L'utilisation des transformations de Schoenberg dans le cadre de l'analyse factorielle

Guillaume Guex

Master en statistique

Professeur responsable : Dr. Alina Matei, Université de Neuchâtel
en collaboration avec le Pr. François Bavaud, Université de Lausanne

Janvier 2011

Table des matières

1 Introduction

Les méthodes factorielles sont une famille de méthodes, appartenant au domaine de l'analyse de données, visant à réduire les dimensions de variables multivariées, tout en gardant un maximum d'informations. Ces méthodes consistent généralement à transformer des variables corrélées en de nouvelles variables indépendantes, appelée *composantes principales*, puis de retenir parmi elles celles qui permettent d'expliquer au maximum la variance du jeu de données.

Cette compression de l'information a plusieurs applications. Elle permet en premier lieu d'effectuer une projection optimale de n variables avec p paramètres, vues comme n points dans un espace à p dimensions, sur un plan, ou dans l'espace, nous donnant alors la "meilleure" visualisation possible des différences entre ces variables. Mais elle est également utilisée dans d'autres domaines, par exemple l'imagerie médicale, comme réductrice du "bruit" contenu dans les données, celui-ci se situant en grande partie sur des composantes principales abandonnées. On abouti alors à une meilleure qualité des données, et donc à une meilleure image.

Il est cependant intéressant de remarquer qu'avant la compression, les méthodes factorielles effectuent un plongement de l'espace des points dans un espace de dimension supérieure. En effet, si nous avons au départ n variables avec p paramètres, nous obtiendrons n composantes principales avec n paramètres. Une propriété remarquable, qui sera utilisée avec nos transformations de Schoenberg, est que ce plongement peut s'effectuer sans avoir la position des points dans l'espace, mais uniquement la matrice des distances relatives entre ces points. Nous reviendrons là-dessus plus tard.

Malheureusement, dans certains cas, il se peut que les données initiales soient particulièrement "mauvaises" : les méthodes factorielles ne permettront alors pas d'obtenir des résultats satisfaisants. C'est ici qu'interviennent des transformations, qui ont été étudiées par Schoenberg en 1938 [SCH1, SCH2] et que l'on nommera tout au long de cet article *transformations de Schoenberg*. Ces dernières sont en fait des fonctions qui envoient une matrice de distances euclidiennes carrées sur une autre matrice de distances euclidiennes carrées. Ensuite, grâce aux méthodes factorielles, il nous est possible de trouver la position de chaque point dans l'espace correspondant à cette matrice de distances euclidiennes modifiée. Ces transformations peuvent donc être vues comme un plongement d'un espace euclidien dans un autre, de dimension supérieure, tout en effectuant une anamorphose de l'espace. Pour l'analyse de données, cette anamorphose peut se révéler très intéressante. Elle pourrait permettre de courber l'espace en certains points, faisant alors apparaître plus de détails, ou au contraire d'aplanir d'autres par-

ties dans l'idée de "cartographier" ces zones. Ces plongements dans un espace de dimension supérieure sont d'ailleurs déjà très utilisés dans le domaine du "statistical machine learning" sous le nom des "méthodes à noyaux" et "méthodes locales". Elles n'ont néanmoins pas le même formalisme qu'ici [DON et références incluses].

Dans cet article, nous allons explorer, d'une manière numérique, les effets de certaines transformations de Schoenberg sur des données multivariées. Grâce au programme de statistiques *R*, nous allons effectuer de nombreux tests sur différents jeux de données synthétiques, et nous analyserons ensuite les résultats pour essayer de mieux comprendre les effets de ces transformations sur l'espace. Nous allons également montrer quelques cas où les données modifiées par une transformation de Schoenberg apportent un avantage par rapport aux données initiales.

Ce rapport est divisé en trois parties principales. La section 2 établit le formalisme mathématique qui entoure les méthodes factorielles et les transformations de Schoenberg. La section 3 explique, d'une manière plus orientée sur la programmation, les différents algorithmes et fonctions numériques mis en place pour étudier ces transformations. Pour finir, la section 4 contient quelques résultats intéressants, ainsi que leur analyse.

2 Théorie

Nous allons poser ici le formalisme mathématique nécessaire pour comprendre en détails la suite du rapport. La section 2.1 est consacrée aux méthodes factorielles et la section 2.2 aux transformations de Schoenberg. Nous verrons ensuite en 2.3 un résumé rapide de la marche à suivre à employer lorsque qu'il s'agit d'effectuer une transformation de Schoenberg couplée à une méthode factorielle, plongeant et courbant ainsi un espace de dimension $\mathbb{R}^p$ dans $\mathbb{R}^n$.

2.1 Les méthodes factorielles

2.1.1 Données de base :

Les méthodes factorielles portent sur des données de la forme suivante :

- une matrice X $(n \times p)$, appelée **matrice des coordonnées**, représentant n objets ayant chacun p paramètres.
- un vecteur f (avec $f_i > 0$ et $\sum_{i=1}^{n} f_i = 1$) de dimension n, qui est le **vecteur des poids relatifs** de ces objets.

Chacun des objets composant X peut être vu comme un point dans $\mathbb{R}^p$. On peut alors définir :

Définition 2.1.1 *Soit D $(n \times n)$ la **matrice des distances euclidiennes carrées** entre chacun des "points-lignes" de X. C'est à dire :*

$$D_{ij} := \sum_{k=1}^{p} (x_{ik} - x_{jk})^2$$

Définition 2.1.2 *l'**inertie** de la configuration (D, f) relativement à un point a se défini comme :*

$$\Delta(a|f) := \sum_{i=1}^{n} f_i D_{ia}$$

Définition 2.1.3 *On pose également l'**inertie minimale** du système comme :*

$$\Delta(f) := \min_{a} \Delta(a|f)$$

Définition 2.1.4 *le **centre de gravité** de la configuration $\bar{x}$ est donné par :*

$$\bar{x}_k := \sum_{i=1}^{n} f_i x_{ik}$$

Le *principe de Huygens fort* nous dit que l'inertie relativement au point a est égale à la distance qui sépare a du centre de gravité ajouté à l'inertie relativement au centre de gravité :

$$\Delta(a|f) = \sum_{i=1}^{n} f_i D_{i\bar{x}} + D_{\bar{x}a}$$

On remarque donc que $\Delta(\bar{x}|f) = \Delta(f)$.

Le *principe de Huygens faible* nous dit que :

$$\Delta(f) = \sum_{i=1}^{n} f_i D_{i\bar{x}} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} f_i f_j D_{ij}$$

2.1.2 Principe des méthodes factorielles :

Les méthodes factorielles ont pour but de déterminer un nouveau système d'axes orthogonaux $u_1, \ldots, u_\beta$, centré sur un point choisi a (généralement on prend $a = \bar{x}$), où il y aura une proportion maximale d'inertie représentée dans les premières dimensions (l'inertie projetée sur u_2 sera plus petite que l'inertie projetée sur u_1, et ainsi de suite). Nous représenterons ensuite nos points initiaux dans ce nouvel espace.

Commençons tout d'abord par définir deux nouvelles matrices :

Définition 2.1.5 *La **matrice des produits scalaires** $(n \times n)$ par rapport au point a, $B^a = (B^a_{ij})$ se définit comme :*

$$B^a_{ij} := \sum_{k=1}^{p} (x_{ik} - a_k)(x_{jk} - a_k)$$

Définition 2.1.6 *La **matrice des produits scalaires pondérés associés** $(n \times n)$ $K^a = (K^a_{ij})$, est :*

$$K^a_{ij} := \sqrt{f_i f_j} B^a_{ij}$$

En regardant de plus près ces matrices, on observe que :

$$B^a_{ii} = D_{ia}$$

et donc

$$tr(K^a) = \sum_{i=1}^{n} K^a_{ii} = \sum_{i=1}^{n} f_i D_{ia} = \Delta(a|f)$$

La trace de la matrice K^a est donc égale à l'inertie du système. Trouver un système d'axes orthogonaux dans lequel l'inertie du système est représentée de manière maximale dans les premières dimensions revient donc à chercher les vecteurs propres $u_1, \cdots, u_n$ associés aux valeurs propres décroissantes $\lambda_1 \leqslant \cdots \leqslant \lambda_n$ de K^a. On peut donc représenter nos points dans un espace en n dimensions au maximum.

Pour trouver ces vecteurs propres, nous allons décomposer spectralement la matrice K^a.

Proposition 2.1.7 *La matrice K^a peut s'écrire :*

$$K^a = U\Lambda U'$$

où $U = (u_{i\beta})$ est orthogonale et contient les vecteurs propres de K^a sur ses colonnes, et Λ est diagonale et contient les valeurs propres associées, classées par valeurs décroissantes, non-négatives.

Preuve : On remarque que

$$\sum_{i=1}^{n}\sum_{j=1}^{n} B^a_{ij} z_i z_j =$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{p}(x_{ik}-a_k)(x_{jk}-a_k)z_i z_j =$$

$$\sum_{k=1}^{p}\sum_{i=1}^{n}(x_{ik}-a_k)z_i \sum_{j=1}^{n}(x_{jk}-a_k)z_j =$$

$$\sum_{k=1}^{p}\left(\sum_{i=1}^{n}(x_{ik}-a_k)z_i\right)^2 \geq 0$$

B^a est donc semi-définie positive. C'est également le cas pour K^a car chacun de ses termes est multiplié par un nombre positif. Par le théorème de décomposition spectrale, K^a peut donc s'écrire comme $K^a = U\Lambda U'$ et ses valeurs propres sont non-négatives.

□

Il est important de noter que la dernière valeur propre de la matrice K^a est la valeur propre triviale $\lambda_n = 0$, elle ne contient donc aucune information sur l'inertie du système et peut généralement être abandonnée.

Définition 2.1.8 *Les* ***coordonnées factorielles*** *ou* ***coordonnées de basses dimensionalités*** $\tilde{x}_\beta$ *de* X *sont définies par :*

$$\tilde{x}_{i\beta} := \frac{\sqrt{\lambda_\beta}}{\sqrt{f_i}} u_{i\beta}$$

On remarque que les $\tilde{x}_\beta$ sont de dimension n, et représentent les "points-colonnes" de notre matrice initiale X. Il suffit maintenant de poser un m tel que $1 \leq m \leq n$ et les coordonnées factorielles de l'objet initial i seront alors $\tilde{x}_{i1}, \ldots, \tilde{x}_{im}$. On peut facilement vérifier que c'est une représentation exacte de notre matrice initiale. En effet si l'on prend comme point de vue $a = \bar{x}$, on a alors :

$$\bar{\tilde{x}}_\beta = \sum_{i=1}^{n} f_i \tilde{x}_{i\beta} = 0 \qquad \forall \beta \in [1, \ldots, n-1]$$

Le centre de gravité de ces points est bien au centre de la représentation. On a également :

$$\sum_{\beta=1}^{n} (\tilde{x}_{i\beta} - \tilde{x}_{j\beta})^2 = \frac{K^a_{ii}}{f_i} + \frac{K^a_{jj}}{f_j} + \frac{K^a_{ij}}{\sqrt{f_i f_j}} = B^a_{ii} + B^a_{jj} - 2B^a_{ij}$$

Comme B^a est la matrice des produits scalaires, la loi des cosinus nous dit que $B^a_{ij} = \frac{1}{2}(D_{ia} + D_{ja} - D_{ij})$ et donc finalement :

$$\sum_{\beta=1}^{n} (\tilde{x}_{i\beta} - \tilde{x}_{j\beta})^2 = D_{ij}$$

Les distances relatives entre les points sont conservées, cette représentation est donc conforme à notre situation initiale.
On peut également remarquer que :

$$\mathbf{var}(\tilde{x}_\beta) = \sum_{i=1}^{n} f_i (\tilde{x}_{i\beta} - \bar{\tilde{x}}_\beta)^2 = \lambda_\beta$$

et

$$\sum_{\beta=1}^{n} \lambda_\beta = \Delta(a|f)$$

Les valeurs propres de K^a nous permettent ainsi de calculer le pourcentage d'inertie, qui est également la variance entre les points, projetée sur chacun des axes factoriels.

2.1.3 Le "distance trick" :

Dans cette partie, nous allons voir que la matrice des distances D peut se substituer à la matrice X pour effectuer une méthode factorielle. C'est une propriété très importante. En effet, cela signifie que les méthodes factorielles sont capables de "reconstituer" la position des points uniquement grâce aux distances relatives. C'est grâce à cette propriété que nos transformations de Schoenberg ne seront plus uniquement des transformations sur des matrices de distances, mais également des anamorphoses d'espaces euclidiens.
Soit notre point de vue a. On peut écrire ce dernier comme une combinaison linéaire de nos points, c'est à dire :

$$a = \sum_{i=1}^{n} \alpha_i x_i$$

On a alors le résultat suivant :

Proposition 2.1.9 *La matrice des produits scalaires B^a peut être obtenu par :*

$$B^a = -\frac{1}{2} H^a D (H^a)'$$

*où $H^a = (h^a_{ij})$ est la **matrice de centration** obtenue par $h^a_{ij} = \delta_{ij} - \alpha_j$.*

Preuve : Tout d'abord, remarquons que D_{ia} peut s'écrire comme :

$$D_{ia} = \sum_{k=1}^{n} \alpha_k D_{ik} - \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} \alpha_k \alpha_l D_{kl}$$

On a, par le théorème du cosinus :

$$B^a_{ij} = \frac{1}{2}(D_{ia} + D_{ja} - D_{ij})$$

$$= -\frac{1}{2} \left(D_{ij} + \sum_{k=1}^{n} \sum_{l=1}^{n} \alpha_k \alpha_l D_{kl} - \sum_{k=1}^{n} \alpha_k D_{ik} - \sum_{k=1}^{n} \alpha_k D_{jk} \right)$$

Ce qui correspond bien à $B^a = -\frac{1}{2} H^a D (H^a)'$ comme nous l'avons défini.

□

2.2 Les transformations de Schoenberg

Cette deuxième partie de la théorie s'intéresse aux transformations que nous avons nommées *transformations de Schoenberg*. Ces dernières sont en fait des fonctions qui s'appliquent aux composantes d'une matrice de distances $\tilde{D}_{ij} = \phi(D_{ij})$, avec certaines contraintes, transformant cette matrice en une nouvelle matrice de distances euclidiennes. Cette classe de fonctions a été découverte et étudiée par Schoenberg en 1938 [SCH1, SCH2].

Définition 2.2.1 *Une* ***transformation de Schoenberg*** *est une fonction* $\phi \in \mathcal{C}^\infty(\mathbb{R}^+, \mathbb{R}^+)$ *avec* $\phi(0) = 0$, $\phi^{(2r-1)}(D_{ij}) \geq 0$ *et* $\phi^{(2r)}(D_{ij}) \leq 0$ $\forall r \in \mathbb{N}^*$.

On voit par cette définition qu'une matrice de distances euclidiennes carrées va nécessairement être envoyé sur une autre matrice de distances euclidiennes carrées. Dans le reste de l'article, on notera $\phi(D)$ lorsque l'on applique $\phi(.)$ à toutes les composantes de D.

Dans son article de 1938, Schoenberg a prouvé [SCH1, p.828] que toutes ces transformations peuvent être données par la proposition suivante.

Proposition 2.2.2 *Soit une fonction* $\phi(D)$ *de* $\mathbb{R}^+$ *vers* $\mathbb{R}^+$. *Alors* $\phi(D)$ *est une transformation de Schoenberg si et seulement si on peut l'écrire sous la forme :*

$$\phi(D) = \int_0^\infty \frac{1 - exp(-\lambda D)}{\lambda} g(\lambda) d\lambda$$

où $g(\lambda)d\lambda$ *est une mesure non-négative sur* $[0, \infty)$ *tel que* $\int_1^\infty \frac{g(\lambda)}{\lambda} d\lambda < \infty$.

