Projet STATIS: Réponse feuille de route

I-Préliminaires pour la situation 1

1.1 Produit scalaire

La matrice diagonale des poids des n individus est W (par défaut, $W = \frac{1}{n}I_n$). On considérera également une matrice diagonale des poids des p colonnes : $C = \frac{1}{p}I_p$ par défaut.

1. Le produit scalaire entre deux matrices A et B de taille (n, p) est :

$$[A|B] = \mathrm{tr}(CA'WB)$$

La norme d'une matrice A correspondant à ce produit scalaire sera notée $[|A|]$.

a) Ecrivons le produit scalaire sous forme $tr(\tilde{A}'\tilde{B})$ (produit scalaire de Frobénius) en explicitant la transformation $Z \quad en \quad \tilde{Z}, \forall Z\ (n, p)$.

Soit $A, B \in M_{n,p}(\mathbb{R})$

$$\begin{aligned}
[A \mid B] &= \mathrm{tr}(CA'WB) \\
&= \mathrm{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}BC^{\frac{1}{2}}) \\
&= \mathrm{tr}((W^{\frac{1}{2}}AC^{\frac{1}{2}})'W^{\frac{1}{2}}BC^{\frac{1}{2}}) \\
&= \mathrm{tr}(\tilde{A}'\tilde{B})
\end{aligned}$$

où $\forall(n, p)\ \tilde{Z} = W^{\frac{1}{2}}ZC^{\frac{1}{2}}$

b) Pour montrer que $[A \mid B] = \mathrm{tr}(\tilde{A}'\tilde{B})$ est un produit scalaire, nous devons démontrer les propriétés suivantes :

(i) Symétrie : $\forall\ A, B \in M_{n,p}(\mathbb{R})$

$$[A \mid B] = [B \mid A]$$

(ii) Linéarité :$\forall\ A, B_1, B_2 \in M_{n,p}(\mathbb{R})$ et $\alpha, \beta in \mathbb{R}$

$$[A \mid (\alpha B_1 + \beta B_2)] = \alpha[A \mid B_1] + \beta[A \mid B_2]$$

(iii) Positivité définie :$\forall\ A \in M_{n,p}(\mathbb{R})$

$$[A \mid A] \geq 0 \quad \text{et} \quad [A \mid A] = 0 \text{ si et seulement si } A = 0.$$

Vérifions ces propriétés :

(i) Soit $A, B \in M_{n,p}(\mathbb{R})$

$$\begin{aligned}[A \mid B] &= \text{tr}(\tilde{A}'\tilde{B}) \\ &= \text{tr}((W^{\frac{1}{2}}AC^{\frac{1}{2}})'W^{\frac{1}{2}}BC^{\frac{1}{2}}) \\ &= \text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}BC^{\frac{1}{2}})\end{aligned}$$

Or, par la propriété de la trace, $\text{tr}(AB) = \text{tr}(BA)$ et l'invariance par transposition des matrices de poids $\forall k \in \mathbb{R}, \quad W^{k'} = W^k$ et $C^{k'} = C^k$, on a alors:

$$\begin{aligned}\text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}BC^{\frac{1}{2}}) &= \text{tr}(C^{\frac{1}{2}}B'W^{\frac{1}{2}}W^{\frac{1}{2}}AC^{\frac{1}{2}}) \\ &= [B \mid A]\end{aligned}$$

(ii) Soit $A, B_1, B_2 \in M_{n,p}(\mathbb{R})$ et $\alpha, \beta \in \mathbb{R}$

$$\begin{aligned}[A \mid (\alpha B_1 + \beta B_2)] &= \text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}(\alpha B_1 + \beta B_2)C^{\frac{1}{2}}) \\ &= \alpha\text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}B_1C^{\frac{1}{2}}) + \beta\text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}B_2C^{\frac{1}{2}}) \\ &= \alpha\text{tr}(\tilde{A}'\tilde{B}_1) + \beta\text{tr}(\tilde{A}'\tilde{B}_2) \\ &= \alpha[A \mid B_1] + \beta[A \mid B_2]\end{aligned}$$

La linéarité de la trace assure que cette propriété est respectée.

(iii) Soit $A \in M_{n,p}(\mathbb{R})$

$$[A \mid A] = \text{tr}(C^{\frac{1}{2}}A'W^{\frac{1}{2}}W^{\frac{1}{2}}AC^{\frac{1}{2}}) \geq 0$$

On suppose que :

$$\begin{aligned}&[A \mid A] = 0 \\ \iff\ & tr(\tilde{A}'\tilde{A}) = 0 \\ \iff\ & \tilde{A}'\tilde{A} = 0 \\ \iff\ & \tilde{A} = 0 \\ \iff\ & A = 0\end{aligned}$$

(car W et C ne sont pas nulle)

Par conséquent, la quantité $[A \mid B] = \text{tr}(\tilde{A}'\tilde{B})$ est un produit scalaire, car elle satisfait toutes les propriétés requises.

c) Écrivons le produit scalaire précédent sous forme de double somme. On note $\tilde{B} = (\tilde{b}_{ij})$, d'où $[\tilde{A}\tilde{B}] = \sum_{k=1}^{n} \tilde{a}_{ik}\tilde{b}_{kj}$ et $[\tilde{A}'\tilde{B}] = \sum_{k=1}^{n} \tilde{a}_{ki}\tilde{b}_{kj}$. On en déduit alors:

$$\begin{aligned}[A \mid A] &= tr(\tilde{A}'\tilde{B}) \\ &= \sum_{l=1}^{p}\sum_{k=1}^{n} \tilde{a}_{kl}\tilde{b}_{kl}\end{aligned}$$

d) Nous pouvons observer le programme du précédent produit scalaire ci-dessous, ainsi que la norme associée à ce produit scalaire.

