



UNIVERSITÉ  
DE MONTPELLIER



INRAE



## HMSN208 : rapport de stage

# Utilisation des méthodes à noyaux en vu de l'analyse des données biologiques

Master Sciences et Numérique pour la Santé  
parcours Bioinformatique, Connaissances, Données



|                       |                  |
|-----------------------|------------------|
| [Candidat]            | Xavier GRAND     |
| [Tuteur pédagogique]  | Thérèse COMMES   |
| [Tuteur scientifique] | Sébastien DEJEAN |
| [Tuteur scientifique] | Jérôme MARIETTE  |

Avril 2020 - Juillet 2020

## Avant-propos et remerciements

Mon choix s'est dirigé vers un stage dans un institut de mathématiques, malgré ma formation initiale en biologie. Toutefois, il y a fort à apprendre de recherches transdisciplinaires, ce dont nous avons discuter avec Sébastien Déjean, mon tuteur durant ce projet. Nous avons tellement à gagner à partager des savoirs et savoirs faire.

Je souhaite te remercier Sébastien pour nos discussions, pour ta disponibilité et tes conseils malgré les conditions toutes particulières de ces derniers mois ; et surtout de m'avoir mené à une démystification les mathématiques. Alban Mancheron avait initié ce travail de démystification à travers le module d'algorithmique, tu as continué ce travail avec brio, tes explications simples et limpides mon permis d'approcher des concepts mathématiques qui me paraissaient obscurs et compliqués. Je ne me sauverai plus en courant à la vue d'une équation désormais !

Je remercie également Jérôme Mariette, pour ton fourmillement d'idées et les discussions que nous avons eu chaque semaine. Rien de tel qu'un projet dont les possibilités d'explorations foisonnent pour être stimulé et pour nourrir abondamment la curiosité ; même si en fin de compte cela aura amplifié la frustration de ne pas être allé plus loin dans ce sujet bien vaste !

Je vous remercie donc tous les deux, pour les perspectives et nouveaux outils que vous m'avez permis d'acquérir et espère pouvoir continuer de travailler avec vous par le biais de collaborations futures.

Je remercie également Thérèse Commes pour votre bienveillance et les remarques pertinentes quant à mon travail de bibliographie et attend avec intérêt celles que vous aurez quant à ce travail de stage.

Enfin, je remercie toute l'équipe pédagogique du Master BCD. Il s'agissait pour moi d'une reprise d'étude afin de faire évoluer mon profil de compétences, chose réussie au vu des résultats, grâce à vos enseignements.

# Table des Matières

|  |           |
|--|-----------|
| <b>1 INTRODUCTION</b>  | <b>4</b>  |
| 1.1 LES MÉTHODES À NOYAUX . . . . .  | 4         |
| 1.1.1 DÉFINITIONS . . . . .  | 4         |
| 1.1.2 QUELQUES NOYAUX USUELS . . . . .   | 5         |
| 1.1.3 DES NOYAUX EN BIOLOGIE . . . . .   | 5         |
| 1.1.4 PROBLÉMATIQUE . . . . .  | 6         |
| 1.1.5 OBJECTIFS . . . . .  | 6         |
| <b>2 DÉCOUVERTE DES MÉTHODES À NOYAUX PAR L'ANALYSE DE JEUX DE DONNÉES SIMULÉS</b>     | <b>6</b>  |
| 2.1 SIMULATION DES JEUX DE DONNÉES . . . . .   | 6         |
| 2.2 CALCUL DES NOYAUX . . . . .  | 8         |
| 2.2.1 PARAMÈTRE $\sigma$ DU NOYAU GAUSSIEN . . . . .                                   | 9         |
| 2.2.2 HEURISTIQUE POUR DÉTERMINER LA VALEUR DE $\sigma$ . . . . .                      | 9         |
| 2.3 MATRICES DE SIMILARITÉS . . . . .  | 10        |
| 2.4 ACP À NOYAUX (KPCA) . . . . .  | 12        |
| 2.5 K-MEANS À NOYAUX . . . . .   | 14        |
| <b>3 MÉTHODES À NOYAUX POUR L'ANALYSE DE DONNÉES DE GÉNOTYPAGE SIMULÉES ET RÉELLES</b> | <b>16</b> |
| 3.1 JEUX DE DONNÉES DE GÉNOTYPAGE SIMULÉS . . . . .                                    | 16        |
| 3.1.1 SIMULATION DE DONNÉES DE GÉNOTYPAGE . . . . .                                    | 16        |
| 3.1.2 CALCUL DE NOYAUX ADAPTÉS AU DONNÉES DE GÉNOTYPAGE . . . . .                      | 17        |
| 3.1.3 "BENCHMARKING" DES FONCTIONS DE CALCUL POUR LE NOYAU ADDITIF . . . . .           | 17        |
| 3.1.4 REPRÉSENTATION . . . . .   | 18        |
| 3.2 JEU DE DONNÉES DE GÉNOTYPAGE RÉEL . . . . .  | 19        |
| 3.2.1 DESCRIPTION . . . . .  | 19        |
| 3.2.2 ANALYSE . . . . .  | 20        |
| <b>4 DISCUSSION ET CONCLUSION</b>  | <b>22</b> |
| 4.1 CONCLUSION . . . . .   | 22        |
| 4.2 DISCUSSION . . . . .   | 22        |
| <b>5 ANNEXES</b>   | <b>24</b> |

