NIPALS

Cristian Preda, Vincent Vandewalle, Quentin Grimonprez

2/5/2019

Simulation des données

```
library(MASS)
set.seed(1234)
mu <- c(1, 2, 4, 3)
n <- 100
p <- 4
sigma <- matrix(
  c(
    0.7, 0, 1.3, 0.5,
    0, 1.2, -0.3, -0.1,
    1.3, -0.3, 3.1, 1.3,
    0.5, -0.1, 1.3, 0.6
  ),
  nrow = p, ncol = p
)

X <- mvrnorm(n, mu, sigma)

cat("Quelques observations des données simulées:")
```

```
Quelques observations des données simulées:
```

```
head(X)
```

```
          [,1]      [,2]     [,3]     [,4]
[1,] 2.0957051 2.1947399 6.153399 3.736707
[2,] 0.8964736 1.5269139 3.523403 2.520930
[3,] 0.2470616 2.2806159 2.099409 2.143699
[4,] 2.9775327 0.9590023 7.978456 4.696517
[5,] 0.6589963 1.1838370 3.190021 2.593942
[6,] 0.8601072 2.2579124 3.120676 2.401118
```

On vérifie que les données sont “bien” simulées :

```
cat("Les moyennes :", round(apply(X, 2, mean), 2), "\n")
```

```
Les moyennes : 1.17 2.01 4.27 3.09
```

```
cat("La matrice de variance-covariance :\n")
```

```
La matrice de variance-covariance :
```

```
print(round(cov(X), 2))
```

```
     [,1]  [,2]  [,3]  [,4]
```

```
[1,] 0.67  0.08  1.27  0.49
[2,] 0.08  1.25 -0.27 -0.07
[3,] 1.27 -0.27  3.19  1.32
[4,] 0.49 -0.07  1.32  0.60
```

On peut également calculer la moyenne par colonne avec

```
colMeans(X)
```

```
[1] 1.174801 2.007708 4.268150 3.085303
```

Avec mvtnorm :

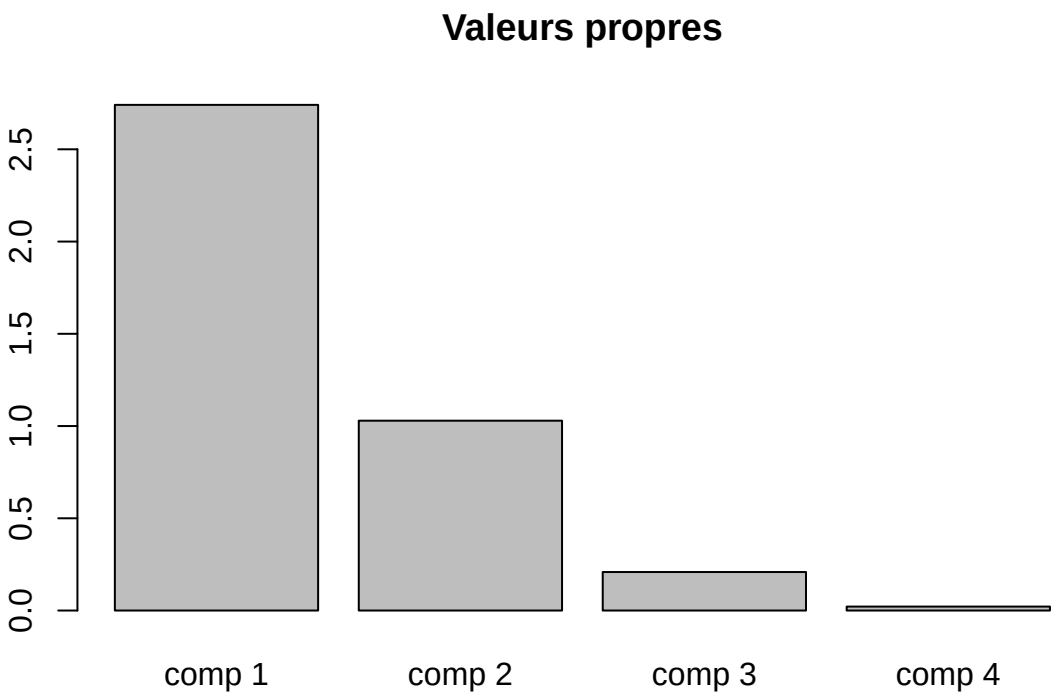
```
library(mvtnorm)
X <- rmvnorm(n, mu, sigma)
head(X)
```