La fonction calculant le produit scalaire de Frobénius

```
prd_scalaire = function(A,B){
  n = length(A[,1]); p = length(A[1,])
  # Les matrices de ponderation
  W = (1/n)*diag(n); C = (1/p)*diag(p)

  # Calcul des matrices A_tild et B_tild
  A_tild = sqrt(W) %*% A %*% sqrt(C)
  B_tild = sqrt(W) %*% B %*% sqrt(C)

  # Produit scalaire
  ps = sum(diag(t(A_tild)%*%B_tild))

  return(ps)
 }
```

Exemple d'application sur le produit scalaire de Frobénius

```
# On initialise des matrices au hasard
A  = matrix(-16:18, nrow =7)
B = matrix(1:35, nrow =7)

# On applique la fonction prd_scalaire
resultat = prd_scalaire(A,B)

# Affichage du résultat
print(resultat)
```

```
## [1] 120
```

La fonction calculant la norme d'une matrice A associée au produit scalaire de Frobénius

```
norme = function(A){
  return(sqrt(prd_scalaire(A,A)))
}
```

Exemple d'application sur la norme

```
# On utilise les matrices précédente afin de calculer leurs normes
rn1 = norme(A)
rn2 = norme(B)

# Affichage du résultat
rn1; rn2
```

```
## [1] 10.14889
```

```
## [1] 20.63977
```

1.2 Coefficient RV

2. On définit le coefficient RV d'Escoufier entre deux matrices A et B de taille (n, p) par :

$$R(A, B) = \frac{[A \mid B]}{[|A|]\,[|B|]}$$

a) En terme géométrique le coefficient RV est le cosinus de A et B.

b) Ci-dessous nous trouverons le programme calculant le coefficient RV ainsi qu'une fonction donnant la matrice des coeffients RV en T tableaux $X_{(n,p)}$

La fonction calculant le coefficient RV d'Escoufier

```
coef_RV = function(T_tableaux) {
  t = length(T_tableaux)
  mat_rv = matrix(rep(NA, t * t), nrow = t, ncol = t)

  for (i in seq_along(T_tableaux)) {
    for (j in seq_along(T_tableaux)) {
      # Stocker le résultat dans une matrice
      prd_sclr = prd_scalaire(T_tableaux[[i]], T_tableaux[[j]])
      norm_i = norme(T_tableaux[[i]])
      norm_j = norme(T_tableaux[[j]])
      mat_rv[i, j] = prd_sclr / (norm_j * norm_i)
    }
  }

  return(mat_rv)
}
```

Exemple d'application sur le coefficient RV d'Escoufier T tableaux X_t(n , p)

```
X1 <- matrix(-12:22, nrow = 7)
X2 <- matrix(-20:14, nrow = 7)
X3 <- matrix(-16:18, nrow = 7)
X4 <- matrix(1:35, nrow = 7)
X5 <- matrix(-8:26, nrow = 7)
X6 <- matrix(-10:24, nrow = 7)
X7 <- matrix(-16:18, nrow = 7)
X8 <- matrix(-4:30, nrow = 7)
X9 <- matrix(-2:32, nrow = 7)
X10 <- matrix(-1:31, nrow = 7)
```

```
## Warning in matrix(-1:31, nrow = 7): data length [33] is not a sub-multiple or
## multiple of the number of rows [7]
```

```
# Noms de lignes
noms_lignes <- c("Ligne1", "Ligne2", "Ligne3", "Ligne4", "Ligne5", "Ligne6", "Ligne7")

# Ajouter les noms de lignes aux matrices
rownames(X1) <- noms_lignes
rownames(X2) <- noms_lignes
```

```r
rownames(X3) <- noms_lignes
rownames(X4) <- noms_lignes
rownames(X5) <- noms_lignes
rownames(X6) <- noms_lignes
rownames(X7) <- noms_lignes
rownames(X8) <- noms_lignes
rownames(X9) <- noms_lignes
rownames(X10) <- noms_lignes


n_tableaux = list(X1, X2, X3, X4, X5, X6, X7, X8,X9,X10)
resultats_n = coef_RV(n_tableaux)
print(data.frame(resultats_n))
```

```
##           X1        X2        X3        X4        X5        X6        X7
## 1  1.0000000 0.7327501 0.9355420 0.8254572 0.9642506 0.9893051 0.9355420
## 2  0.7327501 1.0000000 0.9258808 0.2207369 0.5262283 0.6256552 0.9258808
## 3  0.9355420 0.9258808 1.0000000 0.5728723 0.8084978 0.8740161 1.0000000
## 4  0.8254572 0.2207369 0.5728723 1.0000000 0.9455262 0.8989625 0.5728723
## 5  0.9642506 0.5262283 0.8084978 0.9455262 1.0000000 0.9925901 0.8084978
## 6  0.9893051 0.6256552 0.8740161 0.8989625 0.9925901 1.0000000 0.8740161
## 7  0.9355420 0.9258808 1.0000000 0.5728723 0.8084978 0.8740161 1.0000000
## 8  0.9001813 0.3632406 0.6883266 0.9888942 0.9834084 0.9540787 0.6883266
## 9  0.8685554 0.2991848 0.6375194 0.9966996 0.9688330 0.9315539 0.6375194
## 10 0.7275534 0.1570283 0.4854474 0.9125235 0.8479953 0.8003845 0.4854474
##           X8        X9       X10
## 1  0.9001813 0.8685554 0.7275534
## 2  0.3632406 0.2991848 0.1570283
## 3  0.6883266 0.6375194 0.4854474
## 4  0.9888942 0.9966996 0.9125235
## 5  0.9834084 0.9688330 0.8479953
## 6  0.9540787 0.9315539 0.8003845
## 7  0.6883266 0.6375194 0.4854474
## 8  1.0000000 0.9976953 0.8956236
## 9  0.9976953 1.0000000 0.9058163
## 10 0.8956236 0.9058163 1.0000000
```

