

Rapport d'analyse de données

Mutation de gènes chez les patients affectés par la maladie X

*HARTLEY Marc
HERVÉ Victor*

03/11/2019

Contents

1	Introduction	2
2	Description des individus	2
3	Description des gènes	4
4	Analyse sous R	6
4.1	Présentation du programme	6
4.2	Utilisation	6
4.3	Resultats	7
4.4	Comparaison avec l'existant	10
4.5	Limites	11
5	L'ACP appliquée à notre étude génétique	12
5.1	Conclusion de l'ACP	14
6	Analyse supplémentaire	15
7	Conclusion	20
8	Annexes	22
8.1	Bibliographie	22
8.2	Code complet	22

1 Introduction

Notre jeu de données présente 40 patients. Les 20 premiers sont en bonne santé alors que les autres sont malades. Les patients sont représentés en colonnes et les gènes en lignes.

Nous savons qu'il existe de nombreuses maladies dues à l'expression des gènes. On dénombre entre 6000 et 8000 maladies génétiques dans le monde. Leurs causes ainsi que leurs symptômes sont divers en variés. Ces maladies peuvent être héréditaires ou simplement dues à des mutations génétiques dans la vie adulte. Ainsi la modification d'un gène peut provoquer des protéines déficientes et le fonctionnement anormal de nombreuses cellules. Dans 80% des cas, une maladie génétique serait due à l'altération d'une séquence codante d'un gène. Le but de notre étude est de trouver les gènes ou les groupes responsables des maladies sur nos patients par une analyse en composantes principales (ACP ou PCA en anglais).

2 Description des individus

Sur nos quarante individus, nous remarquons que chaque individu a des gènes qui s'expriment plus que d'autres. Sur le graphique nous pouvons voir que ces gènes sont mis en avant par des valeurs atypiques qui se retrouvent aux extrémités des boîtes à moustaches.

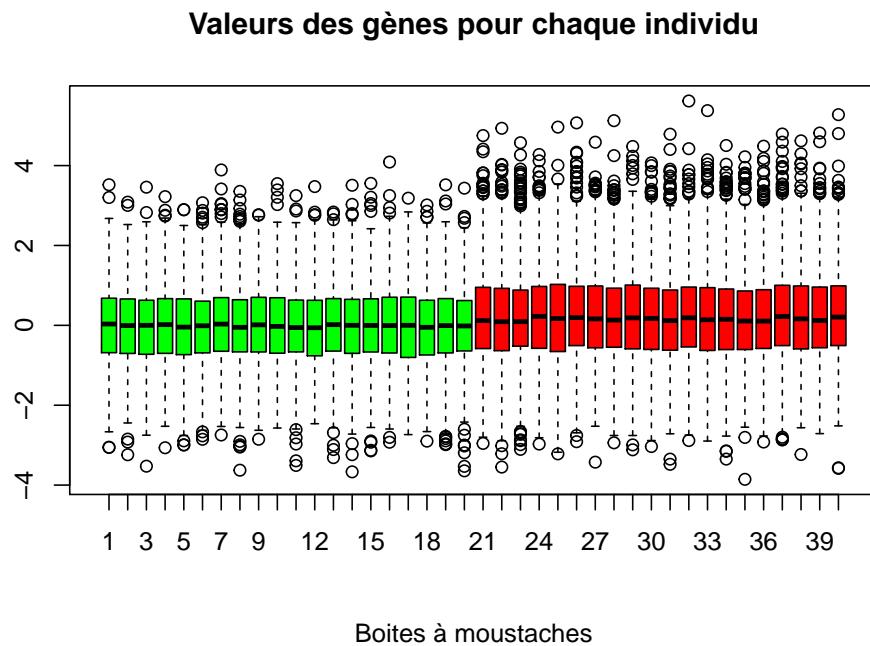
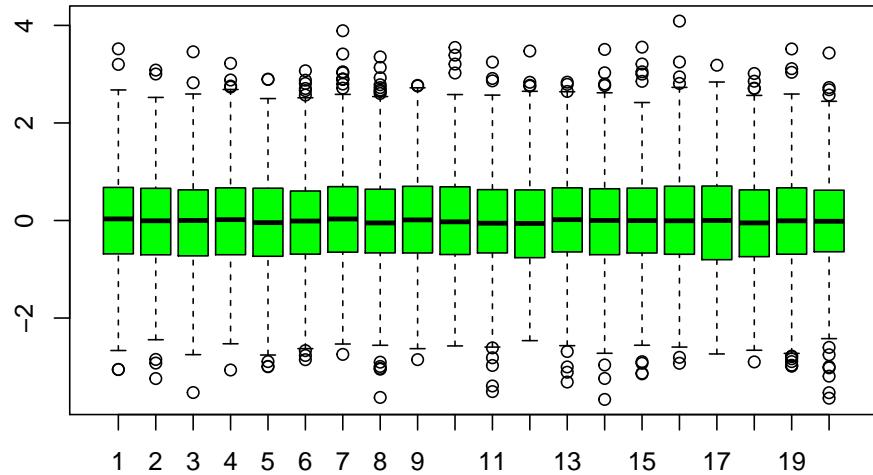


Figure 1: Représentation des gènes par individus

Cependant, l'expression des gènes chez les individus sains diffèrent de leurs expressions par rapport aux individus qui sont malades :

Valeurs des gènes pour chaque patient sain

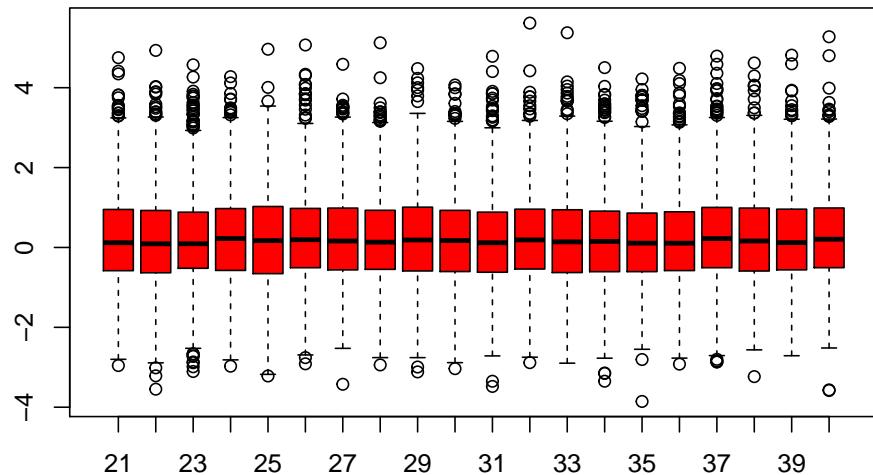


Boites à moustache

Figure 2: Representation des gènes pour les individus sains

En effet nous pouvons remarquer que nous avons environ autant de gènes qui sont sur-exprimés que de gènes sous-exprimés sur l'ensemble des individus sains. Alors que pour les individus malades nous avons beaucoup plus de gènes qui sont sur-exprimés comme le montre le graphique ci-dessous.

Valeurs des gènes pour chaque patient malade



Boites à moustache

Figure 3: Representation des gènes pour les individus malades

C'est pour cela que les boites à moustaches des individus malades ont une médiane tirée vers le haut par rapport aux individus sains.

Nous pouvons premièrement conjecturer que les deux groupes d'individus sont alors différenciés par certains gènes. Nous allons essayer de décrire ces gènes par la suite.

3 Description des gènes

Ici nous pouvons remarquer que la distribution de l'expression des gènes est étendue de -4 à 4.

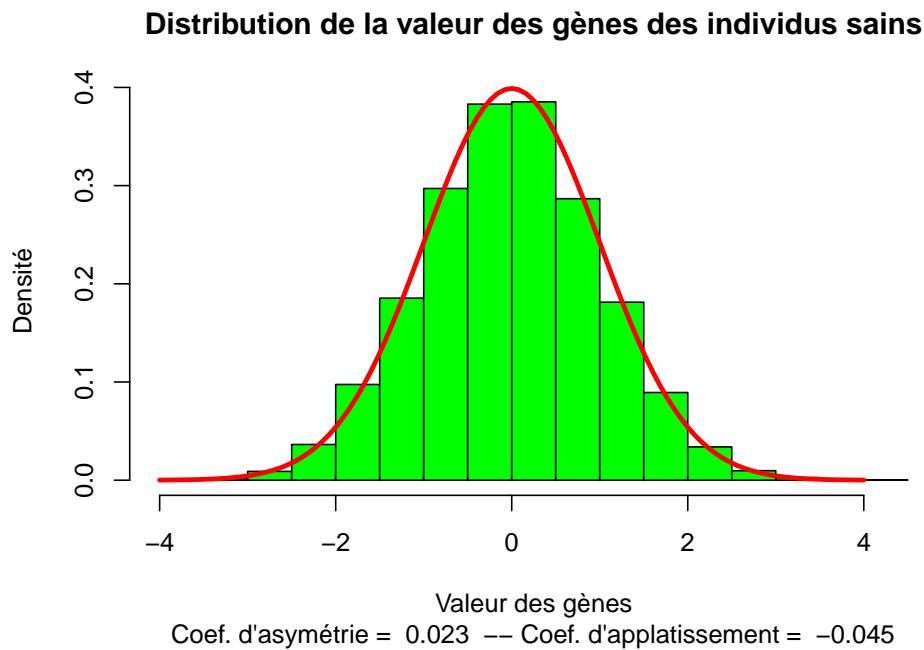


Figure 4: Distribution des gènes des individus sains comparée à la loi normale

Nous pouvons supposer à première vue que l'expression des gènes est distribuée selon une loi Gaussienne qui donne une bonne répartition de la distribution. En effet, le coefficient d'asymétrie et le coefficient d'aplatissement se rapprochent de 0.

Les valeurs d'expression des gènes sont correctement distribuées et ne permettent pas de distinguer un individu sain d'un individu malade. Ce n'est pas le cas pour les individus malades ci-dessous.

Distribution de la valeur des gènes des individus malades

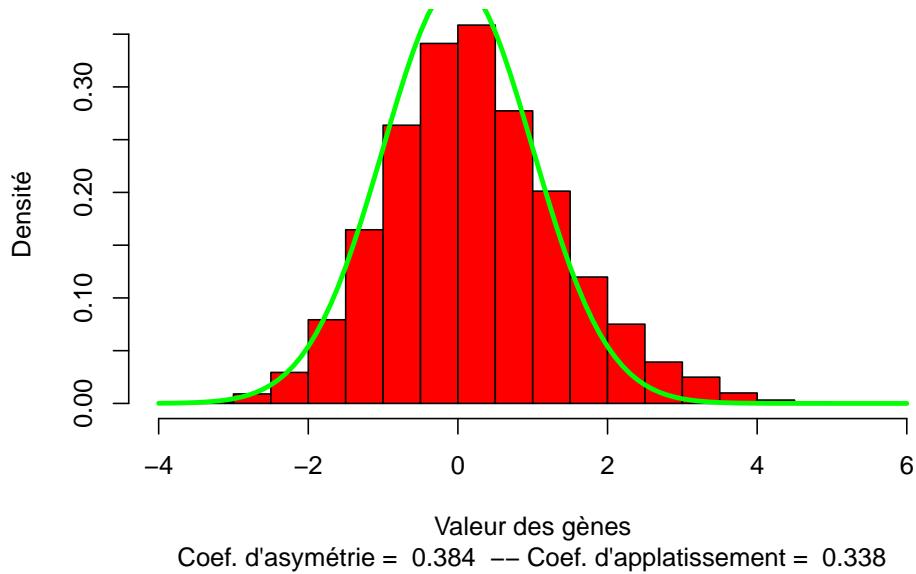


Figure 5: Distribution des gènes des individus malades comparée à la loi normale

En effet, nous remarquons que la distribution des gènes est disymétrique vers la droite lorsque l'individus est malade. Nous pouvons alors penser que certains gènes sont sur-exprimés pour un individu malade. Notre étude est alors de déterminer quels sont les gènes responsables des maladies. L'ACP nous permettra de répondre à cette question en projetant l'information dans un plan en dimension 2 et en mettant en valeur les groupes grâce aux corrélations.