ACP normée sur les données simulées.

```
library(FactoMineR)
acp <- PCA(X, scale.unit = TRUE, graph = FALSE)
# valeurs propres
print(acp$eig)
```

```
       eigenvalue percentage of variance cumulative percentage of variance
comp 1 2.74085999             68.5214999                           68.52150
comp 2 1.02895575             25.7238937                           94.24539
comp 3 0.20873995              5.2184987                           99.46389
comp 4 0.02144431              0.5361078                          100.00000
```

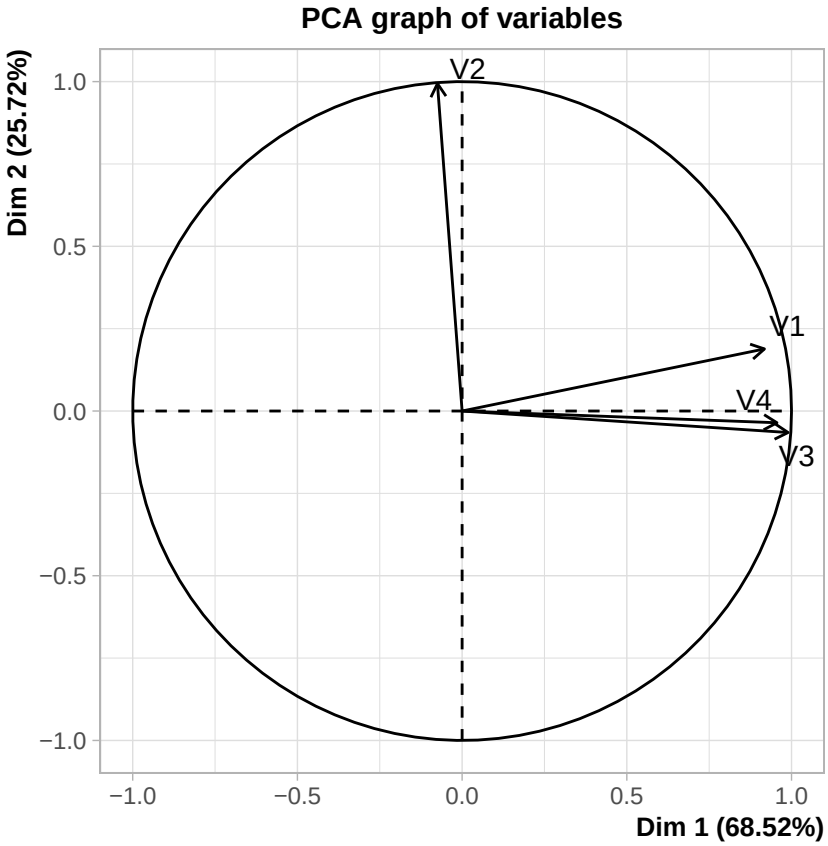
```
# graphe des valeurs propres
barplot(acp$eig[, 1], main = "Valeurs propres")
```




```
# facteurs principaux
print(acp$svd$V)
```

```
           [,1]        [,2]       [,3]        [,4]
[1,]  0.55440697  0.18578678 -0.7590821 -0.28620007
[2,] -0.04520317  0.97985412  0.1740390  0.08690798
[3,]  0.59771542 -0.06432375  0.1231065  0.78958439
[4,]  0.57734380 -0.03509465  0.6151012 -0.53581061
```