2. Programme de STATIS1

2.1. Programme

a) L'expression $\left[\left|\sum_{t=1}^{T} \frac{u_t}{[|X_t|]} X_t\right|\right]^2$ représente l'inertie dans le contexte de l'analyse factorielle. L'inertie est également appelée somme des carrés des corrélations ou variance totale. Dans le cadre de l'analyse factorielle, cette quantité mesure la dispersion totale des données dans l'espace factoriel.

L'objectif du problème d'optimisation associé est de maximiser cette inertie sous la contrainte que la norme du vecteur u est égale à 1. Cela revient à trouver la direction dans laquelle la dispersion des données est maximale.

b) Résolvons le programme ci-dessus:

$$\max_{[|u|]^2=1}\left[\left|\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right|\right]^2=\max_{[|u|]^2=1}\operatorname{tr}\left(C\left(\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right)'W\left(\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right)\right)$$

$$\iff \max_{[|u|]^2=1}\left[\left|\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right|\right]^2=\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{1}{[|X_t|]\,[|X_\tau|]}u_t u_\tau \operatorname{tr}\left(CX'_\tau W X_t\right)$$

1. X_t: $n\times p$ (la matrice X_t a des dimensions $n\times p$).
2. $\frac{u_t}{[|X_t|]}X_t$: $n\times p$ (chaque colonne de X_t est pondérée par $\frac{u_t}{[|X_t|]}$).
3. $\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t$: $n\times p$ (somme des termes précédents sur t).
4. $\left(\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right)'$: $p\times n$ (transposée de la matrice résultante).
5. $\left[\left|\sum_{t=1}^{T}\frac{u_t}{[|X_t|]}X_t\right|\right]^2$: 1×1 (la norme euclidienne au carré est un scalaire).

Le langrangien associée au problème d'optimisation ci-dessus s'écrit:

$$L(u,\lambda)=\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{u_t u_\tau}{[|X_t|]\,[|X_\tau|]}tr(\tilde{X_t'}\tilde{X}_\tau)-\lambda(\|u\|^2-1)$$

On a alors :

$$\begin{cases}\frac{\partial L}{\partial u}(u,\lambda)=2\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{u_\tau}{[|X_t|][|X_\tau|]}tr(\tilde{X_t'}\tilde{X}_\tau)-2\lambda u\\ \frac{\partial L}{\partial \lambda}(u,\lambda)=\|u\|^2-1\end{cases}$$

$$\iff (S)\begin{cases}\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{tr(\tilde{X_t'}\tilde{X}_\tau)}{[|X_t|][|X_\tau|]}u_\tau=\lambda u \quad (*)\\ \|u\|^2=1 \quad (**)\end{cases}$$

D'une part, en multipliant l'équation $(*)$ *par* u' on obtient grâce à l'équation $(**)$ le résultat suivant:

$$\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{u_t tr(\tilde{X_t'}\tilde{X}_\tau)u_\tau}{[|X_t|]\,[|X_\tau|]}=\lambda$$

D'autre part, en multipliant par $Z:=[X_1|...|X_T]$ l'équation $(*)$ on obtient le résultat suivant:

$$\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{tr(\tilde{X_t'}\tilde{X}_\tau)}{[|X_t|]\,[|X_\tau|]}X_t u_\tau=\lambda\sum_{t=1}^{T}X_t u_t$$

D'après, (S) on en déduit que les vecteurs u solution du premier ordre sont les vecteurs propres de la de la matrice $\Gamma=\sum_{t=1}^{T}\sum_{\tau=1}^{T}\frac{\tilde{X}_t'\tilde{X}_\tau}{[|X_t|][|X_\tau|]}$ des coefficients RV d'Escoufier entre T tableaux $X_t(n,p)$.

c) Nous allons écrivez le programme R fournissant les vecteurs u solutions des équations du premier ordre.

Fonction donnant les vecteurs u solution et valeurs propres

```
vecval_prop = function(T_tableau) {
  matrice = coef_RV(T_tableau)# on calcule la matrice contenant les coefs d'Escoufier

  resultat_propre = eigen(matrice)# list ayant les valeurs et vecteurs propres
  val_prop = resultat_propre$values #valeurs propres
  vec_prop = resultat_propre$vectors #vecteurs propres associées

  return(list(val_prop = val_prop, vec_prop = vec_prop))
}
```

