TP n°4 - Décomposition en valeurs singulières & Analyse en composantes principales

Salim Nadir et Guillaume Ostrom

Exercice 1 : Décomposition en valeurs singulières

1. Simulation d'un vecteur gaussien

Nous simulons un vecteur aléatoire gaussien

```
gaussian_50_5 = matrix(rnorm(50*5),ncol = 5, nrow = 50)
```

2. Matrice de covariance

Nous calculons la matrice de covariance empirique.

```
S = cov(gaussian_50_5)
print(S)
```

```
##             [,1]         [,2]        [,3]        [,4]         [,5]
## [1,]  0.85810196 -0.158695503 -0.02887875  0.20172068  0.080275330
## [2,] -0.15869550  0.888426581 -0.12808750 -0.15952957 -0.001276487
## [3,] -0.02887875 -0.128087498  0.77875326  0.04668146 -0.076064504
## [4,]  0.20172068 -0.159529565  0.04668146  0.97165484 -0.098117476
## [5,]  0.08027533 -0.001276487 -0.07606450 -0.09811748  1.190295537
```

La matrice obtenue est carré de taille 4 symétrique à coefficients réels. Nous remarquons que la variance des vecteurs n'est pas de 1 alors qu'ils ont été générés par des loi normales i.i.d. Il faut donc la générer avec d'avantages de données car quand n tend vers l'infini on tend vers la matrice identité.

3. Décomposition en valeurs singulières

La décomposition en valeurs singulère de S de taille n,p implique l'existence de U de taille n,n et V de taille p,p matrices carrés orthogonales réelles et de SIGMA une matrice diagonale de taille n,p de termes croissants, tel que:

$$S = U\Sigma V^T$$

```
svds=svd(S)
U = svds$u
print(U)
```

```
##            [,1]        [,2]       [,3]       [,4]       [,5]
## [1,] -0.3520035  0.43492938 -0.3096627 -0.4755817  0.6040373
## [2,]  0.4531738 -0.28850681 -0.5341771  0.4018474  0.5143646
## [3,] -0.2210213 -0.06619061  0.6874134  0.3826547  0.5725439
## [4,] -0.6251035  0.20423273 -0.3626490  0.6285421 -0.2023735
## [5,]  0.4807518  0.82553501  0.1212944  0.2661782 -0.0425028
```

```
V = svds$v
print(V)
```

```
##            [,1]        [,2]       [,3]       [,4]       [,5]
## [1,] -0.3520035  0.43492938 -0.3096627 -0.4755817  0.6040373
## [2,]  0.4531738 -0.28850681 -0.5341771  0.4018474  0.5143646
## [3,] -0.2210213 -0.06619061  0.6874134  0.3826547  0.5725439
## [4,] -0.6251035  0.20423273 -0.3626490  0.6285421 -0.2023735
## [5,]  0.4807518  0.82553501  0.1212944  0.2661782 -0.0425028
```

```
SIGMA = diag(svds$d)
print(SIGMA)
```

```
##          [,1]     [,2]      [,3]      [,4]      [,5]
## [1,] 1.292864 0.000000 0.0000000 0.0000000 0.0000000
## [2,] 0.000000 1.214859 0.0000000 0.0000000 0.0000000
## [3,] 0.000000 0.000000 0.8532483 0.0000000 0.0000000
## [4,] 0.000000 0.000000 0.0000000 0.7039002 0.0000000
## [5,] 0.000000 0.000000 0.0000000 0.0000000 0.6223606
```

4. Σ, U, V et comparaisons

La diagonale de la matrice Σ comporte les valeurs singulières de S par ordre décroissant.

Une comparaison peut être :

```
print(U-V)
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,] -2.220446e-16  1.665335e-16  1.665335e-16  1.110223e-16  2.220446e-16
## [2,] -2.220446e-16  3.330669e-16 -2.220446e-16  1.110223e-16 -1.110223e-16
## [3,] -2.775558e-17 -4.024558e-16  2.220446e-16  2.220446e-16  2.220446e-16
## [4,]  2.220446e-16  1.942890e-16 -3.330669e-16 -1.110223e-16  1.665335e-16
## [5,] -1.665335e-16 -5.551115e-16  4.996004e-16  5.551115e-17  1.387779e-16
```

```
print(U%*%t(V))
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00  0.000000e+00  3.885781e-16 -2.081668e-16  3.469447e-17
## [2,]  2.775558e-16  1.000000e+00  0.000000e+00 -1.526557e-16  1.387779e-16
## [3,] -1.665335e-16  3.330669e-16  1.000000e+00  5.551115e-17 -6.106227e-16
## [4,] -5.551115e-17 -2.359224e-16 -1.387779e-16  1.000000e+00  4.354156e-16
## [5,] -3.434752e-16 -1.422473e-16  1.422473e-16  2.602085e-17  1.000000e+00
```

On peut aussi utiliser la norme, la norme spectrale et la norme infinie:

```
norm(U-V)
```

```
## [1] 1.651457e-15
```

```
norm(U-V,"2")
```

```
## [1] 1.061709e-15
```

```
norm(U-V,"I")
```

```
## [1] 1.415534e-15
```

Les normes devraient être égales à 0, mais nous obtenons des résultats de l'odre de 10^-15 pour les deux normes. La différence doit provenir d'approximations numériques.

```
print(sum(diag(S)))
```

```
## [1] 4.687232
```

```
print(sum(diag(SIGMA)))
```

```
## [1] 4.687232
```

Nous observons que les traces sont identiques car U et V sont des matrices orthogonales de changement de base.

5. Calcul de $U\Sigma V^T$ et remarques.

Nous calculons $U\Sigma V^T$:

```
print(U%*%SIGMA%*%t(V))
```

```
##             [,1]         [,2]        [,3]        [,4]         [,5]
## [1,]  0.85810196 -0.158695503 -0.02887875  0.20172068  0.080275330
## [2,] -0.15869550  0.888426581 -0.12808750 -0.15952957 -0.001276487
## [3,] -0.02887875 -0.128087498  0.77875326  0.04668146 -0.076064504
## [4,]  0.20172068 -0.159529565  0.04668146  0.97165484 -0.098117476
## [5,]  0.08027533 -0.001276487 -0.07606450 -0.09811748  1.190295537
```

```
print(norm(U%*%SIGMA%*%t(V)-S,'2'))
```

```
## [1] 1.320348e-15
```

Les deux matrices sont quasiment identiques à 10^-16 près. La différence doit provenir d'approximations numériques.

Exercice 2 : Analyse en composantes principales

Analyse préliminaire

```
#install.packages("ade4")
library(ade4)
```

```
## Warning: package 'ade4' was built under R version 3.3.2
```

1. Analyse rapide

Les données contenues dans *cardata.txt* sont des caractéristiques de modèles d'automobiles. Il n'y a pas beaucoup de données et elles sont présentes dans des unités différentes et d'ordre de grandeur différents. Nous devrons donc normaliser celles-ci.

