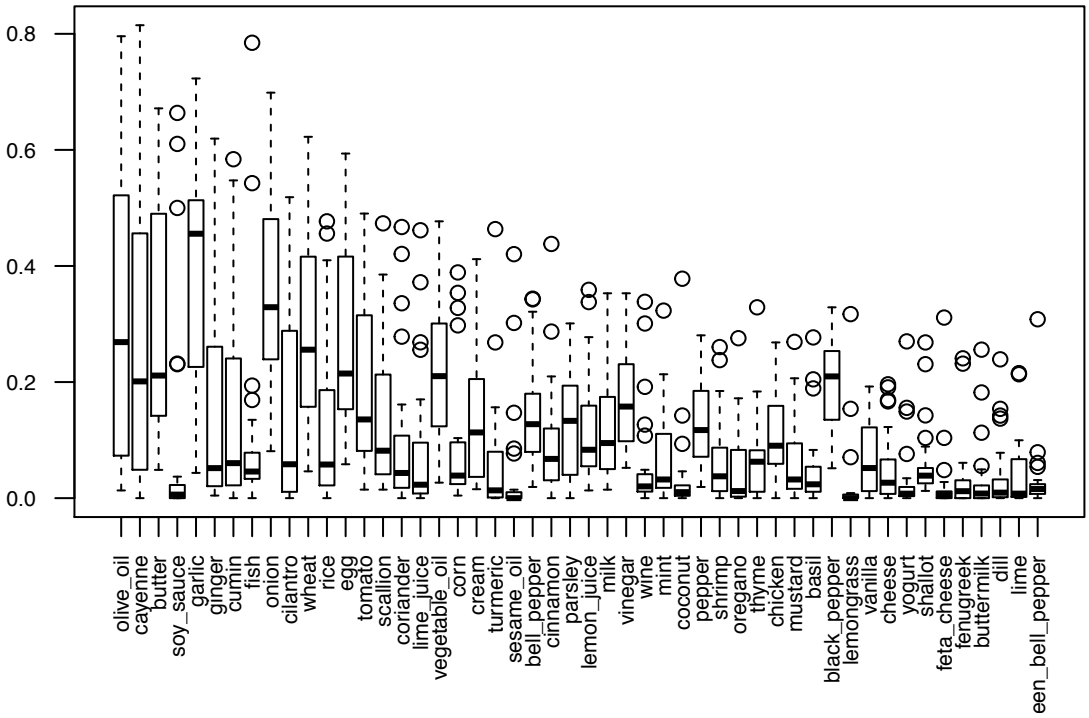
Projet SY09

Aurélien Roy, Aiman Zaki

Cuisine

Analyse exploratoire

Ce diagramme en boîte suivant permet de visualiser l'importance générale de chaque ingrédient.



On remarque que les ingrédients les plus courants, à gauche, dont les bornes supérieures sont plus élvées, sont principalements des sauces ou des épices.

Ces ingrédients se retrouvent dans un grand nombre de plats.

Nous pouvons aussi étudier les corrélations entre ingrédients. Deux ingrédients fortement corrélés positivement vont avoir tendance à avoir une répartition géographique semblable.

Les 10 couples d'ingrédients les plus correlés

```
##      Ingredient 1 Ingredient 2 Pearson cor. factor
## 2417   lime_juice         lime           0.9735043
## 1016    coriander     turmeric           0.9387638
## 1054    soy_sauce   sesame_oil           0.9243978
## 2271     turmeric    fenugreek           0.9228333
## 712          rice     scallion           0.8866110
## 2013          egg      vanilla           0.8862417
## 2190   lemongrass      shallot           0.8832732
## 1981      coconut   lemongrass           0.8777349
```

```
## 2003       butter      vanilla           0.8686524
## 1303       butter         milk           0.8671370
```

Les 10 couples d'ingrédients les moins correlés

```
##      Ingredient 1      Ingredient 2 Pearson cor. factor
## 2239 black_pepper       feta_cheese        0.0039181643
## 1153       butter          cinnamon        0.0037587701
## 1108         fish       bell_pepper       -0.0034788094
## 2223  bell_pepper       feta_cheese       -0.0028335221
## 1657        cumin           oregano        0.0024666484
## 2238        basil       feta_cheese       -0.0023453783
## 2123  bell_pepper            yogurt        0.0022270875
## 1564       tomato            pepper       -0.0017360606
## 1778      vinegar           chicken        0.0011025282
## 2492       cheese green_bell_pepper        0.0007387798
```

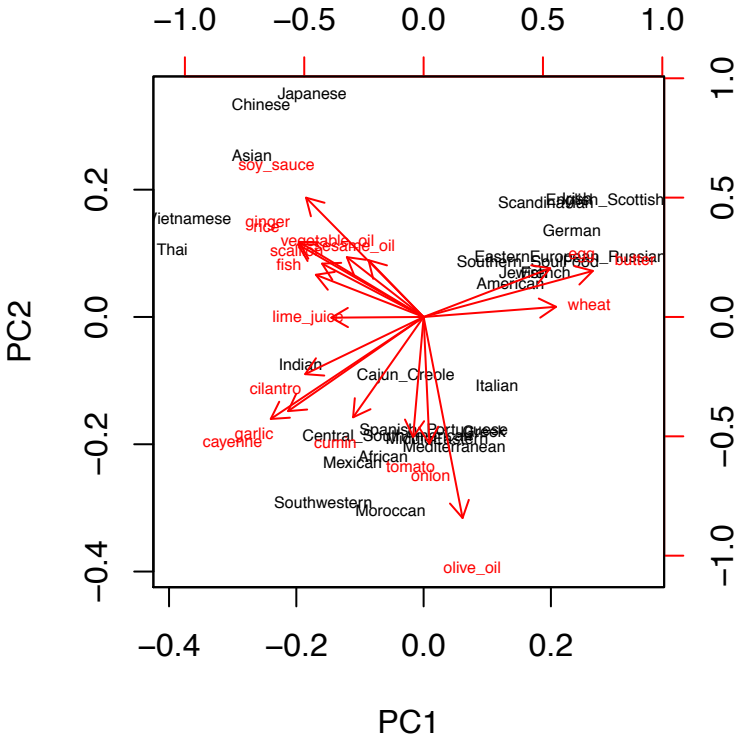
Analyse en composantes principales

On réalise ensuite une analyse en composantes principales. La fonction princomp ne peut être utilisée, car nous avons plus de variables que d'individus. En revanche, la fonction prcomp, gère très bien ce cas.

Inertie des axes

On remarque que 90% de l'intertie est expliqué par les 6 premiers axes. Sur le plan, on obtient la réprésentation suivante (seuls les axes ayant les plus grandes longueurs sont conservés).

```
##  [1] 0.3682744 0.6100062 0.7195924 0.8074238 0.8680652 0.9007243 0.9196207
##  [8] 0.9353748 0.9466509 0.9560694 0.9645050 0.9721877 0.9776894 0.9825509
## [15] 0.9859029 0.9890716 0.9915808 0.9938559 0.9955272 0.9970088 0.9979818
## [22] 0.9988064 0.9994902 0.9997723 1.0000000 1.0000000
```



L'analyse en composante principales nous apprend que certaines cuisines peuvent être rapprochées ensemble (par exemples, toutes les cuisines asiatiques).