4 Analyse sous R

Afin de réaliser notre analyse, nous avons eu besoin de réaliser un programme sous le langage R. Le code source du programme vous est mis à disposition en annexe. Nous allons voir ensemble comment le faire fonctionner afin de poursuivre l'analyse de nos patients.

4.1 Présentation du programme

Nous avons souhaité écrire ce programme afin qu'il soit le plus général possible, réutilisable pour des cas différents. Nous avons essayé de réaliser des fonctions simples à utiliser, complet et graphiquement clair. D'autres bibliothèques sont capable de réaliser des ACP (ade4, FactoMineR, ...), elles nous permettront alors de vérifier que nos résultats sont corrects et nous pouvons nous en inspirer pour restituer les résultats de nos fonctions.

4.2 Utilisation

Notre objectif est de rendre l'utilisation de notre programme le plus simple possible.
Pour utiliser le programme, il faut suivre deux étapes :

4.2.1 Initialisation :

La première étape est l'initialisation. C'est le calcul de l'ACP à proprement parler. Elle fonctionne très simplement, en voici sa signature :

```
nantes_pca <- function(table, center=T, scale=T, numberOfAxis=NULL,
additionalVariables=NULL, additionalIndividus=NULL)
```

On a donc un seul argument obligatoire, c'est la matrice des données brutes. Il est bien sûr possible de réaliser l'analyse en centrée-normée. Ensuite il est possible de choisir le nombre d'axes (ou composantes) à retenir pour l'analyse. Par défaut, le programme tentera de retrouver le nombre d'axes de manière automatique. Nous pouvons aussi ajouter qu'il est possible d'indiquer que certains individus et/ou certaines variables sont supplémentaires.

Au retour de cette fonction, on a une liste avec un maximum d'informations : coordonnées, contribution et qualité des variables/individus, éléments de la SVD (Singular Values Decomposition, ou Décomposition en Valeurs Singulières en français), soit les matrices U et V et les valeurs propres.

Grace à ce résultat, nous pouvons passer à la deuxième étape : la représentation.

4.2.2 Représentation :

Nous avons souhaité réunir tous les résultats que nous avons eu besoin lors des analyses étudiées en cours. Chaque représentation est réalisée par une fonction différente, mais elles sont toutes réunies dans cette fonction :

```
displayFullSummary <- function(pca, useOriginalTable = F, displayLimit = NULL,
colors = NULL, beLike = NULL, numberOfClass = 2,
variableType = "variable")
```

Encore une fois, nous avons misé sur le simple. `pca` est le résultat de la première fonction, c'est encore une fois le seul argument obligatoire.

Ensuite nous avons quelques moyens de personnaliser notre analyse : - `useOriginalTable` nous propose de réaliser une analyse univariée sur les données avant qu'elles soient transformées (centrées-réduites) ou non.

- `displayLimit` est nous permet de réduire la taille de l'affichage lors de la présentation des coordonnées/contributions/qualités des variables et des individus.

- `colors` colore le graphique des individus. Elle peut être utilisée afin de déterminer le nombre de groupes lors de la classification (ex : si on colore les individus en rouge, bleu et noir, cela sera considéré comme 3 groupes). Il est aussi possible de lui donner la valeur "auto" pour que les couleurs se répartissent en fonction du nombre de classes définies par l'argument `numberOfClasses`.

- `beLike` adapte les représentations pour ressembler à d'autres bibliothèques. La représentation par défaut ressemble à celle d'ADE4, mais en passant "FactoMineR" en argument, nous réorientons les graphes pour être identiques aux leurs.

- `numberOfClasses` définit le nombre de groupes pour la classification.

- `variableType` modifie le nom des graphiques "globaux".

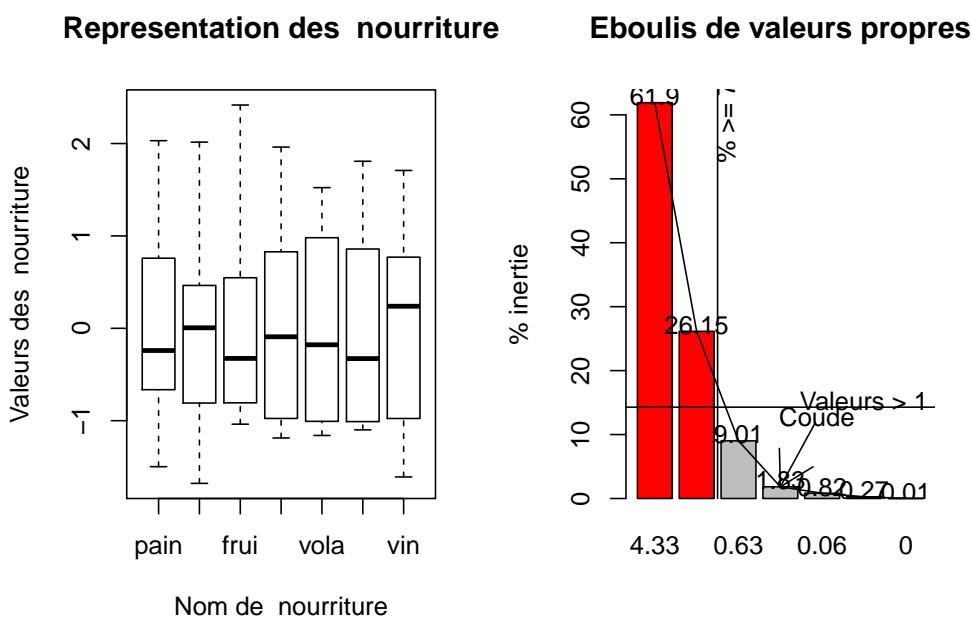
Il suffit de lancer cette fonction pour voir apparaître tous les résultats nécessaires à une bonne analyse en composantes principales.

4.3 Resultats

Voyons à quoi peut ressembler cette analyse avec un exemple : CSP.

Lançons l'initialisation :

Voyons à quoi peut ressembler les représentations :

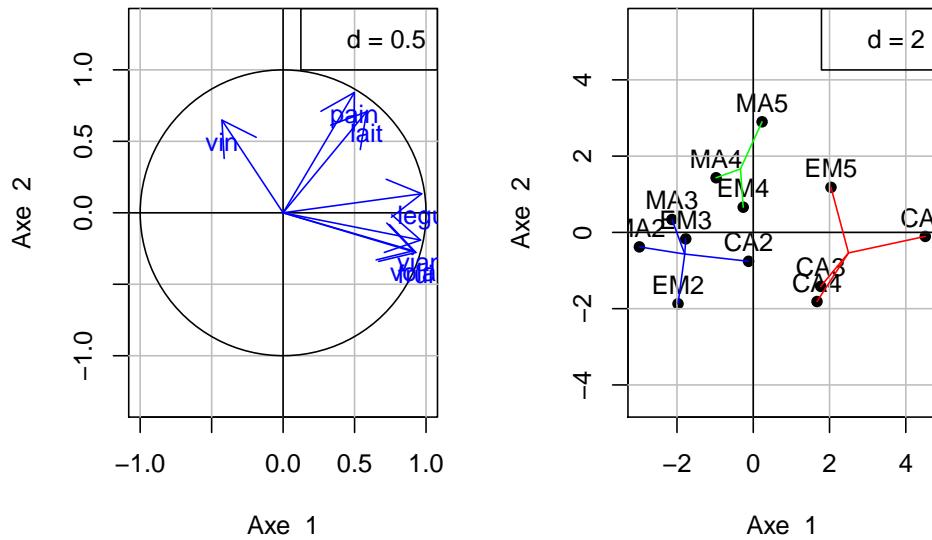


	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
pain	1.0000000	0.5931102	0.1961388	0.3212691	0.2480083	0.855574797
legu	0.5931102	1.0000000	0.8562502	0.8810807	0.8267774	0.662798738
frui	0.1961388	0.8562502	1.0000000	0.9594767	0.9255419	0.332188558
vian	0.3212691	0.8810807	0.9594767	1.0000000	0.9817853	0.374590641

```

vola 0.2480083 0.8267774 0.9255419 0.9817853 1.0000000 0.232891845
lait 0.8555748 0.6627987 0.3321886 0.3745906 0.2328918 1.0000000000
vin 0.3037612 -0.3564682 -0.4862806 -0.4372353 -0.4001570 0.006879604
[,7]
pain 0.303761226
legu -0.356468250
frui -0.486280578
vian -0.437235281
vola -0.400157002
lait 0.006879604
vin 1.0000000000

```



```
[1] "Coordonnees des individus"
```

	Axe 1	Axe 2
MA2	-2.99	-0.38
EM2	-1.97	-1.87
CA2	-0.12	-0.76
MA3	-2.13	0.34
EM3	-1.77	-0.17
CA3	1.77	-1.42
MA4	-0.97	1.43
EM4	-0.26	0.66
CA4	1.67	-1.81
MA5	0.23	2.90
EM5	2.04	1.18
CA5	4.51	-0.11

```
[1] "[CTR en %]"
```

	Axe 1	Axe 2
MA2	205.74	7.88
EM2	89.86	191.45
CA2	0.35	31.37

```

MA3 104.85   6.23
EM3  72.27   1.58
CA3  72.27 109.41
MA4  21.91 112.15
EM4   1.60  23.84
CA4  64.49 179.63
MA5   1.23 459.58
EM5  95.84  76.27
CA5 469.60   0.60
[1] "[QLT en %]"
      Axe 1 Axe 2
1  98.41  1.59
2  52.63 47.37
3  2.57 97.43
4  97.55  2.45
5  99.09  0.91
6  60.99 39.01
7  31.62 68.38
8  13.72 86.28
9  45.95 54.05
10 0.63 99.37
11 74.84 25.16
12 99.95  0.05
[1] "Individus contribuant le plus a l'axe 1 (positif)"
      CA5     EM5     CA3     CA4
469.60 95.84 72.27 64.49
[1] "Individus contribuant le plus a l'axe 1 (negatif)"
      MA2     MA3     EM2     EM3     MA4
205.74 104.85 89.86 72.27 21.91
[1] "Individus contribuant le plus a l'axe 2 (positif)"
      MA5     MA4     EM5     EM4
459.58 112.15 76.27 23.84
[1] "Individus contribuant le plus a l'axe 2 (negatif)"
      EM2     CA4     CA3     CA2
191.45 179.63 109.41 31.37
[1] "Coordonnees des Variables"
      Axe 1 Axe 2
pain  0.50  0.84
legu  0.97  0.13
frui  0.93 -0.28
vian  0.96 -0.19
vola  0.91 -0.27
lait  0.58  0.71
vin   -0.43  0.65
[1] "[CTR en %]"
      Axe 1 Axe 2
pain  5.74 38.70
legu 21.70  0.97
frui 19.92  4.22
vian 21.36  1.99
vola 19.16  3.86
lait  7.88 27.30
vin   4.23 22.95
[1] "[QLT en %]"

```

```

Axe 1 Axe 2
pain 24.87 70.83
legu 94.04 1.77
frui 86.33 7.72
vian 92.56 3.65
vola 83.04 7.07
lait 34.15 49.97
vin 18.34 42.01
[1] "Variables contribuant le plus a l'axe 2 (positif)"
  pain   lait   vin
38.70 27.30 22.95
[1] "Variables contribuant le plus a l'axe 2 (negatif)"
named numeric(0)