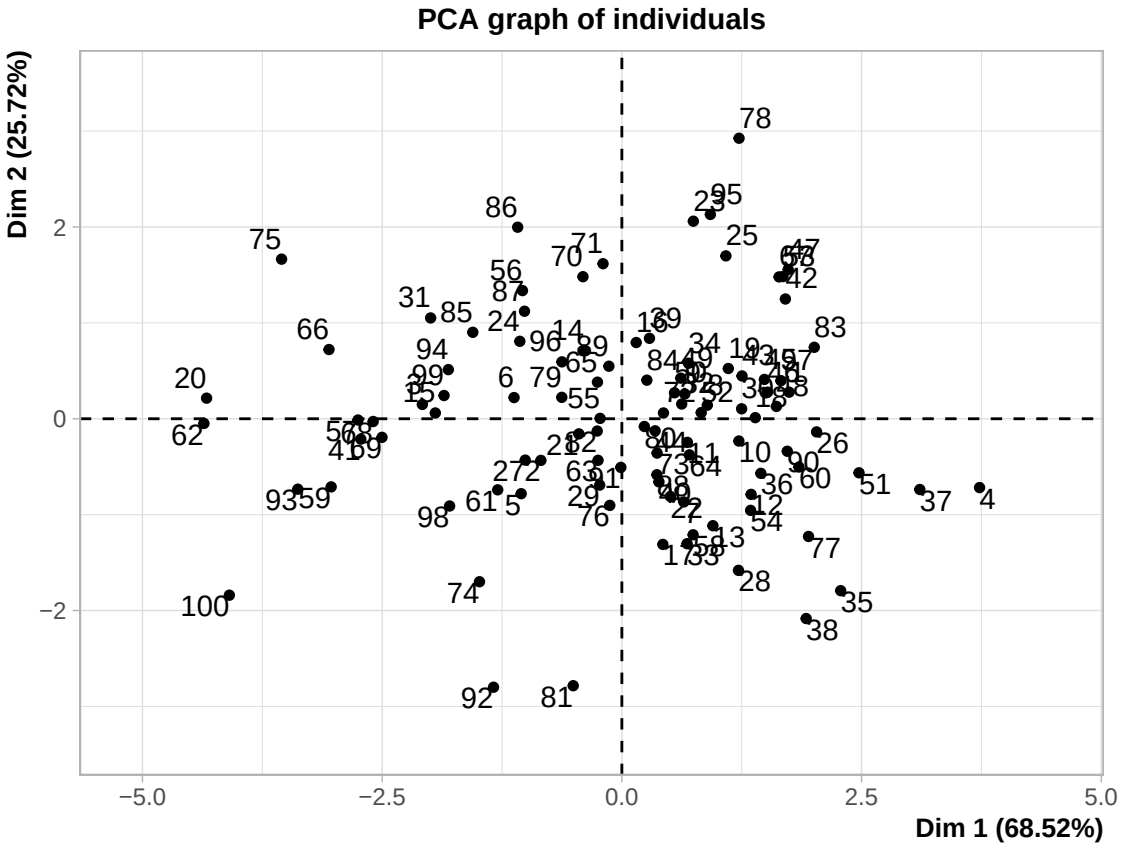
```
# plan des variables
plot(acp, choix = "var", axes = c(1, 2))
```




```
# composantes principales : les 6 premiers individus
print(head(acp$ind$coord))
```

```
      Dim.1      Dim.2       Dim.3       Dim.4
1  1.744297  0.2774305 -0.180939655  0.07417452
2 -0.844467 -0.4340644 -0.317415690  0.12253830
3 -2.080672  0.1491917  0.007667117  0.04016068
4  3.730728 -0.7178369 -0.305401251 -0.19033768
5 -1.050154 -0.7815551 -0.113669128 -0.01946425
6 -1.124327  0.2208457 -0.292807030  0.09684624
```

```
# graphe des individus
plot(acp, choix = "ind", axes = c(1, 2))
```



NIPALS sans traitement des données manquantes

```
NIPALS <- function(X, h = 2, iter = 100) {
  # renvoie les composantes principales (CP), les facteurs principaux (FP) ET
  # les données reconstituées avec h composantes (Xrec)
  n <- nrow(X)
  p <- ncol(X)
  # centrer et réduire matrice X
  m <- apply(X, 2, mean) # calcul des moyennes # m <- colMeans(X)
  s <- apply(X, 2, sd) * sqrt((n - 1) / n) # calcul des écart-types
  Xr <- (X - rep(1, n) %*% t(m)) / (rep(1, n) %*% t(s))
  # autre manière: Xr <- scale(X, center = m, scale = s)

  # on reserve la place pour:
  CP <- matrix(0, nrow = n, ncol = h) # les composantes principales
  FP <- matrix(0, ncol = h, nrow = p) # les facteurs principaux
  Xrec <- matrix(0, nrow = n, ncol = p) # les données reconstituées

  # deroulement de l'algorithme:
  for (i in 1:h) {
    # voir pages 30-32 du cours
    r <- calcul_cp_fp(Xr, iter) # fonction calculant 1ere CP et 1er FP
    CP[, i] <- r$cp
    FP[, i] <- r$fp
    Xr <- Xr - (r$cp) %*% t(r$fp)
  }
```

```
  # Reconstitution des données avec h composantes
  Xrec <- CP %*% t(FP)
  Xrec <- Xrec * (rep(1, n) %*% t(s)) + rep(1, n) %*% t(m)
  return(list(CP = CP, FP = FP, rec = Xrec))
}

# la fonction qui calcule CP_1 et FP_1
calcul_cp_fp <- function(X, iter) {
  cp <- X[, 1]
  for (i in 1:iter) {
    fp <- t(X) %*% cp
    # on normalize fp:
    fp <- fp / sqrt(sum(fp^2))
    cp <- X %*% fp
  }
  return(list(cp = cp, fp = fp))
}
```