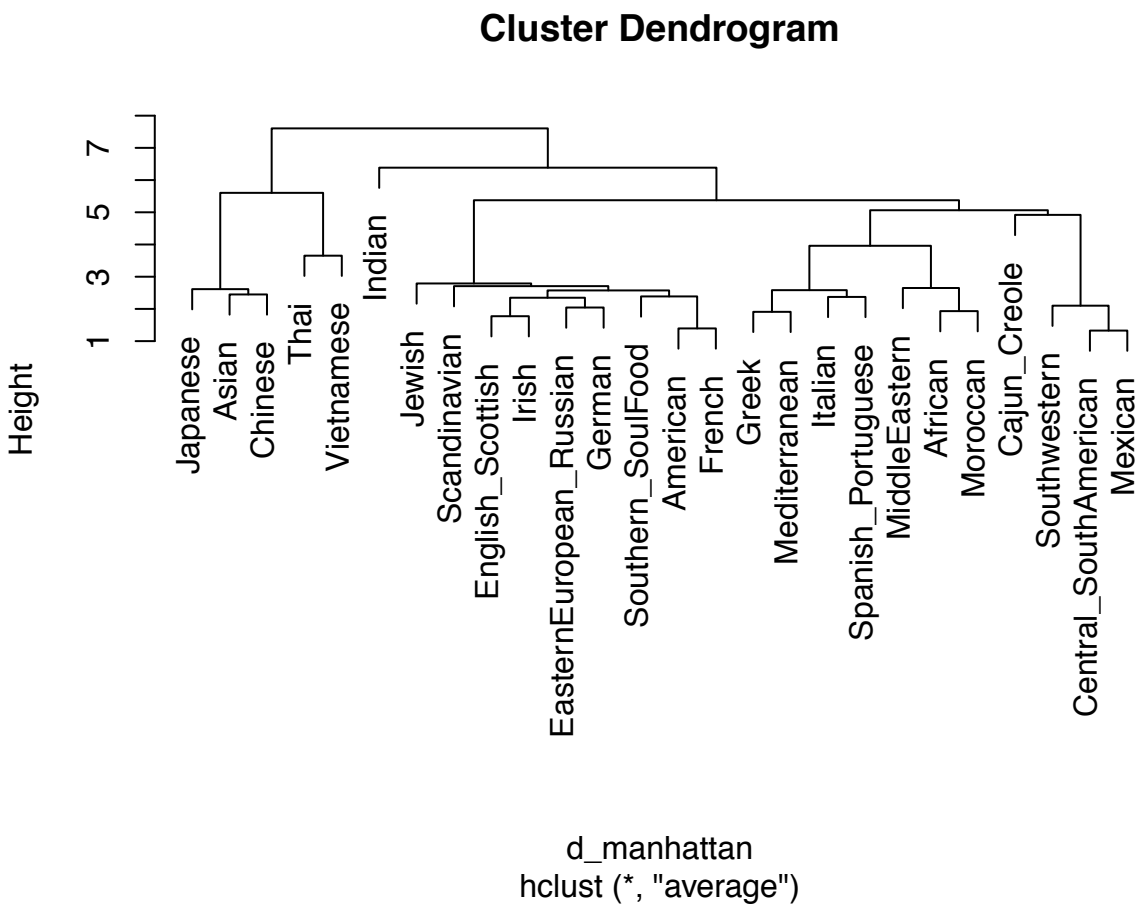
Il est difficile de faire une interprétation du premier axe principal.

En revanche, en prenant la combinaison des deux premiers axes, on observe une série d'axes diagonaux (sauce soja, gingembre, riz) pointant vers toutes les cuisines asiatiques. De la même manière, la seconde composante principale, étudiée seule, semble coder l'aspect plus ou moins tropical/méditaranéen des pays d'origine des recettes

Ainsi, on retrouve une série d'axes pointant dans la même direction (huile d'holive, tomate, oignon), que l'on retrouve beaucoup dans la cuisine méditéranéenne.

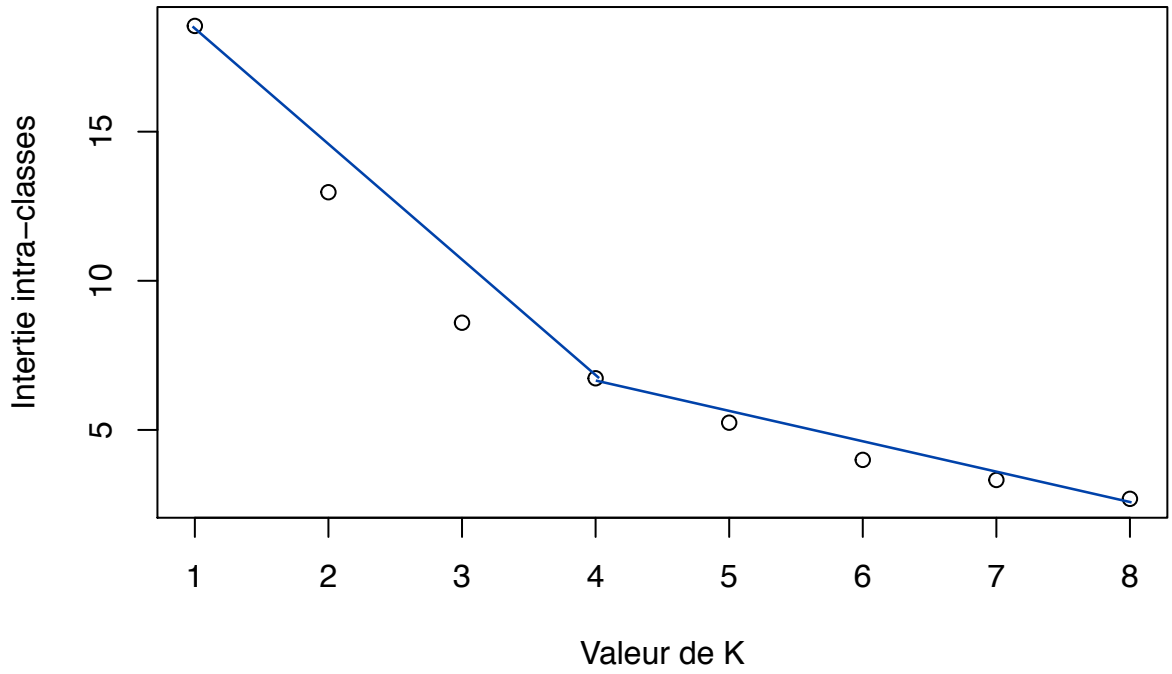
Classification ascendante hiérarchique

```
# Calcul de la distance de Manhattan
d_manhattan <- dist(R, method="manhattan")

# Classifcation hiérarchique avec le critère de distance moyenne
plot(hclust(d_manhattan, method="average"))
```



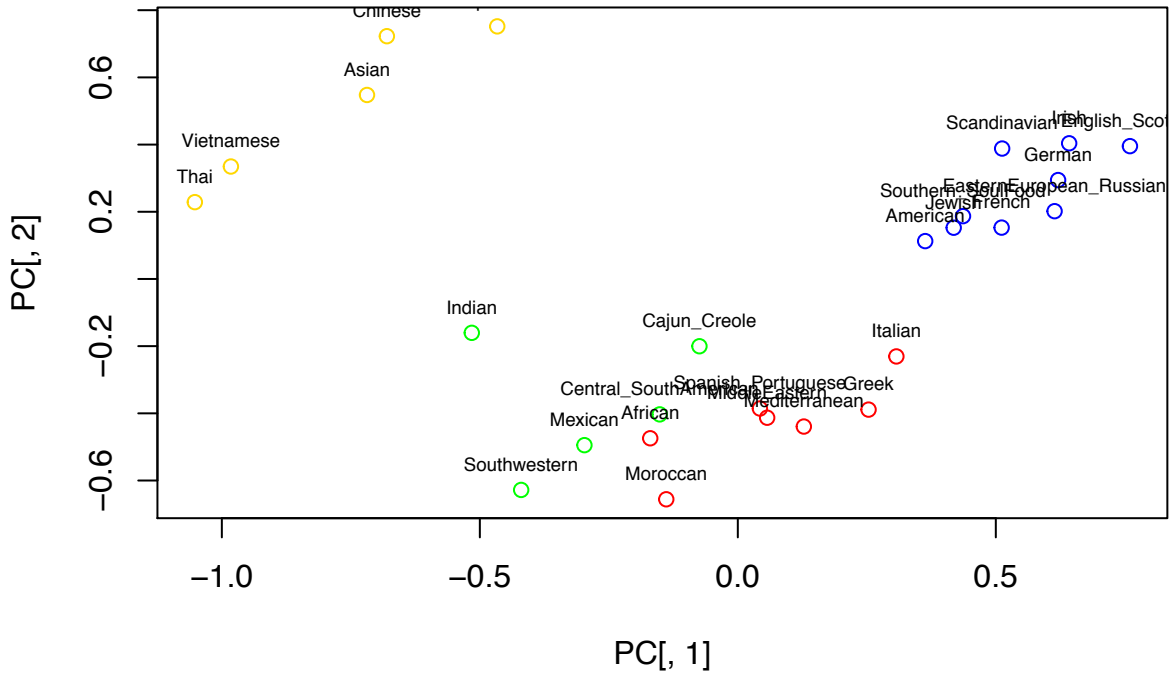
Méthode des K-Means

```
L <- function(D) lapply(1:100, function(t) kmeans(R, centers=D, nstart=1))
wtot <- function(D) min(sapply(L(D), function(l) l$tot.withins))
plot(1:8, sapply(1:8, wtot), ylab="Intertie intra-classes", xlab="Valeur de K")
```



En observant la décroissance des interties intra-classes avec l'augmentation du nombre de classes ; la méthode du coude revèle un changement de pente autour de k = 4.