Grâce à cette proposition, nous pouvons construire toutes les transformations de Schoenberg possibles uniquement en donnant la fonction $g(\lambda)$. En voici quelques exemples :

Fonction $g(\lambda)$	Transformation $\phi(D)$
$\sigma(\lambda - r) \quad r > 0$	$\frac{1-\exp(-rD)}{r}$
$\theta(\lambda \leq \frac{\pi}{2})\lambda \sin(\lambda)$	$\frac{D(D+\exp(-\frac{\pi}{2}D))}{1+D^2}$
$\exp(-r\lambda) \quad r > 0$	$\ln(1 + \frac{D}{r})$
$\lambda \exp(-r\lambda) \quad r > 0$	$\frac{D}{r(r+D)}$
$\frac{r}{\Gamma(1-r)}\lambda^{-r} \quad 0 < r < 1$	D^r
Voir [BEG]	$\frac{D^r}{1+D^r}$

Ces fonctions de Schoenberg peuvent être considérée comme une anamorphose de l'espace Euclidien. En effet, nous avons vu que grâce aux méthodes factorielles, nous pouvions reconstituer une matrice de points X à partir de n'importe quelle matrice de distances euclidiennes D. Ainsi, la transformation de Schoenberg suivie d'une analyse factorielle agit en quelque sorte comme une fonction de $\mathbb{R}^p$ vers $\mathbb{R}^n$, plongeant ainsi les points dans un espace de dimension supérieure et modifiant la structure de l'espace.

$$X \longrightarrow D \xrightarrow{\phi} \tilde{D} \xrightarrow{MF} \tilde{X}$$

C'est cette modification qui va être étudiée dans la suite de l'article, d'une manière numérique ; les connaissances théoriques actuelles sur ces transformations restant pour le moment fortement méconnues.

2.3 Résumé

La marche à suivre suivante va être employée lorsque nous effectuerons une transformation de Schoenberg couplée avec une analyse factorielle.

1. De nos données de bases X et f, trouver la matrice des distances euclidiennes carrées : D.
2. Appliquer une transformation de Schoenberg $\phi(.)$ pour obtenir la matrice des distances euclidiennes carrées modifiée : $\tilde{D} = (\tilde{D}_{ij}) = (\phi(D_{ij}))$.
3. Choisir un point de vue a comme combinaison linéaire des x_i : $a = \sum_{i=1}^{n} \alpha_i x_i$ (généralement nous prendrons $a = \sum_{i=1}^{n} f_i x_i = \bar{x}$).
4. Calculer $H^a = (h_{ij}^a) = (\delta_{ij} \ - \alpha_j)$, puis $B^a = -\frac{1}{2} H^a \tilde{D} (H^a)'$ et finalement $K^a = (K_{ij}^a) = (\sqrt{f_i f_j} B_{ij}^a)$.
5. Décomposer spectralement $K^a = U \Lambda U'$ et ainsi obtenir les axes factoriels $u_{i\beta}$.
6. Obtenir les coordonnées factorielles de nos points $\tilde{x}_{i\beta} = \frac{\sqrt{\lambda_\beta}}{\sqrt{f_i}} u_{i\beta}$.
7. La compression s'effectue ensuite en ne gardant que les m premières coordonnées $\tilde{x}_{i1}, \ldots, \tilde{x}_{im}$ (généralement nous prendrons $m = 2$, afin de pouvoir afficher nos résultats sur un graphique en deux dimensions).

3 Simulations numériques

Pour étudier les effets de nos transformations de Schoenberg sur les données, nous allons procéder à de nombreuses simulations numériques. Ici nous allons travailler uniquement avec des données synthétiques, leur flexibilité permettant de mieux cerner dans quels cas nos transformations peuvent se révéler utiles. Plusieurs types de tests vont être effectués sur ces données.
Pour commencer, nous allons observer quelle forme adoptent les points après une transformation, lorsque ceux-ci sont répartis d'une manière particulière, par exemple s'ils forment une grille ou un anneau. La visualisation de nos nouveaux points pourra alors nous donner une idée de l'anamorphose qu'a subi l'espace.
Dans un deuxième temps, nous allons nous intéresser à la capacité de nos transformations à séparer différents groupes de points. Afin d'automatiser et d'optimiser le processus de recherche, nous réaliserons une cross-validation sur nos points avec la méthode de classification d'analyse discriminante [GNA] (voir la fonction correspondante pour l'algorithme), le taux de points correctement classés nous donnera une bonne idée de la séparation entre les groupes.
Notre troisième étude portera sur la capacité de nos transformations à bien répartir les points dans l'espace. Cette propriété nous permettrait d'avoir de meilleurs graphiques lorsque les données initiales ne s'affichent pas d'une manière satisfaisante, par exemple avec une groupement de points au centre, et quelques points sur les bords. De nouveau, pour faciliter la recherche, nous utiliserons un critère qui jugera de la bonne répartition des points dans l'espace. Ce dernier sera la p-valeur d'un test de Kolmogoroff-Smirnoff effectué sur la distance des points par rapport au centre de gravité de la configuration. Ce critère illustrera ainsi une certaine forme de normalité dans la répartition des points dans l'espace.
Pour finir, notre dernier test portera sur l'effet des transformations sur le centre de gravité de la configuration. Un algorithme spécial va être utilisé à cet effet (voir ce dernier dans la fonction correspondante). Cette approche semble effectivement donner des estimateurs plus robustes, s'approchant plus de la médiane que de la moyenne, et donc moins sensibles à des points "outliers".
Dans cet article, nous utiliserons le langage de programmation R pour faire notre étude, celui-ci permettant un affichage simple des données et incluant de nombreuses méthodes statistiques et mathématiques. Cette partie du rapport est consacrée à l'explication des fonctions principales que nous avons construites pour obtenir nos résultats, mais aucun code ne sera montré, ce dernier se trouvant dans l'annexe.

3.1 Fonctions de génération de données

En premier lieu, il faut bien entendu nous munir de plusieurs fonctions permettant de générer des données aléatoires suivant des lois particulières. Nous pourrons plus tard les combiner entre elles pour créer les données désirées.

Fonction de génération suivant une loi normale :

Nom : gen_norm_X

Entrées : n et p, des entiers. $(c_1, \ldots, c_p)$ et $(s_1, \ldots, s_p)$, des vecteurs de taille p.

Sortie : X, une matrice de dimensions $(n \times p)$.

Description : Cette fonction permet de créer une matrice X $(n \times p)$ de coordonnées. Si on considère les j-ème coordonnées des points, celles-ci suivront une loi normale de paramètre c_j et de variance s_j^2.

Fonction de génération suivant une loi de Cauchy :

Nom : gen_cauchy_X

Entrées : n et p, des entiers. $(c_1, \ldots, c_p)$ et $(s_1, \ldots, s_p)$, des vecteurs de taille p.

Sortie : X, une matrice de dimensions $(n \times p)$.

Description : Cette fonction permet également de créer une matrice X $(n \times p)$ de coordonnées. Mais cette fois-ci, les j-ème coordonnées des points, suivront une loi de Cauchy de paramètre c_j, s_j.

Fonction de génération suivant un anneau :

Nom : gen_anulus_X

Entrées : n et p, des entiers. r et s, des réels.

Sortie : X, une matrice de dimensions $(n \times p)$.

Description : Cette fonction crée une matrice X $(n \times p)$ de coordonnées où les points sont sur un plan, formant un anneau centré à l'origine. L'angle formé entre chaque point et l'origine est tiré d'une uniforme $U(0, 2\pi)$ et la distance à l'origine est tiré d'une normale $N(r, s^2)$.

Fonction de génération suivant une grille :

Nom : gen_gride_X

Entrées : n, un entier étant le carré d'un nombre. p, un entier.

Sortie : X, une matrice de dimensions $(n \times p)$.

Description : Cette fonction crée une matrice X $(n \times p)$ de coordonnées où les points sont sur un plan, formant une grille carrée centrée sur l'origine.

Fonctions de transformation de données

Ici sont regroupées les fonctions effectuant certaines transformations sur les données. La plus importante d'entre elles étant celle qui va nous permettre de faire une analyse en composante principale couplée avec une transformation de Schoenberg.

Fonction centrant et réduisant nos données :

Nom : cent_red

Entrées : X, une matrice de taille $(n \times p)$. $(f_1, \ldots, f_n)$, un vecteur de taille n, avec $\sum_{i=1}^{n} f_i = 1$ et $f_i > 0 \ \forall i$.

Sortie : X_c, une matrice de dimensions $(n \times p)$.

Description : Cette fonction ne fait que centrer et réduire notre matrice initiale X, chaque point i ayant un poids relatif f_i. Pour cela on utilise la méthode suivante. Le centre de gravité c de la configuration est trouvé grace à $c_j = \sum_{i=1}^{n} f_i X_{ij}$. On peut donc trouver la matrice centrée Y avec $Y_{ij} = X_{ij} - c_j$, puis on la réduit en calculant les $s_j^2 = \sum_{i=1}^{n} f_i Y_{ij}^2$ et en prenant $(X_c)_{ij} = Y_{ij}/s_j$.

Fonction de passage de X à D :

Nom : X_to_D

Entrées : X, une matrice de taille $(n \times p)$.

Sortie : D, une matrice de dimensions $(n \times n)$.

Description : Cette fonction transforme la matrice de coordonnées X en la matrice des distances relatives D par calcul direct $D_{ij} = \sum_{k=1}^{p} (x_{ik} - x_{jk})^2$.

Fonction appliquant une transformation de Schoenberg et une méthode factorielle :

Nom : schoen_ACP

Entrées : D, une matrice $(n \times n)$. $(f_1, \ldots, f_n)$, un vecteur de taille n. $\phi(.)$, une fonction. r, un réel.

Sortie : X_b, une matrice de dimensions $(n \times n)$.

Description : C'est dans cette fonction que nous allons faire ce que nous avons appréhendé dans la théorie. Nous allons pour commencer appliquer la transformation $\phi(.)$, définie au préalable, sur la matrice des distances D. Cette fonction prend également comme entrée le paramètre de la fonction de Schoenberg, r. Ensuite nous allons trouver les coordonnées de basses dimensionnalités X_b en faisant une analyse factorielle comme vu dans la théorie.

Fonction permettant d'obtenir les valeurs propres après une transformation de Schoenberg et une méthode factorielle :

Nom : inert_schoen_ACP

Entrées : D, une matrice $(n \times n)$. $(f_1, \ldots, f_n)$, un vecteur de taille n. $\phi(.)$, une fonction. r, un réel.

Sortie : $(\lambda_1, \ldots, \lambda_n)$, un vecteur de longueur n.

Description : Cette fonction est identique à la précédente, seulement elle retourne les valeurs propres de la matrice K^a transformée. Elle nous permettra d'obtenir le pourcentage d'inertie exprimé sur chaque coordonnée factorielle.

3.2 Fonction d'affichage

Cette partie ne comprend qu'une seule fonction, permettant d'afficher nos points sur un graphique.

Nom : display

Entrées : X, une matrice $(n \times p)$. d_1, d_2, deux entiers inférieurs à p. $(g_1, \ldots, g_n)$, un vecteur d'entiers de dimension n.

Sortie : Un graphique en deux dimensions.

Description : Les points de X vont être affichés sur le graphique, en ne prenant compte que des dimensions d_1 et d_2. Le vecteur $(g_1, \ldots, g_n)$ nous permet d'assigner différents groupes aux points, les colorant ainsi de différentes manières. Cette fonction comprend également de nombreux autres paramètres permettant de personnaliser l'affichage (voir le code).

3.3 Fonctions d'évaluation

Cette partie comprend les fonctions permettant l'évaluation des données suivant les critères que nous avons vus au début de cette section.

Fonction de cross-validation :

Nom : cross_valid

Entrées : X_{ext}, une matrice $(n \times p + 2)$. s, un entier. $(d_1, \ldots, d_m)$, un vecteur d'entiers de dimension $m \leq p$.

Sortie : q, une proportion.

Description : X_{ext} est notre matrice de points à laquelle on a ajouté dans les deux dernières colonnes le poids et le groupe de chaque point. La fonction effectue ensuite une cross-validation, en séparant les points en s parties, grâce à la méthode de classification d'analyse discriminante. Cette dernière fonctionne suivant l'algorithme suivant :

- On calcule les centres de gravité de chaque groupe g, c^g définis par $c_k^g = \sum_{i \in g} f_i^g x_{ik}$ où $f_i^g = \frac{f_i}{\sum_{j \in g} f_j}$.
- On va ensuite classer les points suivant leur proximité avec les centres. C'est à dire i est assigné à $\arg\min_g(D_{ic^g})$.

Cette classification se fait en prenant uniquement en compte les dimensions choisies $(d_1, \ldots, d_m)$. Cette fonction nous retourne q, la proportion moyenne de points correctement classés.

Fonction test de Kolmogorov-Smirnoff :

Nom : ks_test

Entrées : X, une matrice $(n \times p)$. $(d_1, \ldots, d_m)$, un vecteur d'entiers de dimension $m \leq p$.

Sortie : s, un réel positif.

Description : Cette fonction effectue le test de Kolmogoroff-Smirnoff sur la distance des points à l'origine en prenant uniquement en compte les dimensions $(d_1, \ldots, d_m)$ choisies.

Fonction changement de centroïde :

Nom : new_centroide

Entrées : D une matrice $(n \times n)$. $(f_1, \ldots, f_n)$ un vecteur de dimension n, $\phi'(.)$ une fonction. r un réel.

Sortie : $(\alpha_1, \ldots, \alpha_n)$ un vecteur de dimension n.

Description : Cette fonction recherche le centroïde qui minimise l'inertie de la configuration après que celle-ci ait été modifiée par la transformation de Schoenberg, de paramètre r et dont la dérivée est $\phi'(.)$. La configuration initiale est donnée par, D la matrice des distances, et $(f_1, \ldots, f_n)$ le poids des points. Pour trouver ce centroïde, c'est l'algorithme itératif suivant qui est utilisé (nous avons volontairement fixé le nombre de boucles à 10).

1. Pour commencer on pose $\alpha_i = f_i$
2. On calcule D_{ia} avec $D_{ia} = \sum_{k=1}^{n} \alpha_k D_{ik} - \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} \alpha_k \alpha_l D_{kl}$.
3. Nos nouveaux α_i seront donnée par $\alpha_i = \frac{f_i \phi'(D_{ia})}{\sum_{k=1}^{n} f_k \phi'(D_{ka})}$
4. On revient au point 2.

Le centroïde résulant est donc donné par sa combinaison linéaire par rapport aux points. A noter que nous l'afficherons ensuite dans la configuration initiale et non dans la configuration transformée.

4 Résultats

Nous avons vu plusieurs exemples de fonctions de Schoenberg dans la théorie, mais l'exploration de ces fonctions étant un sujet très vaste, les résultats dans cet article vont se concentrer uniquement sur trois d'entre elles :

$$\begin{aligned} \phi_1(D) &= \tfrac{1-\exp(-rD)}{r} && r > 0 \\ \phi_2(D) &= \tfrac{D}{r(r+D)} && r > 0 \\ \phi_3(D) &= D^r && 0 < r < 1 \end{aligned}$$

où r est le paramètre de notre fonction.

Les résultats sont divisés en cinq sections. Les sections 4.1 et 4.2 montrent les effets de ces transformations sur un grille et un anneau constitué de 100 points. La section 4.3 contient les résultats sur la classification par la méthode d'analyse discriminante de 3 groupes de 50 points chacun. Cette classification sera évaluée grâce à une cross-validation en séparant nos points en 10 parties de manière aléatoire, puis en classifiant une de ces parties en prenant les 9 autres comme référence. La section 4.4 se penchera sur la visibilité d'un jeu de donnée, grâce à une évaluation avec un test de Kolmogorov-Smirnoff sur la distance des points au centre. Les coordonnées initiales de nos points seront tirées d'une loi de Cauchy, celle-ci générant souvent des valeurs extrêmes. Pour finir, la section 4.5 illustrera brièvement les effets de ces transformations sur le centre de gravité de la configuration initiale.
Il est important de noter que toutes nos données seront centrées et réduites avant d'être transformées, éliminant ainsi le risque d'obtenir des résultats différents à cause d'un changement de l'unité utilisée dans les données.