exemple d'application de la fonction vecval_prop

```
vecval = vecval_prop(n_tableaux)

val_prop <- vecval$val_prop
vec_prop <- vecval$vec_prop

# Affichage des valeurs propres et des vecteurs propres
print("Valeurs propres :")
```

```
## [1] "Valeurs propres :"
```

```
print(val_prop)
```

```
##  [1]  8.123366e+00  1.744723e+00  1.319111e-01  1.796886e-15  2.666068e-16
##  [6]  6.848904e-17  3.697448e-17 -2.917808e-18 -3.263088e-17 -1.225570e-16
```

```
print("Vecteurs propres :")
```

```
## [1] "Vecteurs propres :"
```

```
print(data.frame(vec_prop))
```

```
##            X1           X2          X3            X4            X5
## 1  -0.3466771 -0.115146214 -0.06518138 -7.381755e-01  0.000000e+00
## 2  -0.2185983 -0.590331919  0.16976790  5.466101e-01  1.129394e-01
## 3  -0.3059412 -0.370344120  0.05192976 -8.852231e-02 -8.575430e-02
## 4  -0.3155554  0.324638194 -0.23424259  2.687836e-01 -3.497644e-01
## 5  -0.3480801  0.085994975 -0.14755915 -1.547022e-02  7.916430e-01
## 6  -0.3505635 -0.005465137 -0.11111061 -2.185685e-02 -4.498014e-01
## 7  -0.3059412 -0.370344120  0.05192976 -8.045448e-02 -1.411761e-01
## 8  -0.3347469  0.220158653 -0.19789298  1.185744e-01  9.023557e-02
## 9  -0.3269108  0.268467959 -0.21503580  2.344861e-01  2.150950e-02
## 10 -0.2868342  0.361358829  0.88721247 -3.247402e-15 -1.151856e-15
##               X6            X7            X8            X9           X10
## 1   0.000000e+00  5.633868e-01  0.000000e+00  0.000000e+00  0.000000e+00
## 2   1.685951e-01  4.806686e-01  2.092876e-02  5.096506e-02  2.935908e-02
## 3  -3.298289e-01 -3.739290e-01  4.678997e-01  9.235818e-02 -5.246793e-01
## 4  -4.327154e-01  1.972467e-01  2.315625e-02  5.320606e-01  1.892404e-01
## 5   9.130877e-02 -2.339552e-01 -1.649794e-01  3.629255e-01  2.698022e-02
```

```
## 6   7.256190e-01 -2.583273e-01 -1.513218e-01  1.896027e-01 -9.913298e-02
## 7  -3.105407e-01 -3.633582e-01 -4.226427e-01 -2.663362e-01  5.119750e-01
## 8   1.607411e-01 -2.852187e-02  6.327034e-01 -4.456980e-01  3.905115e-01
## 9  -1.356927e-01  1.360631e-01 -3.886369e-01 -5.182068e-01 -5.127035e-01
## 10  2.012279e-16 -3.677614e-16 -9.584347e-16  9.280771e-16 -5.984796e-16
```

Montrons à présent que les vecteurs u obtenus forment une base I-orthonormée.

La fonction **"vecval_prop()"** ci-dessus nous donne les vecteurs propres de la matrice des coefficients RV Γ associée aux valeurs propres λ. Ensuite, nous utilisons la fonction **"verifier_orthogonalite()"** ci-dessous afin de calculé le produit scalire euclidienne entre les vecteurs propres. Si les vecteurs sont orthogonaux, la fonction nous renverra **"Les vecteurs propres sont orthogonaux entre eux"** dans le cas contraire elle affichera **"Les vecteurs propres ne sont pas orthogonaux entre eux"**.

```r
# Fonction pour vérifier l'orthogonalité des vecteurs propres
verifier_orthogonalite <- function(vec_prop) {
  n <- nrow(vec_prop) # Nombre de vecteurs propres
  orthogonale <- TRUE

  # Vérifier l'orthogonalité pour chaque paire de vecteurs propres
  for (i in 1:(n - 1)) {
    for (j in (i + 1):n) {
      produit_scalaire <- sum(vec_prop[, i] * vec_prop[, j]) # Produit scalaire
      if (abs(produit_scalaire) > 1e-15) { # Vérifier si le produit scalaire est proche de zéro
        orthogonale <- FALSE
        break
      }
    }
    if (!orthogonale) {
      break
    }
  }

  return(orthogonale)
}
```

```r
# fonction pour vérifier si les vecteurs propres sont bien de norme 1
verifier_norme = function(vec_prop){
  n = length(vec_prop[,1])
  v = rep(1,n)
  int = rep(NA,n)

  for (i in 1:n) {
      p = t(vec_prop[,i]) %*% vec_prop[,i]
      int[i] = p
  }

  if(sum(int) == sum(v)){
    print("les vecteurs sont bien de norme 1")
  }else{
    print("il existe un vecteurs qui n'est pas de norme 1")
 }
}
```

Utilisation de les fonctions pour vérifier si les vecteurs propres forment une base orthonormée:

```r
orthogonale = verifier_orthogonalite(vec_prop)

# Affichage du résultat
if (orthogonale) {
  print("Les vecteurs propres sont orthogonaux entre eux.")
} else {
  print("Les vecteurs propres ne sont pas orthogonaux entre eux.")
}
```

```
## [1] "Les vecteurs propres ne sont pas orthogonaux entre eux."
```

```r
verifier_norme(vec_prop)
```

```
## [1] "il existe un vecteurs qui n'est pas de norme 1"
```

2.2. Équivalence avec l'ACP d'un tableau juxtaposé "dépliant" le tableau cubique

a)

Dans le contexte de l'ACP des tableaux juxtaposés, les "variables" sont les différents tableaux $\mathbf{X_t}$, et les "individus" sont les observations à chaque période de temps.

Dans notre cas, chaque $\mathbf{X_t}$ représente un tableau de données à une période de temps spécifique, où les lignes représentent les individus et les colonnes représentent les variables. Par conséquent, chaque colonne de $\mathbf{X_t}$ représente une variable différente observée à cette période.