On choisit donc d'utiliser 4 classes :



La méthode des K-Means révèle quatre groupes : D'abord la quasi-totalité des cuisines asiatiques ont bien été regroupées (thaïlandaise, vietnamienne, japonaise, chinoise...). Seule l'Inde a été placé dans un autre groupe.

Le second cluster contient un certain nombre de cuisines méditéranéenne (Italie, Grèce, Maroc, Espagne, Portugal).

Le troixième cluster rassemble les cuisines d'Amérique latine (la cuisine mexicaine et la cuisine créole), ainsi que l'Inde qui semble faire figure d'intrus du point de vue géographique.

Enfin dans le dernier cluster, on retrouve toute la cuisine occidentale", constituée de l'Europe (sauf Europe du Sud) et de l'Amérique du Nord.

Globalement, on retrouve quand même beaucoup d'associations reflétant une proximité géographique mais cette ressemblance n'est pas parfaite.

On fait le même constat dans la classification hiérarchique ; à quelques exceptions près (ex : France et Amérique) ; les associations faîtes dans le bas de la hirarchie reflètent des proximités géographiques.

Analyse du jeu de données Recettes Echant

```
R_echant <- read.csv('./donnees/recettes-echant.data')
```

Le jeu de données est une liste de 2000 recettes pour lesquelles on dispose de la présence ou non de 51 ingrédients ainsi que son origine géographique.

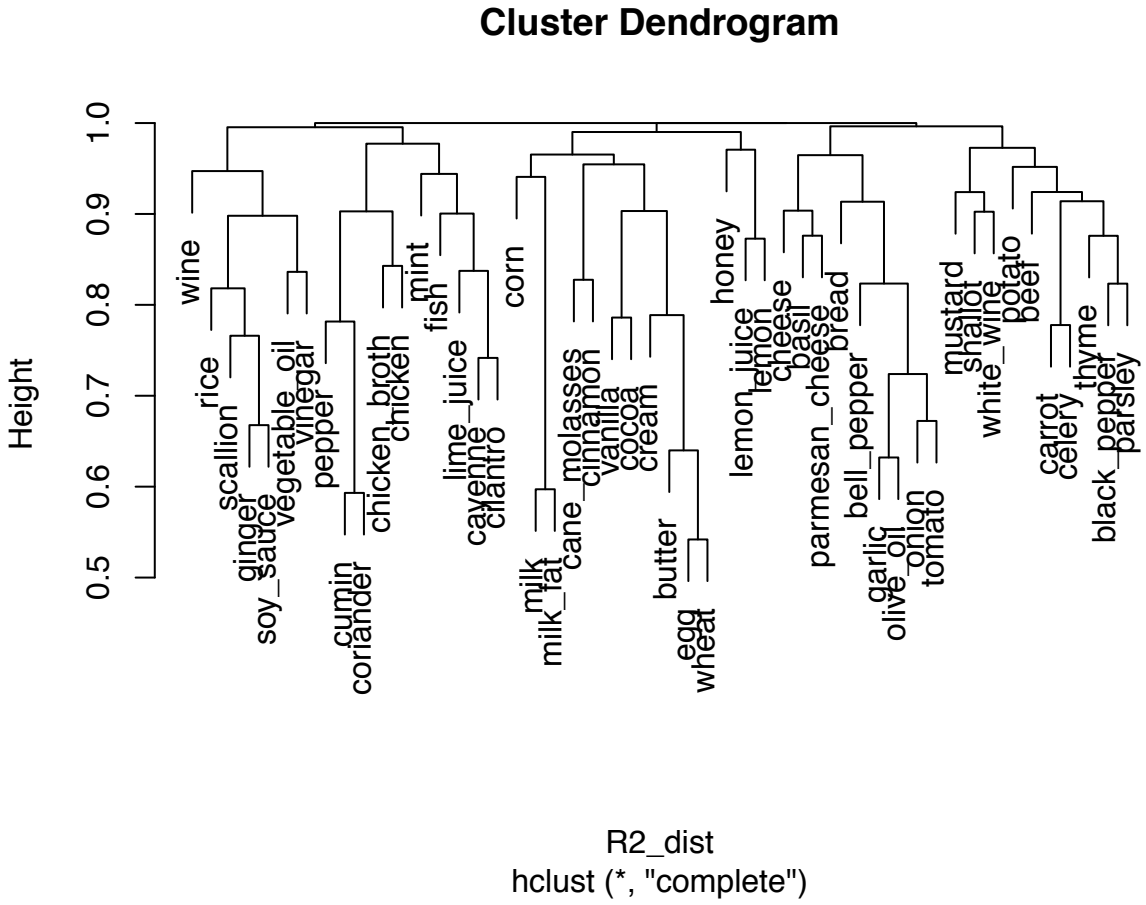
Pour transformer le jeu de données en tableau d'individus-variables sur les ingrédients, on applique simplement la transposée, et on retire la colonne origine.

```
R2 <- t(as.matrix(R_echant[, -1]))
```

Matrice de dissimilarité

Comme notre jeu de données est exclusivement binaire, la distance binaire paraît bien adaptée.

```
R2_dist <- dist(R2, method="binary")
plot(hclust(R2_dist))
```



On observe la présence de six groupes. Toutefois, nous n'observons pas de cohérence dans la séparation proposée.

Algorithme des K-médoïdes

```
library(cluster)
medoids <- pam(R2_dist, k = 6)
print(medoids$medoids)
```

```
## [1] "olive_oil" "egg"       "cilantro"  "soy_sauce" "coriander" "celery"
```

Classification par K-Means avec distance adaptative

Programmation

Ci-dessous, une implémentation en R de l'algorithme des K-Means avec une distance adaptative.

```r
kmeans_adpt <- function(X, K, n_iter = 100, n_ess = 10, precision = 1e-5){
  X = as.matrix(X)
  n = nrow(X)
  p = ncol(X)
  J = Inf

  ro = rep(1, K) # On définit chaque ro_k à 1
  seq_1_K = as.array(seq(K)) # [1, 2, ..., K]

  for(e in 1:n_ess){
    tryCatch({

      # Initialisation des K matrices Vk.
      V_k = sweep(replicate(K, diag(p)), 3, ro**(1-p), "*")
      centers = as.matrix(X[sample(n, K),,drop=FALSE])

      conv = Inf
      i = 0

        while(i < n_iter && conv > precision){
          # Calcul des distances des individus à chaque centre
          distances = apply(seq_1_K, 1, function(k) distXY(X, centers[k,], solve(V_k[,,k])))

          # Calcul du cluster le plus proche pour chaque individu
          clusters = max.col(-distances) # equ. à apply(which.min())

          prev_centers = centers
          # Pour chaque cluster...
          for(k in 1:K){

            elements = X[clusters == k,, drop=FALSE]
            n_k = nrow(elements)

            # Calcul du nouveau centre
            centers[k,] = apply(elements, 2, mean)

            # Calcul du Vk associé
            V_k[,,k] = cov(elements) * (n_k-1) / n_k

            # Normalisation de Vk
            V_k[,,k] = (ro[k] * det(V_k[,,k]))**(-1/p) * V_k[,,k]
          }

          # Calcul de la convergence
          conv = sum(apply(centers-prev_centers, 1, function(x) { sqrt(sum(x**2)) }))

          i = i + 1
        }
```

```
        candidate_J <- sum(apply(seq_1_K, 1, function(k){
          sum(distXY(X[clusters == k,, drop=FALSE], centers[k,], V_k[,,k]))
        }))

        if(candidate_J < J){
          opt_Vk = V_k
          opt_clusters = clusters
          opt_centers = centers

          J = candidate_J
        }

      }, error = function(e){ })

  }
    return (list("cluster" = opt_clusters, "centers" = opt_centers, "vk" = opt_Vk, "tot.distances" = J))
}
```

L'algorithme retourne la partition, les centres des classes, les K matrices Vk ainsi que le critère J optimal, égal à la somme des distances des points à leur centre.