Application de NIPALS aux données simulées

Voici ce qu'on obtient avec h=4 composantes. À comparer avec ce qui est donné par FactoMineR dans l'objet 'acp'.

```
res <- NIPALS(X, h = ncol(X))
cat("Les facteurs principaux :\n")
```

```
Les facteurs principaux :
```

```
cat("  NIPALS :\n")
```

```
  NIPALS :
```

```
print(res$FP)
```

```
            [,1]        [,2]       [,3]        [,4]
[1,]  0.55440697  0.18578678  0.7590821  0.28620007
[2,] -0.04520317  0.97985412 -0.1740390 -0.08690798
[3,]  0.59771542 -0.06432375 -0.1231065 -0.78958439
[4,]  0.57734380 -0.03509465 -0.6151012  0.53581061
```

```
cat("  FactoMineR :\n")
```

```
  FactoMineR :
```

```
print(acp$svd$V)
```

```
            [,1]        [,2]       [,3]        [,4]
[1,]  0.55440697  0.18578678 -0.7590821 -0.28620007
[2,] -0.04520317  0.97985412  0.1740390  0.08690798
[3,]  0.59771542 -0.06432375  0.1231065  0.78958439
[4,]  0.57734380 -0.03509465  0.6151012 -0.53581061
```

```
cat("\nLes composantes principales :\n")
```

```
Les composantes principales :
```

```
cat("  NIPALS :\n")
```

```
  NIPALS :
```

```
print(head(res$CP))
```

```
          [,1]       [,2]         [,3]        [,4]
[1,]  1.744297  0.2774305  0.180939655 -0.07417452
[2,] -0.844467 -0.4340644  0.317415690 -0.12253830
[3,] -2.080672  0.1491917 -0.007667117 -0.04016068
[4,]  3.730728 -0.7178369  0.305401251  0.19033768
[5,] -1.050154 -0.7815551  0.113669128  0.01946425
[6,] -1.124327  0.2208457  0.292807030 -0.09684624
```

```
cat("  FactoMineR :\n")
```

```
  FactoMineR :
```

```
print(head(acp$ind$coord))
```

```
      Dim.1      Dim.2        Dim.3       Dim.4
1  1.744297  0.2774305 -0.180939655  0.07417452
2 -0.844467 -0.4340644 -0.317415690  0.12253830
3 -2.080672  0.1491917  0.007667117  0.04016068
4  3.730728 -0.7178369 -0.305401251 -0.19033768
5 -1.050154 -0.7815551 -0.113669128 -0.01946425
6 -1.124327  0.2208457 -0.292807030  0.09684624
```

La reconstitution complète des données (toutes les composantes principales) La reconstitution des données avec toutes les composantes principales.

```
res <- NIPALS(X, h = ncol(X))
print(head(res$rec))
```

```
          [,1]      [,2]     [,3]     [,4]
[1,] 2.0957051 2.1947399 6.153399 3.736707
[2,] 0.8964736 1.5269139 3.523403 2.520930
[3,] 0.2470616 2.2806159 2.099409 2.143699
[4,] 2.9775327 0.9590023 7.978456 4.696517
[5,] 0.6589963 1.1838370 3.190021 2.593942
[6,] 0.8601072 2.2579124 3.120676 2.401118
```

```
print(head(X))
```

```
          [,1]      [,2]     [,3]     [,4]
[1,] 2.0957051 2.1947399 6.153399 3.736707
[2,] 0.8964736 1.5269139 3.523403 2.520930
[3,] 0.2470616 2.2806159 2.099409 2.143699
[4,] 2.9775327 0.9590023 7.978456 4.696517
[5,] 0.6589963 1.1838370 3.190021 2.593942
[6,] 0.8601072 2.2579124 3.120676 2.401118
```