# Liste des Figures

|    |   |    |
|----|---|----|
| 1  | JEUX DE DONNÉES SIMULÉS . . . . .                                 | 8  |
| 2  | FONCTION GAUSSIENNE . . . . .                                     | 9  |
| 3  | MATRICES DE SIMILARITÉ NOYAU GAUSSIEN . . . . .                   | 11 |
| 4  | MATRICES DE SIMILARITÉ NOYAU GAUSSIEN, VARIATION DE SIGMA . . . . | 12 |
| 5  | KPCA NOYAU GAUSSIEN . . . . .                                     | 13 |
| 6  | KPCA NOYAU GAUSSIEN SIMPLECIRCLES . . . . .                       | 14 |
| 7  | KERNEL K-MEANS NOYAU GAUSSIEN . . . . .                           | 15 |
| 8  | BENCHMARK DES FONCTIONS DE CALCUL DU NOYAU ADDITIF. . . . .       | 18 |
| 9  | NOYAUX DE GÉNOTYPAGE SUR DONNÉES SIMULÉES . . . . .               | 19 |
| 10 | KPCA, NOYAUX DE GÉNOTYPAGE SUR DONNÉES RÉELLES . . . . .          | 20 |
| 11 | DENDROGRAMMES, NOYAUX DE GÉNOTYPAGE SUR DONNÉES RÉELLES . .       | 21 |

# 1 Introduction

Le nombre de publications relatives à l'étude de données omiques a, en 20 ans, considérablement augmenté, en raison notamment des progrès des technologies de séquençage haut débit (Noor *et al.*, 2019 [1]), comme mentionné en introduction du rapport de bibliographie (HMSN307).

Les données dont nous parlions sont des données quantifiées telles que l'expression de gène, l'accumulation de protéines, de métabolites, des phénotypes quantitatifs et qualitatifs. Nous avions fait un état de l'art sur les méthodes d'analyses et d'intégration non supervisées et supervisées à travers le "package" `mixOmics` [2]. Ces méthodes sont l'ACP (Analyse en Composantes Principales) et ses dérivées telles que la PLS ("Partial Least Squares", régression des moindres carrés partiels ou Projection to Latent Structures), ou encore des méthodes basées sur la corrélation telles que la CCA (Analyse des Corrélations Canoniques) par exemple.

Mais il existe des données de biologie qui ne sont pas numériques. De ce fait, les méthodes linéaires ne sont pas adaptées à leur analyse. Il s'agit par exemple des données de la "GeneOntology", de phylogénie, de réseaux de gènes ou d'interactions entre protéines ou encore des données de liens de parentés entre individus dans le cas des études de génétique ("kinship" en sélection et génétique d'association). Pour les données de type graphe, il existe des algorithmes de parcours tels que le parcours en largeur, le parcours en profondeur, des algorithmes d'identification du plus court chemin entre deux noeuds. Ces algorithmes permettent de "mesurer" des distances dans un graphe sous la forme d'un nombre d'arêtes.

Parmi les méthodes possibles pour analyser des données non linéaires, les méthodes à noyaux sont une piste à envisager.