2. Dataframe X

Récupérons les données :

```
setwd("D:\\Centrale\\OMA\\SDMA\\TP4\\")
X=read.table('cardata.txt',sep=';',header=TRUE,row.names = 'model')
nrow(X)
```

```
## [1] 24
```

```
#header(X)
row.names(X)
```

```
##  [1] "HondaCivic "         "Renault19 "          "FiatTipo  "
##  [4] "Peugeot405 "         "Renault21 "          "CitroenBX "
##  [7] "BMW530i "            "Rover827i "          "Renault25 "
## [10] "OpelOmega "          "Peugeot405Break "    "FordSierra "
## [13] "BMW325i "            "Audi90 Quatro "      "Ford Scorpio "
## [16] "Renault Espace "     "Nissan Vanette "     "VW Caravelle "
## [19] "Ford Fiesta "        "Fiat Uno "           "Peugeot 205 "
## [22] "Peugeot 205 Rallye " "Seat Ibiza SXI "     "Citroen AX Sport "
```

```
ncol(X)
```

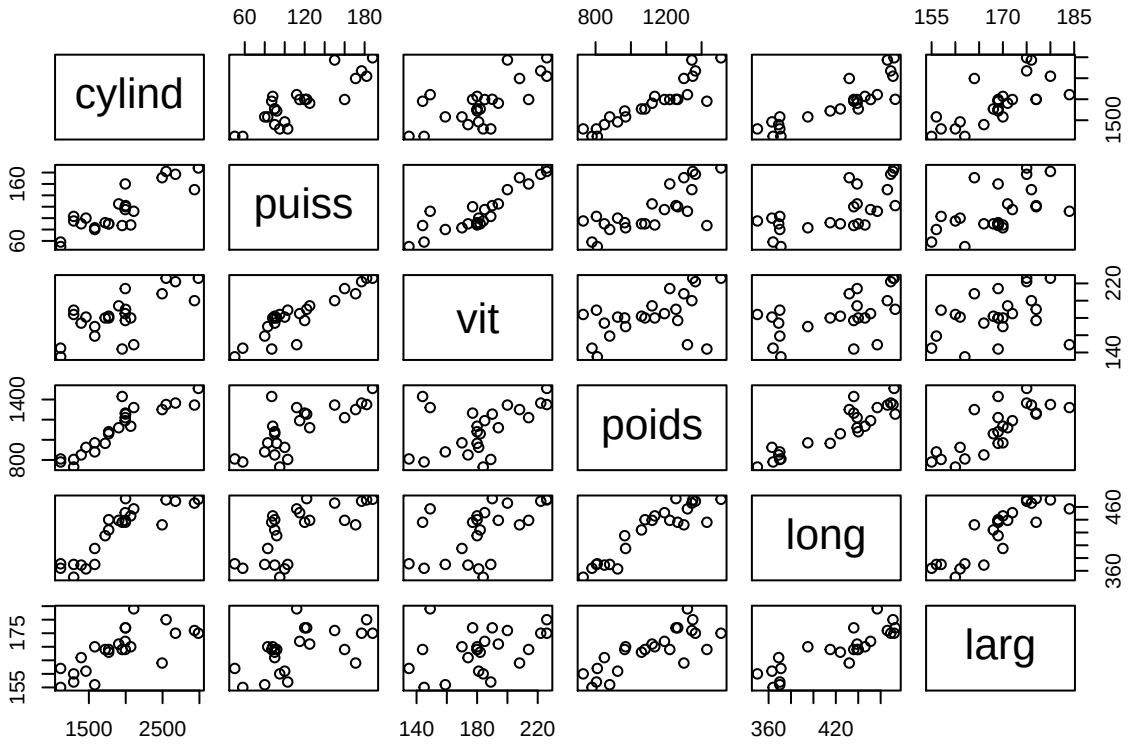
```
## [1] 6
```

Nous observons 6 variables explicatives sur 24 modèles de voitures.

3. Caractérisation

Affichons le graphique de données:

```
plot(X)
```



Nous observons de fortes corrélations entre puissance et vitesse mais aussi entre poids longeur et largeur. Globalement les varaibles sont corrélées.

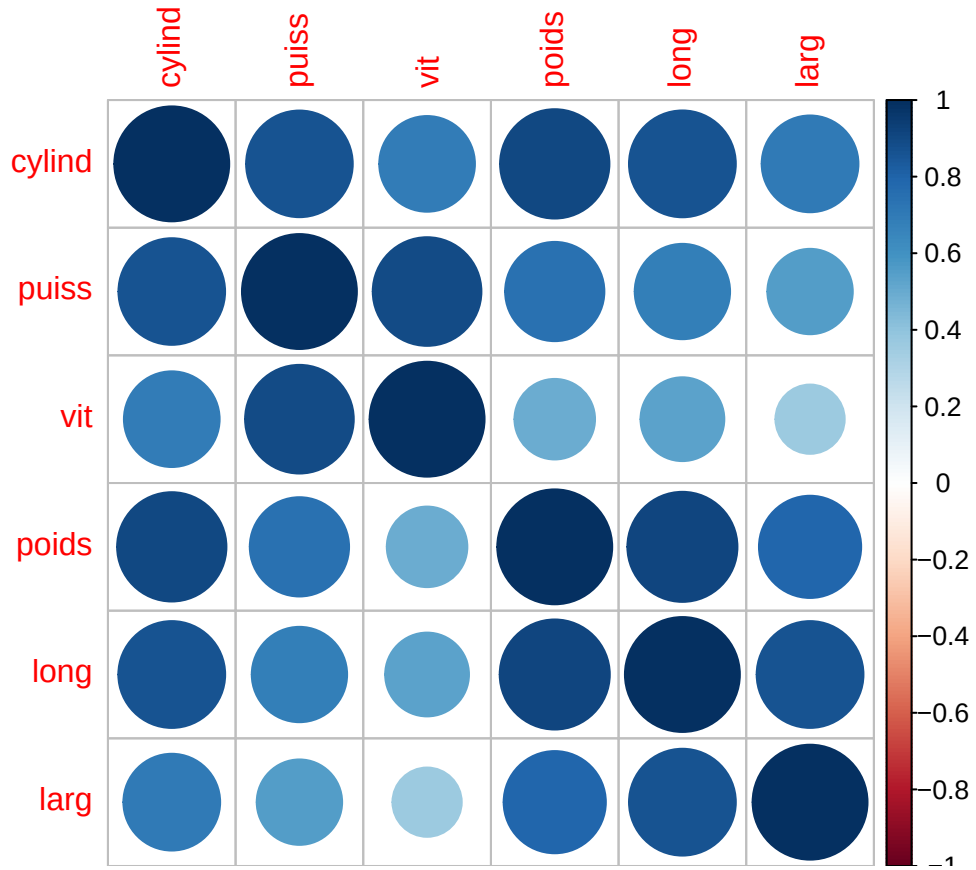
```
cor(X)
```

```
##           cylind     puiss       vit     poids      long      larg
## cylind 1.0000000 0.8609761 0.6933151 0.9049341 0.8642025 0.7090589
## puiss  0.8609761 1.0000000 0.8939873 0.7460794 0.6885147 0.5522800
## vit    0.6933151 0.8939873 1.0000000 0.4914196 0.5319080 0.3632342
## poids  0.9049341 0.7460794 0.4914196 1.0000000 0.9171122 0.7909150
## long   0.8642025 0.6885147 0.5319080 0.9171122 1.0000000 0.8638142
## larg   0.7090589 0.5522800 0.3632342 0.7909150 0.8638142 1.0000000
```

Nous constatons ces fortes corrélations dans la matrice de covariance. Les coefficients proches de 1 indiquent de très forte corrélations entre les variables. Ainsi cylindrée et poids, longueur et poids, puissance et vitesse, cylindrée et puissance sont entre autres fortement corrélés.