Remarquons la présence d'une clause tryCatch au début du corps de la boucle des essais. Nous avons rencontré un problème lors de certaines éxécutions de l'algorithme au niveau du calcul du déterminant des matrices de covariances des classes.

Bien que cela soit en théorie impossible, il arrivait que l'on obtienne des déterminants extrêmement petits négatifs. Le déterminant d'une matrice de covariance ne devrait jamais être négatif, et cette erreur est dû aux limitations des microprocesseurs quant aux calculs avec des valeurs à virgule flottante.

Pour contourner ce problème, on fait en sorte que lorsque la fonction distXY lance une erreur, celle-ci soit rattrapée par l'instruction tryCatch et qu'on passe à l'essai suivant. Il est probable que cette erreur ne réapparaisse pas à l'essai suivant dû à une configuration initiale différente. Nous sommes conscients que cette solution relève de la "bidouille", mais nous n'avons rien trouvé de mieux.

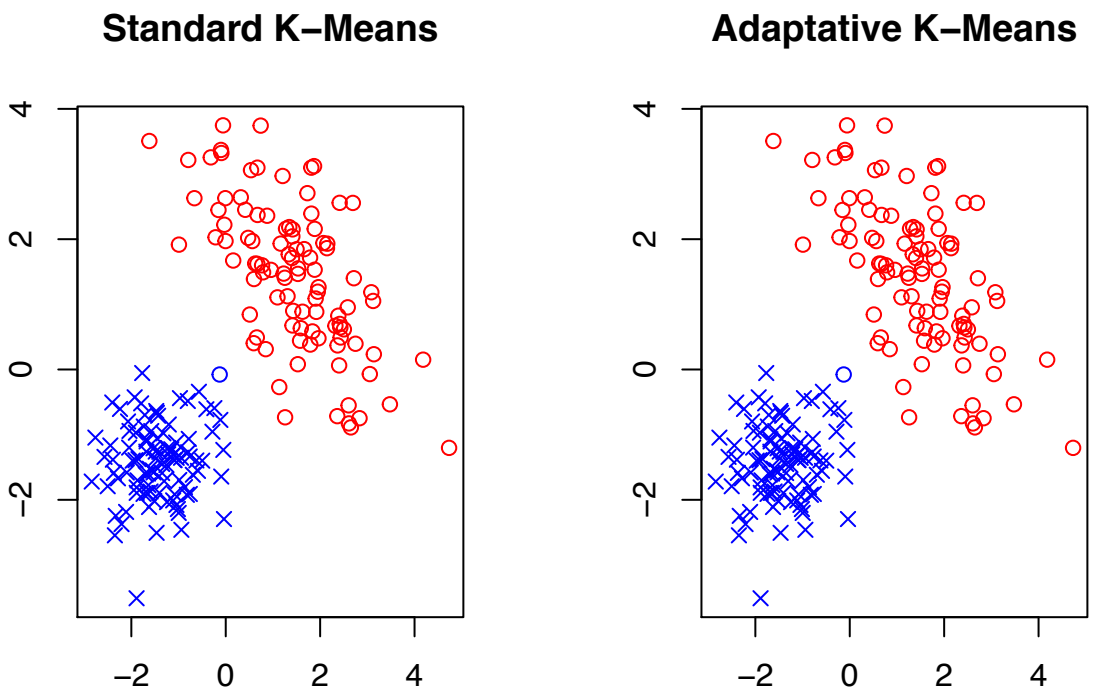
Pour comparer les deux algorithmes, nous avons crée la fonction compare_kmeans, qui lance l'algorithme K_Means standard le même nombre de fois qu'il y a d'essais dans notre algorithme adaptatif (afin qu'on puisse se faire une opignon non biaisée).

Aussi, la fonction tente de faire un semblant d'association des clusters retournés par les deux fonctions, afin que deux clusters semblables sur les partitions des deux algorithmes soient affichés de la même couleur. La fonction prend également en paramètre l'étiquetage des données et utilise la forme des points (croix, cercles, ...) pour représenter la vraie classe de chaque individu.

Applications

Synth1

```
S <- read.csv('./donnees/Synth1.csv', header = T, row.names = 1)
X <- S[, -3]
Z <- S[, 3]

km <- compare_kmeans(X, Z, K=2)
```

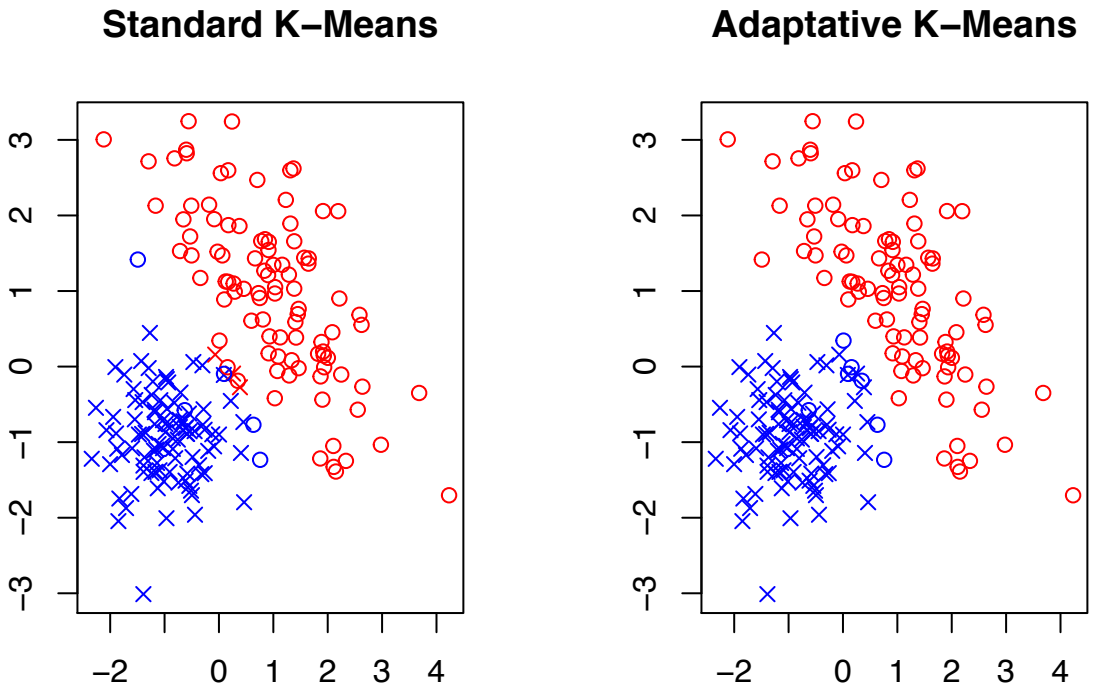



```
## [1] "Standard K-Means : 0.979999505050755"
## [1] "Adaptive K-Means : 0.979999505050755"
```

Sur ce premier jeu de données, les deux classes sont nettement séparées. Si bien, que les deux algorithmes performent tous les deux très bien.