## 1.1 Les méthodes à Noyaux

### 1.1.1 Définitions

D'après Jérôme Mariette, 2018 [3] :

"Les méthodes à noyaux sont des méthodes d'analyse de données basées sur des mesures de similarité entre échantillons. Elles sont plus efficaces en calcul et mémoire que les approches fondées sur la représentation classique des données en tableaux individus  $\times$  variables car elles compressent l'information contenue dans l'ensemble des très nombreuses variables mesurées. En outre, les noyaux offrent un cadre mathématique justifié permettant d'étendre beaucoup d'approches statistiques standards de manière naturelle. Enfin, ils sont adaptés à des données de types très variés et fournissent un cadre commun pour la combinaison de ces données."

Ainsi, une fonction noyau, notée  $K(x, x')$  avec  $x$  et  $x'$  défini dans un espace arbitraire  $X$ , peut être définie comme une fonction qui permet de calculer la similarité entre des données. Cette fonction est symétrique et semi-définie positive.

Le "kernel-trick", ou astuce à noyaux en français, est une méthode qui permet d'analyser des données non linéaires avec des méthodes linéaires, telles que l'analyse en composantes

principales par exemple, rendu possible par une redéfinition de l'espace de données par une fonction noyau.

### 1.1.2 Quelques noyaux usuels

Les noyaux les plus couramment utilisés sont les suivants :

- Noyau linéaire :

$$K(x, x') = \langle x, x' \rangle$$

où  $\langle x, x' \rangle$  représent le produit scalaire usuel entre deux vecteurs.

- Noyau polynomial :

$$K(x, x') = (\text{scale} * \langle x, x' \rangle + \text{offset})^{\text{degree}}$$

- Noyau exponentiel :

$$K(x, x') = \exp^{-\sigma \|x - x'\|}$$

- Noyau Gaussien ou RBF (Radial Basis Function) :

$$K(x, x') = \exp^{-\sigma \|x - x'\|^2}$$

### 1.1.3 Des noyaux en biologie

Les applications des méthodes à noyaux sont multiples en biologie. Le "package" `mixKernel` [4], à l'instar de `mixOmics`, implémente des méthodes d'intégration de données omiques à l'aide de méthodes à noyaux.

Le "package" `SKAT` [5] offre la possibilité de réaliser des analyses de génétique d'association à l'aide de méthodes à noyaux. Dans ce "package", plusieurs fonctions noyaux sont implémentées, avec pour chacune d'elle une application spécifique. Une fonction, le noyau dit linéaire permet de réaliser une régression linéaire entre phénotype et génotype ; le noyau dit quadratique permet de mettre en exergue les variants rares parmi la population étudiée ; le noyau dit additif permet de rendre compte de l'effet additif des variants sur le phénotype. Le détail du calcul de ces noyaux est présenté en partie 3.1.2 de ce rapport. L'avantage de ces méthodes à noyau dans des approches de génétique d'association répond à la problématique posée par les variants rares, dont les méthodes classiques de GWAS ("Genome-Wide Association Study") minimisent l'importance. L'utilisation d'un noyau adapté à des fréquences faibles de variants permet de mieux évaluer leur influence sur le phénotype observé [6].

#### **1.1.4 Problématique**

Les données de biologie sont variées et de nature différentes. Les méthodes d'analyses linéaires telles que l'ACP et ses dérivées permettent d'en analyser une partie. En effet, les données décrites par des valeurs numériques/mesurables/quantitatives, peuvent être analysées et intégrées, comme présenté à travers l'exemple du "package" `mixOmics` (Projet HMSN307).

Cependant, il existe de l'information sous des formes différentes. Comme nous l'avons discuté en introduction, les données de phylogénie sont décrites par des arbres, les données relatives aux interactions entre protéines sont représentées par des graphes, etc. Les méthodes d'analyses linéaires n'étant pas adaptées à ces données, il est intéressant de développer de nouveaux outils permettant d'analyser ces données et de les intégrer.

Les méthodes à noyaux, dont l'objet est de définir des similarités entre les données, sont en ce sens une piste prometteuse pour intégrer des données très hétérogènes. En effet, lorsque toutes les données sont transformées en mesures de similarité, il ne reste qu'à comparer ces similarités. De plus, l'astuce à noyaux dont nous avons parlé en introduction, nous permet d'utiliser des méthodes d'analyse linéaires sur des données qui ne le sont pas.

La question à laquelle nous avons cherché à répondre est : Comment utiliser les méthodes à noyaux pour analyser et intégrer des données biologiques de natures diverses ?