Nous pouvons mettre en évidence ces corrélations avec corrplot:

```
library(corrplot)
corrplot(cor(X))
```



Nous devons utiliser l'analyse en composante principale pour réduire la dimension et décorréler les variables entre elles-mêmes.

4. Analyse en Composante Principale

Nous réalisons une ACP sur des données normalisées :

```
res = prcomp(X,center = TRUE,scale = TRUE)
```

Caractérisons les résulats:

```
attributes(res)
```

```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

L'écart type des composantes principales:

```
res$sdev
```

```
## [1] 2.1577815 0.9566721 0.4903373 0.3204833 0.2542759 0.1447788
```

Nous obtenons la matrice des valeurs propres par ordre décroissant.

Matrice des vecteurs propres:

```
res$rotation
```

```
##              PC1         PC2         PC3         PC4          PC5
## cylind 0.4442019  0.03396424 -0.40143238  0.05002484 -0.798600425
## puiss  0.4144904  0.42122241 -0.03956072  0.48971700  0.306664942
## vit    0.3435401  0.66343624  0.36993499 -0.31990926  0.007184843
## poids  0.4303213 -0.25516926 -0.48446064  0.12315230  0.472589123
## long   0.4302088 -0.29558404  0.04398442 -0.71184868  0.165971527
## larg   0.3776328 -0.47831913  0.68102742  0.36529141 -0.131359532
##                PC6
## cylind  0.01086252
## puiss   0.56154982
## vit    -0.45009675
## poids  -0.52583568
## long    0.43742517
## larg   -0.11879722
```

Chaque colonne est lié à une des composantes principale.

Vérifions que nous obtenons la matrice identité

```
res$rotation%*%t(res$rotation)
```

```
##               cylind         puiss           vit         poids
## cylind  1.000000e+00  1.361758e-16 -5.551115e-17  3.295975e-16
## puiss   1.361758e-16  1.000000e+00  0.000000e+00  5.551115e-17
## vit    -5.551115e-17  0.000000e+00  1.000000e+00 -5.828671e-16
## poids   3.295975e-16  5.551115e-17 -5.828671e-16  1.000000e+00
## long    9.280771e-17  0.000000e+00 -1.942890e-16  5.551115e-17
## larg   -1.556914e-16 -1.804112e-16  1.457168e-16 -2.567391e-16
##                 long          larg
## cylind  9.280771e-17 -1.556914e-16
## puiss   0.000000e+00 -1.804112e-16
## vit    -1.942890e-16  1.457168e-16
## poids   5.551115e-17 -2.567391e-16
## long    1.000000e+00  1.179612e-16
## larg    1.179612e-16  1.000000e+00
```

à 10^-17 près nous obtenons la matrice identité.

Centrons les données:

```
res$center
```

```
##    cylind     puiss       vit     poids      long      larg
## 1906.1250  113.6667  183.0833 1110.8333  421.5833  168.8333
```

Normalisons:

```
res$scale
```

```
##     cylind      puiss        vit      poids       long       larg
## 527.908697  38.784428  25.215448 230.291246  41.340491   7.653738
```

Affichons les valeurs centrées, normalisées dans la base des composantes principales:

```
res$x
```

```
##                              PC1         PC2         PC3          PC4
## HondaCivic           -1.98031246  0.31320636  0.51944454  0.398799200
## Renault19            -0.76212089 -0.13010583  0.43225086 -0.208674274
## FiatTipo             -1.26263558 -0.42506511  0.45912275  0.185979390
## Peugeot405           -0.26805559 -0.45491184  0.18246562 -0.598359864
## Renault21             0.17671129 -0.62373252 -0.06319929 -0.621455359
## CitroenBX            -0.49402237 -0.20323520  0.14787830 -0.406649859
## BMW530i               3.86255660  0.81779768 -0.50472318  0.136070618
## Rover827i             3.12664973  0.75519337 -0.01385351 -0.007428577
## Renault25             3.36720144  0.59641465  0.61488539  0.189098014
## OpelOmega             1.46799561 -0.75934543  0.50120161 -0.392308163
## Peugeot405Break       0.57418132  0.14014832  0.34148947 -0.187094298
## FordSierra            0.72375879 -0.42544911  0.10722306 -0.312307512
## BMW325i               1.67025425  1.33404944 -0.95689442  0.154628514
## Audi90 Quatro         1.38377946  1.06639650  0.14320277 -0.032446536
## Ford Scorpio          2.73620671 -0.11916379 -0.37740116  0.043904414
## Renault Espace        0.90062914 -0.86982953  0.25439714  0.389542444
## Nissan Vanette       -0.02422198 -1.78211534 -1.22233217  0.093879920
## VW Caravelle          1.19626563 -2.33463487  0.29458058  0.656469076
## Ford Fiesta          -3.42518769 -0.88528502 -0.06940878  0.115352017
## Fiat Uno             -3.67857173 -0.01460281 -0.49729388 -0.140199171
## Peugeot 205          -2.56365346  0.40633531 -0.78218086  0.001826213
## Peugeot 205 Rallye   -2.24058198  1.44765724  0.09872414 -0.107853958
## Seat Ibiza SXI       -1.89237182  0.88244127 -0.04654495  0.347202393
## Citroen AX Sport     -2.59445441  1.26783626  0.43696597  0.302025358
##                              PC5          PC6
## HondaCivic           -0.11576686 -0.107856961
## Renault19            -0.22070559 -0.001734473
## FiatTipo             -0.16861466 -0.195002975
## Peugeot405            0.02723234 -0.027763757
## Renault21            -0.32110423 -0.128188412
## CitroenBX            -0.06031364 -0.171571911
## BMW530i              -0.11790140 -0.141283496
## Rover827i            -0.04515210  0.063793928
## Renault25             0.07907649  0.039979286
## OpelOmega             0.29098838  0.087183744
## Peugeot405Break       0.14597274  0.098932156
## FordSierra            0.10587983  0.068222202
## BMW325i               0.08408359  0.150753282
## Audi90 Quatro         0.53331672  0.053226581
## Ford Scorpio         -0.72544974  0.069279889
## Renault Espace        0.14798446 -0.124116749
## Nissan Vanette        0.41860720 -0.266329959
```