Synth2

```
S <- read.csv('./donnees/Synth2.csv', header = T, row.names = 1)
X <- S[, -3]
Z <- S[, 3]

km <- compare_kmeans(X, Z, K=2)
```

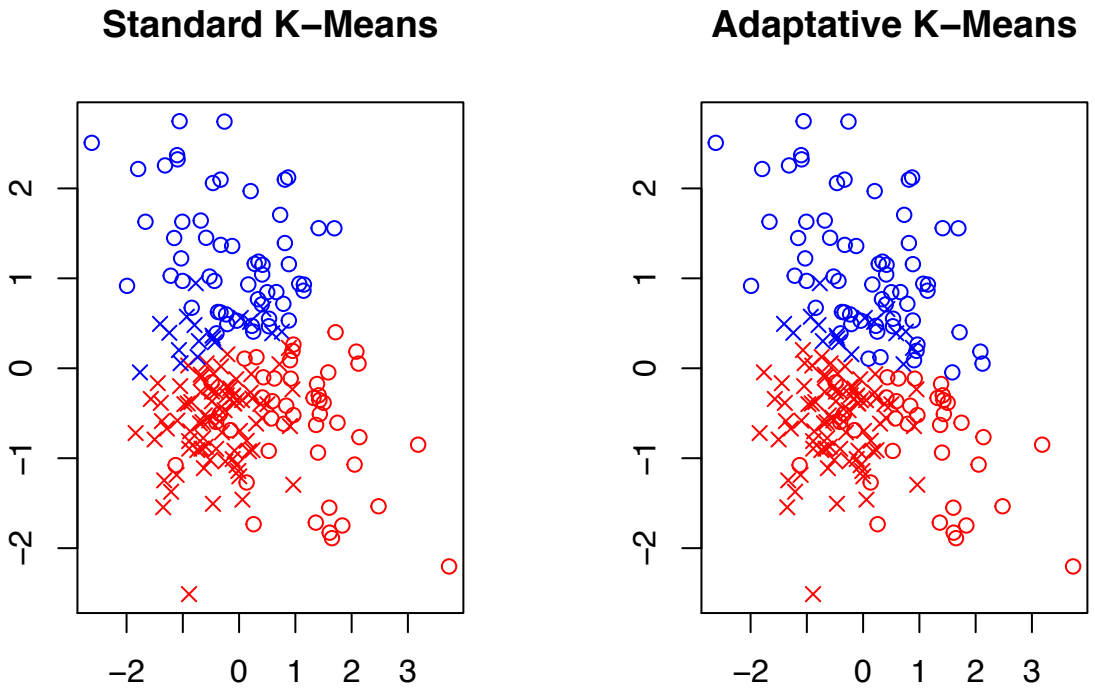



```
## [1] "Standard K-Means : 0.84562455429383"
## [1] "Adaptive K-Means : 0.864221036954136"
```

Sur ce second jeu de données, la séparation sur le plan est moins marquée. L'algorithme K-Means classique a tendance à inclure à tort dans la classe condensée, des points éloignés du centre de la classe allongée. L'algorithme adaptatif propose un meilleur équilibre, les classes proposées semblent un peu plus compactes : on perçoit l'inffluence de la covariance dans le calcul des distances aux centres.

Synth3

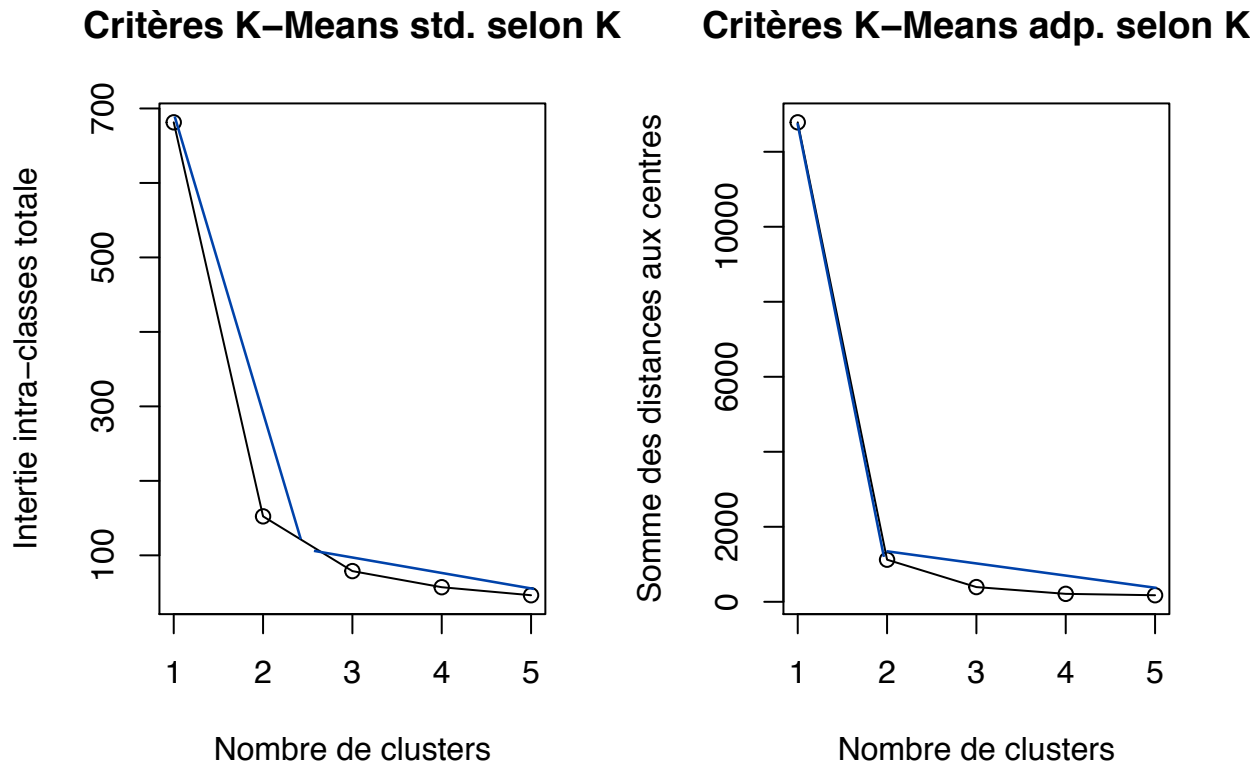
```
S <- read.csv('./donnees/Synth3.csv', header = T, row.names = 1)
X <- S[, -3]
Z <- S[, 3]

km <- compare_kmeans(X, Z, K=2)
```




```
## [1] "Standard K-Means : 0.132860174208587"
## [1] "Adaptive K-Means : 0.217107890935037"
```

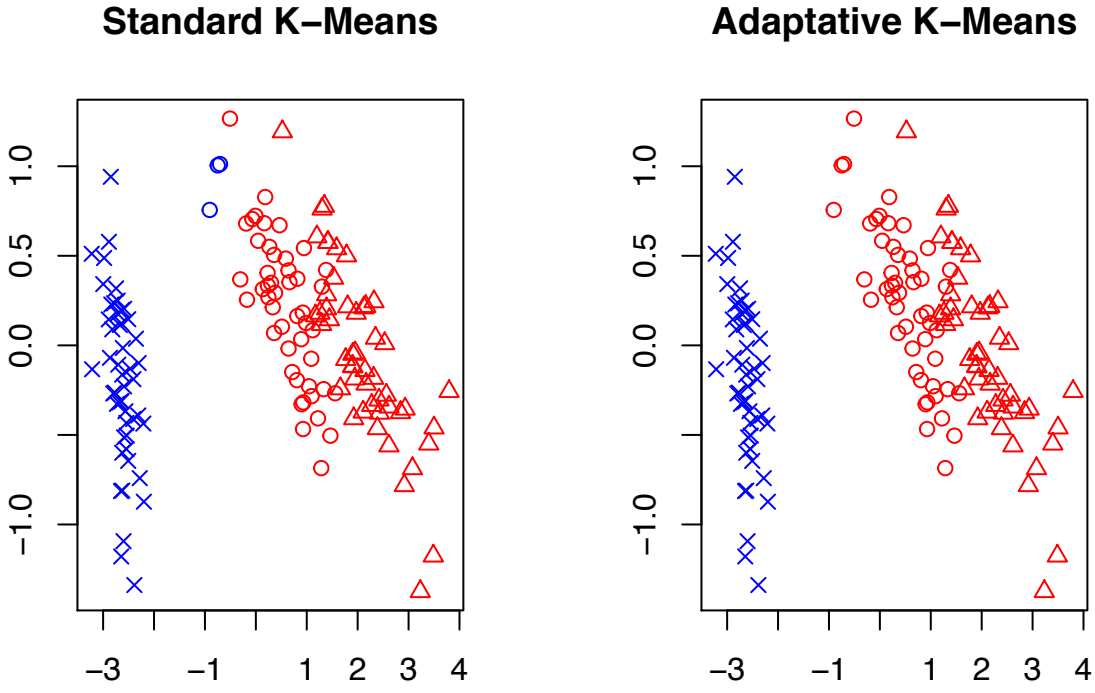
Ici, les deux classes paraissent confondues, mais qu'il y a une différence de densité perceptible (c'est à dire la présence une zone où un grand nombre de points sont concentrés), l'algorithme adaptatif obtient un score bien meilleur que l'algorithme standard.