#### **1.1.5 Objectifs**

Afin de répondre à la problématique sus-énoncée, il est nécessaire de maîtriser les approches à noyaux au sens formel aussi bien qu'au sens pratique. C'est pourquoi les objectifs de ce travail sont multiples :

- Décrire et manipuler des noyaux simples, classiques sur des jeux de données stéréotypés.
- Décrire et manipuler des exemples de noyaux adaptés à la biologie sur des jeux de données simulés.
- Mettre en oeuvre une analyse à noyaux sur des données réelles de biologie.

## **2 Découverte des méthodes à noyaux par l'analyse de jeux de données simulés**

### **2.1 Simulation des jeux de données**

Afin de manipuler les méthodes à noyaux dans un contexte connu et dont les interprétations peuvent être plus facilement anticipées que sur des données réelles, nous avons choisi de définir des jeux de données stéréotypés. Un exemple fréquent dans la littérature est celui des deux cercles concentriques. Il a donc été le premier jeu de données que nous avons simulé. En fin de compte, 9 jeux de données de 2 variables quantitatives ont été simulés. Chaque jeu

comprend de deux à sept catégories. Une catégorie correspond à un ensemble de points qui sont générés à partir d'une même fonction mathématique.

Pour chacun des jeux, une fonction en R a été implémentée. Le paramètre de ces fonctions est le nombre de points par catégorie à générer.

Les jeux de données sont :

- Rim,
- Circle,
- Rafters,
- Moon,
- Spirals,
- Blossum,
- Swiss-roll,
- Hazard,
- Classification,
- SimpleCircles.

La figure 1 est la représentation graphique des jeux de données simulés. Le code des fonctions est disponible en Annexe 1.