Approximation des données avec quelques composantes (ici h=2) Voici la reconstitution des données avec juste deux composantes :

```
res <- NIPALS(X, h = 2)
cat("Données reconstituées :\n")
```

```
Données reconstituées :
```

```
print(head(res$rec))
```

```
          [,1]     [,2]     [,3]     [,4]
[1,] 2.0014656 2.222630 6.088915 3.852965
[2,] 0.7293914 1.576574 3.420921 2.721751
[3,] 0.2611132 2.275243 2.041389 2.156632
[4,] 2.7451794 1.036614 8.312288 4.762611
[5,] 0.5844492 1.207751 3.242192 2.639732
[6,] 0.7022176 2.305289 3.048855 2.579692
```

```
cat("Données :\n")
```

```
Données :
```

```
print(head(X))
```

```
          [,1]      [,2]     [,3]     [,4]
[1,] 2.0957051 2.1947399 6.153399 3.736707
[2,] 0.8964736 1.5269139 3.523403 2.520930
[3,] 0.2470616 2.2806159 2.099409 2.143699
[4,] 2.9775327 0.9590023 7.978456 4.696517
[5,] 0.6589963 1.1838370 3.190021 2.593942
[6,] 0.8601072 2.2579124 3.120676 2.401118
```

NIPALS avec données manquantes.

L'algorithme précédent est adapté aux cas où il y a données manquantes. Les points à modifier sont au niveau du :

- calcul des moyennes (m) et écart-types (s)
- calcul des composantes et facteurs dans la fonction calcul_cp_fp

On ré-écrit donc ces fonctions en les renommant : *NIPALS_dm* et *calcul_cp_fp_dm*

```
NIPALS_dm <- function(X, h = 2, iter = 100) {
  # renvoie les composantes principales (CP), les facteurs principaux (FP) et
  # les données reconstituées avec h composantes (Xrec)
  n <- nrow(X)
  p <- ncol(X)

  # centrer et réduire matrice X
  # calcul des moyennes
  m <- apply(X, 2, mean, na.rm = TRUE) # autre manière: m <- colMeans(X, na.rm = TRUE)
  # calcul des écart-types
  s <- apply(X, 2, sd, na.rm = TRUE) * sqrt((n - 1) / n)
  Xr <- (X - rep(1, n) %*% t(m)) / (rep(1, n) %*% t(s))
  # autre manière: Xr <- scale(X, center = m, scale = s)

  # on réserve la place pour:
  CP <- matrix(0, nrow = n, ncol = h) # les composantes principales
  FP <- matrix(0, ncol = h, nrow = p) # les facteurs principaux
  Xrec <- matrix(0, nrow = n, ncol = p) # les données reconstituées

  # déroulement de l'algorithme:
  for (i in 1:h) {
    # voir pages 30-32 du cours
    r <- calcul_cp_fp_dm(Xr, iter) # fonction calculant 1ere CP et 1er FP
    CP[, i] <- r$cp
    FP[, i] <- r$fp
```

```
    Xr <- Xr - (r$cp) %*% t(r$fp)
  }

  # Reconstitution des données avec h composantes
  Xrec <- CP %*% t(FP)
  Xrec <- Xrec * (rep(1, n) %*% t(s)) + rep(1, n) %*% t(m)

  return(list(CP = CP, FP = FP, rec = Xrec))
}

# la fonction qui calcule CP_1 et FP_1 avec données manquantes
calcul_cp_fp_dm <- function(X, iter) {
  cp <- X[, 1]
  fp <- rep(0, ncol(X))
  for (i in 1:iter) {
    for (j in 1:ncol(X)) {
      fp[j] <- sum(X[, j] * cp, na.rm = TRUE)
    }
    # autre manière qui évite les boucles
    # manière 1: fp = apply(X, 2, function(x) sum(x * cp, na.rm = TRUE))
    # manière 2: fp = colSums(X * cp, na.rm = TRUE)

    # on normalize fp
    fp <- fp / sqrt(sum(fp^2))
    cp <- apply(X, 1, function(x) sum(x * fp, na.rm = TRUE)) # vectoriser le calcul
  }

  return(list(cp = cp, fp = fp))
}
```