```

On a donc, par ordre chronologique :

- Un graphique “global” qui représente toutes les variables sous forme de boîte à moustache. (fonction `displayGlobal()`)
- Un éboulis de valeurs propres, utilisé pour définir le nombre de composantes à sélectionner. (fonction `displayScreePlot()`)
- Une matrice de corrélation des variables. Il est intéressant de savoir que lorsqu'il y a de nombreuses variables, la matrice est représentée graphiquement, comme nous le verrons dans ce rapport lorsque nous analysons 1000 variables. (fonction `displayCorrelation()`)
- Le cercle de corrélation afin de voir graphiquement la contribution des variables sur les composantes principales. (fonction `displayVariables()`)
- Le graphique des individus qui représente la projection de chaque individus sur les composantes.
- (fonction `displayIndividus()`)
- Le détail des coordonnées, contributions et qualité de représentation des variables et des individus selon chaque axes. On indique les plus grosses contributions sur chaque axe. (fonctions `displayInertieIndividus()` et `displayInertieVariables()`)

4.4 Comparaison avec l'existant

Il existe quelques autres bibliothèques permettant de réaliser des ACP. Nous avons ici vérifié nos résultats avec les bibliothèques ADE4 et FactoMineR. Nous avons essayé de comparer les valeurs retrouvées avec celles des autres bibliothèques, et ce par ces fonctions :

```

comparaisonAvecADE4(table, center = T, scale = T, ind.sup = NULL, var.sup = NULL,
                      numberofAxis = 5)
comparaisonAvecFactoMineR(table, center = T, scale = T, ind.sup = NULL,
                           var.sup = NULL, numberofAxis = 5)

```

Une série de tests des différentes valeurs nous permet de savoir si tout est identique ou sinon les valeurs qui diffèrent sont affichées à l'écran. Par chance, nous n'avons jusque là aucune différence avec ces deux bibliothèques :

```

[1] "Comparaison avec ADE4 :"
[1] "-----"
[1] "Valeurs propres :"
[1] "Dimensions identiques : 7 x 1"
[1] "Pas de difference"
[1] "Coordonnees des individus :"
[1] "Dimensions identiques : 12 x 5"

```

```

[1] "Pas de difference"
[1] "Coordonnees des variables :"
[1] "Dimensions identiques : 7 x 5"
[1] "Pas de difference"
[1] TRUE
[1] "Tout est identique! Bravo."

[1] TRUE

[1] "Comparaison avec FactoMineR :"
[1] "-----"
[1] "Valeurs propres :"
[1] "Dimensions identiques : 7 x 1"
[1] "Pas de difference"
[1] "Matrice U :"
[1] "Dimensions identiques : 12 x 5"
[1] "Pas de difference"
[1] "Matrice V :"
[1] "Dimensions identiques : 7 x 5"
[1] "Pas de difference"
[1] "Valeurs singulieres :"
[1] "Dimensions identiques : 7 x 1"
[1] "Pas de difference"
[1] "Coordonnees des individus :"
[1] "Dimensions identiques : 12 x 5"
[1] "Pas de difference"
[1] "Coordonnees des variables :"
[1] "Dimensions identiques : 7 x 5"
[1] "Pas de difference"
[1] TRUE
[1] "Tout est identique! Bravo."

[1] TRUE

```

4.5 Limites

Les limites actuelles de ces fonctions sont :

- mauvaise gestion des valeurs NA - pas encore de gestion des colonnes qualitatives

Toutes ces limites ont comme futur d'être corrigées, mais au jour de notre rendu de rapport, elles sont présentes. Pour la suite de cette analyse, cela ne nous contraindra pas.

5 L'ACP appliquée à notre étude génétique

Nous avons commencé à faire une ACP sur toutes nos variables en même temps. Cependant, l'inertie de nos points se retrouvent au centre de notre cercle de corrélation et nos axes factoriels ne représentent pas bien la réalité de notre étude.

En effet, l'inertie de la première valeur propre synthétise 8,11% de l'information. La deuxième valeur propre est très inférieure à la première (3,45%). Selon la règle du coude, seul le premier axe factoriel est choisi. De plus ce choix est précisé par l'étude en elle-même où l'on recherche qu'un seul axe mettant en valeur les individus malades

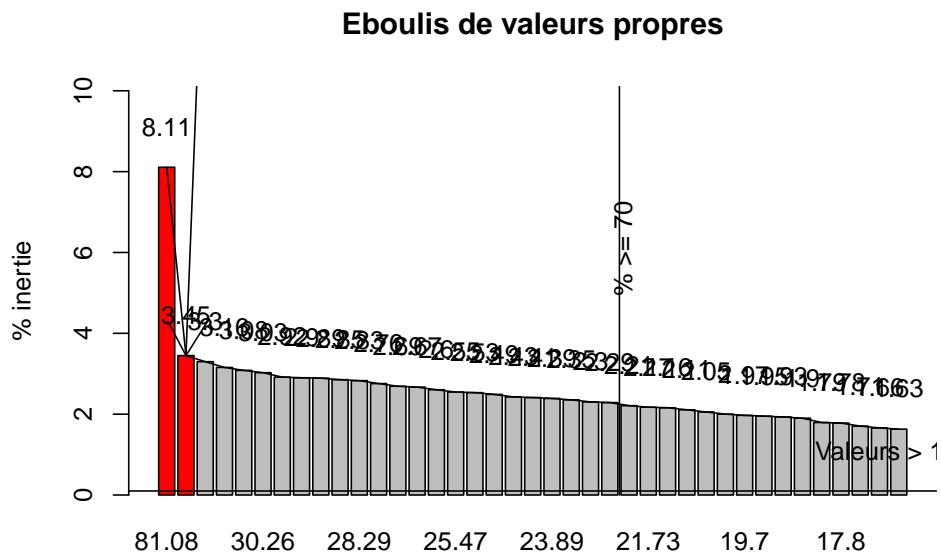


Figure 6: Ébouli de valeurs propres issues de notre fonction R

Dans un soucis d'affichage dans nos graphiques, nous utiliserons 2 axes, bien que le second ne nous semble pas expliquer quoi que ce soit.

Après avoir utilisé l'ACP, on remarque que les individus sont divisés en 2 groupes. On retrouve les sains à gauche de l'axe 1 et les individus malades à droite (figure 7). De plus, sur le cercle de corrélation (figure 8), on peut voir un groupe de gènes qui se démarquent vers la droite avec un cosinus supérieur à 0.5. On pense que c'est l'expression de ces gènes qui déterminent si un individu est malade ou sain. On peut de plus remarquer un gros groupe de variables presque totalement corrélées entre elles, représenté par le carré rouge sur le matrice de corrélation "graphique" (figure 9).

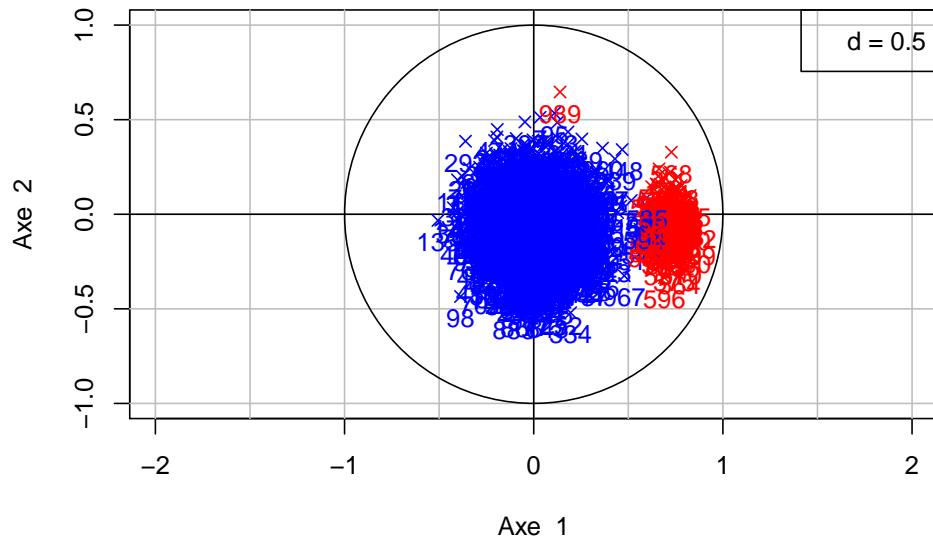


Figure 7: Variables représentées dans un cercle de corrélation

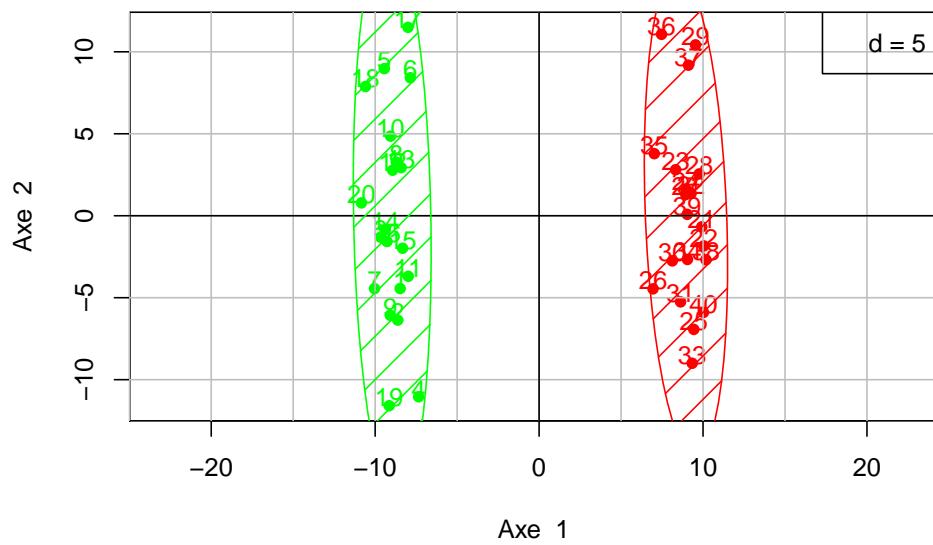


Figure 8: Projection des individus sur les 2 axes principaux, avec une classification K-means

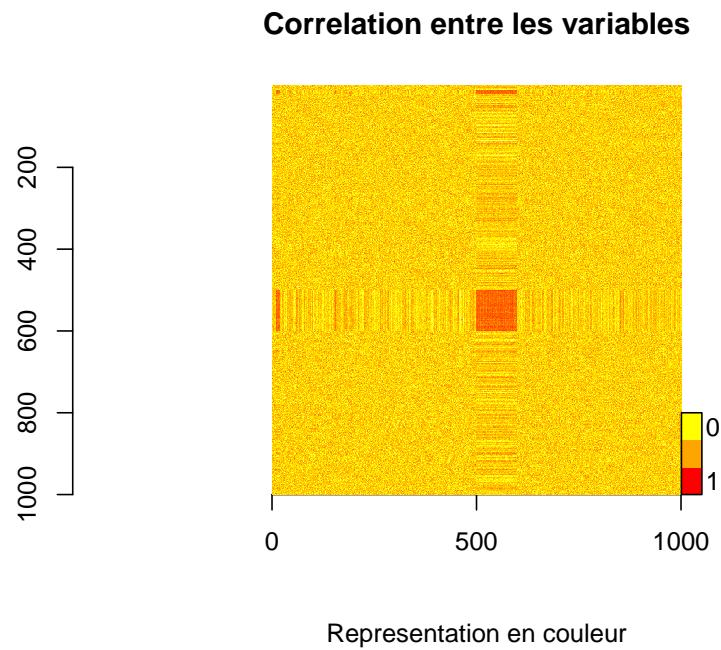


Figure 9: Rendu graphique de la matrice de la corrélation. Chaque pixel représente une valeur allant de 0 (jaune) à 1 (rouge)

5.1 Conclusion de l'ACP

On a bien mis en évidence sur notre cercle de corrélation un groupe de gènes (en rouge) dont l'expression représenterait l'apparition de la maladie. Néanmoins, l'axe 1 projete trop peu d'informations (inertie = 8,11%) donc nous ne pouvons pas savoir si l'expression de ces gènes détermine si un individu est sain ou malade.