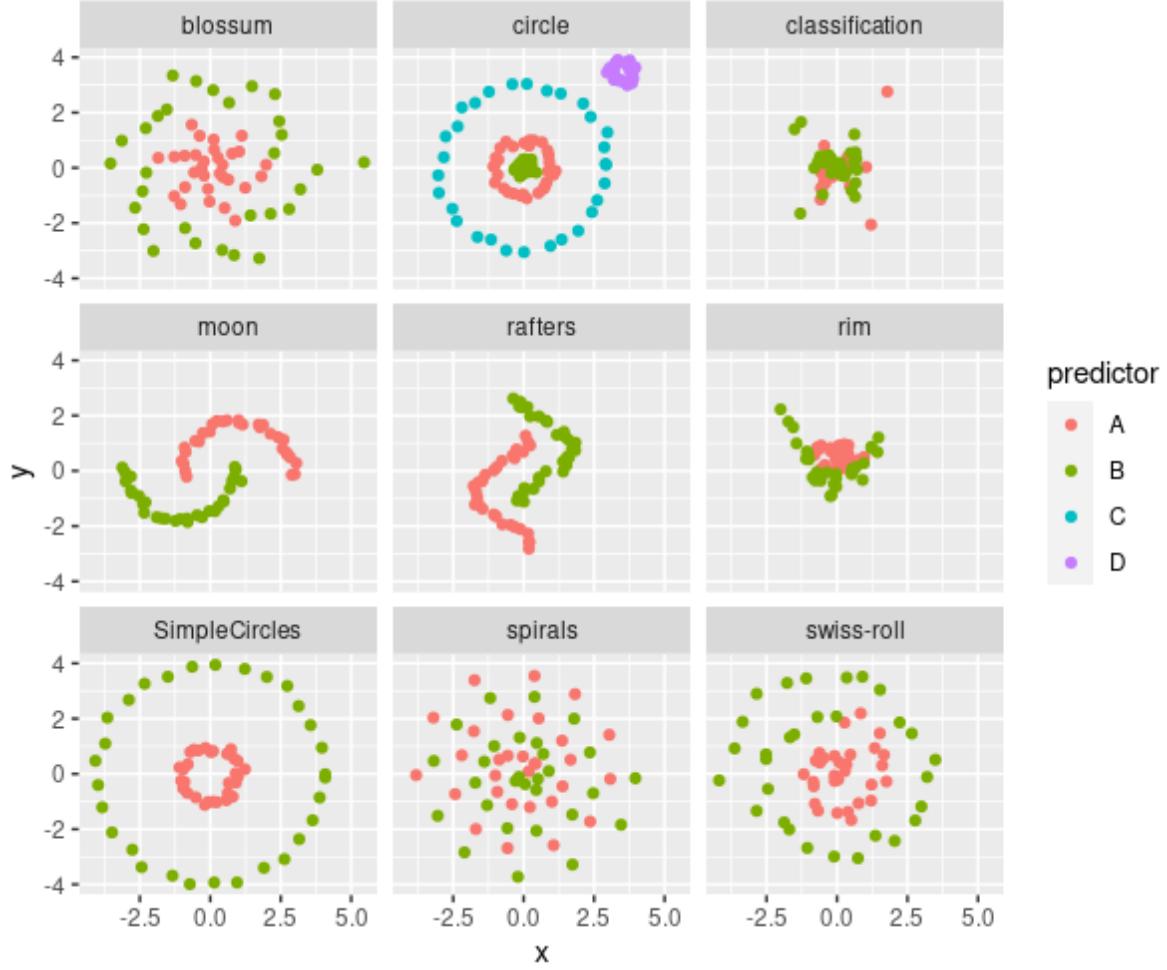


Figure 1: Représentations graphiques des jeux de données simulés.

## 2.2 Calcul des noyaux

Il existe plusieurs "packages" R disponibles pour calculer des noyaux classiques. Nous avons choisi d'utiliser `kernlab` [7], disponible sur le Comprehensive R Archive Network (CRAN, cran.r-project.org). Ce "package" est dédié aux méthodes à noyaux, il permet de calculer de manière simple différents noyaux, notamment linéaire, polynomial ou Gaussien. De plus, il intègre des méthodes d'analyse telles que l'ACP à noyaux. Il s'agit d'un "package" relativement simple, il ne s'agit pas d'une "usine à gaz", mais qui intègre la plupart des fonctions nécessaires à notre projet.

Nous avons, sur des exemples très simples, calculé des noyaux linéaires, polynomiaux et Gaussiens à la main et comparé les résultats obtenus avec les fonctions implémentées dans `kernlab`. De cette façon nous avons pu vérifier la pertinence de ce choix.

Par soucis de concision et parce qu'il s'agit du noyau le plus intuitif, nous ne présenterons que les résultats pour le noyau Gaussien, l'objet de ce travail étant de se familiariser avec les

méthodes à noyau.

### 2.2.1 Paramètre $\sigma$ du noyau gaussien

Afin de visualiser et ainsi de mieux comprendre le calcul d'un noyau, nous avons représenté les fonctions noyaux graphiquement.

La figure 2 représente la fonction noyau Gaussien. Cette fonction prend un hyper-paramètre sigma ( $\sigma$ ), que nous avons fait varier afin de nous rendre compte de son effet.

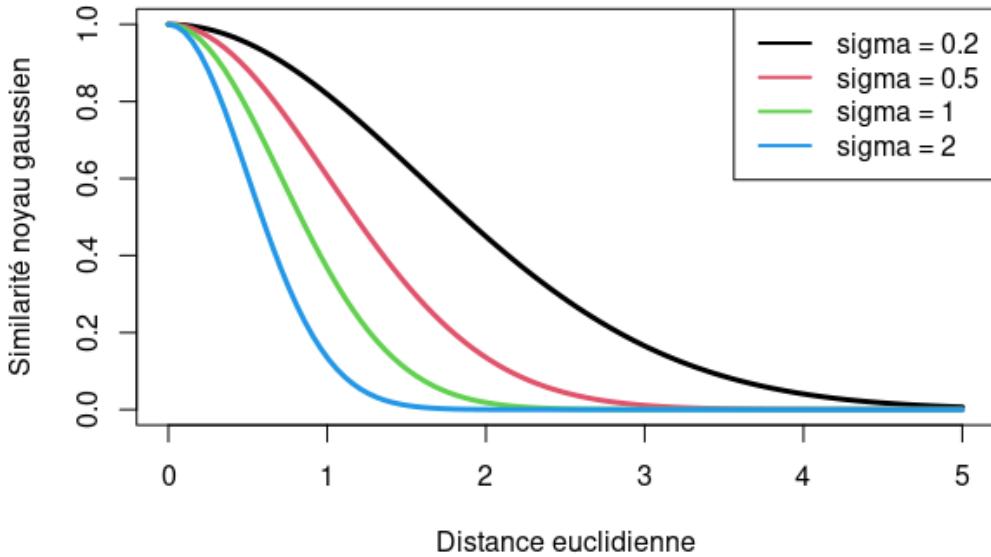


Figure 2: Représentation graphique de la fonction noyau Gaussien.