```
## VW Caravelle        -0.01866639  0.250167314
## Ford Fiesta         -0.02649990  0.177985910
## Fiat Uno             0.07158975  0.218378419
## Peugeot 205         -0.24025152  0.116187417
## Peugeot 205 Rallye   0.21173252  0.063542308
## Seat Ibiza SXI       0.08260299 -0.243812514
## Citroen AX Sport    -0.13864097 -0.049971228
```

5. Etude des Valeurs propres

```
cumsum((res$sdev**2)/6)
```

```
## [1] 0.7760035 0.9285404 0.9686122 0.9857305 0.9965065 1.0000000
```

Il faut donc conserver 3 composantes principales pour expliquer 95% de la variance. Et 4 composantes principales pour expliquer 98%.

6. Etude des Valeurs propres

Voici les coordonnées des axes principaux dans l'ancienne base:

```
res$rotation
```

```
##              PC1         PC2         PC3         PC4          PC5
## cylind 0.4442019  0.03396424 -0.40143238  0.05002484 -0.798600425
## puiss  0.4144904  0.42122241 -0.03956072  0.48971700  0.306664942
## vit    0.3435401  0.66343624  0.36993499 -0.31990926  0.007184843
## poids  0.4303213 -0.25516926 -0.48446064  0.12315230  0.472589123
## long   0.4302088 -0.29558404  0.04398442 -0.71184868  0.165971527
## larg   0.3776328 -0.47831913  0.68102742  0.36529141 -0.131359532
##                PC6
## cylind  0.01086252
## puiss   0.56154982
## vit    -0.45009675
## poids  -0.52583568
## long    0.43742517
## larg   -0.11879722
```

Nous observons que les pondérations sont importantes pour les variables explicatives les plus importante en valeur absolue:

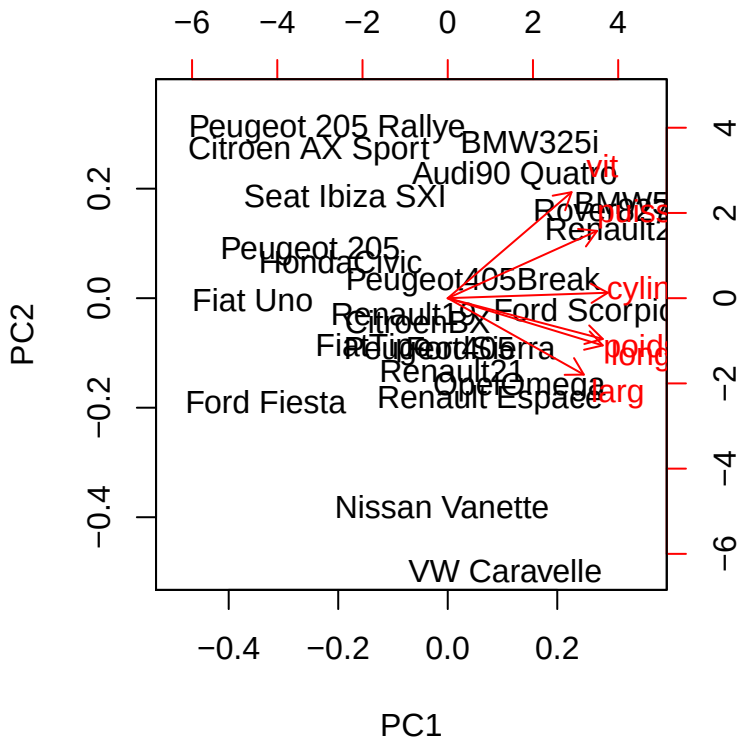
```
res$rotation[,1:4]
```