6 Analyse supplémentaire

Sur notre première ACP, l'inertie de l'axe 1 est peu importante (8%) et on retrouve beaucoup de gènes vers le centre de l'ACP. Ces gènes ne résument donc pas l'information disponible; bien qu'on puisse penser que ces gènes déterminent si un individu est malade ou non, l'inertie de l'axe 1 est trop faible pour que cela soit une certitude. L'ACP contient trop de variables inutiles.

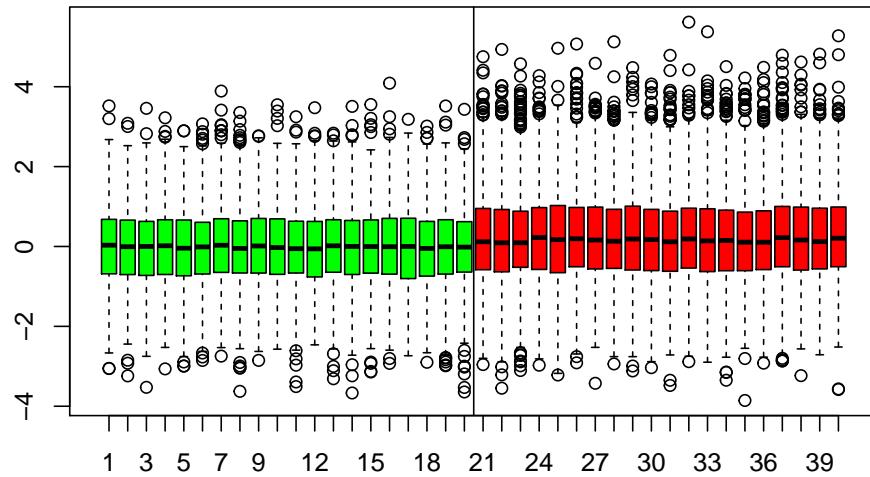


Figure 10: Représentation des gènes par individu, séparée entre individus sains (gauche) et malades (droite)

On s'interesse alors aux valeurs atypiques de ce graphique (c'est à dire les valeurs en dehors de la boîte à moustaches). Les patients malades ont beaucoup plus de gènes qui sont sur-exprimés par rapport aux individus sains. Nous retrouvons donc ce résultat :

```
[1] "Nous considérons 112 variables 'atypiques'"
```

On a donc réalisé une ACP en ne considérant seulement ces variables-ci.
Voici le résultat ci-dessous.

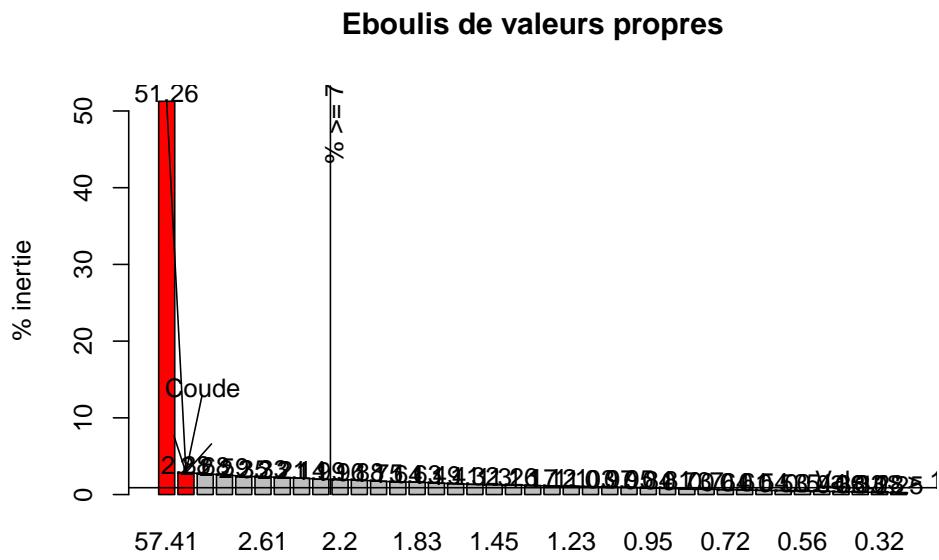


Figure 11: Ébouli de valeurs propres limitées aux gènes dont les valeurs sont 'atypiques'

On voit que dans l'ébouli de valeurs propres, la deuxième valeur propre est beaucoup moins significative que la première.

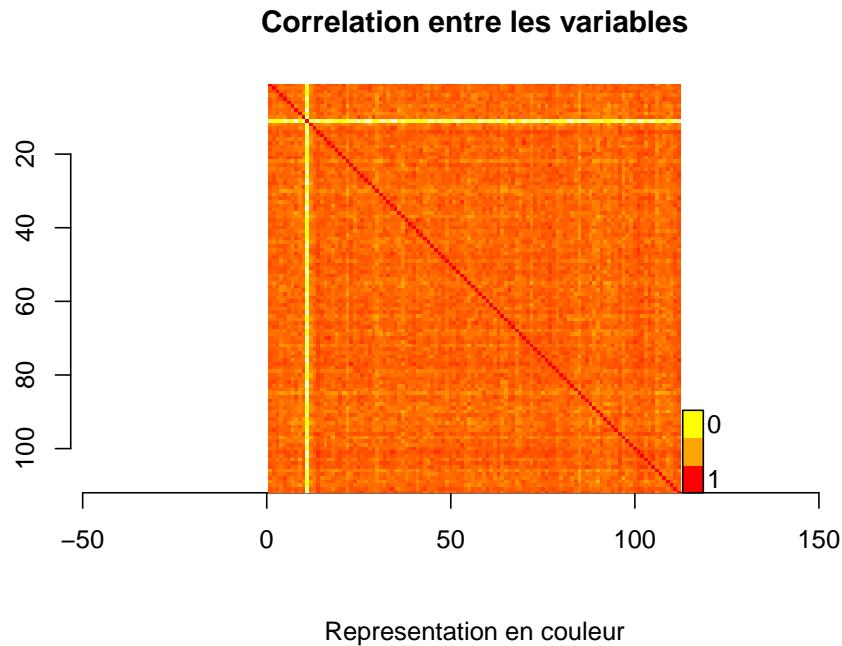


Figure 12: Matrice de corrélation. Nous remarquons la forte corrélation de presque toutes les valeurs

De plus toutes ces variables sont fortement corrélées entre elles. Il est aussi possible de remarquer cela avec

le cercle des corrélations car on peut voir que toutes les flèches sont rapprochées sur l'axe 1.

Les hypothèses posées par rapport à la première étude sont vérifiées. En effet, seul le premier axe est choisi par la règle du coude et par notre étude. De plus, il représente 51% de l'information disponible et la projection des individus malades ci-dessous nous permet de vérifier que la corrélation et la sur-expression des gènes sur notre ACP désigne bien les individus malades.

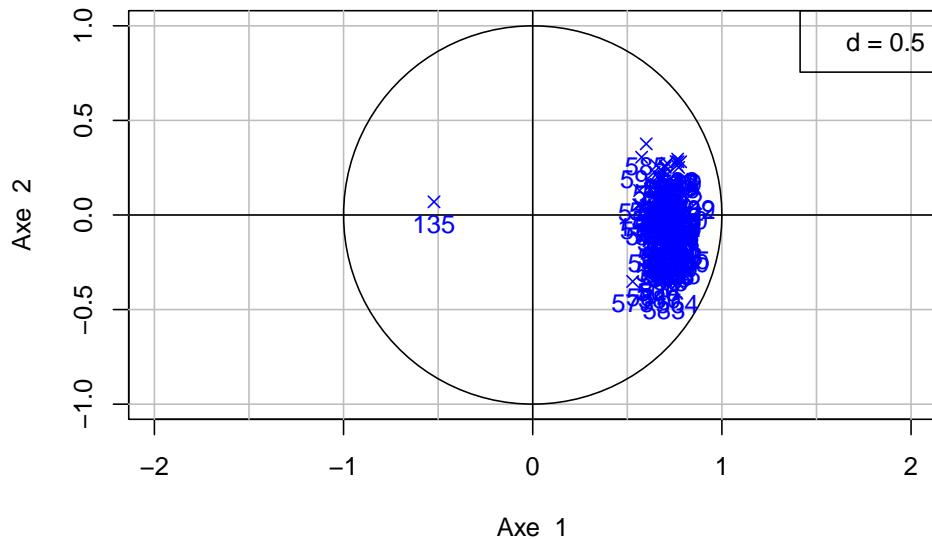


Figure 13: Représentation des gènes sur le cercle des corrélations

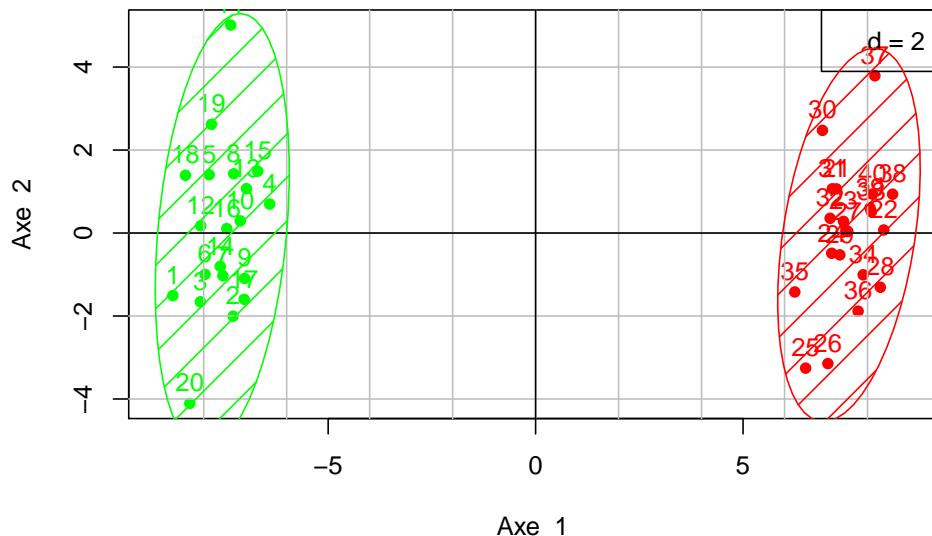


Figure 14: Projection des individus sur les 2 axes principaux de la PCA des valeurs atypiques, avec classification K-means

En appliquant une fonction de classification non-supervisée par “k-means”, on peut bien voir que les deux groupes se distinguent.

En moyenne, on a dans ces groupes une variance intra-groupe de 8,5 et une variance inter-groupe de 14.