Cette visualisation nous permet de constater l'effet de la transformation opérée par le noyau Gaussien. Ainsi, lorsque la distance euclidienne augmente, la valeur de similarité calculée diminue d'autant plus rapidement que la valeur de  $\sigma$  est grande, en suivant une courbe de Gauss. Il est possible d'en déduire que plus les individus sont proches et plus la valeur de similarité calculée est grande. D'autre part, cette valeur est rapidement très faible lorsque la distance passe un seuil.

### 2.2.2 Heuristique pour déterminer la valeur de $\sigma$

Dans le cadre de l'utilisation du noyau Gaussien, l'hyper-paramètre  $\sigma$  a donc un effet sur le calcul du noyau. La question qui se pose alors est comment déterminer sa valeur optimale puisque celle-ci dépend des données à analyser. Nous avons choisi de tester empiriquement plusieurs valeurs de  $\sigma$  dans nos analyses, et d'utiliser une heuristique tel que décrite dans Garreau *et al.*, 2009 [8]. Cette heuristique est défini par l'équation :

$$\sigma = \text{median} \left( \frac{1}{\|x - x'\|^2} \right)$$

## 2.3 Matrices de similarités

Le corrélogramme est une méthode qui permet de représenter graphiquement une matrice de corrélation. Or un noyau est une matrice de similarité. Nous avons alors détourné l'utilisation du corrélogramme pour représenté non pas des coefficients de corrélation entre nos individus simulés, mais leur similarité calculée par les différents noyaux. C'est un bon moyen pour dégager visuellement des groupes d'individus similaires ou au contraire de différencier des individus différents. Nous avons réalisé ces corréogrammes pour tous les jeux de données simulés pour les noyaux linéaire, Gaussien (Figure 3) et polynomial.

La figure 3 représente les corréogrammes gaussiens calculés avec pour hyper-paramètre de la fonction noyau  $\sigma$ , la valeur obtenue avec l'heuristique présentée en partie 2.2.2. De cette manière, il est facile de voir les différences de similarités entre individus calculées par les différents noyaux. Le jeu de données le plus simple à analyser est le jeu nommé "SimpleCircles", les deux cercles concentriques. La neuvième vignette de cette figure représente le corréogramme de ce jeu de données. Les individus du petit cercle sont les individus numérotés de 1 à 10 et ceux du grand cercle sont les individus numérotés de 11 à 20. Nous pouvons constater que la similarité calculée par la fonction noyau pour les individus du petit cercle deux à deux est très élevée, proche de 1. La similarité entre les individus du grand cercle par rapport aux individus du petit cercle est moyenne. Et enfin, la similarité entre les individus du grand cercle entre eux varie de forte à quasi nulle.

Ainsi, les individus proches, au sens de leur distance euclidienne dans l'espace, ont une valeur de similarité calculée élevée. Cette valeur décroît avec l'augmentation de cette distance. C'est pourquoi les individus du grand cercle ont des valeurs de similarité élevée lorsqu'ils sont proches, c'est à dire deux individus consécutifs du cercle par exemple, et des valeurs faibles lorsqu'ils sont éloignés, c'est à dire deux individus de part et d'autre du cercle.