```
##              PC1         PC2         PC3         PC4
## cylind 0.4442019  0.03396424 -0.40143238  0.05002484
## puiss  0.4144904  0.42122241 -0.03956072  0.48971700
## vit    0.3435401  0.66343624  0.36993499 -0.31990926
## poids  0.4303213 -0.25516926 -0.48446064  0.12315230
## long   0.4302088 -0.29558404  0.04398442 -0.71184868
## larg   0.3776328 -0.47831913  0.68102742  0.36529141
```

7. Analyse des individus projetés et cercle des corrélations

Sur le premier plan factoriel nous obtenons:

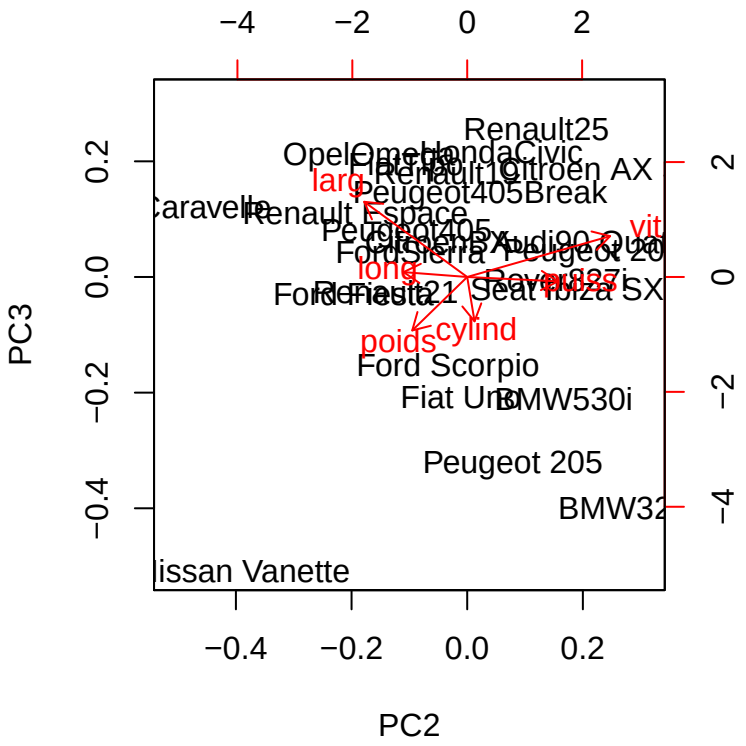
```
biplot(res,choices = c(1,2))
```



Nous remarquons que les sportives sont projetées sur la partie supérieur. Aussi, les familiales sont concentrées sur la partie centrale.

Sur le deuxième plan factoriel nous obtenons:

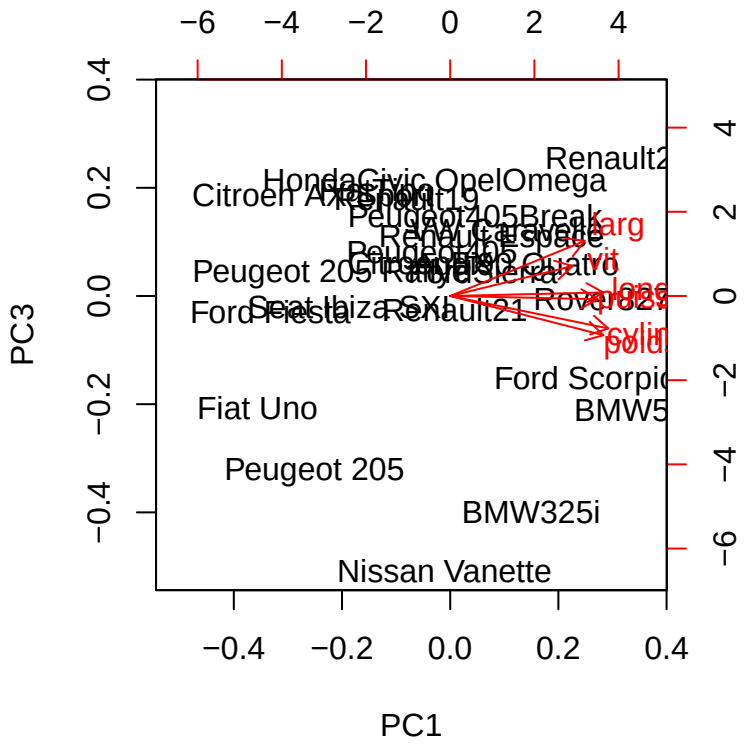
```
biplot(res,choices = c(2,3))
```



On observe que les distances entre chaque points sont inférieurs. Nous remarquons que les données sont plus étalées sur le plan factoriel n°1 que sur le n°2 car le premier explique une plus grande part de la variance.

Sur le troisième plan factoriel nous obtenons:

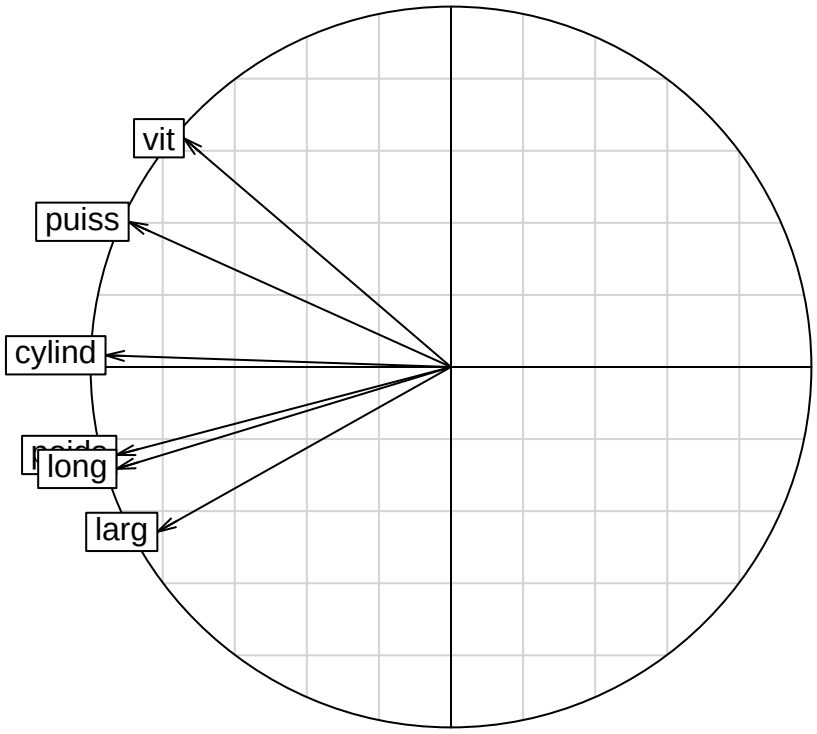
```
biplot(res,choices = c(1,3))
```



Il parait plus complexe de distinguer les groupes.

Ensuite, nous réalisons une ACP avec *ade4*.

```
#install.packages("ade4")
library(ade4)
pca = dudi.pca(X, scan=F)
s.corcircle(pca$co)
```



Nous observons que les correlations sont données par les angles entre les vecteurs.

8. Classification non-supervisée (Kmeans)

Nous implémentons l'algorithme Kmeans pour chaque classes de données tel que :

```
for (k in c(1,2,3,4)){
  result = kmeans(X, k)
  result$centers
    print(result$centers)
  result$cluster
    print(result$cluster)
  result$withinss
    print(result$withinss)
  result$betweenss
    print(result$betweenss)
}
```

```
##     cylind    puiss      vit    poids     long     larg
## 1 1906.125 113.6667 183.0833 1110.833 421.5833 168.8333
##         HondaCivic           Renault19           FiatTipo
##                  1                   1                  1
##         Peugeot405           Renault21           CitroenBX
##                  1                   1                  1
##            BMW530i           Rover827i           Renault25
##                  1                   1                  1
```

```
##          OpelOmega     Peugeot405Break          FordSierra
##                  1                   1                   1
##            BMW325i       Audi90 Quatro        Ford Scorpio
##                  1                   1                   1
##     Renault Espace      Nissan Vanette        VW Caravelle
##                  1                   1                   1
##        Ford Fiesta            Fiat Uno         Peugeot 205
##                  1                   1                   1
## Peugeot 205 Rallye      Seat Ibiza SXI    Citroen AX Sport
##                  1                   1                   1
## [1] 7719474
## [1] 0
##     cylind     puiss      vit     poids     long     larg
## 1 1463.364  84.63636 170.8182  895.9091 384.6364 163.0000
## 2 2280.769 138.23077 193.4615 1292.6923 452.8462 173.7692
##         HondaCivic           Renault19           FiatTipo
##                  1                   1                   1
##         Peugeot405           Renault21           CitroenBX
##                  1                   2                   1
##            BMW530i           Rover827i           Renault25
##                  2                   2                   2
##          OpelOmega     Peugeot405Break          FordSierra
##                  2                   2                   2
##            BMW325i       Audi90 Quatro        Ford Scorpio
##                  2                   2                   2
##     Renault Espace      Nissan Vanette        VW Caravelle
##                  2                   2                   2
##        Ford Fiesta            Fiat Uno         Peugeot 205
##                  1                   1                   1
## Peugeot 205 Rallye      Seat Ibiza SXI    Citroen AX Sport
##                  1                   1                   1
## [1]  728514.2 2023244.6
## [1] 4967715
##     cylind    puiss      vit    poids     long     larg
## 1 1354.750  82.3750 167.1250  843.750 369.0000 160.8750
## 2 2727.200 173.6000 216.4000 1374.000 462.0000 174.0000
## 3 1933.909 109.1818 179.5455 1185.455 441.4545 172.2727
##         HondaCivic           Renault19           FiatTipo
##                  1                   3                   1
##         Peugeot405           Renault21           CitroenBX
##                  3                   3                   3
##            BMW530i           Rover827i           Renault25
##                  2                   2                   2
##          OpelOmega     Peugeot405Break          FordSierra
##                  3                   3                   3
##            BMW325i       Audi90 Quatro        Ford Scorpio
##                  2                   3                   2
##     Renault Espace      Nissan Vanette        VW Caravelle
##                  3                   3                   3
##        Ford Fiesta            Fiat Uno         Peugeot 205
##                  1                   1                   1
## Peugeot 205 Rallye      Seat Ibiza SXI    Citroen AX Sport
##                  1                   1                   1
## [1] 285400.6 226717.2 348584.9
```