4.1 Effets sur une grille

Voici les effets de ces transformations sur une grille constituée de 100 points dans $\mathbb{R}^2$.

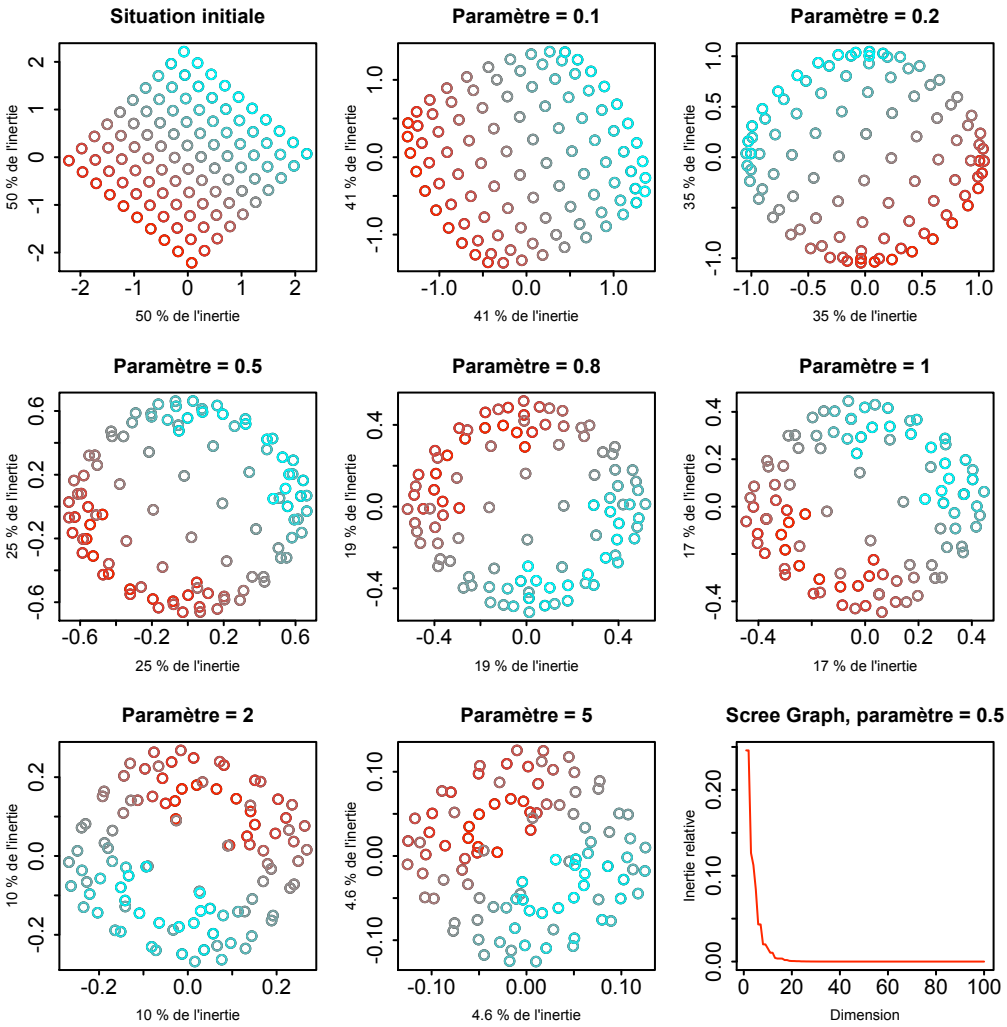

FIG. 1 – Résultats pour $\phi_1(D) = \frac{1-\exp(-rD)}{r}$

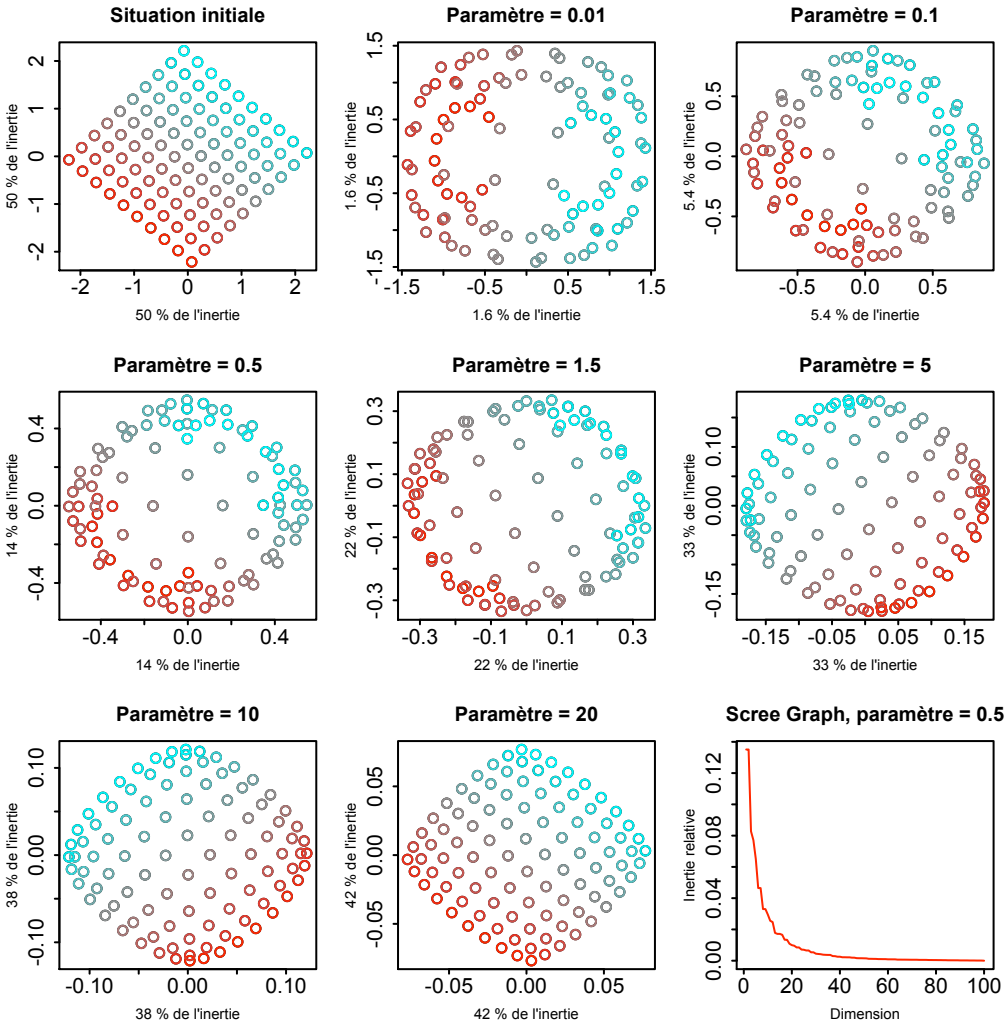

FIG. 2 – Résultats pour $\phi_2(D) = \frac{D}{r(r+D)}$

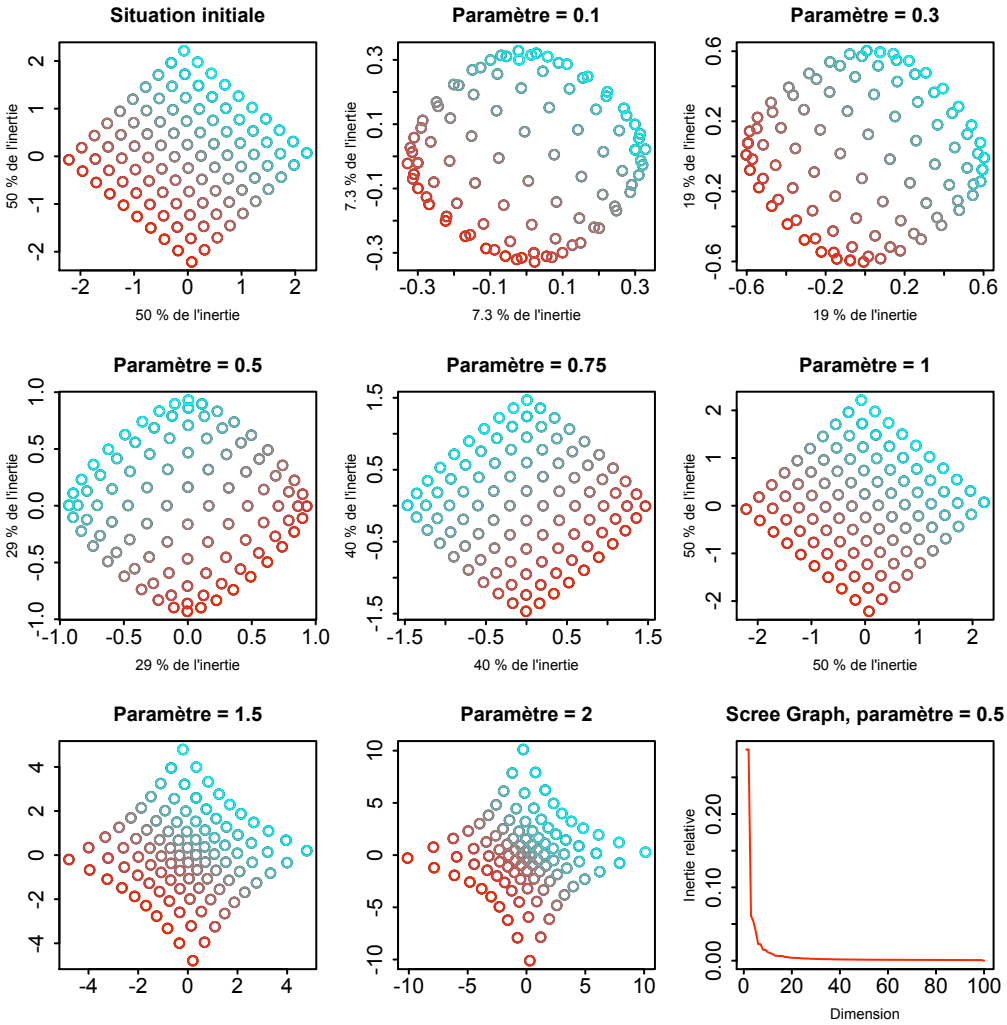

FIG. 3 – Résultats pour $\phi_3(D) = D^r$

Les effets de telles transformations sur une grille sont intéressants. En premier lieu, intéressons-nous à l'inertie exprimée sur les deux premières composantes factorielles. On voit que plus on s'éloigne du paramètre représentant l'identité, c'est à dire $r = 0$ pour ϕ_1, $r \to \infty$ pour ϕ_2 (les points reprennent la même structure, malgré la diminution de l'échelle) et $r = 1$ pour ϕ_3, plus l'inertie et faible dans les deux premières coordonnées, les lignes sur lesquelles les points sont alignés devenant ainsi de plus en plus courbées. Les "Scree Graphs" visent à montrer la répartition de l'inertie sur les différents axes factoriels avec $r = 0.5$.
Pour ϕ_1, il semblerait qu'il y ait un effet "loupe" sur les données lorsque $r < 0.5$. Les points au centre deviennent de plus en plus écartés, alors que ceux sur les bords se répartissent le long d'un cercle. Cet effet est de plus en plus évident lorsqu'on augmente le paramètre, jusqu'aux environs de $r = 0.5$. Pour des valeurs supérieures à 0.5, cet effet disparaît petit à petit pour faire apparaître une structure plus compliquée à comprendre. Comme application de cet effet "loupe", on pourrait penser par exemple à un amas de points sur le centre de gravité qui pourrait être séparé, augmentant ainsi la visibilité de notre projection.
Cet effet "loupe" se retrouve avec ϕ_2 mais dans le sens inverse. Des valeurs du paramètre très proches de zéro nous donnent le même effet qu'avec ϕ_1 pour de fortes valeurs, et vice-versa. Remarquons que la grille retrouve sa structure originale pour de fortes valeurs du paramètre, mais que sa taille diminue. En effet si $r \to \infty$ alors $\phi_2 \to 0$, la distance entre les points diminuera de plus en plus. Pour ϕ_3, on retrouve notre effet "loupe" pour $r < 1$. Si on augmente encore le paramètre, on observe une sorte de regroupement de points par "courbes de niveau", avec un effet d'écartement sur les diagonales. Mais il faut noter que ce ne sont plus là des transformations de Schoenberg : ces illustrations servent uniquement à montrer ce qu'il se passe lorsque l'on dépasse la valeur du paramètre autorisée.