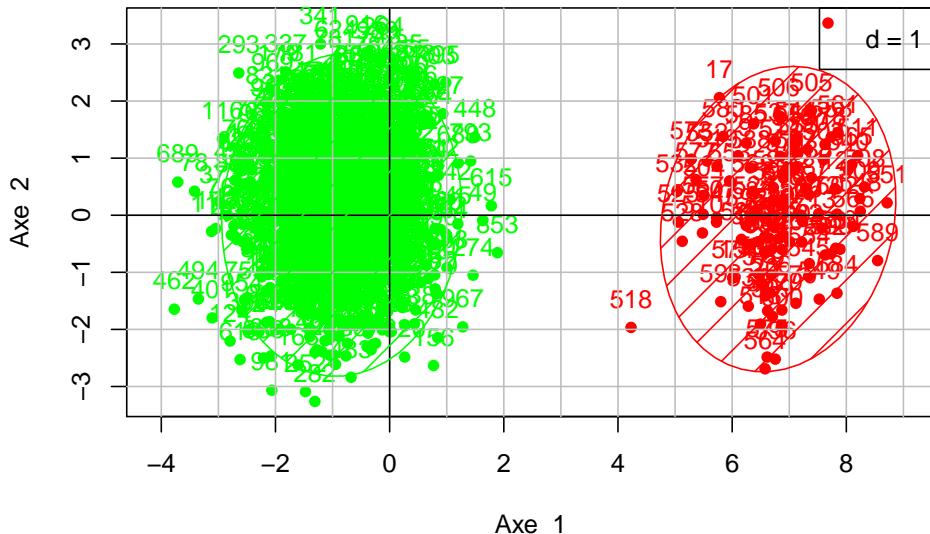


Figure 15: Représentation des gènes, avec classification K-means. On voit bien une séparation entre gènes neutres et gènes liés à la maladie

D'après la figure 15 représentant une classification type K-means sur les gènes, on voit une nette séparation entre les deux groupes avec le barycentre des gènes liés à la maladie sur la droite de l'axe 1 contre le barycentre des gènes neutres au centre du graphique. La variance intra-groupe est bien plus faible que la variance inter-groupe, l'algorithme du K-means a bien retrouvé nos deux groupes de gènes en différenciant les gènes dont l'expression détermine la présence de la maladie ou au contraire, les gènes dont l'expression ne permettent pas de déterminer qu'un individu est malade ou non (gènes neutres).

Pour plus de précision sur la corrélation entre les variables et l'axe 1, on a représenté les contributions des variables ci-dessous.

```
[1] "Coordonnées des Variables"
 1   2
-0.05 -0.02
[1] "[CTR en %]"
1 2
0 0
[1] "[QLT en %]"
 1   2
0.22 0.04
[1] "Variables contribuant le plus à l'axe 1 (positif)"
 502 589 565 590 600 551 593 538 584 509 511 570 599 535 528
0.90 0.89 0.84 0.84 0.84 0.77 0.77 0.76 0.76 0.75 0.75 0.75 0.74 0.74 0.74 0.73
 592 566 564 540 508 554 11 548 536 563 588 555 520 522 558
0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.72 0.72 0.72 0.72 0.71 0.71 0.71 0.70
 574 559 530 569 514 15 561 513 16 575 505 503 549 582 539
0.70 0.70 0.70 0.70 0.70 0.70 0.69 0.69 0.69 0.68 0.68 0.68 0.68 0.67 0.66 0.66
 13 526 515 516 562 568 547 527 12 571 534 579 523 541 507
0.66 0.66 0.66 0.66 0.65 0.65 0.65 0.65 0.64 0.64 0.64 0.64 0.64 0.63 0.63
```

```

550   18  504  545  529  586  533  544  521  531  595  512  596  517  546
0.63  0.62  0.62  0.62  0.60  0.60  0.60  0.60  0.60  0.60  0.59  0.59  0.59  0.59
 542  587  583  598  591  557  501  580  524  572  20  553  17  532  597
0.59  0.58  0.58  0.58  0.57  0.57  0.56  0.55  0.55  0.55  0.55  0.54  0.54  0.54
 519  567  556  506  577  537  576  560  14   19   552  525  543  581  585
0.54  0.53  0.53  0.53  0.52  0.52  0.52  0.51  0.51  0.50  0.48  0.47  0.47  0.46
 578  594  510  156  518  573  967  448  615  853  939  243  457  331  715
0.44  0.42  0.41  0.39  0.38  0.33  0.28  0.27  0.26  0.24  0.23  0.20  0.19  0.19
 900  860  752  802  61   686  326  346  142  49   680  214  634  852  675
0.17  0.16  0.16  0.16  0.15  0.14  0.14  0.14  0.14  0.14  0.14  0.13  0.13  0.13
 684  665  54   258  651  460  830  265  896  280  638  128  277  620  424
0.13  0.13  0.13  0.12  0.12  0.12  0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11
 813  737
0.10  0.10
[1] "Variables contribuant le plus a l'axe 1 (negatif)"
 135  172  393  116  98   462  751  389  77   293  337  768  104  263  373
0.32  0.22  0.20  0.20  0.18  0.18  0.18  0.18  0.18  0.16  0.15  0.15  0.14  0.14
 871  469  622  689  709  130  10   957  611  102  806  67   459  224  444
0.13  0.13  0.13  0.13  0.13  0.12  0.12  0.12  0.12  0.11  0.11  0.11  0.11  0.11
 678  375  427  36   251  159  391
0.11  0.11  0.11  0.11  0.10  0.10  0.10

```

On peut donc en conclure que ce sont les gènes ayant les plus grosses contributions à l'axe 1 qui sont responsables (ou liées) à la maladie :

Table 1: Gènes responsables à la maladie

11	128	448	511	524	537	550	563	576	589
12	142	457	512	525	538	551	564	577	590
13	156	460	513	526	539	552	565	578	591
14	214	501	514	527	540	553	566	579	592
15	243	502	515	528	541	554	567	580	593
16	258	503	516	529	542	555	568	581	594
17	265	504	517	530	543	556	569	582	595
18	277	505	518	531	544	557	570	583	596
19	280	506	519	532	545	558	571	584	597
20	326	507	520	533	546	559	572	585	598
49	331	508	521	534	547	560	573	586	599
54	346	509	522	535	548	561	574	587	600
61	424	510	523	536	549	562	575	588	615

Table 2: Gènes ‘inversionement’ responsables à la maladie

10	67	98	104	130	159	224	263	337	375
36	77	102	116	135	172	251	293	373	389

7 Conclusion

Aussi bien à l'aide de notre ACP que notre classification K-means, on a su trouver les gènes responsables des maladies dans notre échantillon de 40 individus. On peut très fortement supposer que les gènes liés à

l'apparition de la maladie chez les patients sont numérotés de 11 à 30 et de 501 à 600, et quelques autres gènes supplémentaires. On trouve aussi quelques gènes qui semblent plus forts chez les patients sains, mais leur importance et leur effectif sont bien inférieurs aux précédents. Il peut néanmoins être utile de s'y intéresser car ils peuvent peut-être être une solution de soins face à la maladie. Les méthodes utilisées (aussi bien en prenant en compte la décomposition de la variance et la corrélation entre les individus) ont toutes été efficaces pour le problème posé.

8 Annexes

8.1 Bibliographie

http://geai.univ-brest.fr/ https://fr.wikipedia.org/wiki/Kurtosis https://bookdown.org/ https://www.rdocumentation.org/ https://genetique-medicale.fr/

8.2 Code complet

```
library(ade4)
library(FactoMineR)
library(gplots)
library(crayon)
library(stringr)
library(car)
library(varhandle)
library(stats)
library(plotrix)

epsilonPower = 8 # Les nombres inferieurs a 10^(-8) seront consideres comme nuls
roundingValue = 2 # Nombre de chiffres apres la virgule a l'affichage

# Calcul de l'inter-quartile
interQuartile <- function(valeurs) {
  return(quantile(valeurs)[4] - quantile(valeurs)[2])
}

# Quelles valeurs sont extremes parmis la colonne
extremValues <- function(column, coef = 1.5) {
  return(column < quantile(column)[2] - coef*interQuartile(column) |
         quantile(column)[4] + coef*interQuartile(column) < column)
}

# Calcule le coefficient d'asymetrie
skewness <- function(values) {
  center <- mean(values)
  first <- sum((values - center)**3)/length(values)
  second <- (sum((values - center)**2) / length(values))**(3/2)
  return(first/second)
}

# Calcule le coefficient d'aplatissement
# InspirÃ© par la bibliothÃ¨que "moments"
kurtosis <- function(values) {

  if (is.matrix(values))
    return(apply(values, 2, kurtosis))
  else if (is.vector(values)) {
    n <- length(values)
    return(n * (sum((values - mean(values))**4))/(sum((values - mean(values))**2)**2) - 3)
  }
  else if (is.data.frame(values)) {
    return(sapply(values, kurtosis))
  }
  else {
```

```

        return(kurtosis(as.vector(values)))
    }
}

# Produit Vectoriel
produitVect <- function(X, Y) {
  return(t(X) %*% Y)
}

# Recupere l'extension du fichier donnee en parametre
getExtension <- function(file){
  ex <- strsplit(basename(file), split="\\".")[[1]]
  return(ex[-1])
}

# Fonction pour recuperer le contenu du fichier sous forme de matrice
recuperer_donnees_en_matrice <- function(fichier, entete=FALSE, transpose = F) {
  if(getExtension(fichier) == "csv") { # Si le fichier est un CSV, on utilise la fonction read.csv
    donnees <- read.csv(fichier, sep=",", header=entete, stringsAsFactors = F) # On recupere les donnees
  } else { # Sinon on utilise la fonction read.table, plus generique
    donnees <- read.table(fichier, header = entete, stringsAsFactors = F, dec=",")
  }
  # On transpose les donnees si demande
  if(transpose) donnees <- t(donnees)
  if(!entete) {
    colnames(donnees) <- 1:ncol(donnees)
    rownames(donnees) <- 1:nrow(donnees)
  }
  return(as.matrix(donnees)) # On les transforme en matrice
}

# Recupere la variance d'une colonne de valeurs
getVariance <- function(column) {
  sums <- sum((column - mean(column))**2)
  return(sums/length(column))
}

# Centrer et reduire un tableau de valeurs
centerAndScale <- function(values, center = T, scale = T) {
  if(center){
    values <- apply(values, 2, function(x){ return(x - mean(x))}) # Chaque colonne est centre avec sa moyenne
  }
  if(scale){
    values <- apply(values, 2, function(x){ return(x / sqrt(getVariance(x)))})
    # On divise chaque valeur avec la variance de sa colonne
  }
  return(values)
}

# Fonction pour passer les valeurs a l'intervalle [-1;1]
toPercents <- function(values) { # Normalise chaque valeur d'un vecteur
  return(values/sum(values))
}

# Tentative de recuperation du coude d'un eboulis de valeurs
# On cherche la premiere valeur qui marque un "coude", c'est a dire qui reduit l'écart avec la valeur precedente
# On cherche un écart entre i et i+1 "négligence fois" inférieur à l'écart entre i-1 et i