Ainsi, en considérant les tableaux $\mathbf{X_t}$ comme les variables, la matrice $\mathbf{A}$ est calculée en utilisant les produits scalaires entre les différentes combinaisons de tableaux $\mathbf{X_t}$. Les valeurs propres et les vecteurs propres de cette matrice $\mathbf{A}$ correspondent alors aux directions dans lesquelles la dispersion des données est maximale dans l'espace des tableaux juxtaposés.

b) Pour déduire que les composantes principales $F^1, \ldots, F^k, \ldots$ sont orthogonales au sens du produit scalaire convenable, nous devons considérer la façon dont ces composantes sont construites à partir des vecteurs propres obtenus.

Lorsque nous obtenons les vecteurs propres u_k de la matrice des coefficients RV, nous avons montré précédemment qu'ils forment une base orthogonale au sens du produit scalaire euclidien. Cela signifie que ces vecteurs propres sont mutuellement orthogonaux.

Ensuite, les composantes principales F^k sont obtenues en projetant les tableaux X_t sur les vecteurs propres u_k. Comme chaque vecteur propre u_k est orthogonal aux autres vecteurs propres, les projections des tableaux sur ces vecteurs propres le seront également. Par conséquent, les composantes principales F^k seront orthogonales les unes aux autres.

c) Dépliage du tableau cubique en un tableau juxtaposé :

```r
# Création des tableaux X_t dépliés
X1_jux <- matrix(-12:22, nrow = 35)
X2_jux <- matrix(-20:14, nrow = 35)
X3_jux <- matrix(-16:18, nrow = 35)
X4_jux <- matrix(1:35, nrow = 35)
X5_jux <- matrix(-8:26, nrow = 35)
```

```r
X6_jux <- matrix(-10:24, nrow = 35)
X7_jux <- matrix(-16:18, nrow = 35)
X8_jux <- matrix(-4:30, nrow = 35)
X9_jux <- matrix(-2:32, nrow = 35)
X10_jux <- matrix(-3:31, nrow = 35)
mat_jux <- cbind(X1_jux, X2_jux, X3_jux, X4_jux, X5_jux, X6_jux, X7_jux, X8_jux, X9_jux, X10_jux)
mat_jux_centr <- scale(mat_jux) # Centrage-réduction des données
w_jux <- diag(35)/35 # Poids uniformes
mat_sym <- t(mat_jux_centr) %*% w_jux %*% mat_jux_centr # Matrice de variance-covariance
```