4.2 Effets sur une anneau

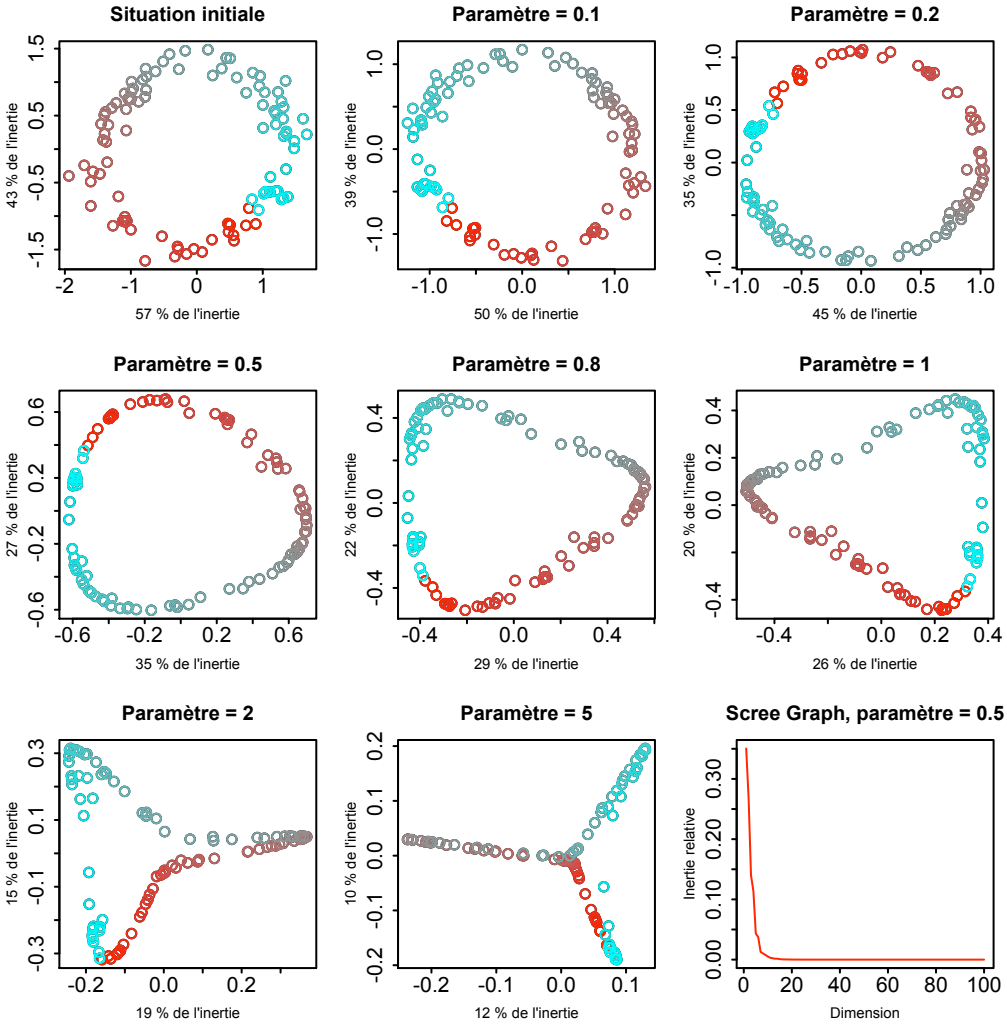

FIG. 4 – Résultats pour $\phi_1(D) = \frac{1-\exp(-rD)}{r}$

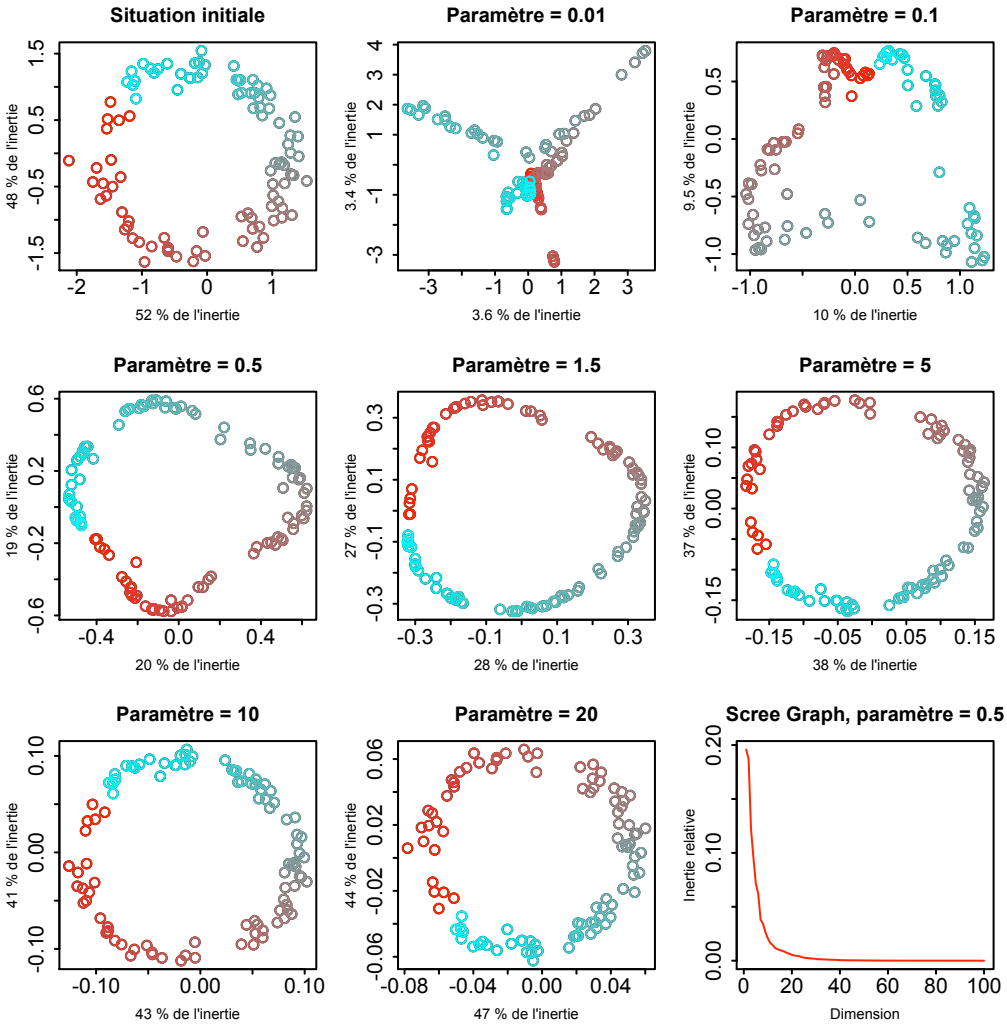

FIG. 5 – Résultats pour $\phi_2(D) = \frac{D}{r(r+D)}$

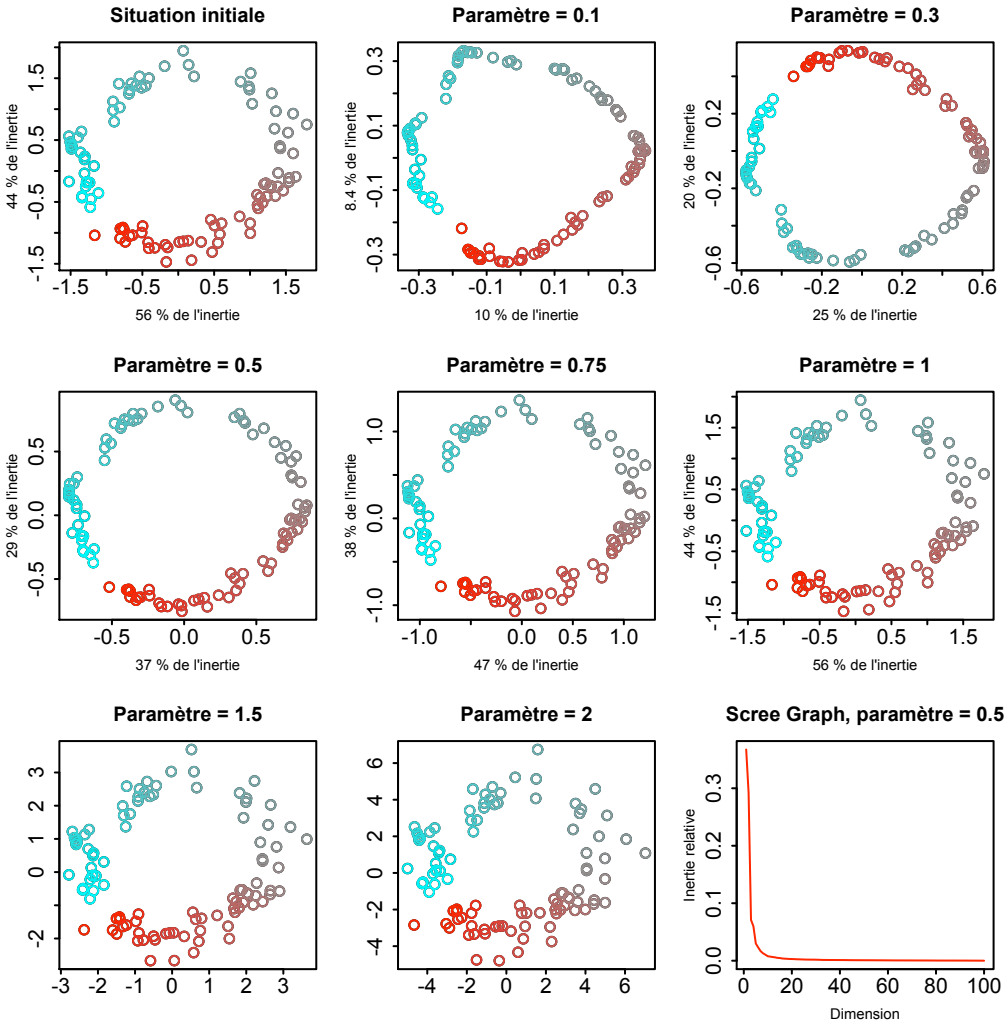

FIG. 6 – Résultats pour $\phi_3(D) = D^r$

Mise à part la structure parfois étrange que prennent les données, les trois transformations ont un point commun : pour certaine valeur du paramètre ($\phi_1(D)$ avec $r \geq 0.5$, $\phi_2(D)$ avec r aux alentours de 1.5 et $\phi_3(D)$ avec $r \leq 0.3$), elles ont tendance à aligner les points, formant ainsi une courbe de points sans épaisseur. Le nuage initial disparaît donc pour laisser place à une courbe. C'est un phénomène qu'il serait intéressant d'étudier, car il pourrait peut-être permettre de développer une nouvelle forme de compression des données. En effet, mise à part la structure étrange de la courbe, les points sont, en quelque sorte, définis par leur place dans un espace à une seule dimension. Il serait intéressant de chercher à retrouver les points initiaux uniquement grâce à la structure de la courbe et la position des points sur celle-ci. Tout ceci reste bien entendu très hypothétique, mais des applications d'un tel phénomène pour l'analyse de données peuvent effectivement exister.

4.3 Classification

Nous allons maintenant tester les effets de nos transformations lorsqu'on les applique avant une classification. Nos données vont comporter trois groupes de 50 points dans $\mathbb{R}^{10}$. Les points de chaque groupe ont des coordonnées issues d'une loi normale de même centre, mais ayant respectivement pour le groupe 1, 2 et 3 une variance de 1, 4 et 25. La classification va ensuite se faire par rapport au deux premières coordonnées de nos points : on ne classe que ce que l'on "voit" sur le graphique.

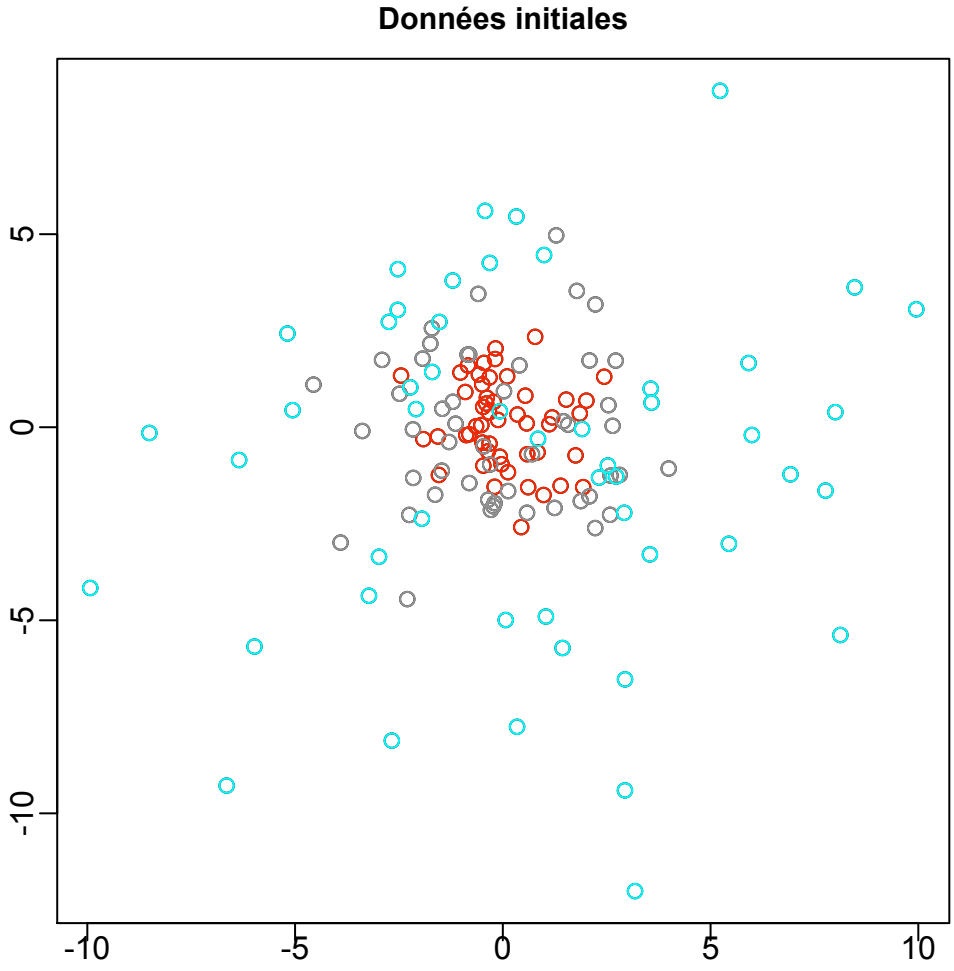

FIG. 7 – Nuage de 150 points formant 3 groupes avec variance 1, 4 et 25

Si l'on fait cross-validation de ces données avec la méthode d'analyse discriminante, le résultat est de 38% de points correctement classés, un résultat très mauvais. Rien d'étonnant : les centres de gravité de chaque groupe sont quasiment confondus et la méthode de classification d'analyse discriminante classe les différents points suivant la distance entre ces points et ces centres.

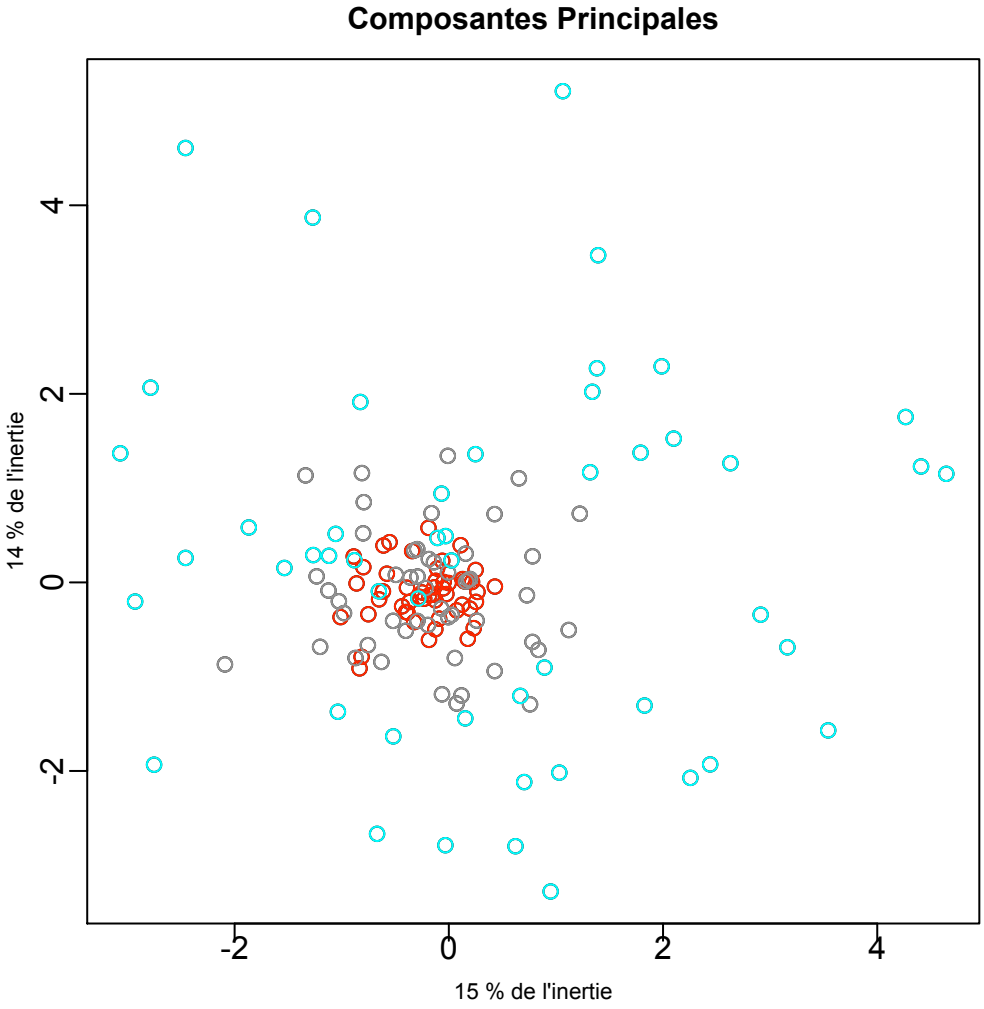

FIG. 8 – Données après une décomposition en composantes principales

En effectuant une décompostion en composantes principales, le résultat de notre cross-validation passe à 42% de points correctement classés. C'est déjà un peu mieux, mais loin d'être suffisant. Voyons maintenant les résulats avec nos fonctions de Schoenberg