```

```

getCoude <- function(values, negligence = 4) {
  slope <- 0 # On commence avec un ecart minimal
  for(i in 1:(length(values) -1)){
    newSlope <- values[i]-values[i+1] # On a le nouvel ecart entre i et i+1
    if((newSlope*negligence) < slope) { # Si cet ecart est suffisamment faible,
      # on dit que la valeur a recuperer est i-1 (nombre d'axes principaux)
      return(i-1)
    }
    slope <- newSlope # Sinon on recommence
  }
  return(length(values))
}

# Fonction pour "estimer" le nombre de composantes principales de notre jeu de donnees
# On utilise 3 regles :
# - critere de Keiser (inertie superieure a I/p)
# - critere de pourcentage represente (cumul des valeurs propres doivent representer 70% de l'inertie totale)
# - regle du coude
getNbAxes <- function(data) {
  rule1 <- length(which(data$eig > 1))
  rule2 <- getCumSumForValue(toPercents(data$eig), 0.7)
  rule3 <- getCoude(data$eig)
  return(c(rule1, rule2, rule3)) # On retourne tous les resultats
}

# Fonction pour connaitre l'indice pour lequel la somme cumulee est superieure ou egale a une certaine valeur
getCumSumForValue <- function(values, wantedSum) {
  cumul <- 0
  for(i in 1:length(values)) {
    cumul <- cumul + values[i]
    if(isTRUE(cumul >= wantedSum)) {
      return(i)
    }
  }
  return(length(values))
}

# Fonction pour recuperer les valeurs propres superieures a 0
getEigenValues <- function(data) {
  eigenValues <- eigen(data$correlation)$values
  eigenValues[eigenValues < 0] <- 0
  return(eigenValues)
}

## Fonctions d'affichage
# Affichage de l'ebouli de valeurs
displayScreePlot <- function(data) {
  heights <- data$eig[data$eig > 10**(-epsilonPower)] # Les valeurs du barplot sont les plus grandes
  # valeurs propres superieures a 0
  percents <- toPercents(heights)*100
  br <- barplot(height = percents,
                width = 2,
                ylab = "% inertie",
                names.arg = round(heights, roundingValue),
                col = c(rep("red", data$numberOfAxis), # On colore en rouge les X composantes choisies
                       rep("gray", length(heights)-data$numberOfAxis)),

```

```

        main = "Eboulis de valeurs propres",
        ylim = c(0, percents[1]+2))
text(x = br,
      y = percents+1,
      labels = round(percents, roundingValue)) # On affiche le pourcentage represente par chaque compo

lines(x = br,
      y = percents) # On ajoute une courbe pour reperer le coude

# On cherche a afficher une droite horizontale pour voir les valeurs propres superieures a 1
heightFor1 <- 100 / sum(data$eig)
abline(h = heightFor1)
text(x = br[length(br)-1],
      y = heightFor1+1, labels="Valeurs > 1")

# On va afficher une droite verticale pour distinguer les composantes principales qui
# expliquent jusu'a 70% de l'inertie
 maxValueFor70 <- getCumSumForValue(percents, 70)
 widthFor70 <- (br[maxValueFor70] + br[maxValueFor70 +1]) / 2
 abline(v = widthFor70)
text(x = widthFor70 + 0.5,
      y = percents[1]-2,
      labels = "% >= 70",
      srt = 90)

# Affichage du coude avec une fleche
coudeIndex <- getCoude(heights)
x0 <- br[coudeIndex+1]
y0 <- percents[coudeIndex+1]
arrows(x0 = x0, y0 = y0, x1 = x0 + 2, y1 = y0 + 10, code=1)
text(x = x0 + 2.1,
      y = y0 + 11,
      labels = "Coude")
}

# Affichage des individus sur un plan 2D
# Il est possible de choisir le modele voulu : FactoMineR ou ADE4
displayIndividus <- function(data, axe1=1, axe2=2, colors=NULL, beLike=NULL) {
  coord <- data$ind$coord
  if(ncol(coord) == 1) axe2 = axe1
  if(!is.null(data$ind$sup)) {coord.sup <- data$ind$sup$coord}
  if(!is.null(beLike))
  {
    if(grepl(toupper(beLike), "ADE", fixed=T)) {
      coord[,axe2] = -coord[,axe2]
      if(!is.null(data$ind$sup)) {
        coord.sup[,axe1] = -coord.sup[,axe1]
      }
    } else if(grepl(toupper(beLike), "FACTOMINER", fixed=T)) {
      # Rien a faire
    }
  }
}
```

```

labelOffset <- 0.5 # Valeur arbitraire

# On affiche les individus selon les coordonnees calculees precedemment
plot(coord[,c(axe1, axe2)],
      xlim = c(min(coord[,axe1]), max(coord[,axe1])),
      ylim = c(min(coord[,axe2]), max(coord[,axe2])),
      xlab = paste("Axe ", axe1, collapse=""),
      ylab = paste("Axe ", axe2, collapse=""),
      col=colors,
      pch=16,
      asp = 1)

labels = 1:data$nbInd
if(length(rownames(data$table)) > 0) labels = rownames(data$table)
text(x = coord[,axe1],
      y = coord[,axe2]+labelOffset,
      labels = labels,
      col = colors)

# S'il y a des individus supplementaires, on les affiche aussi
if(!is.null(data$ind$sup)) {
  points(x = coord.sup[,axe1],
         y = coord.sup[,axe2],
         pch = 4)
  text(x = coord.sup[,axe1],
        y = coord.sup[,axe2] + labelOffset,
        labels = 1:data$ind$sup$nbSup,
        font = 3)
}
displayQuadrillage()
}

# Affichage du quadrillage
displayQuadrillage <- function(interval = NULL, color="gray") {
  # On reecupere les parametre du plot
  param <- par()
  maxX <- max(abs(c(param$xaxp[1], param$xaxp[2])))
  maxY <- max(abs(c(param$yaxp[1], param$yaxp[2])))
  if(is.null(interval))
  {
    interval <- min(abs(c((param$xaxp[1] - param$xaxp[2])/param$xaxp[3],
                           (param$yaxp[1] - param$yaxp[2])/param$yaxp[3])))
  }
  abline(h=seq(from=-maxY*2, to=maxY*2, by=interval),
         v=seq(from=-maxX*2, to=maxX*2, by=interval), col="gray")
  abline(h=0, v=0)
  legend(x="topright", legend = paste("d =", interval, collapse=""))
}

# Affichage de la matrice de correlation sous forme d'image
# pour les grandes matrices. Rouge = correlation, jaune = pas de correlation
displayCorrelationAsHeatmap <- function(correlation) {
  image(x = 1:nrow(correlation),

```

```

y = 1:ncol(correlation),
z = correlation, #t(apply(correlation, 2, rev)),
col=rev(heat.colors(100)),
#xaxt='n',
#yaxt='n',
ann=FALSE,
bty='n',
asp=1,
ylim = rev(c(1, ncol(correlation))),
xlim = c(1, ncol(correlation)))
title(main = "Correlation entre les variables", sub = "Representation en couleur")
param <- par()
x <- ncol(correlation) + 1
x2 <- x + ncol(correlation)/20
y2 <- ncol(correlation)
y <- y2 - ncol(correlation)/5
color.legend(xl = x,xr = x2,yt = y, yb = y2, c("1", "", "0"),
c("red", "orange", "yellow"), gradient="y", align = "rb")
}

# Affichage des variables dans le cercle des correlations
displayVariables <- function(data, axe1=1, axe2=2, beLike=NULL, useArrows=T, colorWeakValues = F) {
  coord <- data$var$coord
  if(ncol(coord) == 1) axe2 = axe1
  coord <- coord[,c(axe1, axe2)]
  if(!is.null(data$var$sup)) {

    coord.sup <- data$var$sup$coord[,c(axe1, axe2)]
  }
  if(!is.null(beLike))
  {
    if(grepl(toupper(beLike), "ADE", fixed=T)) {
      coord[,1] <- -coord[,1]
      if(!is.null(data$var$sup)) {
        coord.sup[,1] <- -coord.sup[,1]
      }
    } else if(grepl(toupper(beLike), "FACTOMINER", fixed=T)){
      # Rien à faire
    }
  }
  # On initialise le graphique
  plot(x = coord,
    xlim=c(-1,1),
    ylim=c(-1,1),
    type="n",
    xlab = paste("Axe ", axe1, collapse=""),
    ylab = paste("Axe ", axe2, collapse=""),
    asp=1)
  displayQuadrillage()

  # Affichage du cercle de correlation
  piSeq <- seq(from=0, to=2*pi, length=100)
  circleValuesX = cos(piSeq)
}

```

```

circleValuesY = sin(piSeq)
lines(x = circleValuesX, y = circleValuesY)
color = "blue"
if(colorWeakValues) {
  color = apply(coord, 1, FUN=function(row){
    if(max(row[c(axe1, axe2)]) < colorWeakValues) {
      return("blue")
    } else {
      return("red")
    }
  })
}

if(useArrows) {
  # On utilise des flèches pour représenter les variables
  arrows(x0=rep(0, n=data$nbVar),
         y0=rep(0, data$nbVar),
         x1=coord[,1],
         y1=coord[,2],
         col=color)
} else {
  points(x=coord[,1],
         y=coord[,2],
         col=color,
         pch = 4)
}

# Affichage des noms de variables
if(!is.null(names(data$table))) {
  label <- names(data$table)
} else if(length(rownames(data$var$coord)) > 0) {
  label <- rownames(data$var$coord)
} else {
  label <- 1:data$nbVar
}
text(x = coord[,1],
      y = coord[,2],
      labels = label,
      col = color,
      pos = 1)

# Affichage des variables supplémentaires
if(!is.null(data$var$sup)) {
  if(useArrows){
    arrows(x0=rep(0, n=data$var$sup$nbSup),
           y0=rep(0, data$var$sup$nbSup),
           x1=coord.sup[,1],
           y1=coord.sup[,2],
           col="green",
           lty = "dotted")
  } else {
    points(x=coord.sup[,1],
           y=coord.sup[,2],

```

```

        col="green",
        pch = 9)
    }
    if(length(rownames(data$var$sup$coord)) > 0) {
      label.sup <- rownames(data$var$sup$coord)
    } else {
      label.sup <- 1:data$var$sup$nbSup
    }
    text(x = coord.sup[,1],
          y = coord.sup[,2],
          labels = label.sup,
          col = "green",
          pos = 1)
  }
}

# Fonction SVD
SVD <- function(data) {
  X <- data$table
  row.w <- rep(1/nrow(X), nrow(X))
  col.w <- rep(1, ncol(X))
  ncp <- data$numberOfAxis
  row.w <- row.w / sum(row.w)
  # X <- t(t(X)*sqrt(col.w))*sqrt(row.w)
  if (ncol(X) < nrow(X)){
    mSVD <- svd(X, nu=ncp, nv=ncp)
    if (names(mSVD)[[1]]=="message"){
      mSVD <- svd(t(X), nu=ncp, nv=ncp)
      if (names(mSVD)[[1]]=="d"){
        aux <- mSVD$u
        mSVD$u <- mSVD$v
        mSVD$v <- aux
      } else{
        bb <- eigen(crossprod(X,X), symmetric=TRUE)
        mSVD <- vector(mode = "list", length = 3)
        mSVD$d[mSVD$d<0]=0
        mSVD$d <- sqrt(mSVD$d)
        mSVD$v <- bb$vec[,1:ncp]
        #           mSVD$u <- sweep(X*%mSVD$v, 2, mSVD$d[1:ncp], FUN="/")
        mSVD$u <- t(t(crossprod(t(X), mSVD$v))/mSVD$d[1:ncp])
      }
    }
    U <- mSVD$u
    V <- mSVD$v
    if (ncp >1){
      mult <- sign(as.vector(crossprod(rep(1,nrow(V)), as.matrix(V))))
      mult[mult==0] <- 1
      U <- t(t(U)*mult)
      V <- t(t(V)*mult)
    }
    U <- U/sqrt(row.w)
    V <- V/sqrt(col.w)
  }
}