```
## [1] 6858772
##     cylind     puiss      vit    poids     long    larg
## 1 1243.400  79.20000 165.4000  795.000 364.8000 160.000
## 2 2727.200 173.60000 216.4000 1374.000 462.0000 174.000
## 3 1646.667  89.16667 175.3333  980.000 401.1667 165.500
## 4 2001.750 116.12500 179.1250 1241.875 447.1250 173.625
##         HondaCivic           Renault19           FiatTipo
##                  1                   3                  3
##         Peugeot405           Renault21           CitroenBX
##                  3                   4                  3
##            BMW530i           Rover827i           Renault25
##                  2                   2                  2
##          OpelOmega     Peugeot405Break          FordSierra
##                  4                   4                  4
##            BMW325i       Audi90 Quatro        Ford Scorpio
##                  2                   4                  2
##     Renault Espace      Nissan Vanette        VW Caravelle
##                  4                   4                  4
##        Ford Fiesta            Fiat Uno         Peugeot 205
##                  1                   1                  3
## Peugeot 205 Rallye      Seat Ibiza SXI    Citroen AX Sport
##                  1                   3                  1
## [1]  73344.0 226717.2 114079.8 108376.9
## [1] 7196956
```

Exercice 3 : Caractères manuscrits

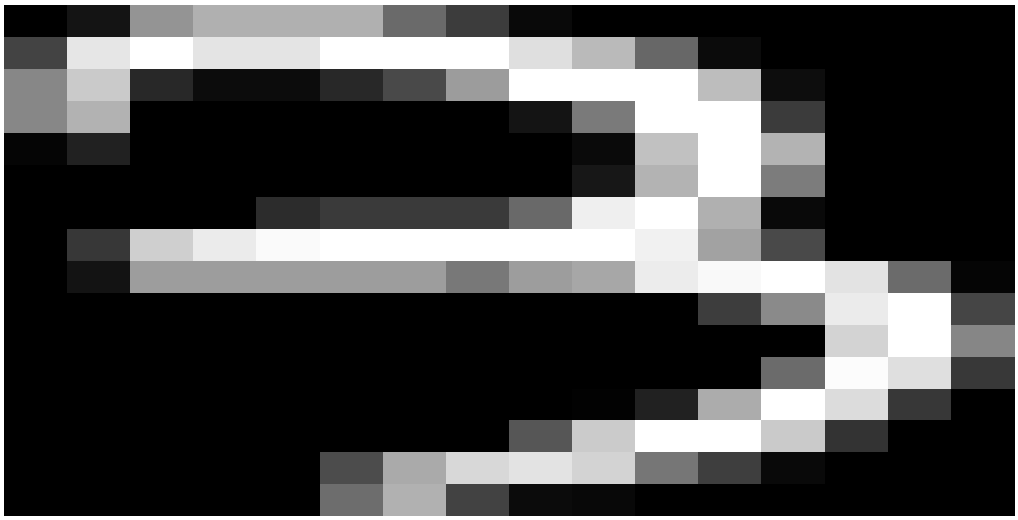
Partie A

```
load("digits3-8.RData")
```

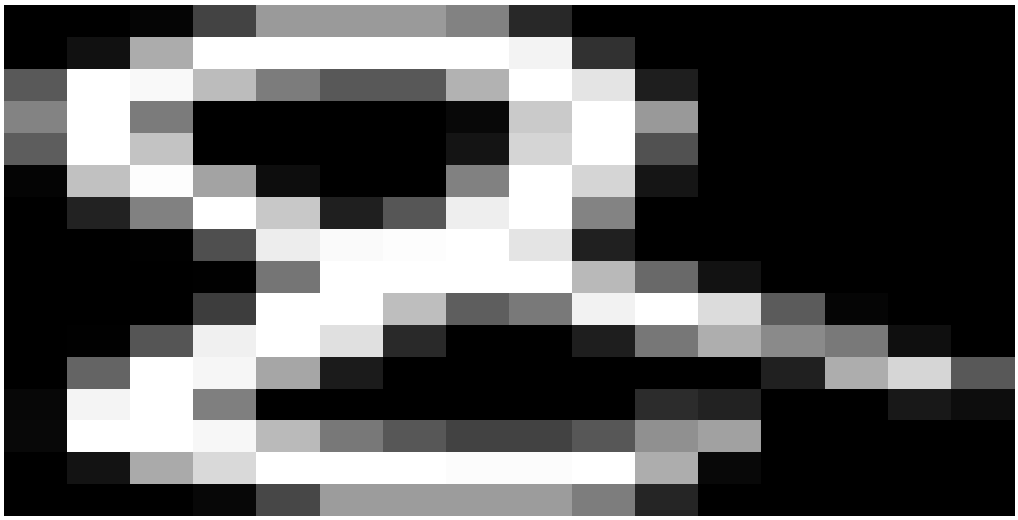
1. Définition de la fonction mImage

```
mImage = function(vect)
{
  image(t(matrix(vect,16,16)), axes=FALSE, col = gray(0:255/255))
}
```

```
mImage(d3[1,])
```



```
mImage(d8[1,])
```



Nous observons les chiffres 3 et 8 manuscritement écrit.

Partie B

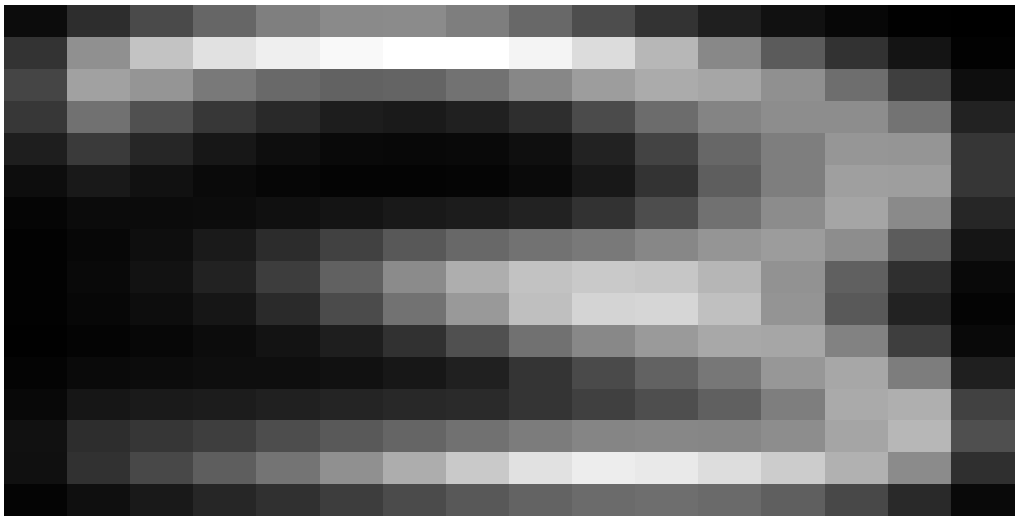
2. Sélection des données

