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

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

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

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

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

Or, par la propriété de la trace, $\mathrm{tr}(AB) = \mathrm{tr}(BA)$ 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:

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

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

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

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

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

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

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 $[A \mid B] = \mathrm{tr}(\tilde{A}'\tilde{B})$ 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

```r
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

```r
# 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

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

Exemple d'application sur la norme

```r
# 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 et 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)

```
set.seed(123)
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(rnorm(35), nrow = 7)
X10 <- matrix(rpois(35,0.5), nrow = 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
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
## 1   1.00000000  0.73275013  0.93554203 0.82545722 0.96425063 0.9893050661
```

```
## 2   0.73275013  1.00000000  0.92588083 0.22073690 0.52622833 0.6256551962
## 3   0.93554203  0.92588083  1.00000000 0.57287231 0.80849776 0.8740160582
## 4   0.82545722  0.22073690  0.57287231 1.00000000 0.94552623 0.8989624962
## 5   0.96425063  0.52622833  0.80849776 0.94552623 1.00000000 0.9925900690
## 6   0.98930507  0.62565520  0.87401606 0.89896250 0.99259007 1.0000000000
## 7   0.93554203  0.92588083  1.00000000 0.57287231 0.80849776 0.8740160582
## 8   0.90018126  0.36324063  0.68832665 0.98889421 0.98340842 0.9540786773
## 9  -0.00635103 -0.03731433 -0.02289429 0.02184908 0.00659435 0.0007175268
## 10  0.19613819 -0.23188264 -0.01146283 0.47346177 0.33538952 0.2745490570
##              X7         X8            X9         X10
## 1   0.93554203 0.90018126 -0.0063510302  0.19613819
## 2   0.92588083 0.36324063 -0.0373143284 -0.23188264
## 3   1.00000000 0.68832665 -0.0228942893 -0.01146283
## 4   0.57287231 0.98889421  0.0218490773  0.47346177
## 5   0.80849776 0.98340842  0.0065943497  0.33538952
## 6   0.87401606 0.95407868  0.0007175268  0.27454906
## 7   1.00000000 0.68832665 -0.0228942893 -0.01146283
## 8   0.68832665 1.00000000  0.0151855616  0.41694407
## 9  -0.02289429 0.01518556  1.0000000000  0.05613269
## 10 -0.01146283 0.41694407  0.0561326881  1.00000000
```

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:

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

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

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:

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

On a alors :

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

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

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

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

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

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

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]  6.739603e+00  1.788305e+00  9.924783e-01  4.796133e-01  1.218479e-15
##  [6]  2.182024e-16  2.093232e-16 -1.218379e-16 -2.104815e-16 -2.236445e-16
```

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

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

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

```
##              X1          X2           X3          X4            X5
## 1  -0.384679579 -0.03763255  0.003890890 -0.01688904 -6.378123e-01
## 2  -0.270659986 -0.49935328  0.058538293  0.34460882  5.853027e-01
## 3  -0.354063226 -0.28008529  0.032544800  0.16949385 -1.492991e-01
## 4  -0.326838432  0.36027010 -0.042980130 -0.31005531  4.430808e-01
## 5  -0.375294366  0.14742738 -0.017933318 -0.15529814  8.933387e-02
## 6  -0.382969124  0.06389311 -0.008086995 -0.09322615 -2.982176e-02
## 7  -0.354063226 -0.28008529  0.032544800  0.16949385 -1.387732e-01
## 8  -0.353458270  0.26805964 -0.032137072 -0.24367148  6.226940e-02
## 9   0.001435391  0.09907872  0.994782163 -0.02428570 -1.619798e-16
## 10 -0.093668974  0.59586096 -0.039763829  0.79661451 -6.522560e-16
##                X6            X7            X8            X9           X10
## 1   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  6.659583e-01
## 2  -2.390914e-01 -2.951559e-02 -8.156556e-02 -1.318286e-02  3.844029e-01
## 3   4.123488e-01 -2.459876e-01 -3.388825e-01  5.193143e-01 -3.592267e-01
## 4   5.931537e-01  2.225746e-01  6.074869e-02  4.859803e-02  2.483081e-01
## 5  -1.461612e-01 -7.459941e-01  4.382562e-01 -1.378447e-01 -1.267269e-01
## 6  -8.295332e-02  6.186864e-02 -5.779387e-01 -6.581339e-01 -2.484835e-01
## 7   8.707038e-02  4.905047e-01  5.851545e-01 -1.861758e-01 -3.491458e-01
## 8  -6.206020e-01  2.969001e-01 -7.954750e-02  4.908913e-01 -1.353757e-01
## 9  -9.454243e-17 -1.807026e-16  8.166753e-17 -3.985121e-17  1.076071e-17
## 10  1.266348e-16 -3.707971e-17 -1.188286e-16 -1.561251e-17  7.546047e-17
```

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 sont orthogonaux entre eux."
```

```r
verifier_norme(vec_prop)
```

```
## [1] "les vecteurs sont bien 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)
d)

2.3. ACP et d'autres dépliages du tableau cubique

II - Situation 2