Données iris



En utilisant la méthode du coude, on conclut que le nombre optimal de clusters pour l'algorithme standard se situe entre 2 et 3. Pour l'algorithme K-Means adaptatif, la partition optimale est nettement obtenue pour K = 2 clusters.

```
km <- compare_kmeans(X_iris, Z_iris, K=2)
```

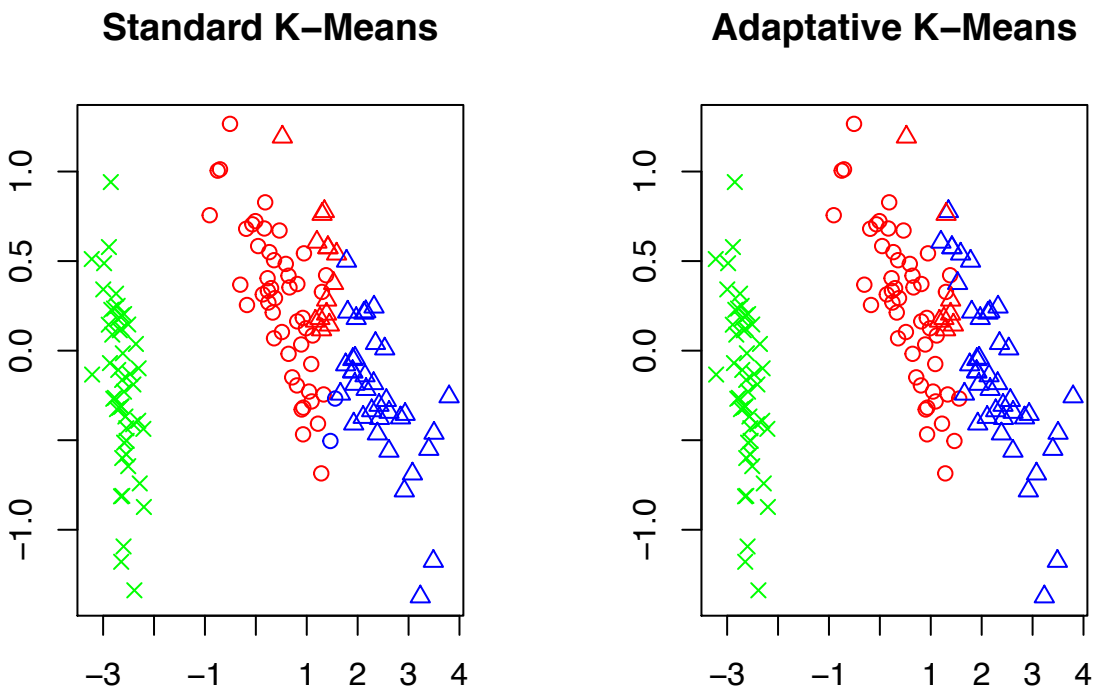


```
## [1] "Standard K-Means : 0.539921829420712"
## [1] "Adaptive K-Means : 0.568115942028986"
```

Sur une classification à K = 2, l'algorithme adaptatif reconnaît parfaitement l'une des classes réelles du jeu de données (la classe de gauche).

Cette classe est mal délimitée par l'algoirthme standard des K-Means, du fait que certains points de classe de droite sont plus proche du centre de la classe de gauche

```
km <- compare_kmeans(X_iris, Z_iris, K=3)
```

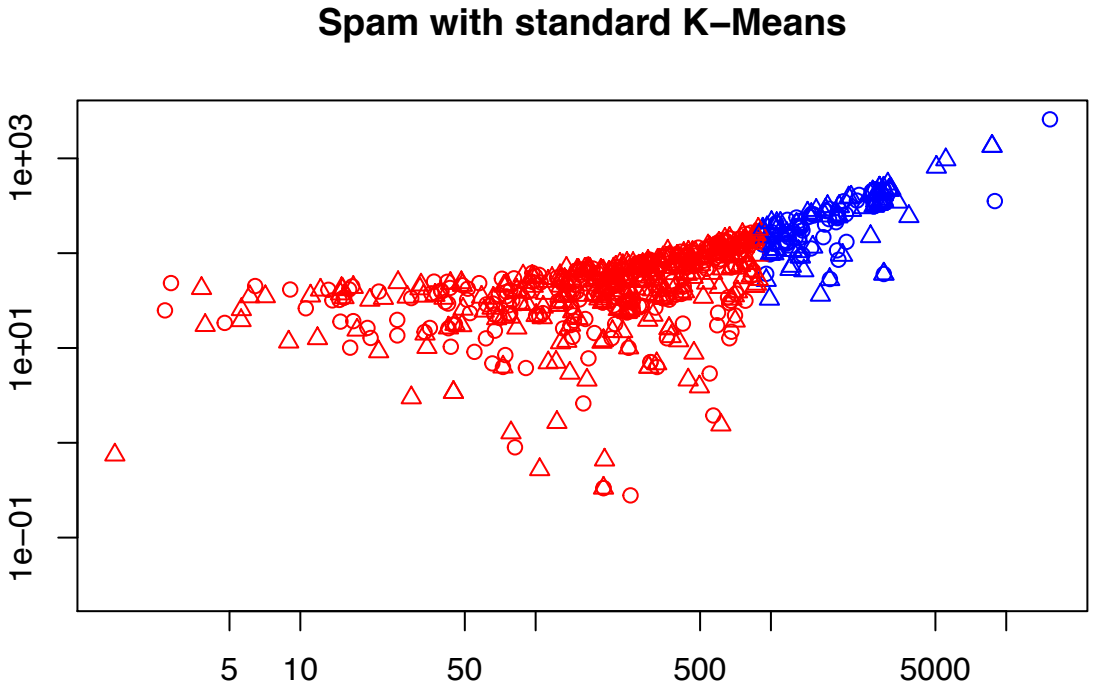



```
## [1] "Standard K-Means : 0.73023827228347"
## [1] "Adaptive K-Means : 0.851456931395815"
```

Dès qu'on passe à K = 3, l'algorithme des K-Means délimite correctement la classe de gauche, du fait du repositionnement des autres centres. En revanche, l'algorithme adaptatif s'en sort toujours mieux pour séparer les deux classes de droites qui semblent assez confondues sur le plan.