On vérifie que la version modifiée *NIPALS_dm* donne les mêmes résultats que *NIPALS* lorsqu'il n'y a pas de données manquantes.

```
res_dm <- NIPALS_dm(X, h = ncol(X))
cat("Les facteurs principaux :\n")
```

```
Les facteurs principaux :
```

```
print(res_dm$FP)
```

```
            [,1]        [,2]       [,3]        [,4]
[1,]  0.55440697  0.18578678  0.7590821  0.28620007
[2,] -0.04520317  0.97985412 -0.1740390 -0.08690798
[3,]  0.59771542 -0.06432375 -0.1231065 -0.78958439
[4,]  0.57734380 -0.03509465 -0.6151012  0.53581061
```

```
cat("Les composantes principales :\n")
```

```
Les composantes principales :
```

```
print(head(res_dm$CP))
```

```
          [,1]       [,2]         [,3]        [,4]
[1,]  1.744297  0.2774305  0.180939655 -0.07417452
[2,] -0.844467 -0.4340644  0.317415690 -0.12253830
[3,] -2.080672  0.1491917 -0.007667117 -0.04016068
[4,]  3.730728 -0.7178369  0.305401251  0.19033768
```

```
[5,] -1.050154 -0.7815551  0.113669128  0.01946425
[6,] -1.124327  0.2208457  0.292807030 -0.09684624
```

```
cat("Les données reconstituées :\n")
```

```
Les données reconstituées :
```

```
print(head(res_dm$rec))
```

```
          [,1]      [,2]     [,3]     [,4]
[1,] 2.0957051 2.1947399 6.153399 3.736707
[2,] 0.8964736 1.5269139 3.523403 2.520930
[3,] 0.2470616 2.2806159 2.099409 2.143699
[4,] 2.9775327 0.9590023 7.978456 4.696517
[5,] 0.6589963 1.1838370 3.190021 2.593942
[6,] 0.8601072 2.2579124 3.120676 2.401118
```

Parfait!