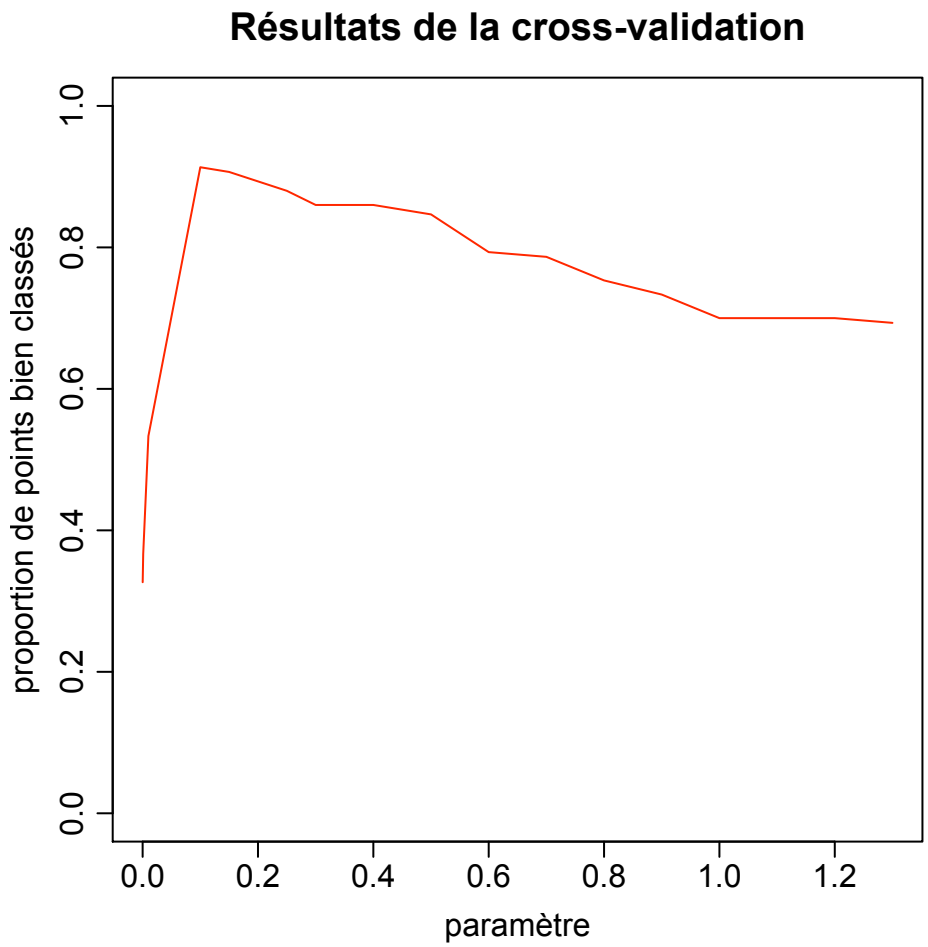

FIG. 9 – Résultats de la cross-validation pour $\phi_1(D) = \frac{1-\exp(-rD)}{r}$

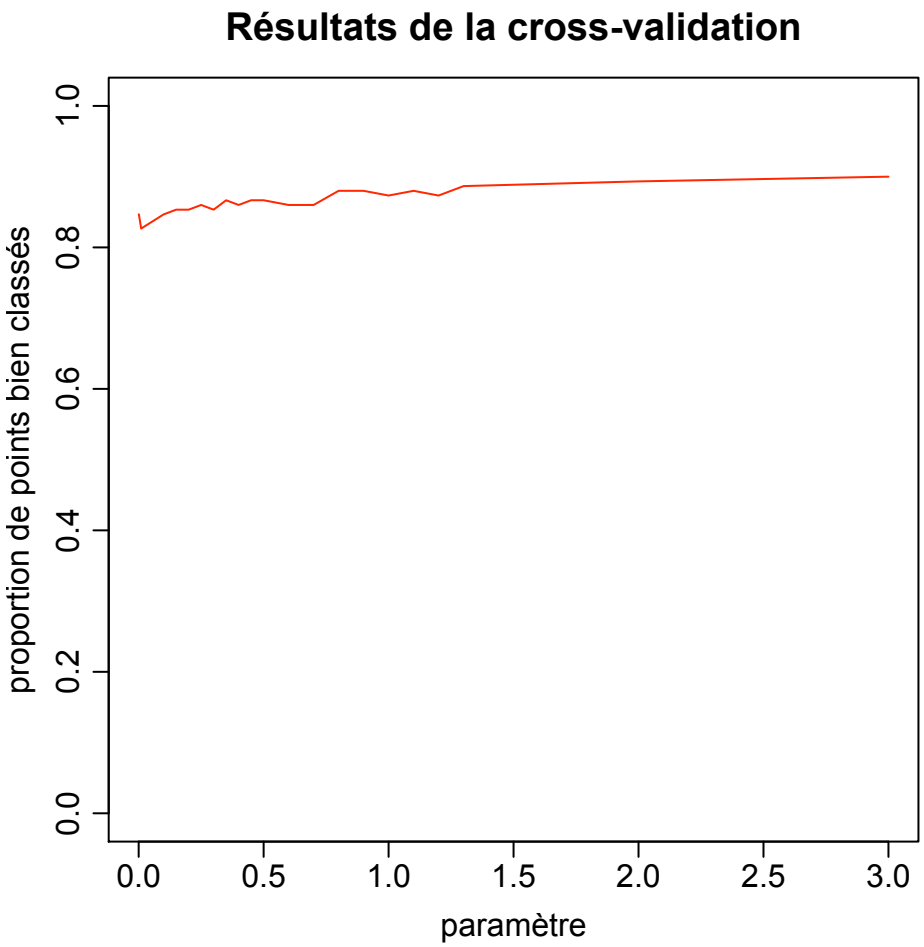

FIG. 10 – Résultats de la cross-validation pour $\phi_2(D) = \frac{D}{r(r+D)}$

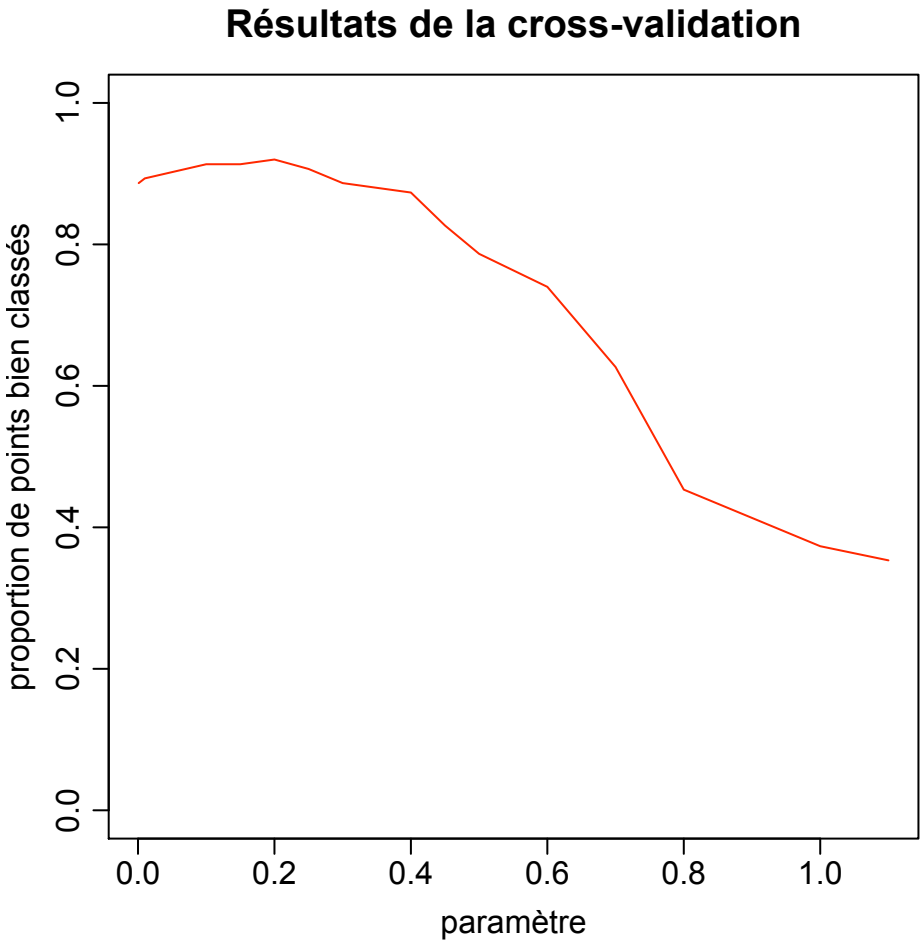

FIG. 11 – Résultats de la cross-validation pour $\phi_3(D) = D^r$

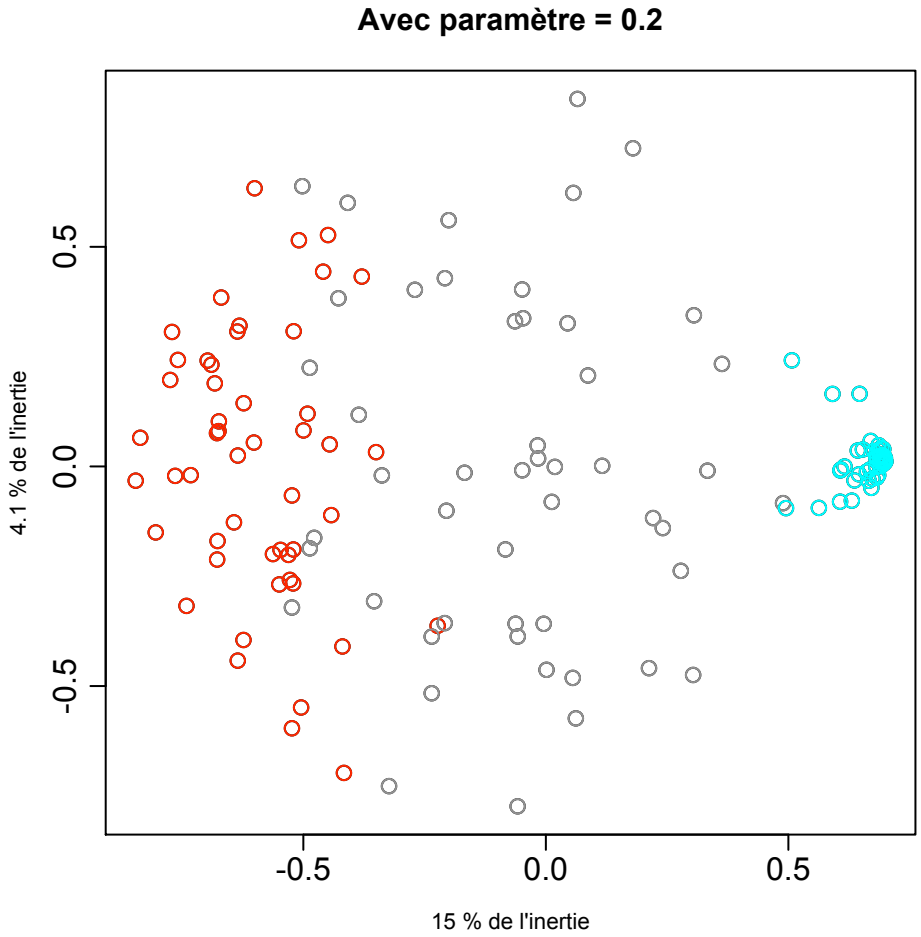

FIG. 12 – Points pour $\phi_1(D) = \frac{1-\exp(-0.2D)}{0.2}$, 93% de points bien classés

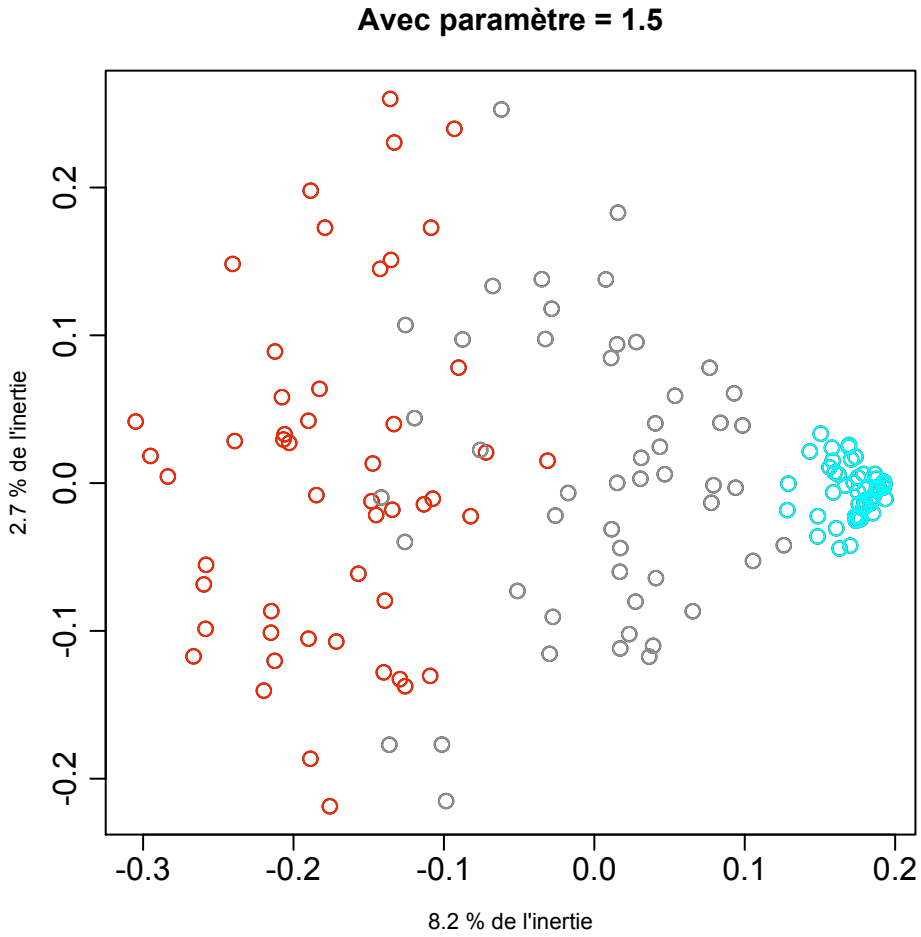

FIG. 13 – Points pour $\phi_2(D) = \frac{D}{1.5(1.5+D)}$, 87% de points bien classés

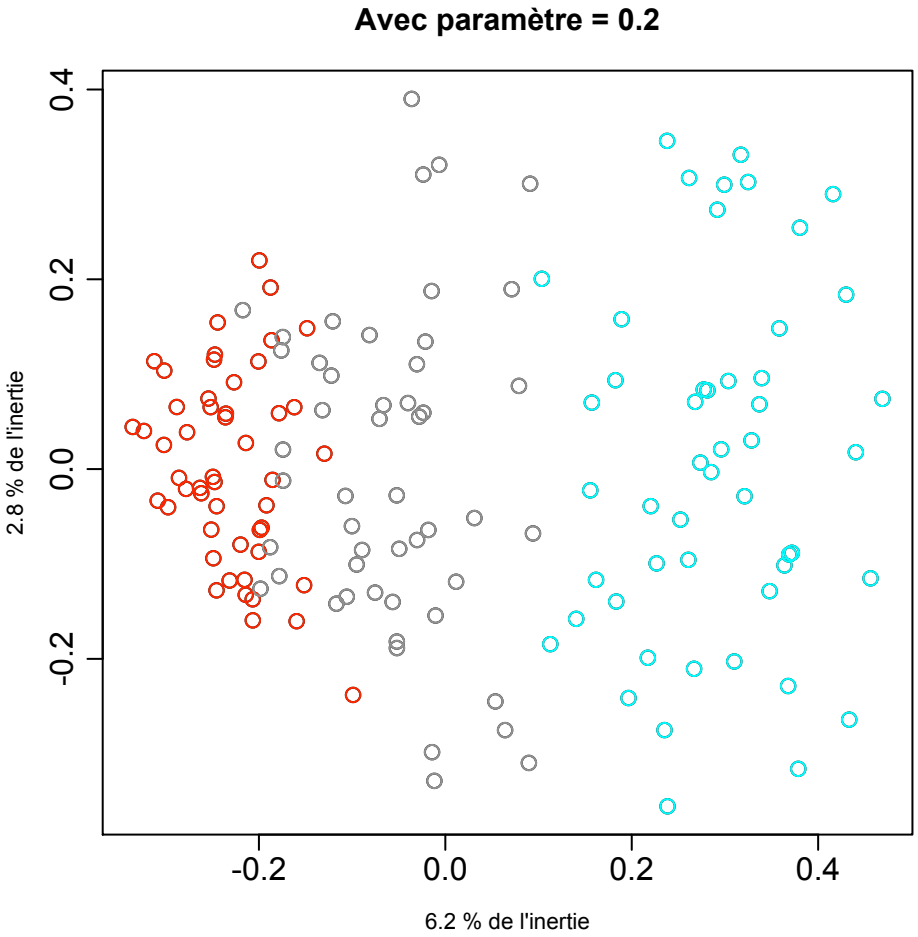

FIG. 14 – Points pour $\phi_2(D) = D^{0.2}$, 91% de points bien classés

Dans les trois cas, on peut trouver une valeur de r pour laquelle la cross-validation donne des résultats aux alentours des 90%, ce qui est un résultat nettement supérieur. Dans le cas de ϕ_1, le taux de points correctement classés augmente très rapidement jusqu'à ce que le paramètre soit égal à 0.2. Ensuite il diminue lentement, au fur et à mesure que le paramètre augmente. Pour ϕ_2, les résultats sont très bons pour des valeurs du paramètre entre 0 et 3 ; ensuite, le résultat chute lentement. Nous avions vu grâce à la grille que les points retrouvent leur structure initiale pour des valeurs du paramètre élevées, ce qui explique cette chute. Pour ϕ_3, on obtient de bons résultats avec un r petit, puis on revient à la situation initiale lorsque le paramètre s'approche de 1.