Données Spam

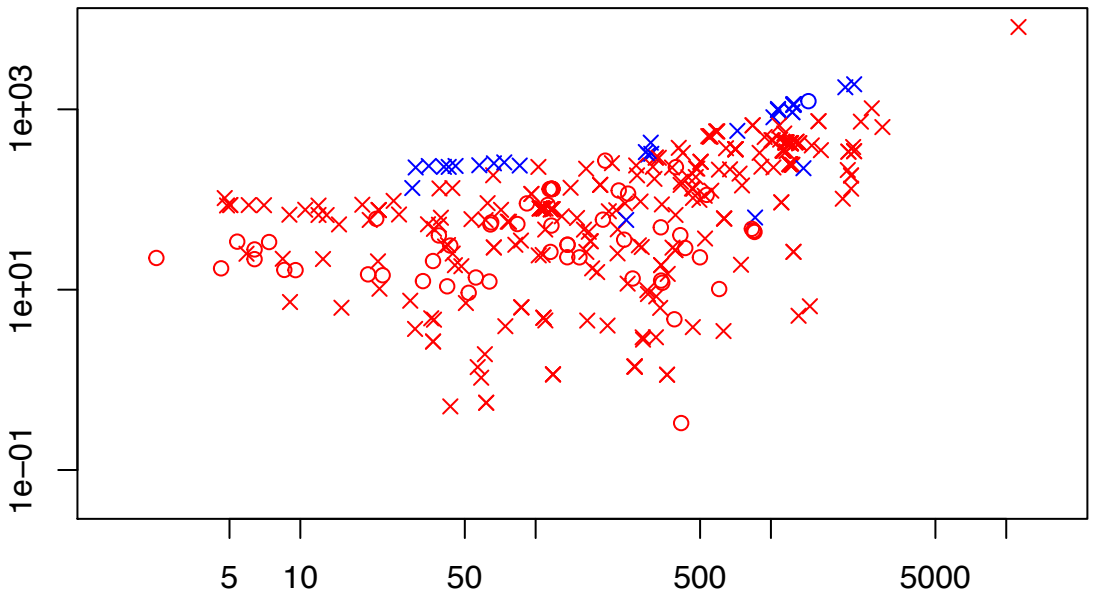
```
Spam <- read.csv("./donnees/spam.csv", header=T, row.names=1)
X_spam <- Spam[,-58]
Z_spam <- Spam[,58]

X_PC <- princomp(X_spam)$scores
km <- std_kmeans(X_spam, K=2, n_ess=10)
plot(X_PC[, 1], X_PC[, 2], col=c("red", "blue")[km$cluster], pch=c(1, 2), xlab="", ylab="", main="Spam
```



Remarquons d'abord que l'algorithme des K-Means standard gère très mal ce jeu de données. Cela peut s'expliquer par l'éventuelle confusion des deux classes (bien qu'il soit difficile de la visualiser compte tenu du nombre de dimensions) mais surtout par la présence de points très éloignés.

Il s'agit peut-être de points aberrants, mais ne connaissant pas la nature des données ni le protocole qui a permis des les obtenir, nous ne pouvons en être sûr. Ainsi, nous n'allons pas retirer ces points éloignés. Notons que les graphiques de cette partie utilisent une échelle logarithmique.

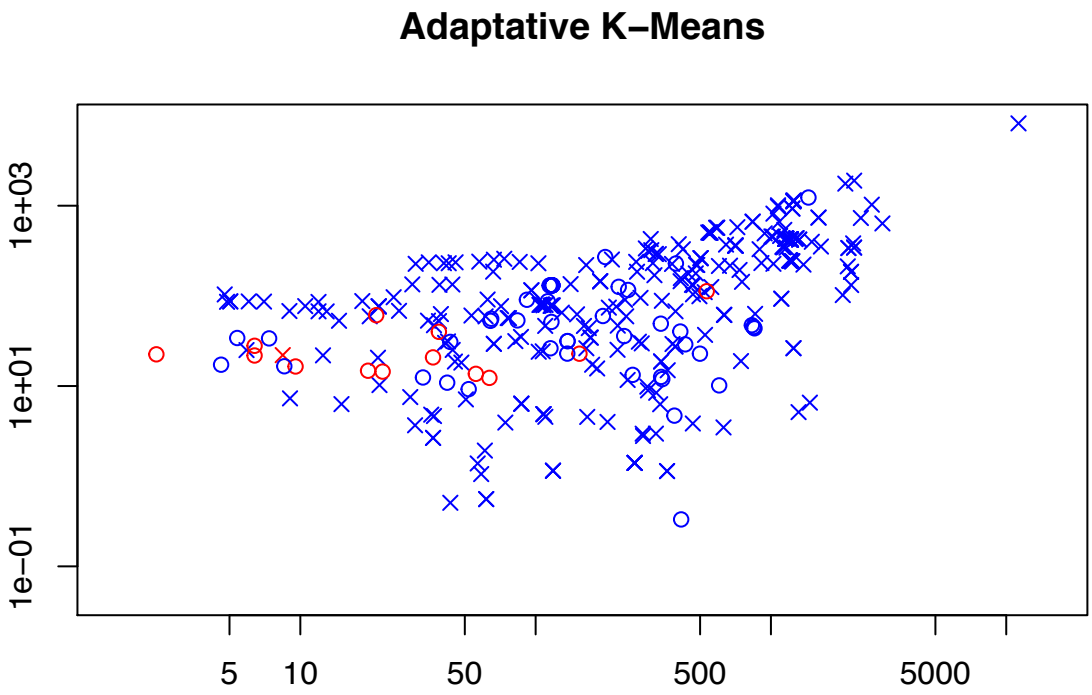
```
km <- compare_kmeans(X_PC[, 1:3], Z_spam, K=2, plot_std=FALSE, log_scale = TRUE)
```




```
## [1] "Standard K-Means : 0.039417249005261"
## [1] "Adaptive K-Means : 0.0166838639173034"
```

L'application directe de l'algorithme des K-Means adaptatifs ne fonctionne pas, en raison du trop grand nombre de dimensions qui conduit à un manque de précision dans les calculs (obtention de déterminants négatifs pour des matrices de covariance). Néanmoins, en réalisant une ACP, on remarque que 96% de l'information peut être représenté par 3 dimensions.

```
km <- compare_kmeans(X_PC[, 1:50], Z_spam, K=2, plot_std=FALSE, log_scale = TRUE)
```




```
## [1] "Standard K-Means : 0.039417249005261"
## [1] "Adaptive K-Means : 0.394936027230473"
```

Les résultats ne sont pas convaincants. Nous retentons avec cette fois-ci en conservant les 50 premiers axes principaux, ce qui donne un résultat passable.

Justification

Les pages suivantes traitent la justification.

1) $\frac{\partial L(\nu_k, M_k, \lambda_k)}{\nu_k}$

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{\nu_k} = \frac{\partial\left(J(\{\nu_k, M_k\}_{k=1,\dots,K}) - \sum_{k=1}^{K} \lambda_k\,(det\ M_k - \rho_k)\right)}{\partial \nu_k}$$

$$= \frac{\partial J(\{\nu_k, M_k\}_{k=1,\dots,K})}{\partial \nu_k} - \frac{\partial\left(\sum_{k=1}^{K} \lambda_k\,(det\ M_k - \rho_k)\right)}{\partial \nu_k}$$

Et comme $\sum_{k=1}^{K} \lambda_k\,(det\ M_k - \rho_k)$ ne dépend pas de ν_k

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{\nu_k} = \frac{\partial J(\{\nu_k, M_k\}_{k=1,\dots,K})}{\partial \nu_k}$$

L'opérateur dérivée étant linéaire, la dérivée d'une somme est la somme des dérivées, soit :

$$\frac{\partial(\nu_k, M_k, \lambda_k)}{\nu_k} = \frac{\partial\left(\sum_k^K \sum_{i=1}^n z_{ik} d_{ik}^2\right)}{\partial \nu_k} = \sum_{k=1}^{K}\sum_{i=1}^{n} \frac{\partial\left(z_{ik} d_{ik}^2\right)}{\partial \nu_k} = \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik} \frac{\partial\left(d_{M_k}^2(x_i, \nu_k)\right)}{\partial \nu_k}$$

$$= \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik} \frac{\partial((x_i - \nu_k)^T M_k (x_i - \nu_k))}{\partial \nu_k}$$

M_k étant la matrice de Mahalanobis de covariance, elle est symétrique, et selon le Cookbook (86) :

$$\frac{\partial((x_i - \nu_k)^T M_k (x_i - \nu_k))}{\partial \nu_k} = -2M_k(x_i - \nu_k)$$

$$d'o\grave{u}$$

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{\nu_k} = -2\sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik} M_k (x_i - \nu_k)$$

2) $\frac{\partial L(\nu_k, M_k, \lambda_k)}{M_k}$

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{M_k} = \frac{\partial\left(J(\{\nu_k, M_k\}_{k=1,\dots,K}) - \sum_{k=1}^{K} \lambda_k\,(det\ M_k - \rho_k)\right)}{\partial M_k} = \frac{\partial J((\nu_k, M_k)_{k=1,\dots,K})}{\partial M_k} - \frac{\partial\left(\sum_{k=1}^{K} \lambda_k\ det\ M_k - \sum_{k=1}^{K} \lambda_k \rho_k\right)}{\partial M_k}$$