```

```

else{
  mSVD <- svd(t(X), nu=ncp, nv=ncp)
  if (names(mSVD)[[1]]=="message"){
    mSVD <- svd(X, nu=ncp, nv=ncp)
    if (names(mSVD)[[1]]=="d"){
      aux <- mSVD$u
      mSVD$u <- mSVD$v
      mSVD$v <- aux
    } else{
      bb <- eigen(crossprod(t(X), t(X)), symmetric=TRUE)
      mSVD <- vector(mode = "list", length = 3)
      mSVD$d[mSVD$d<0]=0
      mSVD$d <- sqrt(mSVD$d)
      mSVD$v <- bb$vec[,1:ncp]
      mSVD$u <- t(t(crossprod(X, mSVD$v))/mSVD$d[1:ncp])
    }
  }
  U <- mSVD$v
  V <- mSVD$u
  mult <- sign(as.vector(crossprod(rep(1, nrow(V)), as.matrix(V))))
  mult[mult==0] <- 1
  V <- t(t(V)*mult)/sqrt(col.w)
  U <- t(t(U)*mult)/sqrt(row.w)
}
vs <- mSVD$d[1:min(ncol(X), nrow(X)-1)]
num <- which(vs[1:ncp]<1e-15)
if (length(num)==1){
  U[,num] <- U[,num, drop=FALSE]*vs[num]
  V[,num] <- V[,num, drop=FALSE]*vs[num]
}
if (length(num)>1){
  U[,num] <- t(t(U[,num])*vs[num])
  V[,num] <- t(t(V[,num])*vs[num])
}
res <- list(vs = vs, U = U, V = V)
return(res)
}

# Affichage du rapport d'ACP
displayFullSummary <- function(pca, colors = NULL, beLike = NULL,
                               displayLimit = 20, numberOfClasses = 2, useOriginalTable = F,
                               bestContributions = 30, variableType = "variables") {
  data <- pca
  # On retrouve les couleurs utilisees pour les classes
  classColors = colors
  colorToBlack <- F
  if(!is.null(classColors)) {
    colorToBlack <- T
    if(!grepl(toupper(classColors), "AUTO", fixed=T)) {
      classColors = levels(as.factor(classColors))
      colorToBlack <- F
    }
  }
}

```

```

}

if(colorToBlack) {
  colors = "black"
}

# Nombre de variables maximal a afficher par graphique
maxGenesPerPlot <- 50
usedTable = data$table
if(useOriginalTable) usedTable <- data$originalTable

ylim = c(min(usedTable), max(usedTable))

# Si toutes les variables tiennent dans un graphique
if(data$nbVar <= maxGenesPerPlot) {
  boxplot(usedTable,
    main = paste("Representation des ", variableType),
    ylim = ylim,
    xlab= paste("Nom de ", variableType),
    ylab= paste("Valeurs des ", variableType))
}
else { # Sinon on realise plusieurs graphiques
  for(i in 1:(ceiling(data$nbVar) / maxGenesPerPlot)) {
    start <- (i-1)*maxGenesPerPlot+1
    end <- min(i*maxGenesPerPlot, data$nbVar)
    boxplot(usedTable[,start:end],
      main=paste("Representation des ", variableType, " ", start, " a ",
                 end, collapse=""),
      ylim = ylim,
      xlab = paste("Nom de ", variableType),
      ylab = paste("Valeurs des ", variableType))
  }
}

# Eboulis de valeurs propres avec lignes additionnelles
displayScreePlot(data)

# Heatmap de la matrice de correlation
displayCorrelation(data)

# affichage des variables et des individus
if(data$numberOfAxis > 1) { # Cas classique avec plusieurs composantes principales
  for(i in 1:(data$numberOfAxis-1)) {
    for(j in (i+1):data$numberOfAxis) { # On fait chaque combinaison d'axes possibles
      # Affichage des variables
      displayVariables(data, axe1 = i, axe2 = j, beLike=beLike)
      # Affichage des individus
      displayIndividus(data, colors=colors, axe1 = i, axe2 = j, beLike=beLike)
      # Classification
      displayClasses(data, axe1 = i, axe2 = j, colors = classColors,
                     usePreviousGraph = T, beLike=beLike, nbCenters = numberOfClasses)
    }
  }
}

```

```

} else {
  # Si il n'y a qu'un axe, on gère ce cas
  displayIndividus(data, colors = colors, axe1 = 1, axe2 = 1)
}

# Affichage dans la console de l'inertie des individus et des variables
displayInertiaIndividus(data, displayLimit = displayLimit, bestContribution = bestContributions)
displayInertiaVariables(data, displayLimit = displayLimit, bestContribution = bestContributions)

}

# Fonction pour afficher le tableau de corrélation
displayCorrelation <- function(data) {
  if(data$nbVar < 10) { # Pour peu de variables, on affiche le tableau de corrélation en console
    print(data$correlation)
  } else { # Sinon on l'affiche graphiquement
    displayCorrelationAsHeatmap(data$correlation)
  }
}

# Affichage de l'inertie et contribution des individus sur les axes
displayInertiaIndividus <- function(data, bestContribution=NULL,
                                      displayLimit = Inf, numberofAxis = Inf) {
  numberofAxis <- min(data$numberofAxis, numberofAxis)
  displayLimit = min(displayLimit, data$nbInd)

  # Affichage des coordonnées
  print("Coordonnées des individus")
  print(round(data$ind$coord[1:displayLimit, 1:numberofAxis], roundingValue))

  # Affichage des contributions
  print("[CTR en %]")
  print(round(data$ind$contrib[1:displayLimit, 1:numberofAxis] * 100, roundingValue))

  # Affichage de la qualité de représentation
  print("[QLT en %]")
  print(round(data$ind$quality[1:displayLimit, 1:numberofAxis] * 100, roundingValue))

  # Pour chaque axes, on affiche les plus grosses contributions
  for(axe in 1:numberofAxis) {
    # Contributions positives
    bestPlus <- sort(data$ind$contrib[,axe][data$ind$coord[,axe] > 0], decreasing = T)
    bestPlus <- bestPlus[bestPlus > 1/ data$nbInd]
    print(paste("Individus contribuant le plus à l'axe", axe, "(positif)", collapse=" "))
    if(!is.null(bestContribution)) bestPlus <- head(bestPlus, n = bestContribution)
    print(round(bestPlus * 100, roundingValue))

    # Contributions négatives
    bestMoins <- sort(data$ind$contrib[,axe][data$ind$coord[,axe] < 0], decreasing = T)
    bestMoins <- bestMoins[bestMoins > 1/ data$nbInd]
    print(paste("Individus contribuant le plus à l'axe", axe, "(négatif)", collapse=" "))
    if(!is.null(bestContribution)) bestMoins <- head(bestMoins, n = bestContribution)
    print(round(bestMoins * 100, roundingValue))
  }
}

```

```

}

}

# Affichage des inerties et qualites des variables
displayInertiaVariables <- function(data, bestContribution=NULL,
                                      displayLimit = Inf, numberOfWorkAxis = Inf) {
  numberOfWorkAxis <- min(data$numberOfWorkAxis, numberOfWorkAxis)
  displayLimit = min(displayLimit, data$nbVar)

  # Affichage des coordonnees
  print("Coordonnees des Variables")
  print(round(data$var$coord[1:displayLimit, 1:numberOfWorkAxis], roundingValue))

  # Affichage des contributions
  print("[CTR en %]")
  print(round(data$var$contrib[1:displayLimit, 1:numberOfWorkAxis] * 100, roundingValue))

  # Affichage de la qualite de la representation
  print("[QLT en %]")
  print(round(data$var$quality[1:displayLimit, 1:numberOfWorkAxis] * 100, roundingValue))

  # Pour chaque axes, on affiche les plus grosses contributions
  for(axe in numberOfWorkAxis) {
    # Contributions positives
    bestPlus <- sort(data$var$contrib[,axe] [data$var$coord[,axe] > 0], decreasing = T)
    bestPlus <- bestPlus[bestPlus > 1/ data$nbVar]
    print(paste("Variables contribuant le plus a l'axe", axe, "(positif)", collapse=" "))
    if(!is.null(bestContribution)) bestPlus <- head(bestPlus, n = bestContribution)
    print(round(bestPlus * 100, roundingValue))

    # Contributions negatives
    bestMoins <- sort(data$var$contrib[,axe] [data$var$coord[,axe] < 0], decreasing = T)
    bestMoins <- bestMoins[bestMoins > 1/ data$nbVar]
    print(paste("Variables contribuant le plus a l'axe", axe, "(negatif)", collapse=" "))
    if(!is.null(bestContribution)) bestMoins <- head(bestMoins, n = bestContribution)
    print(round(bestMoins * 100, roundingValue))
  }
}

# Classification non-supervisee par K-means
displayClasses <- function(data, nbCenters = 2, axe1 = 1, axe2 = 2,
                           colors = rainbow(nbCenters), usePreviousGraph = F, beLike=NULL) {
  coord = data$ind$coord
  if(!is.null(beLike)) {
    # Pour un affichage faisan "FactoMineR", on inverse l'axe 1
    if(grepl(toupper(beLike), "ADE", fixed=T))
    {
      coord[,axe2] = -coord[,axe2]
    }
    else if(grepl(toupper(beLike), "FACTOMINER", fixed=T))
    {
      # Rien a faire
    }
  }
}

```

```

}

# On retrouve les couleurs utilisees pour les classes
classColors = colors
colorToBlack <- F
if(!is.null(classColors)) {
  colorToBlack <- T
  if(!grepl(toupper(classColors), "AUTO", fixed=T)) {
    classColors = levels(as.factor(classColors))
    colorToBlack <- F
  }
}
colors = classColors
# Utilisation de la fonction kmeans de la library "stats"
kmean <- kmeans(coord[, c(axe1, axe2)], nbCenters, nstart = 1)
# Si on souhaite un nouveau graphique, on lance l'affichage des individus
if(!usePreviousGraph){
  displayIndividus(data, axe1=axe1, axe2=axe2, colors=colors[kmean$cluster])
}
# Affichage de chaque groupes
for(i in 1:nbCenters) {
  valeurs <- coord[which(kmean$cluster == i),]
  if(!is.matrix(valeurs)) {
    # Leger traitement dans le cas où il y a un groupe avec un seul individu
    valeurs <- t(as.matrix(valeurs))
  }
  if(identical(toupper(colors), "AUTO"))
  {
    color = rainbow(n = nbCenters)[i]
    # Si on ne souhaite pas de couleurs, on affiche les groupes en tracant une ligne entre les points
    segments(x0=valeurs[,axe1],
              y0=valeurs[,axe2],
              x1=rep(kmean$centers[i, 1], nrow(valeurs)),
              y1=rep(kmean$centers[i, 2], nrow(valeurs)),
              col = color)
  }
  else {
    # Sinon on affiche une ellipse d'inertie
    ell <- dataEllipse(valeurs[,axe1], valeurs[,axe2], levels = 0.95, draw = F)
    polygon(ell, col=colors[i], density = 5)
  }
}
}

# Comparer deux variables pour savoir s'il y a des differences significatives entre les bibliotheques
compareValues <- function(v1, v2) {
  v1 <- as.matrix(v1)
  v2 <- as.matrix(v2)
  # Il est possible qu'une matrice est transposee par rapport a l'autre, on corrige grossierement
  if(ncol(v1) != ncol(v2) & ncol(v1) == nrow(v2)) v2 <- t(v2)

  # On verifie les dimensions
  if(identical(dim(v1), dim(v2))) {
    print(paste("Dimensions identiques : ", dim(v1)[1], " x ", dim(v2)[2], collapse=""))
  }
}