Le jeu de données que nous avons testé ici est seulement un exemple parmi d'autres, et l'on pourrait se demander, lors d'une classification, sur quels autres jeux de données ces transformations ont un effet bénéfique. Ayant testé plusieurs autres données, nous avons remarqué que les transformations sont surtout utiles lorsqu'il y a une différence de variance entre les différents groupes. Les différences entre les lois et les centres de gravité que suivent les points des différents groupes n'ont pas beaucoup d'influence. Cet exemple à été choisi comme un des cas les plus critiques : en effet, les centres sont confondus, la loi est la même et les variances sélectionnées sont telles que les résultats reflètent de manière optimale l'amélioration permise par les transformations.

Ces résultats sont néanmoins très étranges. En effet, si l'on regarde parmi les points du groupe comportant la variance la plus faible, on devrait trouver des points appartenant aux deux autres groupes. Pourtant, après une transformation de Schoenberg, ces points sont complètement séparés (les points des groupes 1 et 3 sont même très distincts). Il semblerait que ces transformations permettent en quelque sorte de déceler la variance intra-groupe, et de l'exprimer dans la modification de l'espace. Ce phénomène mériterait d'être approfondi, car trouver une explication à cette séparation semble, en l'état, bien compliqué.

4.4 Le test de Kolmogorov-Smirnoff

L'aptitude de nos transformations à rendre les données plus "lisibles" peut être observé comme suit : nous avons généré 30 points dans $\mathbb{R}^{10}$, dont les coordonnées suivent une loi de Cauchy centrée en 0 et avec un paramètre d'échelle de 4. Nous avons sélectionné cette loi car elle fournit quelques points extrêmes, très éloignés du centre de gravité. Si l'on affichait ces données dans un graphique avec une échelle normale sur les axes, un regroupement de points au centre du graphique rendrait ce dernier peu "lisible". Nous allons observer les effets des transformations sur ces données.

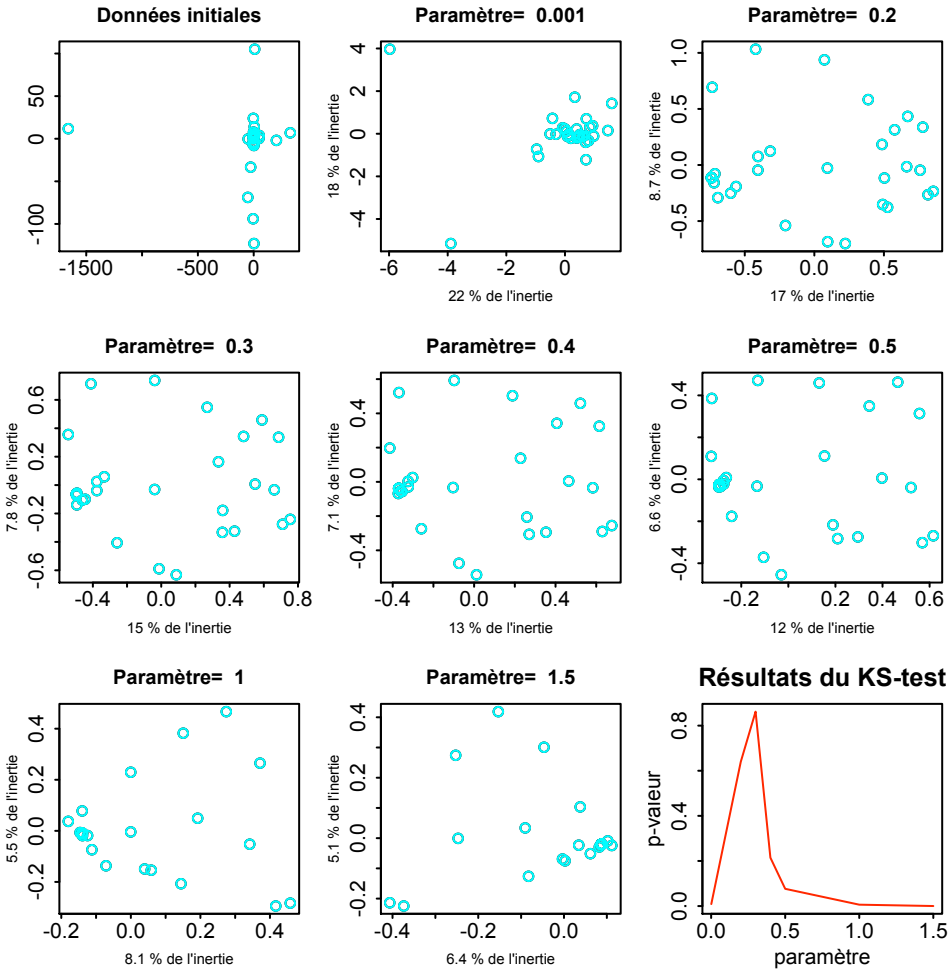

FIG. 15 – Résultats du test de Kolmogorov-Smirnoff pour $\phi_1(D) = \frac{1-\exp(-rD)}{r}$

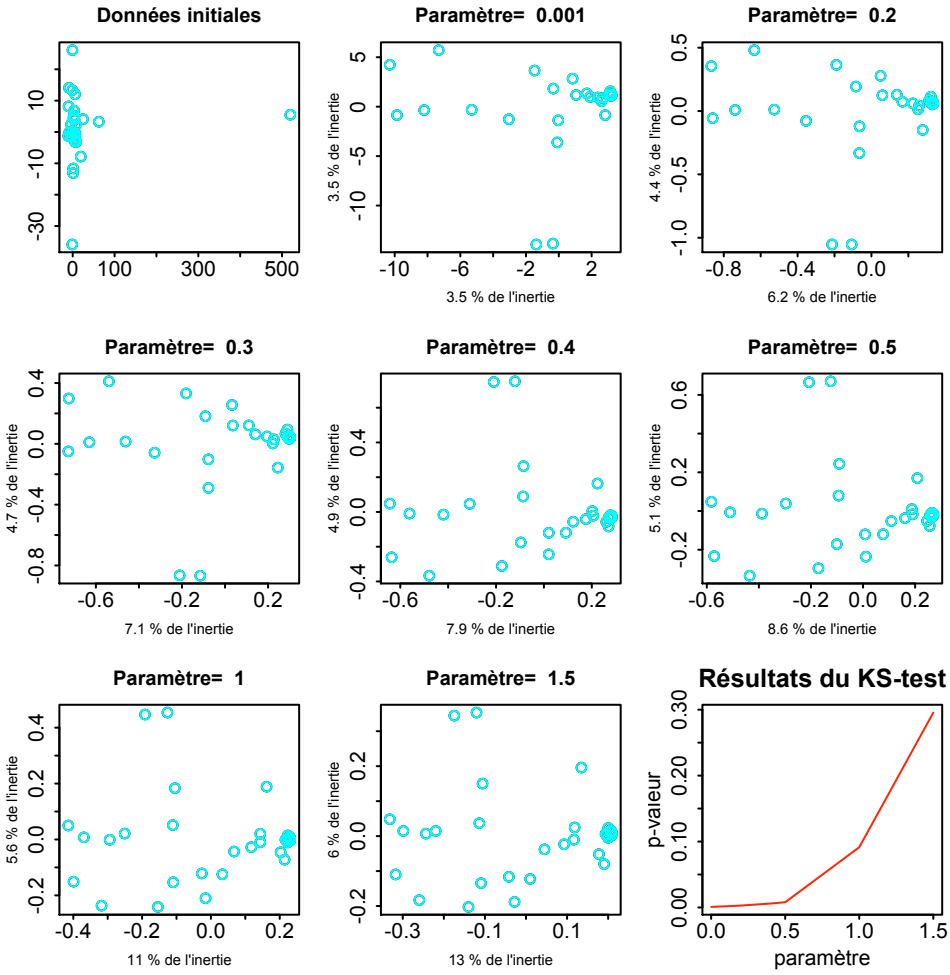

FIG. 16 – Résultats du test de Kolmogorov-Smirnoff pour $\phi_2(D) = \frac{D}{r(r+D)}$

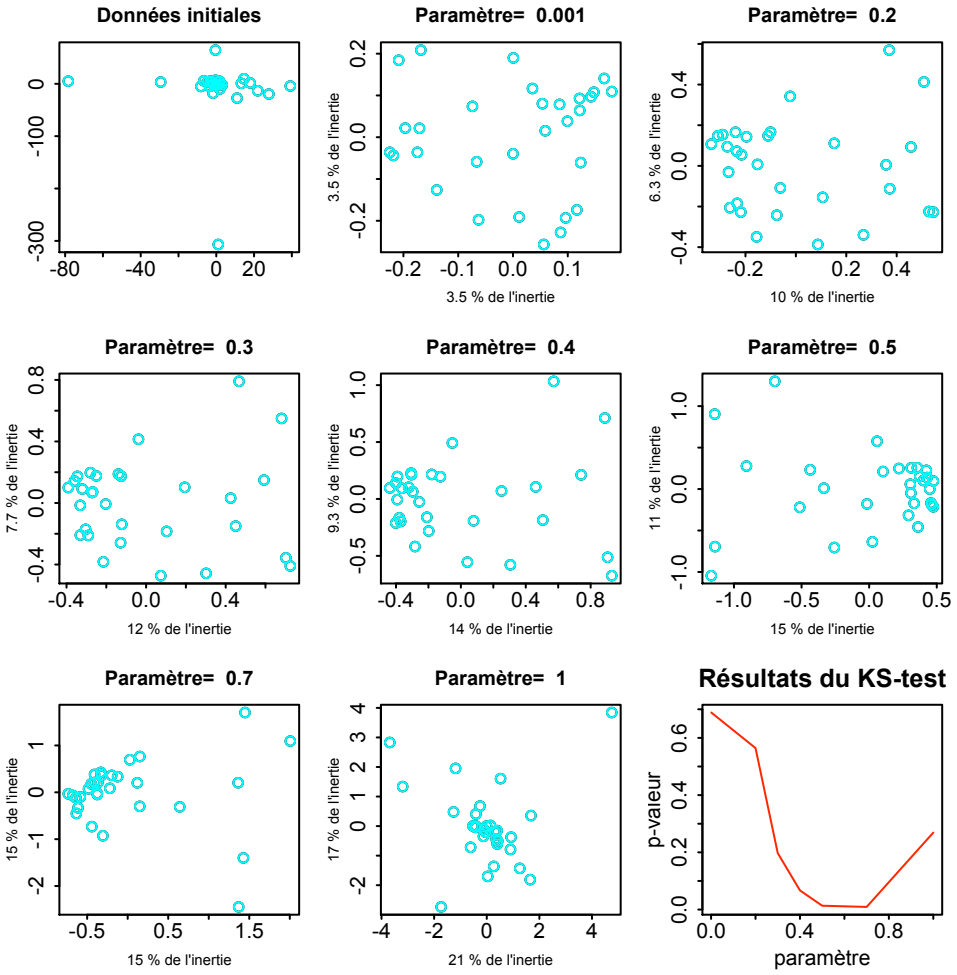

FIG. 17 – Résultats du test de Kolmogorov-Smirnoff pour $\phi_3(D) = D^r$

L'effet recherché ici est le fameux effet "loupe" que nous avions remarqué dans la section 4.1. En effet les points centraux se séparent les uns des autres tandis que ceux dans la périphérie se rapprochent du centre. Les meilleurs résultats sont obtenus avec $a = 0.4, 1.5$ et 0.001 respectivement pour ϕ_1, ϕ_2 et ϕ_3. Cependant, il est difficile de savoir si la "structure" de la configuration est bien respectée, l'affichage des points n'aurait alors plus aucun sens. Ce test sert surtout à illustrer une autre application possible des transformations de Schoenberg.

4.5 Changement de centroïde

Lors d'une transformation de Schoenberg, le centre de gravité se modifie. Observons 50 points dans $\mathbb{R}^{10}$ répartis sur un anneau de rayon 1 couplé avec 20 points "outliers", ayant toutes leurs coordonnées égales à 3. Le centroïde initial va être affiché en bleu foncé, et les centroïdes modifiés par un dégradé du bleu clair au rouge, au fur et à mesure que le paramètre augmente de 0.2 à 10, par étapes de 0.2.