```
data = 1:nrow(d3)
train = sample(nrow(d3), 1000)
test = data[is.na(pmatch(data, train))]
d3train = d3[train,]
d3test = d3[test,]
d8train = d8[train,]
d8test = d8[test,]
```

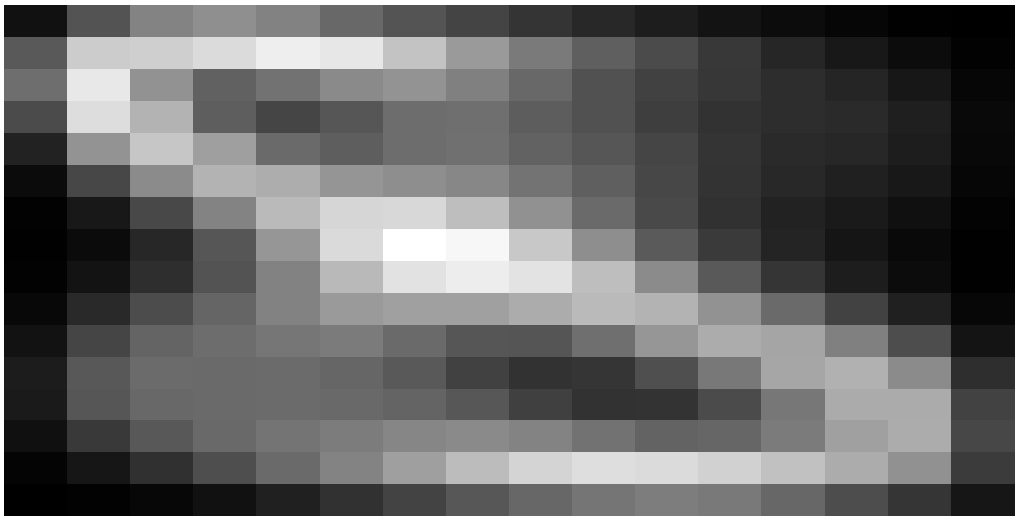
3. “3” & “8” moyens

Affichons les 3 et 8 moyens:

```
mean3 = colMeans(d3train,1)
mImage(mean3)
```



```
mean8 = colMeans(d8train,1)
mImage(mean8)
```



4. Matrices de covariances.

Voici les matrices de covariances :

```
d3train = t(scale(t(d3train)))
d8train = scale(d8train)
cov_3 = cov(d3train)
cov_8 = cov(d8train)
#print(cov_3)
#print(cov_8)
```

5. Composantes principales

La relation est

$$les valeurs propres de la matrice de covariance = \frac{1}{n}(valeurs singulière de d3train)^2$$

```
svd3=svd(d3train)$d
eigen3=eigen(cov_3)$values
```

La relation est bien vérifiée:

```r
norm(as.matrix(svd3^2-eigen3*999))
```

```
## [1] 75368.43
```

```r
norm(as.matrix(svd3^2-eigen3*999),"2")
```

```
## [1] 55241.2
```

Nous obtenons un résultat à 10^-10, la relation est vérifiée à l'erreur numérique près.

Pour 8 on obtient :

```r
svd8=svd(d8train)$d
eigen8=eigen(cov_8)$values
```

```r
norm(as.matrix(svd8^2-eigen8*999))
```

```
## [1] 3.972884e-10
```

```r
norm(as.matrix(svd8^2-eigen8*999),"2")
```

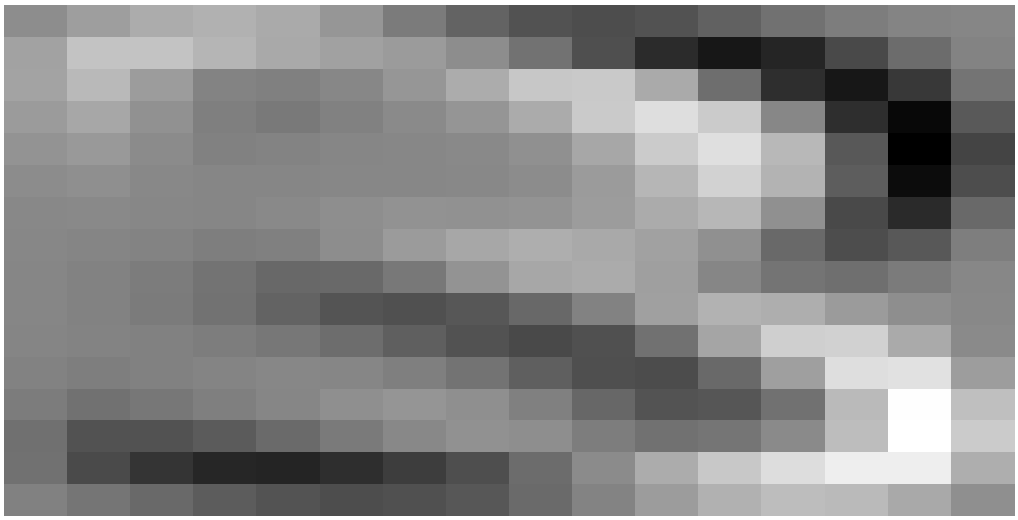
```
## [1] 1.266786e-10
```

Nous obtenons un résultat à 10^-10, la relation est vérifiée à l'erreur numérique près.

6. Modes propres

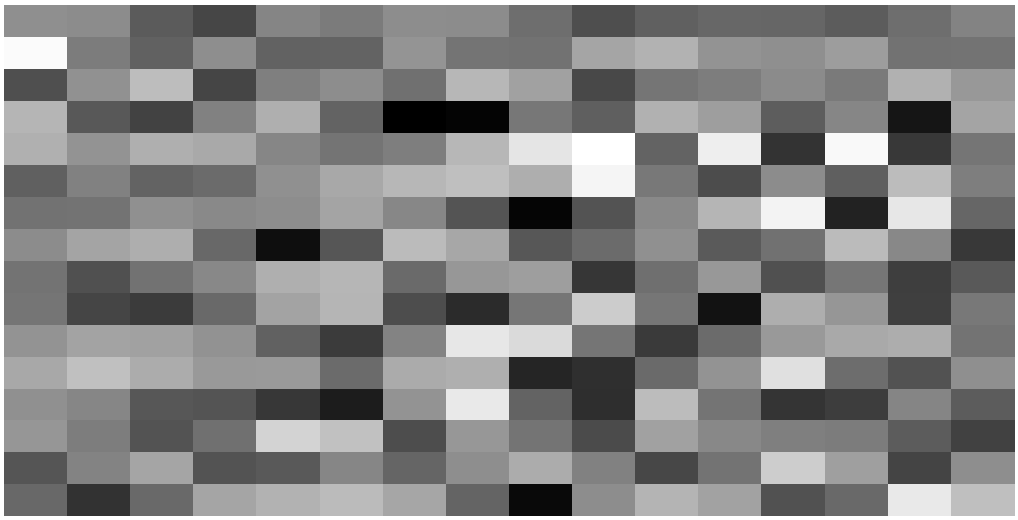
```r
eigen3_mp=eigen(cov_3)$vectors
eigen8_mp=eigen(cov_8)$vectors
```