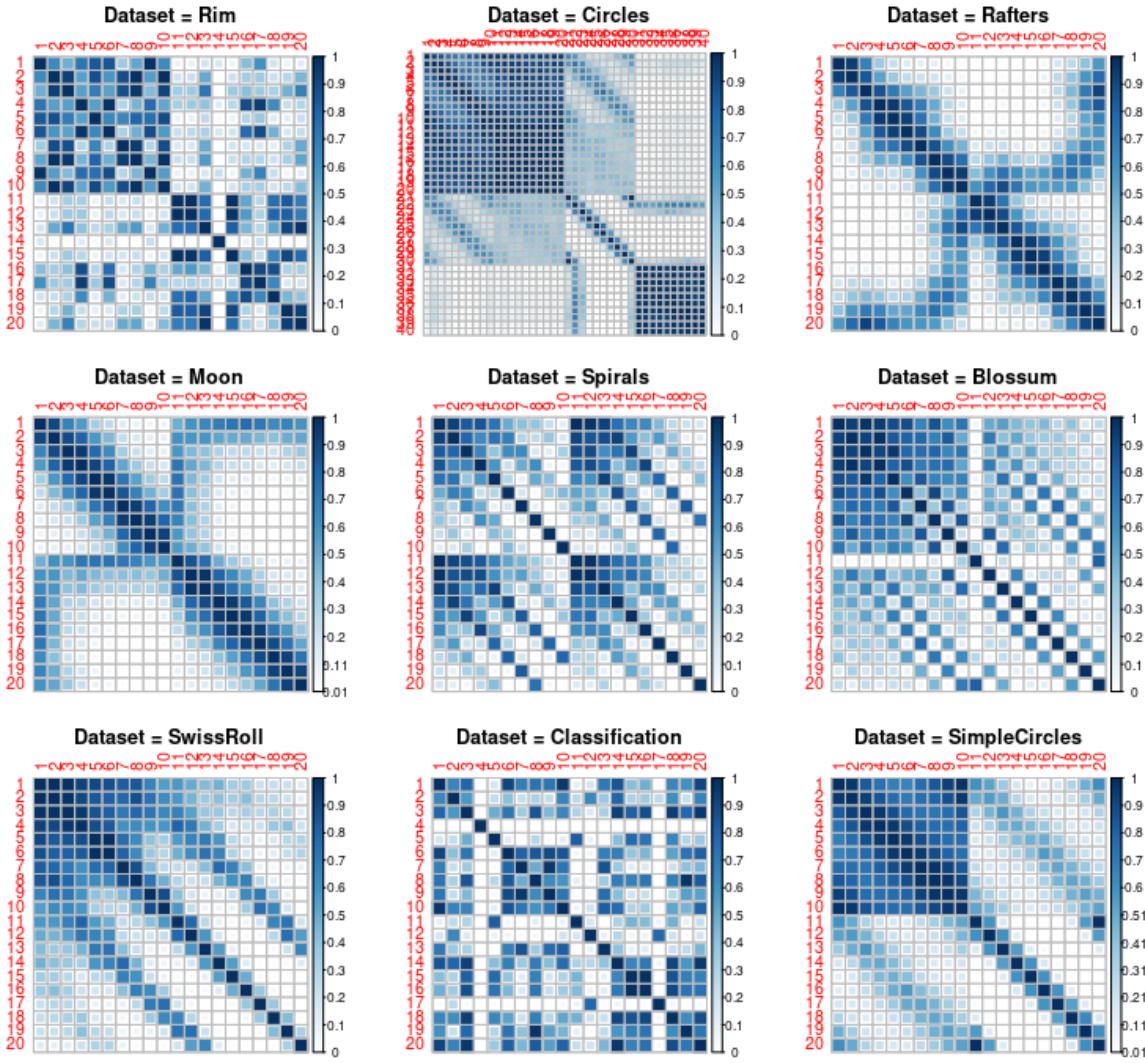


Figure 3: Matrices de similarité Noyau Gaussien obtenues avec le package `corrplot` sur les jeux de données simulés.

Pour appréhender l'effet de l'hyper-paramètre  $\sigma$ , la figure 4 représente des corréogrammes pour le jeu de données "SimpleCircles" pour des valeurs de  $\sigma$  croissantes, de 0.01 à 1 et la valeur calculée par l'heuristique (Cf partie 2.2.2). De cette manière, nous pouvons constater que la similarité entre les individus diminue avec la distance d'autant plus rapidement que la valeur de  $\sigma$  augmente.