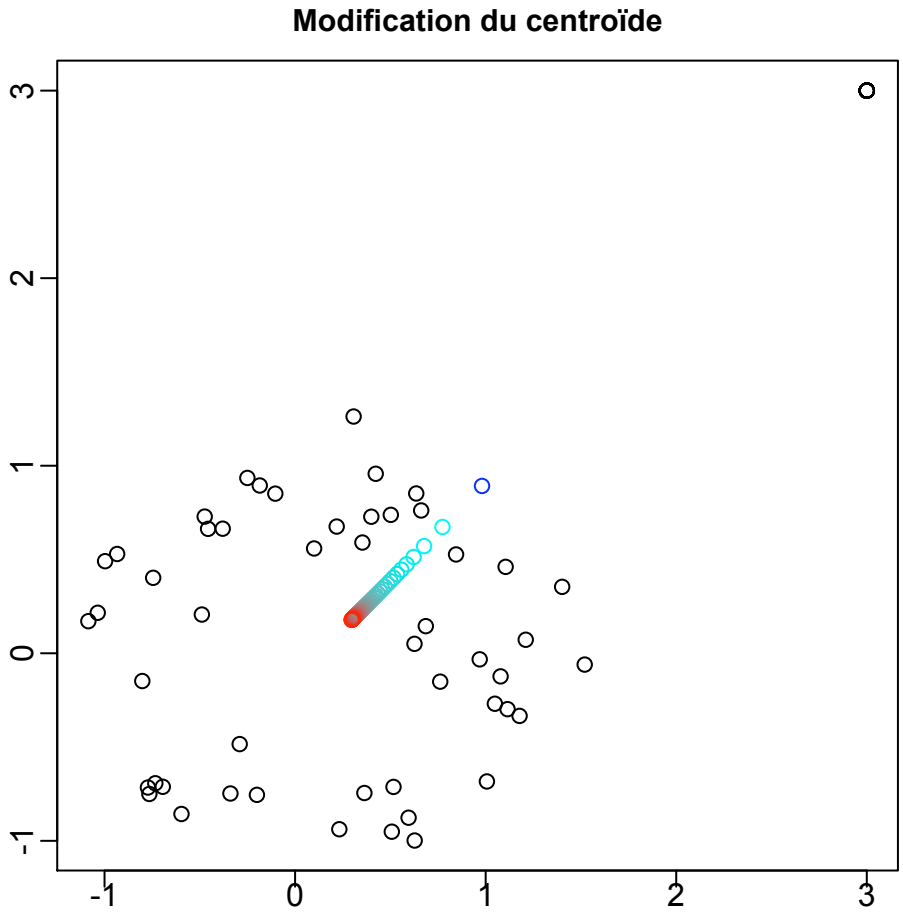

FIG. 18 – Résultats pour $\phi_1(D) = \frac{1-\exp(-rD)}{r}$

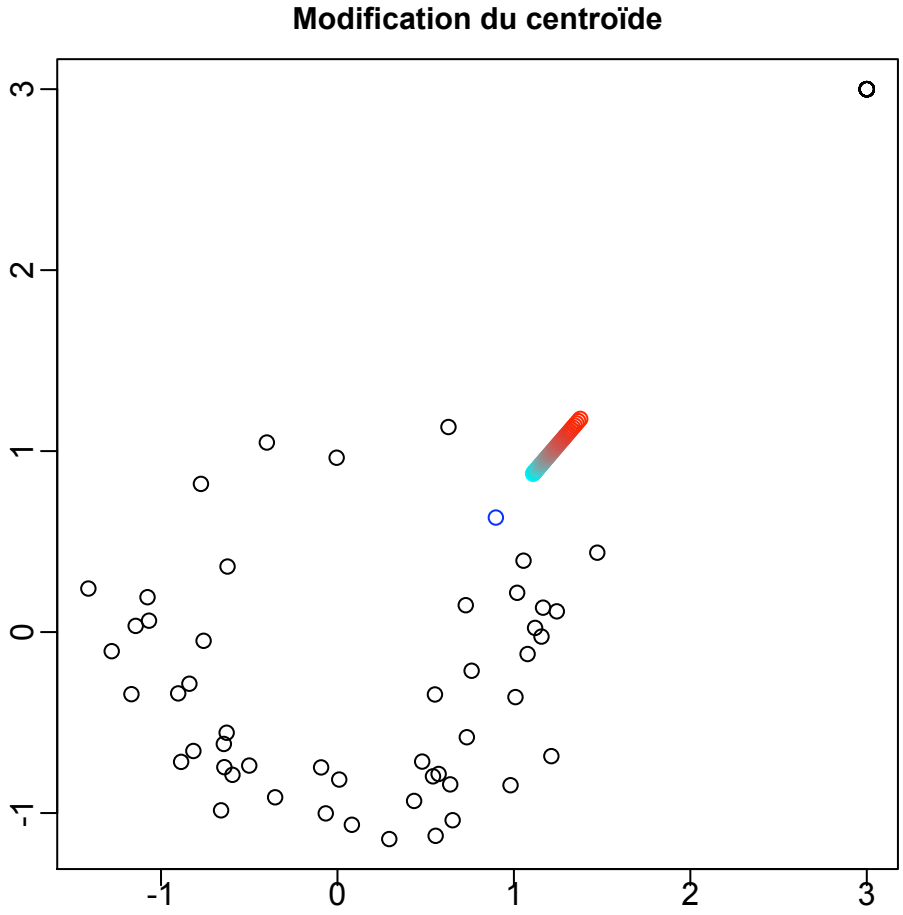

FIG. 19 – Résultats pour $\phi_2(D) = \frac{D}{r(r+D)}$

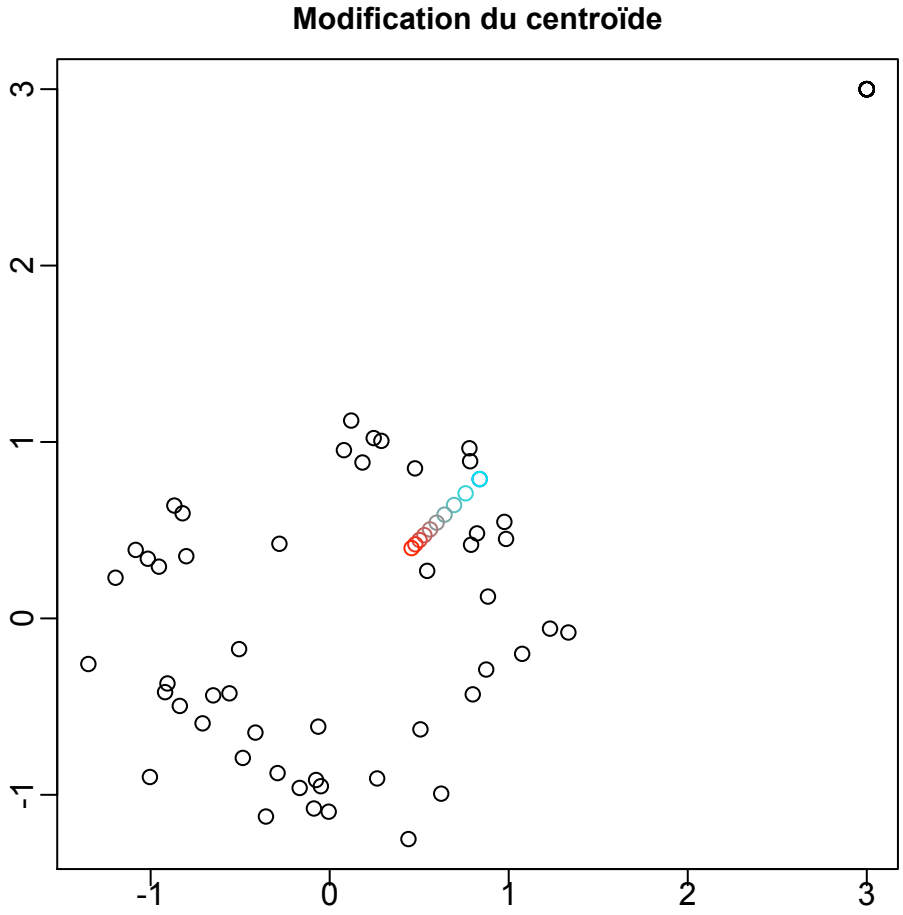

FIG. 20 – Résultats pour $\phi_3(D) = D^r$

Dans le premier et le dernier cas, on voit que plus le paramètre évolue, plus on se rapproche du centre de gravité "robuste" de la configuration, comme si les 20 points "outliers" n'existaient plus. On remarque que cette convergence est plus rapide pour ϕ_1 que pour ϕ_3. Pour ce qui est de ϕ_2, les résultats sont surprenants. En effet, le centre se déplace lentement vers les points "outliers". Cet effet est certainement dû au fait que le centroïde est calculé par un processus itératif, qui prend comme point initial le centre de gravité de la configuration non transformée. Il se peut qu'il ait donc plusieurs points de convergence, ϕ_2 nous amenant dans un autre de ces "puits" en partant du centroïde original. Quoi qu'il en soit, cette méthode peut être très utile pour créer de nouveaux estimateurs, appelés "M-estimators", plus robuste que la moyenne.

5 Conclusions

Bien que connues sous une forme dans le milieu du "statistical learning" [DON], les méthodes effectuant une anamorphose d'un espace dans un espace de dimension supérieure sont encore très méconnues dans le monde de l'analyse de données. Pourtant, comme nous l'avons montré dans ce rapport, elles gagnent vraiment à être connues. Bien que leurs effets sur l'espace des points reste encore compliqué à appréhender d'une manière formelle, on ne peut nier leur utilité dans certains cas, comme dans l'exemple que nous avons vu, où les transformations arrivent à séparer des points appartenant à des groupes différents se chevauchant. On pourrait se demander si le passage dans un espace de dimension supérieur permet en quelque sorte de faire ressortir d'une manière explicite des caractéristiques inhérentes aux données, difficile à cerner au premier abord.

Pour ce qui est des différentes transformations de Schoenberg données en exemple dans cet article, il semblerait finalement que celles-ci soient assez similaires si l'on modifiait leur paramètre. On retrouve souvent les même résultats pour deux transformations différentes, mais avec d'autres valeurs de r. Cependant, il se peut que, dans certains cas, de subtiles changements apparaissent entre deux transformations. Nous n'avons pas réussi tirer de conclusions définitives jusque-là, malgré les nombreux tests effectués : en effet, l'exploration de ces transformations est un travail gigantesque. Il est non seulement possible de modifier le paramètre et la transformation que l'on applique, mais également le type de données, l'unité de mesure et le point de vue initial a. Toutes ces modifications nous donneraient à chaque fois un résultat différent. N'oublions pas non plus que cette recherche se base sur différents critères, qui varient selon que l'on recherche à séparer les données, à les rendre plus lisible où à courber l'espace d'une certaine manière. Bien entendu, l'exploration numérique que nous avons effectuée dans ce rapport n'est finalement que le début de l'étude sur ces transformations de Schoenberg, la finalité étant de trouver des résultats théoriques expliquant les phénomènes observés, posant ainsi un cadre théorique solide et cernant d'une manière plus pertinente les cas concrets où ces transformations devraient être utilisées.

6 Remerciements

Je tiens à remercier différentes personnes qui m'ont beaucoup aidé durant ce travail :

Pour commencer, je remercie le Pr. François Bavaud, sans qui tout ce travail n'aurait pas pu être possible. Merci beaucoup pour votre motivation communicative, vos connaissances et vos idées.
Ensuite, je remercie la Dr. Alina Matei, dont les cours ont non seulement été très agréables à suivre, mais qui se sont également révélés très utiles lors de ce mémoire. Merci beaucoup pour vos talents pédagogiques et la prévenance envers vos étudiants dont vous avez fait preuve tout au long de ce master.
Un grand merci à Mme Elodie Grossenbacher, dont les talents orthographiques ont été mis à rude épreuve lors de ses nombreuses relectures.
Un merci spécial à Mme Caroline Tschumi, pour son soutient moral tout au long de ce travail, et pour sa bonne humeur communicative.
Je remercie également, et pour finir, tous les professeurs de l'Université de Neuchâtel pour leurs cours et les connaissances qu'ils ont partagé avec moi pendant cette année de master. J'ai beaucoup appris à vos côtés et j'espère sincèrement pouvoir vous le rendre un jour.

7 Biblographie

[AND] Anderson, T.W. (1958) *An Introduction to Multivariate Statistical Analysis* J.Wiley, New York.

[BAV1] Bavaud, F. (2006) *Spectral clustering and multidimensional scaling : a unified view.* In : Data science and classification (Batagelj, V., Bock, H.-H., Ferligoj, A., Ziberna, A., eds.), Springer, 131-139.

[BAV2] Bavaud, F. (2010) *On the Schoenberg Transformations in Data Analysis : Theory and Illustrations.* The Department of Computer Science and Mathematical Methods, Department of Geography, University of Lausanne.

[BEN] Bernstein, S. (1929) *Sur les fonctions absolument monotones.* Acta Mathematica 52, 1-66.

[BEG] Berg, C., Mateu, J., Porcu, E. (2008) *The Dagum family of isotropic correlation functions.* Bernoulli 14, 1134-1149.

[CHE] Chen, D., He, Q., Wang, X. (2007) *On linear separability of data sets in feature space.* Neurocomputing 70, 2441-2448.

[DON] Donoho, D.L., Grimes C. (2003) *Hessian eigenmaps : Locally linear embedding techniques for high-dimensional data.* Departement of Statistics, Standford University.

[GNA] Gnanadesikan R. (1989) *Discriminant analysis and clustering, panel of experts.* Statistical Science, 4, n°1, 34-69.

[LEB] Lebart L., Morineau A., Warwick K. (1984) *Multivariate Descriptive Statistical Analysis.* J.Wiley, New York.

[SCH1] Schoenberg, I.J. (1938) *Metric Spaces and Completely Monotonic Functions.* The Annals of Mathematics 39, 811-841.

[SCH2] Schoenberg, I.J. (1938) *Metric Spaces and Positive Definite Functions.* Transactions of the American Mathematical Society 44, 522-536.