```

```

} else {
  print(paste("Dimension 1 : ", dim(v1)[1], " x ", dim(v1)[2], collapse=""))
  print(paste("Dimension 2 : ", dim(v2)[1], " x ", dim(v2)[2], collapse=""))
}
# On calcule la difference entre les matrices
dimensions <- c(min(dim(v1)[1], dim(v2)[1]), min(dim(v1)[2], dim(v2)[2]))
difference <- abs(v1[1:dimensions[1], 1:dimensions[2]]) - abs(v2[1:dimensions[1], 1:dimensions[2]])
if(any(which(difference > 10^(-epsilonPower)) > 0)) { # S'il y a une difference, on affiche la matrice
  print("Difference remarquée")
  print(v1[1:dimensions[1], 1:dimensions[2]] - v2[1:dimensions[1], 1:dimensions[2]])
  return(FALSE)
} else { # Sinon, on est content
  print("Pas de difference")
  return(TRUE)
}
}

# Comparaison des valeurs trouvées entre notre fonction et celle de FactoMineR
comparaisonAvecFactoMineR <- function(table, center=T,
                                         scale=T, ind.sup=NULL,
                                         var.sup=NULL, numberOfAxis = 5)
{
  print("Comparaison avec FactoMineR :")
  print("-----")
  # Il semblerait que FactoMineR ne peut pas centrer sans reduire
  centreReduire <- center & scale
  # On lance les PCA avec FactoMineR et notre fonction pour les comparer
  facto <- FactoMineR::PCA(table, scale.unit = centreReduire,
                            ind.sup = ind.sup, quanti.sup = var.sup,
                            graph = F, ncp = numberOfAxis)
  nantes <- nantes_pca(table, center = centreReduire, scale = centreReduire,
                        additionalIndividus = ind.sup, additionalVariables = var.sup,
                        numberofAxis = numberofAxis)

  egal <- TRUE
  # SVD :
  print("Valeurs propres :")
  egal <- compareValues(facto$eig[,1], nantes$eig[nantes$eig > 10^(-epsilonPower)])

  print("Matrice U :")
  egal <- compareValues(facto$svd$U, nantes$svd$U)

  print("Matrice V :")
  egal <- compareValues(facto$svd$V, nantes$svd$V)

  print("Valeurs singulières :")
  egal <- compareValues(facto$svd$vs, nantes$svd$vs)

  # Individus :
  print("Coordonnées des individus :")
  egal <- compareValues(facto$ind$coord, nantes$ind$coord)

  # Variables :

```

```

print("Coordonnees des variables :")
egal <- compareValues(facto$var$coord, nantes$var$coord)

print(egal)
if(egal == TRUE) {
  print("Tout est identique! Bravo.")
  return(TRUE)
}
return(FALSE)
}

# Comparaison des valeurs trouvées entre notre fonction et celle de ADE4
comparaisonAvecADE4 <- function(table, center=T, scale=T, ind.sup=NULL,
                                 var.sup=NULL, numberofAxis = 5)
{
  print("Comparaison avec ADE4 :")
  print("-----")
  # Il semblerait que ADE4 ne traite pas les individus et variables supplémentaires
  ind.sup = NULL
  var.sup = NULL
  # On lance les PCA avec FactoMineR et notre fonction pour les comparer
  ade <- ade4::dudi.pca(table, center = center, scale = scale,
                        scannf = F, nf = numberofAxis, )
  nantes <- nantes_pca(table, center = center, scale = scale,
                        additionalIndividus = ind.sup,
                        additionalVariables = var.sup,
                        numberofAxis = numberofAxis)

  egal <- TRUE
  # SVD :
  print("Valeurs propres :")
  egal <- compareValues(ade$eig, nantes$eig[nantes$eig > 10^(-epsilonPower)]) 

  # Individus :
  print("Coordonnees des individus :")
  egal <- compareValues(ade$li, nantes$ind$coord)

  # Variables :
  print("Coordonnees des variables :")
  egal <- compareValues(ade$co, nantes$var$coord)

  print(egal)
  if(egal == TRUE) {
    print("Tout est identique! Bravo.")
    return(TRUE)
  }
  return(FALSE)
}

# Notre fonction d'ACP
nantes_pca <- function(table, center=T, scale=T, scannf = F,
                       numberofAxis=NULL, additionalVariables=NULL,
                       additionalIndividus=NULL, minAxis=2) {

```

```

data <- list()
originalTable <- table
data$originalTable <- table # On garde en memoire notre tableau d'origine avant modifications

ind.sup = NULL
var.sup = NULL
if(!is.null(additionalIndividus)) { # S'il y a des individus supplementaires
  ind.sup <- as.matrix(originalTable[additionalIndividus,]) # On les place dans une variable
  # Peut-être a decommenter
  if(!is.null(additionalVariables)) { # S'il y a en plus des variables supplementaires, on les retire
    ind.sup <- ind.sup[, -additionalVariables]
  }
  # Ou peut-être a commenter...
  table <- table[-additionalIndividus,] # On supprime les individus de notre table
}
if(!is.null(additionalVariables)) { # S'il y a des variables supplementaires
  var.sup <- as.matrix(originalTable[, additionalVariables]) # On les place dans une variable
  if(!is.null(additionalIndividus)) { # On retire les lignes concernees par des individus supplementaires
    var.sup <- as.matrix(var.sup[-additionalIndividus, ])
  }
  table <- table[,-additionalVariables] # On retire les variables de notre table
}
# En cours : retirer les variables "qualitatives" de notre table
# qualitatifs <-

nbIndividus <- nrow(table)
nbParameters <- ncol(table)
data$nbInd <- nbIndividus
data$nbVar <- nbParameters

#if (!exists("rowW") | is.null(rowW))
rowW <- rep(1, nbIndividus)
#if (!exists("colW") | is.null(colW))
colW <- rep(1/nbParameters, nbParameters)
if(is.null(colnames(table))) colnames(table) <- 1:ncol(table)
D <- diag(rep(1/nbIndividus, nbIndividus))
Q <- diag(rep(1, nbParameters))

# On recuper la table centre non-reduite
centered <- centerAndScale(table, center=T, scale=F)
# la table centre-reduite
centeredReduced <- centerAndScale(table, center=T, scale=T)
# et celle demandee
data$table <- centerAndScale(table, center=center, scale=scale)

# On stock la moyenne
cent <- colMeans(table)
# Et les écarts-types
norm <- apply(table, 2, function(x){ sqrt(getVariance(x)) })
data$centeredReduced <- centeredReduced

# La table de covariance se recuper avec S = Xc' D Xc

```

```

covariance <- t(centered) %*% D %*% centered
# La table de correlation se calcule avec  $R = X_{cr}' D X_{cr} Q$ 
correlation <- (t(centeredReduced) %*% D %*% centeredReduced %*% Q)
data$correlation <- correlation

# Recuperation des valeurs propres
eigenValues <- getEigenValues(data)
data$eig <- eigenValues

# Le rang est defini par le nombre de valeurs propres (superieures a 0)
rank <- length(eigenValues[eigenValues>0])

# Si on ne demande pas un nombre de composantes exact, on le calcul "automatiquement"
if(scannf){
  data$numberOfAxis = 0
  displayScreePlot(data)
  numberofAxis <- as.integer(readline(prompt = "Nombre d'axes à garder : "))
} else if(is.null(numberofAxis)){
  numberofAxis <- min(getNbAxes(data))
  numberofAxis <- max(numberofAxis, minAxis)
}
data$numberOfAxis <- numberofAxis

# Les composantes de la SVD (U et V) sont calculees dans une fonction a part
svd <- SVD(data)
U <- svd$U
V <- svd$V
data$svd <- svd

# On va raccourcir la liste de valeurs propres si elle est trop grande
# pour les donnees, on reduit
# (Solution peut-être simplifiee, mais suffisante...)
if(length(eigenValues) > min(nbIndividus, nbParameters)) {
  eigenValues <- eigenValues[1:min(nbIndividus, nbParameters)]
  data$rank <- min(nbIndividus, nbParameters)
}
# Les valeurs singulieres sont les racines carrees des valeurs propres
VS <- sqrt(eigenValues)

# Les coordonnees des individus et variables
# Individus = U * VS
ind.coord <- t(t(as.matrix(U)) * VS[1:numberOfAxis])
# Variables = V * VS
var.coord <- t(t(as.matrix(V)) * VS[1:numberOfAxis])

# Ajout de noms si possible aux matrices
colnames(ind.coord) <- paste("Axe", 1:ncol(ind.coord), sep=" ")
if(length(rownames(table)) > 0) {
  rownames(ind.coord) <- rownames(table)
} else {
  rownames(ind.coord) <- 1:nbIndividus
}
colnames(var.coord) <- paste("Axe", 1:ncol(var.coord), sep=" ")

```

```

if(length(colnames(table)) > 0) {
  rownames(var.coord) <- colnames(table)
} else {
  rownames(var.coord) <- 1:nbParameters
}

# La contribution par axe est calcule par (coord^2)/valeur propre de l'axe
ind.contrib <- t(t(ind.coord**2) / eigenValues[1:ncol(ind.coord)])
var.contrib <- t(t(var.coord**2) / eigenValues[1:ncol(var.coord)])
# Si la valeur propre est tres proche de 0, la contribution
# sera proche de +infini, dans le doute, on passe a 0...
ind.contrib[ind.contrib > 100] <- 0
var.contrib[var.contrib > 100] <- 0

# Distance entre les individus : somme des coordonnees ^2
ind.distanc <- rowSums(ind.coord**2)
# Distance entre les variables :
var.distanc <- as.vector(t(rep(1, nrow(centeredReduced))) %*%
                           as.matrix((centeredReduced**2)/nbIndividus))

ind.quality <- ind.coord**2 / ind.distanc
colnames(ind.quality) <- paste("Axe", 1:ncol(ind.quality), sep=" ")
rownames(ind.quality) <- 1:nbIndividus

var.cor <- var.coord / sqrt(var.distanc)
var.quality <- var.cor**2
colnames(var.quality) <- paste("Axe", 1:ncol(var.quality), sep=" ")
if(length(colnames(table)) > 0) {
  rownames(var.quality) <- colnames(table)
} else {
  rownames(var.quality) <- 1:nbParameters
}

data$ind <- list(coord = ind.coord,
                  contrib = ind.contrib,
                  quality = ind.quality, dist = ind.distanc)
data$var <- list(coord = var.coord,
                  contrib = var.contrib,
                  quality = var.quality, dist = var.distanc, cor = var.cor)
data$call <- list()

if(!is.null(ind.sup)) {
  ind.sup.centered <- t(t(ind.sup) - cent)
  ind.sup.centeredReduced <- t(t(ind.sup.centered) / norm)

  ind.sup.coord <- ind.sup.centeredReduced %*% svd$V
  ind.sup.distanc <- rowSums(ind.sup.coord**2)
  ind.sup.contrib <- 1#t(t((ind.sup.coord**2) * rowW / sum(rowW)) / eigenValues)
  ind.sup.quality <- ind.sup.coord**2 / ind.sup.distanc

  data$ind$sup <- list(nbSup = nrow(ind.sup),
                      coord = ind.sup.coord,
                      contrib = ind.sup.contrib,
                      quality = ind.sup.quality,
                      dist = ind.sup.distanc)
}

```

```

}

if(!is.null(var.sup)) {
  var.sup.centered <- t(t(var.sup) - colMeans(originalTable)[additionalVariables])
  var.sup.norm <- sqrt(colSums((var.sup**2)/ncol(var.sup)))
  var.sup.centeredReduced <- t(t(var.sup.centered) / var.sup.norm)

  var.sup.coord <- t(var.sup.centeredReduced) %*% svd$U
  var.sup.contrib <- t(t(var.sup.coord**2) / eigenValues)

  var.sup.distanc <- as.vector(produitVect(rep(1,nrow(centeredReduced)),
                                             as.matrix(centeredReduced**2)))

  var.sup.cor <- var.sup.coord / sqrt(var.sup.distanc)
  var.sup.quality <- var.sup.cor**2

  data$var$sup <- list(nbSup = ncol(var.sup),
                       coord = var.sup.coord,
                       contrib = var.sup.contrib,
                       quality = var.sup.quality,
                       dist = var.sup.distanc)
}

return(data)
}

```