with defined target frequency (expected much lower
        # than non-target freq)
        if(mirprior & length(init) == 1) {

            k <- init

            stopifnot(expectedTargetFreq > 0 & expectedTargetFreq < 1)

            expectTargetCnt <- round(expectedTargetFreq * n)

            backgroundCnt <- rep((n - expectTargetCnt)/(k-1), k-1)

            prior <- list(
                alpha = c(expectTargetCnt, backgroundCnt),
                kappa = 1,
                m = as.matrix(rowMeans(X)),
                v = d+1,
                M = diag(1,d,d) # M = inv(W)
            )
        } else { # more general prior with equal alpha

            if(length(init)>1) k <- ncol(init) else k <- init

            prior <- list(
                alpha = rep(1,k),
                kappa = 1,
                m = as.matrix(rowMeans(X)),
                v = d+1,
                M = diag(1,d,d) # M = inv(W)
            )
        }
    } else {

        stopifnot(
            all(names(prior) %in% c("alpha","kappa","m","v","M")) &
            all(sapply(prior, is.numeric)) & nrow(prior$m) == d &
            ncol(prior$m) == 1 &
            nrow(prior$M) == d & ncol(prior$M) == d)
    }

    if(mirprior & length(init) != 1) {
        warning("mirprior is TRUE but init is not scalar (k)! Proceed as if mirprior were FALSE") }

    # lower variational bound (objective function)
    L <- rep(-Inf, maxiter)
    converged <- FALSE
    t <- 1

    model <- list()

    model$R <- initialization(X, init) # initialize responsibility R

    while(!converged & t < maxiter) {
        t <- t + 1
        model <- vmax(X, model, prior)
        model <- vexp(X, model)
        L[t] <- vbound(X, model, prior)/n
        converged <- abs(L[t] - L[t-1]) < tol * abs(L[t])

        if(verbose) message(sprintf("VB-EM-%d: L = %.6f", t, L[t]))
    }

    L <- L[2:t]

    model <- sort_components(model)

    label <- rep(0, n)

    label <- apply(model$R, 1, which.max)

    # unique to have consecutive label eg 2,3,6 changed to 1,2,3
    # label <- match(label, sort(unique(label)))
    if(converged) message(sprintf("Converged in %d steps.\n", t-1)) else
    warnings(sprintf("Not converged in %d steps.\n", maxiter))

    list(label=label, R=model$R, mu=model$m, full.model=model, L=L)
}


############ Initialization of responsibility (intialization) ############
initialization <- function(X, init) {

    d <- nrow(X)

    n <- ncol(X)

    stopifnot(length(init) %in% c(1, n) ||
        (nrow(init) == d  & ncol(init) == k))

    if(length(init) == 1) { # init = k gaussian components

        k <- init

        idx <- sample(1:n, k)

        m <- X[,idx,drop=F]

        label <- apply(bsxfun.se("-", crossprod(m, X),
            as.matrix(dot.ext(m,m,1)/2)), 2, which.max)

        # unique to have consecutive label eg 2,3,6 changed to 1,2,3
        label <- match(label, sort(unique(label)))

        while(k != length(unique(label))) {

            idx <- sample(1:n, k)

            m <- X[,idx,drop=F]

            label <- apply(bsxfun.se("-", crossprod(m, X),
                as.matrix(dot.ext(m,m,1)/2)), 2, which.max)

            label <- match(label, unique(label))
        }

        R <- as.matrix(sparseMatrix(1:n, label, x=1))

    } else {

        if(length(init) == n) { # initialize with labels

            label <- init
            k <- max(label)
            R <- as.matrix(sparseMatrix(1:n, label, x=1))
        } else {

            if(!is.null(dim(init))) {

                if(nrow(init) == d  & ncol(init) == k) { # initialize with centers

                    k <- ncol(init)
                    m <- init
                    label <- apply(bsxfun.se("-", crossprod(m, X),
                        as.matrix(dot.ext(m,m,1)/2)), 2, which.max)
                    R <- as.matrix(sparseMatrix(1:n, label, x=1))

                } else stop(message("Invalid init."))
            }
        }
    }

    R
}


############ Variational-Maximimization ############
vmax <- function(X, model, prior) {

    alpha0 <- prior$alpha
    kappa0 <- prior$kappa
    m0 <- prior$m;
    v0 <- prior$v;
    M0 <- prior$M;
    R <- model$R;

    nk <- apply(R, 2, sum) # 10.51
    alpha <- alpha0 + nk # 10.58
    nxbar <- X %*% R
    kappa <- kappa0 + nk # 10.60
    m <- bsxfun.se("*", bsxfun.se("+", nxbar, kappa0 * m0), 1/kappa) # 10.61
    v <- v0 + nk # 10.63 (NB: no 1 in the matlab code)

    d <- nrow(m)
    k <- ncol(m)

    M <- array(0, c(d, d, k))
    sqrtR <- sqrt(R)

    xbar <- bsxfun.se("*", nxbar, 1/nk) # 10.52
    xbarm0 <- bsxfun.se("-", xbar, m0)
    w <- (kappa0 * nk) * (1/(kappa0 + nk))

    for(i in 1:k) {

        Xs <- bsxfun.se("*", bsxfun.se("-", X, xbar[,i]), t(sqrtR[,i]))

        xbarm0i <- xbarm0[,i]
```

```r
        # 10.62
        M[,,i] <- M0 + tcrossprod(Xs, Xs) + w[i] * tcrossprod(xbarm0i, xbarm0i)
    }

    model$alpha <- alpha
    model$kappa <- kappa
    model$m <- m
    model$v <- v
    model$M <- M # Whishart: M = inv(W)

    model
}


############ Variational-Expectation ############
vexp <- function(X, model) {

    alpha <- model$alpha    # Dirichlet
    kappa <- model$kappa    # Gaussian
    m <- model$m            # Gasusian
    v <- model$v            # Whishart
    M <- model$M            # Whishart: inv(W) = V'*V

    n <- ncol(X)
    d <- nrow(m)
    k <- ncol(m)

    logW <- array(0, dim=c(1,k))
    EQ <- array(0, dim=c(n,k))

    for(i in 1:k) {

        U <- chol(M[,,i])

        logW[i] <- -2 * sum(log(diag(U)))