Composantes principales des tableaux dépliés :

```r
# On veut donner la fonction des composantes principales correspondant aux vecteurx u (vec_prop)
composantes_principales = function(u, X_t){
  F <- X_t %*% u # calcul de la matrice des composantes principales

  return(F)
}
print(composantes_principales(vec_prop, mat_jux)) # Affichage des composantes principales
```

```
##              [,1]     [,2]        [,3]      [,4]      [,5]      [,6]        [,7]
##  [1,]  27.150242 22.22909 -3.65759079 0.2973766 -1.216759 -1.916943 -0.08329683
##  [2,]  24.010393 22.03807 -3.46777342 0.5213515 -1.226927 -1.979457  0.03597695
##  [3,]  20.870544 21.84706 -3.27795606 0.7453264 -1.237096 -2.041971  0.15525072
##  [4,]  17.730696 21.65605 -3.08813869 0.9693012 -1.247265 -2.104484  0.27452450
##  [5,]  14.590847 21.46504 -2.89832132 1.1932761 -1.257433 -2.166998  0.39379828
##  [6,]  11.450999 21.27402 -2.70850396 1.4172510 -1.267602 -2.229512  0.51307206
##  [7,]   8.311150 21.08301 -2.51868659 1.6412259 -1.277770 -2.292025  0.63234584
##  [8,]   5.171301 20.89200 -2.32886922 1.8652008 -1.287939 -2.354539  0.75161962
##  [9,]   2.031453 20.70098 -2.13905186 2.0891756 -1.298108 -2.417053  0.87089340
## [10,]  -1.108396 20.50997 -1.94923449 2.3131505 -1.308276 -2.479566  0.99016718
## [11,]  -4.248245 20.31896 -1.75941712 2.5371254 -1.318445 -2.542080  1.10944096
## [12,]  -7.388093 20.12795 -1.56959976 2.7611003 -1.328613 -2.604594  1.22871473
## [13,] -10.527942 19.93693 -1.37978239 2.9850751 -1.338782 -2.667107  1.34798851
## [14,] -13.667791 19.74592 -1.18996502 3.2090500 -1.348951 -2.729621  1.46726229
## [15,] -16.807639 19.55491 -1.00014766 3.4330249 -1.359119 -2.792135  1.58653607
## [16,] -19.947488 19.36389 -0.81033029 3.6569998 -1.369288 -2.854648  1.70580985
## [17,] -23.087337 19.17288 -0.62051292 3.8809747 -1.379456 -2.917162  1.82508363
## [18,] -26.227185 18.98187 -0.43069556 4.1049495 -1.389625 -2.979676  1.94435741
## [19,] -29.367034 18.79086 -0.24087819 4.3289244 -1.399794 -3.042189  2.06363119
## [20,] -32.506883 18.59984 -0.05106082 4.5528993 -1.409962 -3.104703  2.18290496
## [21,] -35.646731 18.40883  0.13875654 4.7768742 -1.420131 -3.167217  2.30217874
## [22,] -38.786580 18.21782  0.32857391 5.0008491 -1.430300 -3.229730  2.42145252
## [23,] -41.926429 18.02680  0.51839128 5.2248239 -1.440468 -3.292244  2.54072630
## [24,] -45.066277 17.83579  0.70820864 5.4487988 -1.450637 -3.354758  2.66000008
## [25,] -48.206126 17.64478  0.89802601 5.6727737 -1.460805 -3.417271  2.77927386
## [26,] -51.345975 17.45377  1.08784338 5.8967486 -1.470974 -3.479785  2.89854764
## [27,] -54.485823 17.26275  1.27766074 6.1207235 -1.481143 -3.542299  3.01782142
## [28,] -57.625672 17.07174  1.46747811 6.3446983 -1.491311 -3.604812  3.13709519
## [29,] -60.765521 16.88073  1.65729548 6.5686732 -1.501480 -3.667326  3.25636897
## [30,] -63.905369 16.68971  1.84711284 6.7926481 -1.511648 -3.729840  3.37564275
## [31,] -67.045218 16.49870  2.03693021 7.0166230 -1.521817 -3.792353  3.49491653
## [32,] -70.185067 16.30769  2.22674758 7.2405978 -1.531986 -3.854867  3.61419031
```

```
## [33,] -73.324915 16.11667  2.41656494 7.4645727 -1.542154 -3.917381  3.73346409
## [34,] -76.464764 15.92566  2.60638231 7.6885476 -1.552323 -3.979895  3.85273787
## [35,] -79.604613 15.73465  2.79619968 7.9125225 -1.562492 -4.042408  3.97201165
##               [,8]      [,9]      [,10]
##  [1,] -0.040017280 0.3161816 0.04417691
##  [2,] -0.022909990 0.3138527 0.05572735
##  [3,] -0.005802701 0.3115239 0.06727779
##  [4,]  0.011304588 0.3091950 0.07882823
##  [5,]  0.028411878 0.3068661 0.09037867
##  [6,]  0.045519167 0.3045373 0.10192911
##  [7,]  0.062626457 0.3022084 0.11347954
##  [8,]  0.079733746 0.2998795 0.12502998
##  [9,]  0.096841036 0.2975507 0.13658042
## [10,]  0.113948325 0.2952218 0.14813086
## [11,]  0.131055615 0.2928930 0.15968130
## [12,]  0.148162904 0.2905641 0.17123174
## [13,]  0.165270194 0.2882352 0.18278218
## [14,]  0.182377483 0.2859064 0.19433262
## [15,]  0.199484773 0.2835775 0.20588306
## [16,]  0.216592062 0.2812486 0.21743350
## [17,]  0.233699352 0.2789198 0.22898394
## [18,]  0.250806641 0.2765909 0.24053438
## [19,]  0.267913931 0.2742620 0.25208482
## [20,]  0.285021220 0.2719332 0.26363526
## [21,]  0.302128509 0.2696043 0.27518570
## [22,]  0.319235799 0.2672754 0.28673614
## [23,]  0.336343088 0.2649466 0.29828658
## [24,]  0.353450378 0.2626177 0.30983702
## [25,]  0.370557667 0.2602889 0.32138746
## [26,]  0.387664957 0.2579600 0.33293790
## [27,]  0.404772246 0.2556311 0.34448834
## [28,]  0.421879536 0.2533023 0.35603878
## [29,]  0.438986825 0.2509734 0.36758922
## [30,]  0.456094115 0.2486445 0.37913966
## [31,]  0.473201404 0.2463157 0.39069010
## [32,]  0.490308694 0.2439868 0.40224054
## [33,]  0.507415983 0.2416579 0.41379098
## [34,]  0.524523273 0.2393291 0.42534142
## [35,]  0.541630562 0.2370002 0.43689186
```

```r
# On veut maintenant programmer la fonction calculant la kième composante F_k ainsi que la composante n
composante_k = function(u, X_t, k){

  F_k = X_t %*% u # calcul des composantes principales


  f_k = F_k / sqrt(sum(F_k^2)) # calcul de la kième composante principale normée

  return(list(F_k = F_k[,k], f_k = f_k[,k]))
}
print(composante_k(vec_prop, mat_jux, 1)) # Affichage de la première composante principale et de sa ver
```

```
## $F_k
##  [1]  27.150242  24.010393  20.870544  17.730696  14.590847  11.450999
```