Simulation des données manquantes sur la matrice X

```
# pourcentage des données manqunates
pm <- 0.1
# génération des valeurs manquantes
# on se rappelle n=100, p=4.
Xm <- X
# indices des valeurs manquantes
im <- which(runif(n * p) < pm)
Xm[im] <- NA
summary(Xm)
```

```
       V1                V2                V3                V4
 Min.   :-1.0321   Min.   :-0.9992   Min.   :-0.6178   Min.   :1.012
 1st Qu.: 0.6905   1st Qu.: 1.3269   1st Qu.: 3.1553   1st Qu.:2.567
 Median : 1.3561   Median : 2.0095   Median : 4.5705   Median :3.181
 Mean   : 1.1802   Mean   : 1.9634   Mean   : 4.2530   Mean   :3.077
 3rd Qu.: 1.7606   3rd Qu.: 2.5756   3rd Qu.: 5.5921   3rd Qu.:3.591
 Max.   : 2.9775   Max.   : 4.2508   Max.   : 8.0416   Max.   :5.206
 NA's   :11        NA's   :11        NA's   :9         NA's   :9
```

```
cat("Voici les valeurs qui ont été déclarées manquantes :\n")
```

```
Voici les valeurs qui ont été déclarées manquantes :
```

```
print(X[im])
```

```
 [1]  0.6589963  1.5539468  2.1682722  2.4873297 -0.9565933  1.8670051
 [7]  1.4618377  0.5531494  1.9216448  1.5807108 -0.8514763  0.5999619
[13]  0.5660848  1.2227212  2.4360024  2.2707255  2.0012074  3.6023816
[19]  3.8534424  5.1037784  1.8926856  2.4823749  4.8672514  4.7933822
[25]  4.9166337  4.8056831  6.3678514  4.5616907  2.5293602  4.6772452
[31]  2.2737444  2.1057144  3.9212124  4.1683168  3.2185435  3.7980385
[37]  3.5619919  2.7768853  1.7167419  3.2897089
```

```
print(im)
```

```
 [1]   5  11  42  47  62  72  73  74  90  97 100 113 117 140 149 150 157 167 171
[20] 178 180 197 217 232 252 272 277 280 285 288 299 327 336 338 339 342 348 356
```

```
[39] 357 389
```

```
print(Xm)
```

```
             [,1]       [,2]       [,3]     [,4]
  [1,]  2.09570509  2.1947399  6.1533987 3.736707
  [2,]  0.89647357  1.5269139  3.5234032 2.520930
  [3,]  0.24706164  2.2806159  2.0994089 2.143699
  [4,]  2.97753268  0.9590023  7.9784558 4.696517
  [5,]          NA  1.1838370  3.1900214 2.593942
  [6,]  0.86010719  2.2579124  3.1206759 2.401118
  [7,]  1.88019022  0.8575888  4.8965106 2.961106
  [8,]  1.89994217  2.0164588  4.7718952 3.387853
  [9,]  1.46167003  2.2734029  5.0396001 3.367350
 [10,]  1.33432042  1.8017967  5.5721775 3.970892
 [11,]          NA  1.6652321  4.6893127 3.443804
 [12,]  1.54492043  1.1140984  5.7178470 3.846842
 [13,]  1.88820906         NA  5.2571405 3.181547
 [14,]  0.54223109  2.9871444  4.0260130 3.298995
 [15,]  0.11604643  2.2341181  2.3792647 2.315176
 [16,]  1.41964787  2.8459038  4.1062151 3.180198
 [17,]  1.13770920         NA         NA 3.342461
 [18,]  1.59845048  2.0094799  5.5253116 3.990979
 [19,]  1.59521419  2.5755878  5.5622388 3.633164
 [20,] -0.64589042  2.4328521 -0.2354642 1.012243
 [21,]  1.17670678  1.7859511  3.6891002 2.733514
 [22,]  1.47200687  1.0362344  4.7863950 3.204381
 [23,]  2.36601797  4.0492694  4.7570332 2.885475
 [24,]  1.06533380  2.8704226  3.2736022 2.259060
 [25,]  1.89184862  3.8158314  5.2748846 3.525295
 [26,]  1.83776630  1.8305908  6.6456156 4.113257
 [27,]  1.50796623  1.3156008  2.6961492       NA
 [28,]  1.71092454  0.1517234  5.7676546 3.445752
 [29,]  0.75426640  1.3269040  3.9549842 3.263639
 [30,]  1.76056476  2.0574901  5.6932101 3.580797
 [31,]  0.23386649  3.3190304  2.1027079 2.325628
 [32,]  1.35614693  2.1797118         NA 3.526137
 [33,]  1.01592856  0.6371222  5.2967936 3.601369
 [34,]  1.54788687  2.6112195  4.8581910 3.439753
 [35,]  2.07632254 -0.1103114  6.7808429 4.065654
 [36,]  1.39682127  1.4254483  6.1927782       NA
 [37,]  1.49800280  1.3503015  8.0415601 5.206119
 [38,]  1.72434914 -0.3690155  6.1647522       NA
 [39,]  1.35622678  2.9332771  4.5704537       NA
 [40,]  1.40606735         NA  4.8193384 3.100714
 [41,] -0.12167064  1.9256975  1.4393117 1.900490