En remplacent $J(\{\nu_k, M_k\}_{k=1,\dots,K}) = \sum_k^K \sum_{i=1}^n z_{ik} d_{ik}^2$:

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{M_k} = \frac{\partial\left(\sum_k^K \sum_{i=1}^n z_{ik} d_{ik}^2\right)}{\partial M_k} - \sum_{k=1}^{K} \lambda_k \frac{\partial(det\ M_k)}{\partial M_k} + 0$$

Selon le Cookbook (49) :

$$\frac{\partial(det\ M_k)}{\partial M_k} = \det M_k (M_k^{-1})^T$$

Et comme $(M_k^{-1})^T = \left(M_k^T\right)^{-1}$ or $M_k^T = M_k$ car symétrique, on obtient :

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{M_k} = \sum_{k=1}^{K}\sum_{i=1}^{n} \frac{\partial\left(z_{ik} d_{ik}^2\right)}{\partial M_k} - \sum_{k=1}^{K} \lambda_k\ det\,(M_k) M_k^{-1}$$

$$= \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}\frac{\partial\left(d^2_{M_k}(x_i,\nu_k)\right)}{\partial M_k} - \sum_{k=1}^{K}\lambda_k\, det\,(M_k)M_k^{-1}$$

$$avec\ d^2_{M_k}(x_i,\nu_k) = (x_i-\nu_k)^T M_k (x_i-\nu_k)$$

Et selon le Cookbook (70) :

$$\frac{\partial\left((x_i-\nu_k)^T M_k (x_i-\nu_k)\right)}{\partial M_k} = (x_i-\nu_k)(x_i-\nu_k)^T$$

$$soit$$

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{M_k} = \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}(x_i-\nu_k)(x_i-\nu_k)^T - \sum_{k=1}^{K}\lambda_k\, det\,(M_k)M_k^{-1}$$

3) $\frac{\partial L(\nu_k,M_k,\lambda_k)}{\lambda_k}$

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{\lambda_k} = \frac{\partial\left(J\left(\{\nu_k,M_k\}_{k=1,\dots,K}\right) - \sum_{k=1}^{K}\lambda_k\,(det\,M_k-\rho_k)\right)}{\partial\lambda_k} = 0 - \frac{\partial\left(\sum_{k=1}^{K}\lambda_k\,(det\,M_k-\rho_k)\right)}{\lambda_k}$$

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{\lambda_k} = -\sum_{k=1}^{K}\left(\frac{\partial\lambda_k}{\partial\lambda_k}(det\,M_k-\rho_k)\right) = \sum_{k=1}^{K}(\rho_k - det\,M_k)$$

Le Lagrangien atteint donc un point stationnaire lorsque :

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{\partial\nu_k} = 0 \Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}M_k(x_i-\nu_k) = 0$$

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{\partial M_k} = 0 \Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}(x_i-\nu_k)(x_i-\nu_k)^T = \sum_{k=1}^{K}\lambda_k\, det\,(M_k)M_k^{-1}$$

$$\frac{\partial L(\nu_k,M_k,\lambda_k)}{\partial\lambda_k} = 0 \Leftrightarrow \sum_{k=1}^{K} det\,M_k = \sum_{k=1}^{K}\rho_k$$

Questions :

10.

Le produit matriciel étant distribuable :

$$\sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}M_k(x_i - \nu_k) = 0 \Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n}(z_{ik}M_k x_i - z_{ik}M_k\nu_k) = 0 \Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n}(z_{ik}M_k x_i) - \sum_{k=1}^{K}\sum_{i=1}^{n}(z_{ik}M_k\nu_k) = 0$$

$$\Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}M_k x_i = \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}M_k\nu_k$$

Pour une classe k fixé, la mise à jour ce fait tel que :

$$\sum_{i=1}^{n} z_{ik}M_k x_i = \sum_{i=1}^{n} z_{ik}M_k\nu_k$$

Comme $z_{ik} = \begin{cases} 1 \ si \ x_i \in P_k \\ 0 \ s \quad inon \end{cases}$, $soit \ \sum_{i=1}^{n} M_k z_{ik}\nu_k = 1M_k\nu_k + 0 + 1M_k\nu_k + 1M_k\nu_k + 1M_k\nu_k + 0 + 0 \ldots =$ $n_k M_k \nu_k$, avec $n_k = card(P_k)$

$$\sum_{i=1}^{n} z_{ik}M_k x_i = \mathrm{n_k}M_k\nu_k$$

$$\frac{1}{n_k}\sum_{i=1}^{n} M_k z_{ik}x_i = M_k\nu_k$$

$$M_k{}^{-1}\frac{1}{n_k}\sum_{i=1}^{n} M_k z_{ik}x_i = \nu_k$$

Comme $M_k{}^{-1}\sum_{i=1}^{n} M_k z_{ik}x_i = M_k{}^{-1}1M_k x_1 + 0 + M_k{}^{-1}1M_k x_3 + 1M_k x_4 + M_k{}^{-1}1M_k x_5 + 0 = \sum_{i=1}^{n} z_{ik}x_{\mathrm{i}}$

$$\frac{1}{n_k}\sum_{i=1}^{n} z_{ik}x_i = \nu_k$$

On a la valeur optimale de $\nu_k = \bar{x}_k$ qui est le centre de gravité de la classe k

11.

$$\frac{\partial L(\nu_k, M_k, \lambda_k)}{\partial M_k} = 0 \Leftrightarrow \sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}(x_i - \nu_k)(x_i - \nu_k)^T = \sum_{k=1}^{K} \lambda_k \, det\,(M_k)M_k^{-1}$$

$$\sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}(x_i - \nu_k)(x_i - \nu_k)^T = \sum_{k=1}^{K} \lambda_k \, det\,(M_k)M_k^{-1}$$

En combinant avec la troisième équation $\frac{\partial L(\nu_k, M_k, \lambda_k)}{\lambda_k} = 0 \Leftrightarrow \sum_{k=1}^{K} det\, M_k = \sum_{k=1}^{K} \rho_k$

$$\sum_{k=1}^{K}\sum_{i=1}^{n} z_{ik}(x_i - \nu_k)(x_i - \nu_k)^T = \sum_{k=1}^{K} \lambda_k\rho_k \, M_k^{-1}$$

Et Pour une classe k fixé, la mise à jour ce fait tel que :

$$\sum_{i=1}^{n} z_{ik}(x_i - \nu_k)(x_i - \nu_k)^T = \lambda_k\rho_k M_k^{-1}$$

Soit :

$$M_k^{-1} = \frac{1}{\lambda_k\rho_k}\sum_{i=1}^{n} z_{ik}(x_i - \nu_k)(x_i - \nu_k)^T$$

Si on montre que $\frac{\mathrm{n_k}}{\lambda_k\rho_k} = (\rho_k \mathrm{detV_k})^{\frac{-1}{\mathrm{p}}}$ alors on a bien $M_k^{-1} = (\rho_k detV_k)^{-\frac{1}{p}} V_k$

On admet que $\frac{\partial^2 L(v_k,M_k,\lambda_k)}{\partial {v_k}^2}$, $\frac{\partial^2 L(v_k,M_k,\lambda_k)}{\partial {M_k}^2}$ et $\frac{\partial^2 L(v_k,M_k,\lambda_k)}{\partial {\lambda_k}^2}$ sont définies positif au point optimal

Le point minimal optimal de la fonction $J(\{v_k,M_k\}_{k=1,\dots,K})$ sous la contrainte $det\, M_k = \rho_k$ est atteint lorsque $M_k^{-1} = (\rho_k det V_k)^{-\frac{1}{p}} V_k$ et $v_k = \bar{x}_k$. Ce qui représente le minimum de la somme, pour chaque classe, des distances de Mahalanobis des individus à leur centre de gravité de leur classe respective. En choisissant ces valeurs pour notre algorithme on montre donc que c'est le choix optimal.