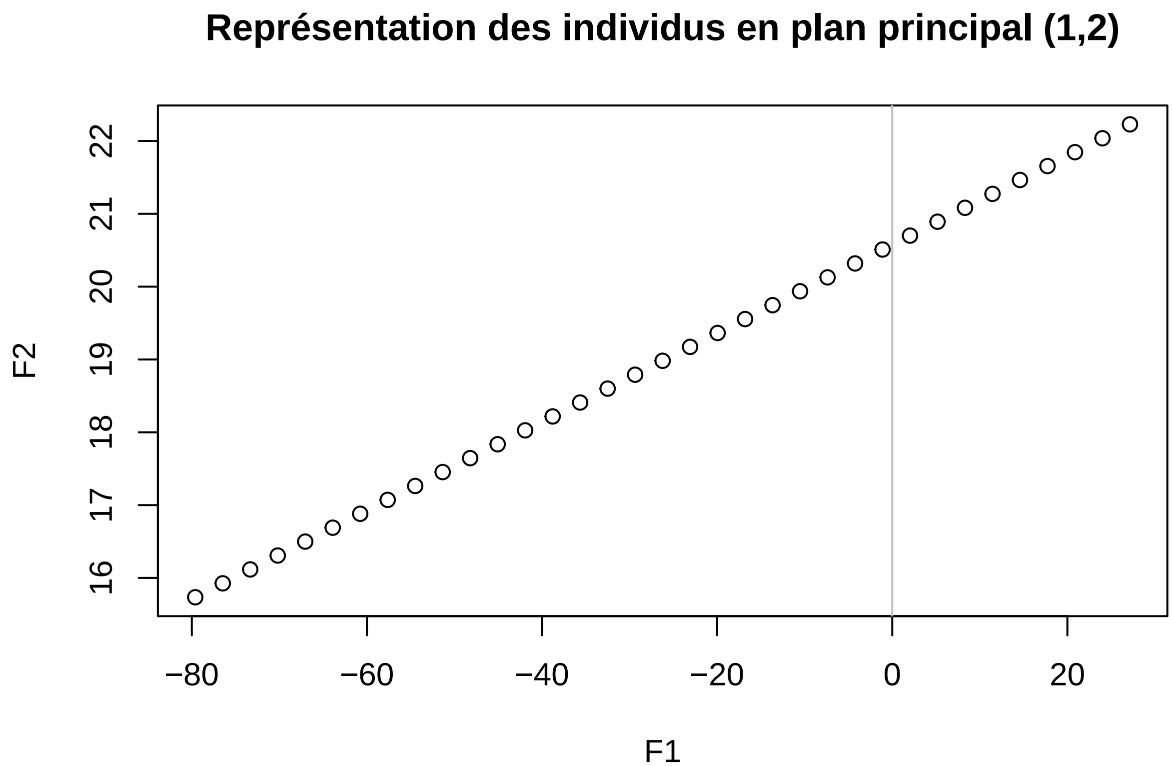
```
##  [7]   8.311150   5.171301   2.031453  -1.108396  -4.248245  -7.388093
## [13] -10.527942 -13.667791 -16.807639 -19.947488 -23.087337 -26.227185
## [19] -29.367034 -32.506883 -35.646731 -38.786580 -41.926429 -45.066277
## [25] -48.206126 -51.345975 -54.485823 -57.625672 -60.765521 -63.905369
## [31] -67.045218 -70.185067 -73.324915 -76.464764 -79.604613
##
## $f_k
##  [1]  0.100145176  0.088563669  0.076982163  0.065400657  0.053819151
##  [6]  0.042237644  0.030656138  0.019074632  0.007493126 -0.004088381
## [11] -0.015669887 -0.027251393 -0.038832899 -0.050414406 -0.061995912
## [16] -0.073577418 -0.085158925 -0.096740431 -0.108321937 -0.119903443
## [21] -0.131484950 -0.143066456 -0.154647962 -0.166229468 -0.177810975
## [26] -0.189392481 -0.200973987 -0.212555493 -0.224137000 -0.235718506
## [31] -0.247300012 -0.258881518 -0.270463025 -0.282044531 -0.293626037
```

```r
print(composante_k(vec_prop, mat_jux, 2)) # Affichage de la deuxième composante principale et de sa ver
```

```
## $F_k
##  [1] 22.22909 22.03807 21.84706 21.65605 21.46504 21.27402 21.08301 20.89200
##  [9] 20.70098 20.50997 20.31896 20.12795 19.93693 19.74592 19.55491 19.36389
## [17] 19.17288 18.98187 18.79086 18.59984 18.40883 18.21782 18.02680 17.83579
## [25] 17.64478 17.45377 17.26275 17.07174 16.88073 16.68971 16.49870 16.30769
## [33] 16.11667 15.92566 15.73465
##
## $f_k
##  [1] 0.08199322 0.08128866 0.08058410 0.07987954 0.07917498 0.07847042
##  [7] 0.07776585 0.07706129 0.07635673 0.07565217 0.07494761 0.07424305
## [13] 0.07353848 0.07283392 0.07212936 0.07142480 0.07072024 0.07001568
## [19] 0.06931111 0.06860655 0.06790199 0.06719743 0.06649287 0.06578831
## [25] 0.06508374 0.06437918 0.06367462 0.06297006 0.06226550 0.06156094
## [31] 0.06085637 0.06015181 0.05944725 0.05874269 0.05803813
```

```r
# Fonction donnant la représentation graphique des individus en plan principal (k,l)
representation_graphique = function(vec_prop, X_t, k, l){
  F_k = composante_k(vec_prop, X_t, k)$F_k # Récupération de la kième composante principale
  F_l = composante_k(vec_prop, X_t, l)$F_k # Récupération de la lième composante principale

  plot(F_k, F_l, xlab = paste("F", k, sep = ""), ylab = paste("F", l, sep = ""), main = paste("Représent
  abline(h = 0, col = "gray")
  abline(v = 0, col = "gray")
}
representation_graphique(vec_prop, mat_jux, 1, 2) # Affichage de la représentation graphique des indivi
```