 [42,]          NA  3.2721338  5.8845332       NA
 [43,]  1.42841202  2.5478637  5.5239554 3.991282
 [44,]  1.49782171  1.7905834  4.3547465 3.203086
 [45,]  2.06843060  2.3286741  5.7159433 3.621370
 [46,]  1.93873955  2.2178541  5.6119873 3.823101
 [47,]          NA  3.5279967  5.9899678 3.512743
 [48,]  1.97852951  2.0561240  6.3693713       NA
 [49,]  1.52948215         NA  4.9619377 3.293774
 [50,]  1.48405900         NA  4.9111193 3.261809
```

```
 [51,]  2.30890231  1.2416972  7.3114902 3.943412
 [52,]  1.83704370  1.9452637         NA 3.291350
 [53,]  2.18275551  3.5354591  6.0499639 3.690866
 [54,]  1.83785154  0.8337745  5.7231631 3.555836
 [55,]  0.66933356  2.1559718  4.4854335 3.153717
 [56,]  0.69051136  3.5867321  3.0478620       NA
 [57,]  0.32970149         NA  0.8542290       NA
 [58,]  0.96409285  0.7662628  5.4603127 3.666158
 [59,] -0.25672269  1.3729984  1.1341768 1.718492
 [60,]  2.05928437  1.3173680  5.8408558 4.000181
 [61,]  0.75916139  1.1727355  2.6767227 2.406026
 [62,]          NA  2.2191345 -0.6178101 1.417740
 [63,]  1.01636780  1.5410720  4.0342519 2.978071
 [64,]  0.87990235  1.7370906  5.2915123 3.819273
 [65,]  1.14956111  2.4290069  4.0489276 2.890014
 [66,] -0.37799886  3.0357264  0.8329142 2.023512
 [67,]  1.95186393         NA  6.1863242 3.789582
 [68,] -0.41408705  2.2380581  1.6438380 2.263671
 [69,]  0.07913449  1.9040065  1.6296048 1.925464
 [70,]  1.32153564  3.6046576  3.2525443 2.952726
 [71,]  1.11363248         NA  4.1256012 3.043173
 [72,]          NA  1.9028670         NA 2.786756
 [73,]          NA  1.5421984  4.5700351 3.153223
 [74,]          NA  0.1315533  2.7283120 2.260832
 [75,]  0.29012393  3.8614086  0.3578396 1.013521
 [76,]  0.78069996  1.0906787  4.2650780 3.228242
 [77,]  2.12813589  0.4872301         NA 3.788593
 [78,]  2.28816695         NA  5.3898212 3.365282
 [79,]  0.86864459  2.2997266  3.5172453 2.881602
 [80,]  1.31970996         NA         NA 3.127860
 [81,]  0.27971760 -0.9223165  4.4050308 3.004132
 [82,]  1.11384636  1.8663332  4.4721301 2.699195
 [83,]  1.90945881  2.8105829  6.5416000 4.115470
 [84,]  1.54822173  2.3674007  4.1581627 3.161666
 [85,]  0.27782814  3.1729051         NA 2.672393
 [86,]  0.95124928  4.2508310  2.7420120 2.646007
 [87,]  0.53849944  3.3898822  3.0586226 2.928132
 [88,]  1.16273345  1.3786289         NA 3.366234
 [89,]  0.64486714  2.7919634  4.5810625       NA
 [90,]          NA  1.5431180  6.0668074 3.873282
 [91,]  1.38035170  1.3586095  3.8845037 3.021079
 [92,]  0.24226002 -0.9992476  3.6859554 2.248732
 [93,] -1.03213636  1.5604124  1.1407601 1.967958
 [94,]  0.17953489  2.7362326  2.2131857 2.539924
 [95,]  2.27892741  4.1698678  4.8163665 3.181205
 [96,]  1.45627526  2.5350218  3.2210412 2.493763
 [97,]          NA         NA  6.0577315 4.150167
 [98,]  0.10544714  1.1424466  2.3972861 2.456678
 [99,]  0.41254042  2.3545550         NA 2.220075
[100,]          NA  0.1791382  0.2372655 1.182134
```

Imputation des valeurs manquantes avec NIPALS :

```
imp_nip <- NIPALS_dm(Xm, h = 4) # par défaut, on utilise donc juste deux comp (h=2)
cat("Voici les valeurs estimées par NIPALS pour les données manquantes :\n")
```

```
Voici les valeurs estimées par NIPALS pour les données manquantes :
```

```
print(imp_nip$rec[im])
```

```
 [1] 0.9150562 1.2655608 1.4314306 1.5974282 0.2685685 1.1654394 1.2101373
 [8] 0.7413522 1.5162153 1.5462975 0.2469707 1.8580783 1.9559459 1.9147179
[15] 1.8895679 1.8967408 2.1159183 1.7653483 1.9765384 1.8197410 1.9561402
[22] 1.8644776 4.5270528 4.7939709 4.7213307 3.9893524 5.4640141 4.3588392
[29] 3.4128527 4.6024221 3.0174737 2.7127222 3.4958667 3.4540665 3.1595830
[36] 3.3026905 3.5338922 2.8440868 2.3498870 3.1737683
```

Cela a l'air pas mal!