        Q <- solve(t(U), bsxfun.se("-", X, m[,i]))

        EQ[,i] <- d/kappa[i] + v[i] * dot.ext(Q,Q,1)    # 10.64
    }

    vd <- bsxfun.se("-", matrix(rep(v+1, d),nrow=d,byrow=T), as.matrix(1:d))/2

    ElogLambda <- colSums(digamma(vd)) + d*log(2) + logW    # 10.65
    Elogpi <- digamma(alpha) - digamma(sum(alpha))          # 10.66

    logRho <- (bsxfun.se("-", EQ, 2*Elogpi + ElogLambda - d*log(2*pi)))/(-2) # 10.46
    logR <- bsxfun.se("-", logRho, logsumexp(logRho, 1))    # 10.49
    R <- exp(logR)

    model$logR <- logR
    model$R <- R
    model
}



############ Variational-(lower)-Bound Evaluation ############
vbound <- function(X, model, prior) {

    alpha0 <- prior$alpha
    kappa0 <- prior$kappa
    m0 <- prior$m
    v0 <- prior$v
    M0 <- prior$M

    alpha <- model$alpha    # Dirichlet
    kappa <- model$kappa    # Gaussian
    m <- model$m            # Gasusian
    v <- model$v            # Whishart
    M <- model$M            # Whishart: inv(W) = V'*V
    R <- model$R
    logR <- model$logR

    d <- nrow(m)
    k <- ncol(m)
    nk <- colSums(R)                                    # 10.51

    Elogpi <- digamma(alpha) - digamma(sum(alpha))      # 10.66

    Epz = dot(nk, Elogpi)                               # 10.72
    Eqz = dot(as.numeric(R), as.numeric(logR))          # 10.75
    # logCalpha0 = lgamma(k * alpha0) - k * lgamma(alpha0) # for scalar alpha0
    logCalpha0 = lgamma(sum(alpha0)) - sum(lgamma(alpha0))
    # Eppi <- logCalpha0+(alpha0-1)*sum(Elogpi) # for scalar alpha0
    Eppi <- logCalpha0+dot(alpha0-1, Elogpi)            # 10.73
    logCalpha <- lgamma(sum(alpha))-sum(lgamma(alpha))
    Eqpi = dot(alpha-1, Elogpi) + logCalpha             # 10.76

    # part of 10.70
    L <- Epz - Eqz + Eppi - Eqpi

    U0 <- chol(M0)
    sqrtR <- sqrt(R)
    xbar <- bsxfun.se("*", X %*% R, 1/nk)               # 10.52

    logW <- array(0, dim = c(1, k))
    trSW <- array(0, dim = c(1, k))
    trM0W <- array(0, dim = c(1, k))
    xbarmWxbarm <- array(0, dim = c(1, k))
    mm0Wmm0 <- array(0, dim = c(1, k))

    for(i in 1:k) {

        U <- chol(M[,,i])
        logW[i] <- -2 * sum(log(diag(U)))

        Xs <- bsxfun.se("*", bsxfun.se("-", X, as.matrix(xbar[,i,drop=F])), t(sqrtR[,i,drop=F]))
        V <- chol(tcrossprod(Xs, Xs)/nk[i])
        Q <- solve(U, V)
        # equivalent to tr(SW)=trace(S/M)
        trSW[i] <- dot(as.numeric(Q), as.numeric(Q))
        Q <- solve(U, U0)
        trM0W[i] <- dot(as.numeric(Q), as.numeric(Q))

        q <- solve(t(U), xbar[,i,drop=F]-m[,i,drop=F])
        xbarmWxbarm[i] = dot(q, q)
        q <- solve(t(U), m[,i,drop=F]-m0)
        mm0Wmm0[i] <- dot(q, q)
    }

    vd <- bsxfun.se("-", matrix(rep(v+1, d),nrow=d,byrow=T), as.matrix(1:d))/2
    ElogLambda <- colSums(digamma(vd)) + d*log(2) + logW    # 10.65

    # first half of 10.74
    Epmu <- sum(d*log(kappa0/(2*pi))+ElogLambda-d*kappa0/kappa-kappa0*(v*mm0Wmm0))/2
    logB0 <- v0*sum(log(diag(U0)))-0.5*v0*d*log(2)-logmvgamma(0.5*v0,d) # B.79
    # second half of 10.74
    EpLambda <- k*logB0+0.5*(v0-d-1)*sum(ElogLambda)-0.5*dot(v,trM0W)


    Eqmu <- 0.5*sum(ElogLambda+d*log(kappa/(2*pi)))-0.5*d*k # 10.77 (1/2)
    logB <- (-v) * (logW+d*log(2))/2 - logmvgamma(0.5*v, d) # B.79
    EqLambda <- 0.5*sum((v-d-1)*ElogLambda-v*d)+sum(logB)   # 10.77 (2/2)

    EpX <- 0.5*dot(nk, ElogLambda-d/kappa-v*trSW-v*xbarmWxbarm-d*log(2*pi)) # 10.71

    L <- L+Epmu-Eqmu+EpLambda-EqLambda+EpX  # 10.70

    L
}
```