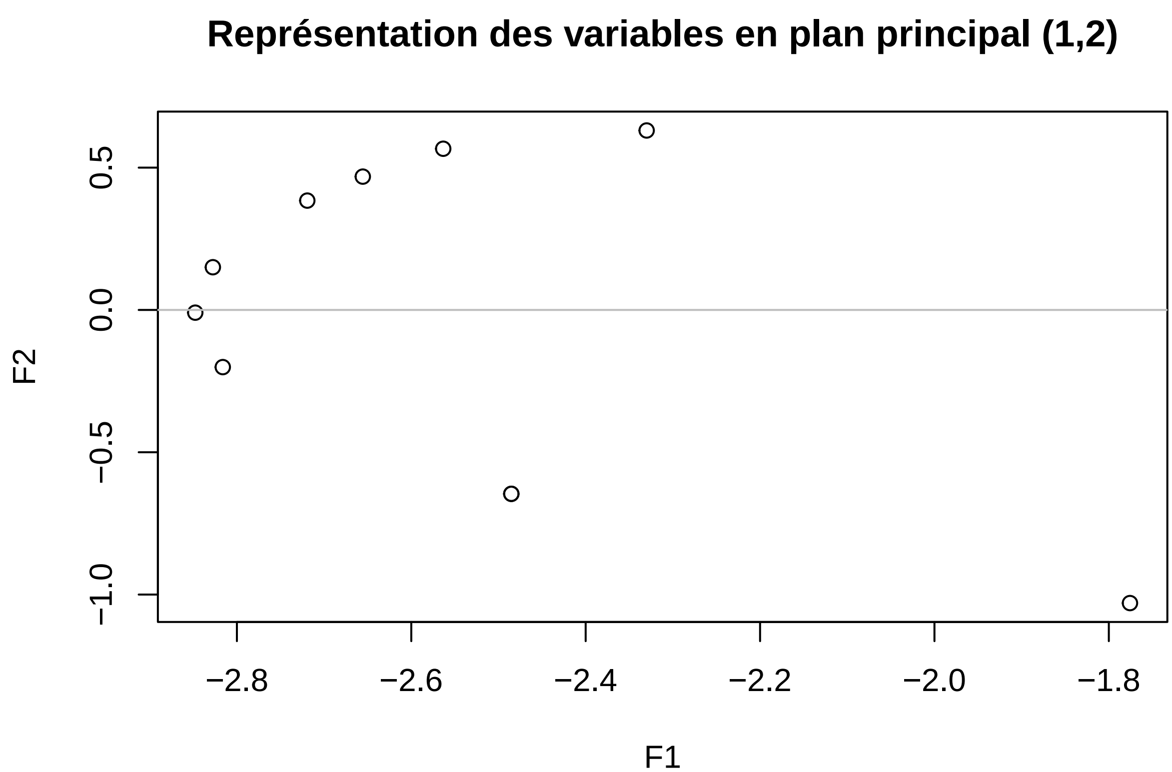
d) Cosinus entre les tableaux et les composantes principales :

```
# Calcul des cosinus entre Xt (resultats_n) et chaque composante principale F_k
cosinus = function(X_t, F_k){
  for (k in 1:ncol(F_k)){
    cos = sum(X_t * F_k[,k]) / (sqrt(sum(X_t^2)) * sqrt(sum(F_k^2))) # Calcul du cosinus entre Xt et la
    print(cos)
  }
}
cosinus(resultats_n, composantes_principales(vec_prop, resultats_n)) # Affichage du produit scalaire en
```

```
## [1] -3.000639
## [1] -0.008420703
## [1] 4.78334e-05
## [1] 1.069286e-16
## [1] 1.489574e-17
## [1] -5.015876e-17
## [1] -3.527471e-16
## [1] -1.530142e-17
## [1] -7.815522e-17
## [1] -8.352907e-17
```

```
# Fonction donnant la représentation graphique des variables en plan principal (k,l)
representation_graphique_variables = function(vec_prop, X_t, k, l){
  F_k = composante_k(vec_prop, X_t, k)$F_k # Récupération de la kième composante principale
  F_l = composante_k(vec_prop, X_t, l)$F_k # Récupération de la lième composante principale
```

```
  plot(F_k, F_l, xlab = paste("F", k, sep = ""), ylab = paste("F", l, sep = ""), main = paste("Représen
  abline(h = 0, col = "gray")
  abline(v = 0, col = "gray")
}
representation_graphique_variables(vec_prop, resultats_n, 1, 2) # Affichage de la représentation graphi
```



2.3. ACP et d'autres dépliages du tableau cubique

Pour cette parties, les réponses sont données dans un contexte sans data. Nous utiliserons par la suite le set de données chickenk de la librairie ade4 et modifierons nos réponses en conséquence, mais ous devons d'abord le fonctionnement de ce set : par exemple comment fonctionne le dépliage si les tableaux ne font pas la même dimension ? Est-ce possible de le transformer en tableau cubique ? Devons-nous changer de set ? Une compréhension des variables est aussi nécessaire pour répondre aux questions b) et c).

a) Il y a deux autres dépliages possibles du tableau cubique selon ce que l'on définit comme individus et variables.

b) Quelle(s) ACP des tableaux dépliés pouvez-vous envisager? Pour chacune: quels sont ses "individus" et ses "variables"? Quel centrage paraît opportun? Quelle réduction? Quels éléments sont utiles à projeter en supplémentaire? Les ACP à envisager dépendent du jeu de données que l'on souhaite étudier. Ici nous avons créé notre jeu de données sans réelle signification. De même les indidus et variables restent à déterminer. Ce qui est sûr c'est que les individus regrouperont les tableaux dépliés. Les éléments à projeter en supplémentaire dépendent de ce que l'on souhaite étudier.

c) Quels sont les avantages et les inconvénients de chaque ACP d'un tableau "déplié"? Finalement, le "dépliage" opéré par STATIS1 était-il le plus opportun? Le principal inconvénient est qu'on perde une grosse partie de l'information qui sera noyée dans le tableau déplié. Cependant, cela permet de mieux visualiser les données et de les traiter plus facilement.

II - Situation 2