8 Annexes

Code R

Fonctions.r

Voici le code des fonctions décrites dans la section 3.

```
#####################################################
###### SCRIPT POUR LE MEMOIRE DE STATISTIQUES ######
######             Guillaume Guex             ######
######         oct 2010 - janvier 2011        ######
#####################################################

########################################
# FONCTIONS DE GENERATIONS DE DONNEES
########################################

############# gen_norm_X
#### Fonction pour generer des coordonnees de points suivant une loi normale
#### Entree :  Les dimensions <n> et <p>, un centre <c> (p) et un vecteur de
#### deviation sd <std> (p) (indép. des coordonnees)
#### Sortie :  Une matrice <X> (nxp) de coordonnees centrees en c et
#### de variance <var>

gen_norm_X = function(n,p,c=rep(0,p),std=rep(1,p)){
X = c()
for(i in 1:n){
point = rnorm(p,c,std)
X = rbind(X,point)
}
return(X)
}

############# gen_cauchy_X
#### Fonction pour generer des coordonnees de points suivant une loi de cauchy
#### Entree :  Les dimensions <n> et <p>, une localisation <loc> (p) et
#### un vecteur de scale <sca> (p) (indép. des coordonnees)
#### Sortie :  Une matrice <X> (nxp) de coordonnees centrees en c et
#### de variance <var>

gen_cauchy_X = function(n,p,loc=rep(0,p),sca=rep(1,p)){
X = c()
for(i in 1:n){
point = rcauchy(p,loc,sca)
```

```
X = rbind(X,point)
}
return(X)
}

############## gen_anulus_X
#### Fonction créer des point sur un anneau
#### Entree :  Les dimensions <n> et <p>, le rayon de l'anneau <r>,
#### la deviation sd du contour <std>
#### Sortie :  Une matrice <X> (nxp) de coordonnees de points formant un anneau.

gen_anulus_X = function(n,p,r=2,std=0.2){
X = c()
radius = rnorm(n,r,std)
theta = sort(runif(n,0,2*pi))
x = radius*cos(theta)
y = radius*sin(theta)
for (i in 1:(p-2)){
X = cbind(X,rep(0,n))
}
X = cbind(x,y,X)
return(X)
}

############## gen_gride_X
#### Fonction créer une grille de point
#### Entree :  Les dimensions <n> et <p>, la taille de la grille <size>
#### Sortie :  Une matrice <X> (nxp) de coordonnees de points aligné
#### sur une grille.

gen_gride_X = function(n,p,si = 1){
nbr = floor(n^(1/2))
X = c()
for(i in 1:nbr){
for(j in 1:nbr){
point = c((-nbr/2+i)*si,(-nbr/2+j)*si,rep(0,p-2))
X = rbind(X,point)
}
}
return(X)
}
```

```
########################################
# FONCTIONS DE TRANSFORMATIONS
########################################

############# cent_red
#### Fonction pour centrer et réduire une matrice de distances
#### Entree :  Une matrice <X> (nxp) de coordonnees quelconques et
#### un vecteur de poids <f> (n)
#### Sortie :  Une matrice <X_c> (nxp) de coordonnees centrees reduites

cent_red = function(X,f=rep(1/dim(X)[1],dim(X)[1])){
z_M = colSums(X*f)
center_X = t(t(X)-z_M)
s_z_2 = colSums(center_X^2*f)
X_c = t((t(X)-z_M)/(sqrt(abs(s_z_2+1e-24))))

return(X_c)
}

############# X_to_D
#### Fonction pour la construction de la matrice des distance
#### Entree :  Une matrice <X> (nxp) de coordonnees.
#### Sortie :  Une matrice <D> (nxn) des distances entre les points

X_to_D = function(X){
n = dim(X)[1]
one = rep(1,n)
aux = diag(X %*% t(X)) %*% t(one)
D = aux + t(aux) - 2*X %*% t(X)

return(D)
}

############# schoen_ACP
#### Fonction pour une ACP et une transformation de schoenberg
#### Entree :  Une matrice de distance <D> (nxn), un vecteur de poids <f> (n),
#### une fonction de schoenberg <schoen>(à définir au préalable) et
#### le parametre de cette fonction <prm>.
#### Sortie : Une matrice de coordonnees en basses dimensions (nxn)

schoen_ACP = function(D,f=rep(1/dim(D)[1],dim(D)[1]),schoen = NULL,prm = 1){

### Transformation de Schoenberg
```

```
if (!is.null(schoen)){
D = schoen(D,prm)
}
n = dim(D)[1]

### Choix du point de vue
alpha = f

### Construction des matrices B puis K
H = t(t(diag(rep(1,n))) - alpha)
B = -1/2 * H %*% D %*% t(H)
K = diag(sqrt(f)) %*% B %*% diag(sqrt(f))

### Decomposition spectrale
specK = eigen(K)
vapK = specK$values
vepK = specK$vectors

### Matrice des coordonnées de basse dimensionalité
X_b = diag(1/sqrt(f)) %*% vepK %*% diag(sqrt(abs(vapK + 1e-12)))

return(X_b)
}

############## inert_schoen_ACP
#### Fonction pour l'inertie dans les différentes dimentions d'une ACP
#### et d'une transformation de schoenberg
#### Entree :  Une matrice de distance <D> (nxn), un vecteur de poids <f> (n),
#### une fonction de schoenberg <schoen> (à définir au préalable) et
#### le parametre de cette fonction <prm>.
#### Sortie : Un vecteur (n) contenant le pourcentage d'inertie dans
#### les différentes coordonnees factorielles.

inert_schoen_ACP = function(D,f=rep(1/dim(D)[1],dim(D)[1]),schoen = NULL,prm = 1){

### Transformation de Schoenberg

if (!is.null(schoen)){
D = schoen(D,prm)
}
n = dim(D)[1]

### Choix du point de vue
alpha = f
```

```
### Construction des matrices B puis K
H = t(t(diag(rep(1,n))) - alpha)
B = -1/2 * H %*% D %*% t(H)
K = diag(sqrt(f)) %*% B %*% diag(sqrt(f))

### Decomposition spectrale
specK = eigen(K)
vapK = specK$values
vepK = specK$vectors

return(vapK/sum(vapK))
}


##########################################
# FONCTIONS D'AFFICHAGE
##########################################

############# display
#### Fonction pour faire un graphique suivant deux dimensions
#### d'une matrice de coordonnées.
#### Entree :  Une matrice de coordonnees <X> (nxp), un vecteur
#### de dimensions <dime> (dime[1],dime[2] <= p }) et
#### un vecteur de groupe <groups> (n)
#### Sortie : Affichage du nuage de points suivant les axes choisis

display = function(X,dime=c(1,2),groups = NULL,main = NULL,xlab="",ylab=""){
n = dim(X)[1]
p = dim(X)[2]
if (dime[1] > p) dime[1] = p
if (dime[2] > p) dime[2] = p
plot(X[,dime[1]],X[,dime[2]],xlab=xlab,ylab=ylab,main = main,cex.main=0.95,
cex.lab=0.7)
if (!is.null(groups)){
m = max(groups)-1
for(i in 1:n){
points(X[i,dime[1]],X[i,dime[2]], col = rgb(1-(groups[i]-1)/m, (groups[i]-1)/m,
(groups[i]-1)/m))
}
}
}
```

```
########################################
# FONCTIONS D'EVALUATIONS DES DONNEES
########################################

########## CROSS-VALIDATION

############# discriminant_classifier
#### Fonction permettant de classifier un point donné par rapport à
#### des points de plusieurs groupes, grâce à la méthode d'analyse discriminante.
#### Entree : Une matrice de points + poids + groupes <ext_X> (nx(p+2)),
#### un point <point> (p) et un vecteur des dimensions prises en compte <dim_taken>.
#### Sortie : L'indice du groupe choisi pour le point.

discriminant_classifier = function(ext_X,point,dim_taken=c(1:(dim(ext_X)[2]-2))){

new_X = ext_X[,c(dim_taken,dim(ext_X)[2]-1,dim(ext_X)[2])]
reduced_point = point[dim_taken]
n = dim(new_X)[1]
p = dim(new_X)[2]-2
max_groups = max(new_X[,p+2])

centroides = reduced_point

for(i in 1:max_groups){
group = new_X[new_X[,p+2]==i,]
center = colSums(group[,1:p]*group[,p+1]/sum(group[,p+1]))
centroides = rbind(centroides,center)
}


d = X_to_D(centroides)
dist = d[1,-1]
answ = which.min(dist)
return(answ)
}
```

```
############## sample_classifier
#### Fonction permettant de classifier un groupe de points donnés par rapport
#### à des points de plusieurs groupes, grâce à la méthode du discriminant
#### Entree : Une matrice de points + poids + groupes <ext_X> (nx(p+2)),
#### une matrice de points + poids + groupes à tester <tested_X> (mx(p+2))
#### et un vecteur des dimensions prises en compte <dim_taken>.
#### Sortie : L'indice du groupe choisi pour le point.

sample_classifier = function(ext_X,tested_X, dim_taken=c(1:(dim(ext_X)[2]-2)) ){
class_result = c()
for(i in 1:dim(tested_X)[1]){
class_result = c(class_result,discriminant_classifier(ext_X,
tested_X[i,1:(dim(tested_X)[2]-2)],dim_taken) == tested_X[i,dim(tested_X)[2]])
}
mean(class_result)
}


############## cross_valid
#### Fonction permettant une cross-validation d'un ensemble de points grâce
#### à la méthode d'analyse discriminante.
#### Entree : Une matrice de points + poids + groupes <ext_X> (nx(p+2)),
#### un nombre de parties <fold> .
#### et un vecteur des dimensions prises en compte <dim_taken>.
#### Sortie : Le taux moyen de points correctement classifiés.

cross_valid = function(ext_X,fold=10,dim_taken=c(1:(dim(ext_X)[2]-2)) ){
n = dim(ext_X)[1]
p = dim(ext_X)[2]-2

indice = sample(1:n)
mixed_X= c()
for(i in indice) mixed_X = rbind(mixed_X,ext_X[i,])

step = n/fold
if(step < 2) step = 2
if(step > n/2) step = n/2

answ_vect = c()
for(i in seq(0,n-1,step)){
tested_X = mixed_X[(i+1):(i+step),]
training_X = mixed_X[setdiff(1:n,(i+1):(i+step)),]
answ_vect = c(answ_vect,sample_classifier(training_X,tested_X,dim_taken))
```

```
}
return(mean(answ_vect))
}

########## KS-TEST

############## ks_test
#### Fonction permettant de faire un ks.test sur un ensemble de point,
#### sur 2 coordonnees choisies.
#### Entree : Une matrice de points <X> (n,p) et
#### un vecteur de dimension dime (dime[1],dime[2] <= p)
#### Sortie : Le resultat du ks.test sur les points.

ks_test = function(X,dime=c(1,2)){

p = dim(X)[2]
  if (dime[1] > p) dime[1] = p
if (dime[2] > p) dime[2] = p

points = sqrt(X[,dime[1]]^2 + X[,dime[2]]^2)
points = (points-mean(points))/sd(points)

return(ks.test(points,"pnorm")$p.value)
}
```

```
########## MODIFICATION DU CENTROIDE

############# new_centroide
#### Fonction permettant de trouver du centroide "modifie" par
#### la transformation de schoenberg
#### Entree :  Une matrice de distance <D> (nxn), un vecteur de poids <f> (n),
#### la derivee d'une fonction de schoenberg <schoen_der> et
#### le parametre de cette fonction <prm>
#### et <iter> le nombre d'iterations.
#### Sortie : Les "poids" alpha du nouveau centroide (n).

new_centroide = function(D,f=rep(1/dim(D)[1],dim(D)[1]),schoen_der = NULL,
prm = 1,iter = 10){

alpha = f

for(i in 1:iter){
new_D = schoen_der(D*alpha - as.numeric(t(alpha)%*%D%*%alpha),prm)
alpha = rowSums(new_D*alpha)/sum(rowSums(new_D*alpha))
}

return(alpha)
}

############# alpha_to_x
#### Fonction permettant de trouver les coordonnees d'un centroide
#### en fonction des "poids"
#### Entree :  la matrice des points <X> (nxp), les poids <alpha> (n).
#### Sortie : les coodronnes du centroide (p).

alpha_to_x = function(X,alpha){
colSums(X*alpha)
}
```

Gride.r

Voici un des codes utilisé pour obtenir les résultats de la section 4.1

```
####################################
##### GRIDE.R
####################################

X = gen_gride_X(100,2)
gr = 1:100
display(X,main="Situation initiale",groups=gr)

X_ACP = schoen_ACP(X_to_D(cent_red(X)))
display(X_ACP,main="Composantes Principales")

schoen_fct = function(D,parm){
return((1-exp(-parm*D))/parm)
#return(D/(parm*(parm + D)))
#return(D^parm)
}

par(mfrow = c(3,3),mex=0.45,tck=-0.02)

display(X_ACP,main="Situation initiale",,groups=gr)

par_2 = c(0.1,0.2,0.5,0.8,1,2,5)

for(i in par_2){
X_schoen = schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=i)
display(X_schoen,main=paste("Paramètre =",i),groups=gr)
}

par_3 = 0.5
plot(inert_schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=par_3),
col="red",type="l",main=paste("Scree Graph, paramètre =",par_3),
xlab="Dimension",ylab="Inertie relative",cex.lab=0.7,cex.main=0.95)
```

Anulus.r

Idem pour les résultats de la section 4.2

```
###################################
##### ANULUS.R
###################################

X = gen_anulus_X(100,2)
gr = 1:100
display(X,main="Situation initiale",groups=gr)

X_ACP = schoen_ACP(X_to_D(cent_red(X)))
display(X_ACP,main="Composantes Principales")
p_1 = inert_schoen_ACP(X_to_D(cent_red(X)))[1]
p_2 = inert_schoen_ACP(X_to_D(cent_red(X)))[2]

schoen_fct = function(D,parm){
return((1-exp(-parm*D))/parm)
#return(D/(parm*(parm + D)))
#return(D^parm)
}

par(mfrow = c(3,3),mex=0.45,tck=-0.02)

display(X_ACP,main="Situation initiale",,groups=gr)

par_2 = c(0.1,0.2,0.5,0.8,1,2,5)

for(i in par_2){
X_schoen = schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=i)
display(X_schoen,main=paste("Paramètre =",i),groups=gr)
}

par_3 = 0.5
plot(inert_schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=par_3),
col="red",type="l",main=paste("Scree Graph, paramètre =",par_3),
xlab="Dimension",ylab="Inertie relative",cex.lab=0.7,cex.main=0.95)
```

Cross-valid.r

Pour les résultats de la section 4.3

```
####################################
##### CROSS-VALID.R
####################################

## Donnees

gr_1 = gen_norm_X(50,10,std=rep(1,10))
gr_2 = gen_norm_X(50,10,rep(-0.8,10),std=rep(2,10))
gr_3 = gen_norm_X(50,10,rep(0.8,10),std=rep(2,10))

X_1 = rbind(gr_1,gr_2,gr_3)
groups_1 = c(rep(1,50),rep(2,50),rep(3,50))

ext_X_1 = cbind(X_1,rep(1/dim(X_1)[1],dim(X_1)[1]),groups_1)

## Affichage

display_X(X_1,groups=groups_1,main="Données initiales")
cross_valid_X(ext_X_1,dim_taken=c(1,2))

## Avec APC simple

X_1_ACP = schoen_ACP(X_to_D(cent_red(X_1)))
ext_X_1_ACP = cbind(X_1_ACP,rep(1/dim(X_1)[1],dim(X_1)[1]),groups_1)

display_X(X_1_ACP,groups=groups_1,main="Composantes Principales",dime=c(1,2))
cross_valid_X(ext_X_1_ACP,dim_taken=c(1,2))

####### AVEC SCHOENBERG

#########################
#ICI CHANGER LA FONCTION#
#########################

schoen_fct = function(D,parm){
#return((1-exp(-parm*D))/parm)
#return(D * (D + exp(-pi*D/2)) / (1+D^2))
#return(log(1 + D/parm))
#return(D/(parm*(parm + D)))
#return(D^parm)
return(D^parm/(1+D^parm))
```

```
}

param = c(0.0001,0.001,0.01,0.1,0.15,0.20,0.25,0.30,0.35,0.4,0.45,0.5,0.6,
0.7,0.8,0.9,1,1.1,1.2,1.3,1.5,2,3)
#param = c(0.0001,0.001,0.01,0.1,0.15,0.20,0.25,0.30,0.35,0.4,0.45,0.5,0.6,
0.7,0.8,0.9,1,1.1,1.2,1.3,1.4,1.5,2,5,10,20,30)
#param = 1/c(10,20,30,40,50,60,70,80,100,1000)

### GRAPHIQUE

results_1 = c()

for(i in param){
X_schoen_1 = schoen_ACP(X_to_D(cent_red(X_1)),schoen=schoen_fct,prm=i)
ext_X_schoen_1 = cbind(X_schoen_1,rep(1/dim(X_1)[1],dim(X_1)[1]),groups_1)

re_1 = c()

for (j in 1:1){
re_1 = c(re_1,cross_valid_X(ext_X_schoen_1,dim_taken=c(1,2)))
}

results_1 = c(results_1,mean(re_1))
}

plot(param,results_1,type="l",ylim=c(0,1),xlab="paramètre",
ylab="proportion de points bien classés",main="Résultats de la cross-validation",
col="red")

### Situation particuliere

par_2 = 0.001

X_schoen_2 = schoen_ACP(X_to_D(cent_red(X_1)),schoen=schoen_fct,prm=par_2)
ext_X_schoen_2 = cbind(X_schoen_2,rep(1/dim(X_1)[1],dim(X_1)[1]),groups_1)

display_X(X_schoen_2,groups=groups_1,main=c("Avec paramètre =",par_2))
cross_valid_X(ext_X_schoen_2,dim_taken=c(1,2))
```

KS_Test.r

Pour les résultats de la section 4.4

```
########################################
# KS_TEST.R
########################################

X = gen_cauchy_X(30,10,sca=rep(4,100))
gr = rep(3,30)
display(X,main="Données initiales")
ks_test_X(X)

####### AVEC SCHOENBERG

schoen_fct = function(D,parm){
#return((1-exp(-parm*D))/parm)
#return(log(1 + D/parm))
#return(D/(parm*(parm + D)))
return(D^parm)
#return(D^parm/(1+D^parm))
}

###### GRAPHIQUES

param = c(0.001,0.2,0.3,0.4,0.5,0.7,1)

par(mfrow = c(3,3),mex=0.52,tck=-0.02)
display(X,main="Données initiales",groups=gr)
answ = c()
for(i in param){
mini_answ = c()
for(j in 1:10){
X_t = schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=i)
mini_answ = c(mini_answ,ks_test(X_t))
}
p_1 = inert_schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=i)[1]
p_2 = inert_schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=i)[2]
display(X_t,main=paste("Paramètre= ",i),groups=gr,,xlab=paste(signif(p_1*100,2),
"% de l'inertie"),ylab=paste(signif(p_2*100,2),"% de l'inertie"))
answ = c(answ,mean(mini_answ))
}

plot(param,answ,type="l",xlab="paramètre",ylab="p-valeur",
main="Résultats du KS-test",col="red")
```

```
## Situation particuliere
par_2 = 0.8

X_s = schoen_ACP(X_to_D(cent_red(X)),schoen=schoen_fct,prm=par_2)
ks_test_X(X_s)
display_X(X_s,main=c("Avec paramètre =",par_2))
```

Nv_centroide.r

Et pour finir, le code des résultats de la section 4.5

```
########################################
# NV_CENTROIDE.R
########################################

### Donnees

X_1 = gen_anulus_X(50,10,1)

for(i in 1:20){
X_1 = rbind(X_1,rep(3,10))
}

f = rep(1/dim(X_1)[1],dim(X_1)[1])

### Schoenberg

schoen_fct = function(D,parm){
#return((1-exp(-parm*D))/parm)
#return(log(1 + D/parm))
#return(D/(parm*(parm + D)))
return(D^parm)
#return(D^parm/(1+D^parm))
}

schoen_der_fct = function(D,parm){
#return(exp(-parm*D))
#return(1/(parm + D))
#return(1/(parm + D)^2)
return(parm*D^(parm-1))
#return(parm*D^(parm-1)/(1+D^parm)^2)

}

### Graphique

parameter = 1:50/5

coord_1 = c()
coord_2 = c()

for (i in parameter){
```

```
alpha_1 = new_centroide(X_to_D(cent_red(X_1)),schoen_der=schoen_der_fct,prm=i)
coord_1 = c(coord_1,alpha_to_x(X_1,alpha_1)[1])
coord_2 = c(coord_2,alpha_to_x(X_1,alpha_1)[2])
}

display_X(X_1,main="Modification du centroïde")
points(alpha_to_x(X_1,f)[1],alpha_to_x(X_1,f)[2],col="blue")

for (i in 1:length(parameter)){
m = length(parameter)
points(coord_1[i],coord_2[i],col = rgb(i/m,1-i/m,1-i/m))
}
```