```r
mImage(eigen3_mp[,1])
```

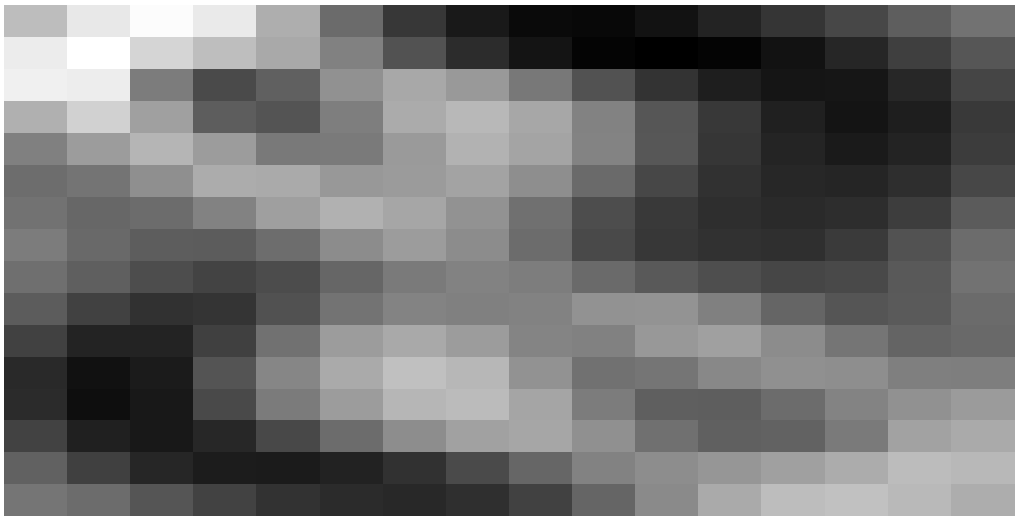


```
mImage(eigen3_mp[,10])
```

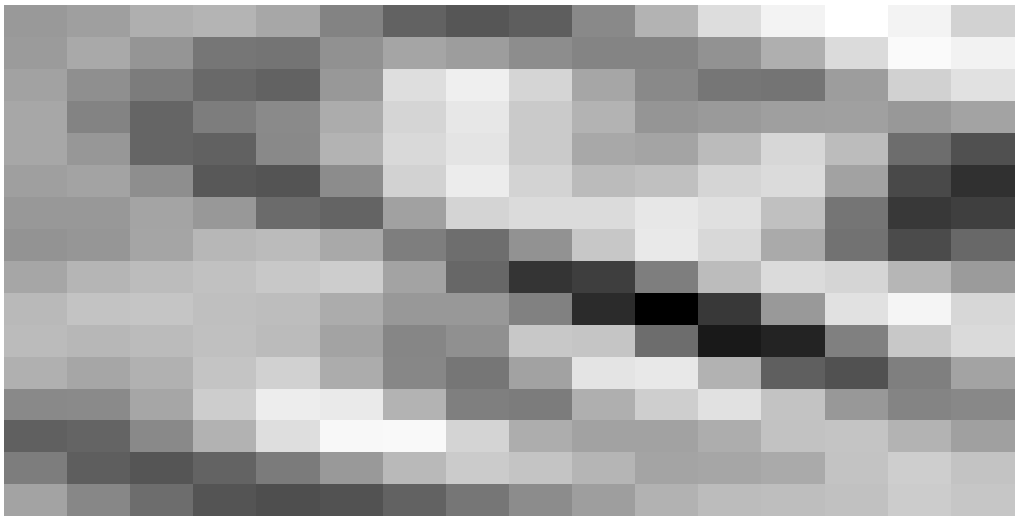
```
mImage(eigen3_mp[,100])
```



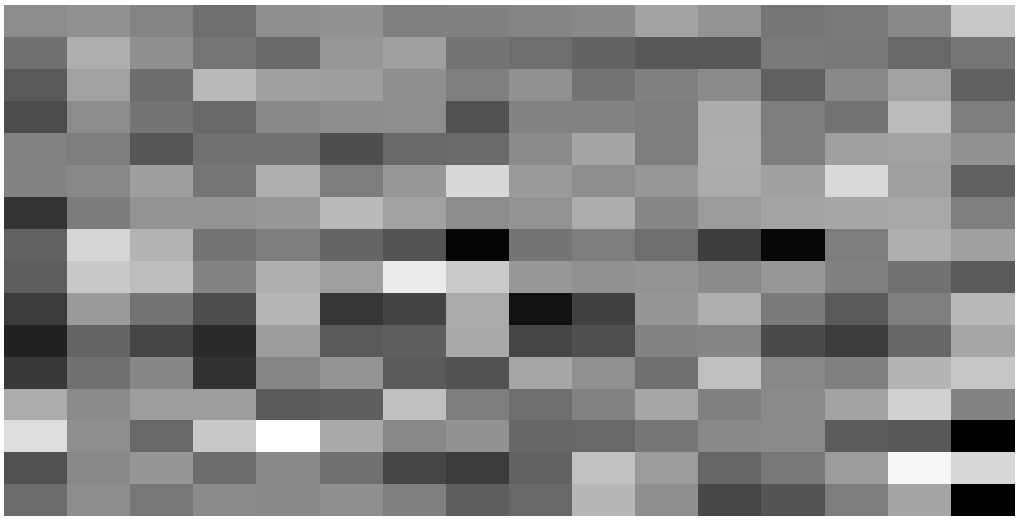
```
mImage(eigen8_mp[,1])
```



```
mImage(eigen8_mp[,10])
```



```
mImage(eigen8_mp[,100])
```



On reconnait les formes d'un 3 et d'un 8. Mais cela s'estompe selon la décroissance du mode propres.

7. Matrice de projection

Calculons la matrice de projection sur le sous espace vectoriel engendré par les 5 premières composantes.

```
projection3 = rbind(t(eigen3_mp[,1:5]),matrix(0,nr = 251, ncol = 256));
max(abs(projection3%*%projection3 - projection3))
```

```
## [1] 0.2585171
```

Il s'agit bien de la matrice de projection de 3.

```
projection8 = rbind(t(eigen8_mp[,1:5]),matrix(0,nr = 251, ncol = 256));
max(abs(projection8%*%projection8 - projection8))
```

```
## [1] 0.1951713
```

Il s'agit bien de la matrice de projection de 8.

8. Reconstruction des images

Une méthode de reconstruction permet à l'aide des coordonnées sur les 5 composantes principales seulement de reconstruire des images qui seront de moins bonnes qualité que les images originales mais qui nécessiteront

moins d'informations pour leur stockage. Etapes:
1) Normalisation en utilisant les matrices de projection dans la base des vecteurs propres.
2) Stockage
3) Reconstruction avec la matrice inverse

Avec cette méthode nous obtenons un gain de 256x256 à 5x256 valeurs.