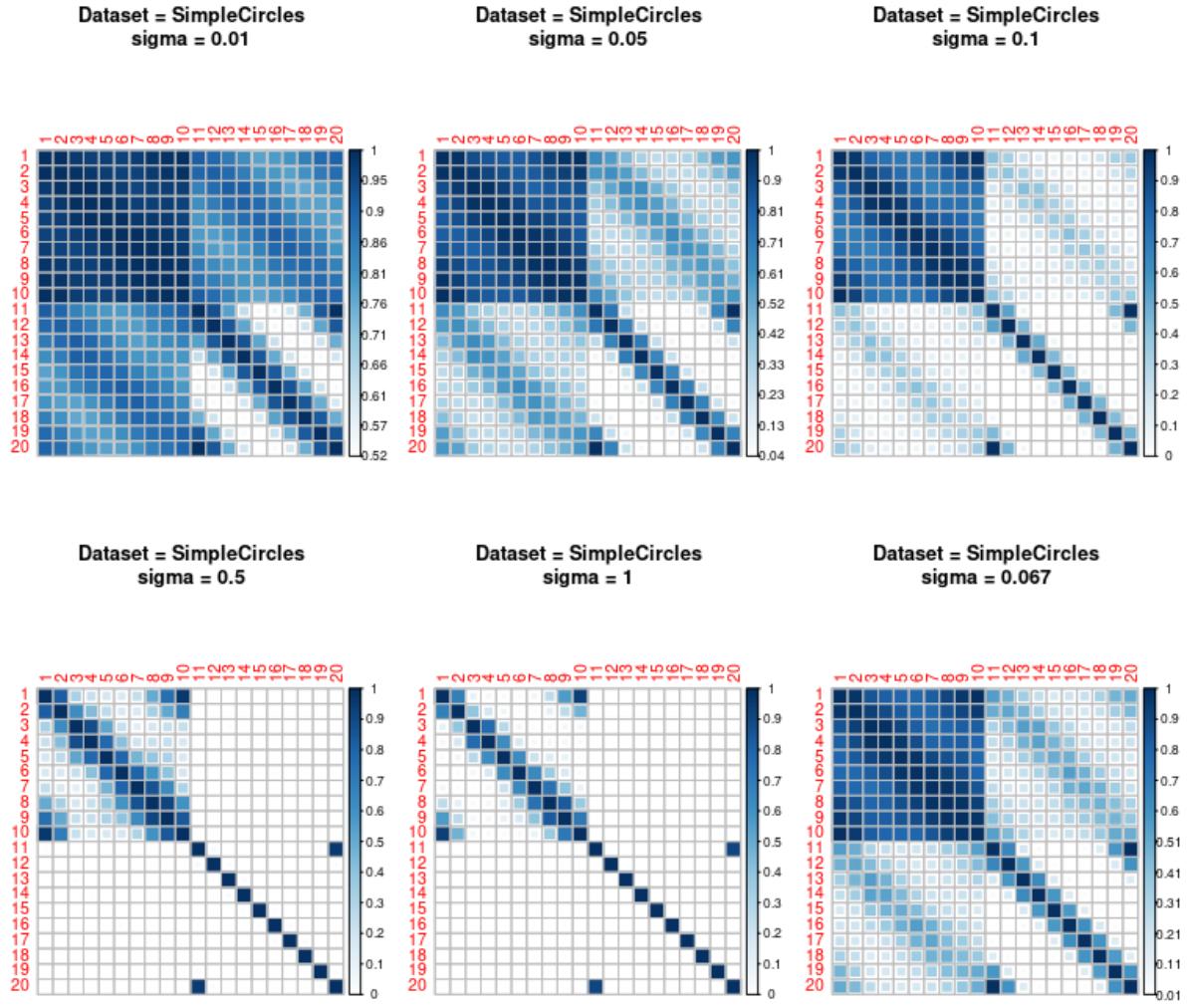


Figure 4: Matrices de similarité Noyau Gaussien sur le jeu de données "SimpleCircles" pour lesquelles la valeur de l'hyper-paramètre  $\sigma$  varie.

## 2.4 ACP à noyaux (KPCA)

Comme nous l'avons évoqué en introduction, un des avantages des méthodes à noyaux est de pouvoir utiliser des méthodes de statistiques linéaires sur les données non linéaires. L'ACP est une méthode très utile pour décrire les données. Le principe de l'ACP est de projeter sur un premier axe, dans l'espace des données, tous les individus, en maximisant leur variance. Puis sur un second axe orthogonal au premier et ainsi de suite. Ainsi, on nomme composante principale, l'ensemble des coordonnées de chaque individu sur ces axes. La représentation graphique des individus sur les axes de l'ACP, permet de dégager des groupes d'individus, si tant est qu'il y en est.

Dans le cadre des analyses à noyau, l'ACP à noyau, ou KPCA, permet de réaliser une ACP sur les valeurs du noyau plutôt que sur les valeurs originales des données, potentiellement

non linéaires. La figure 5 représente les ACP à noyau Gaussien pour l'ensemble des jeux de données simulés. La valeur de l'hyper-paramètre  $\sigma$  est celle calculée par l'heuristique (Cf. 2.2.2).

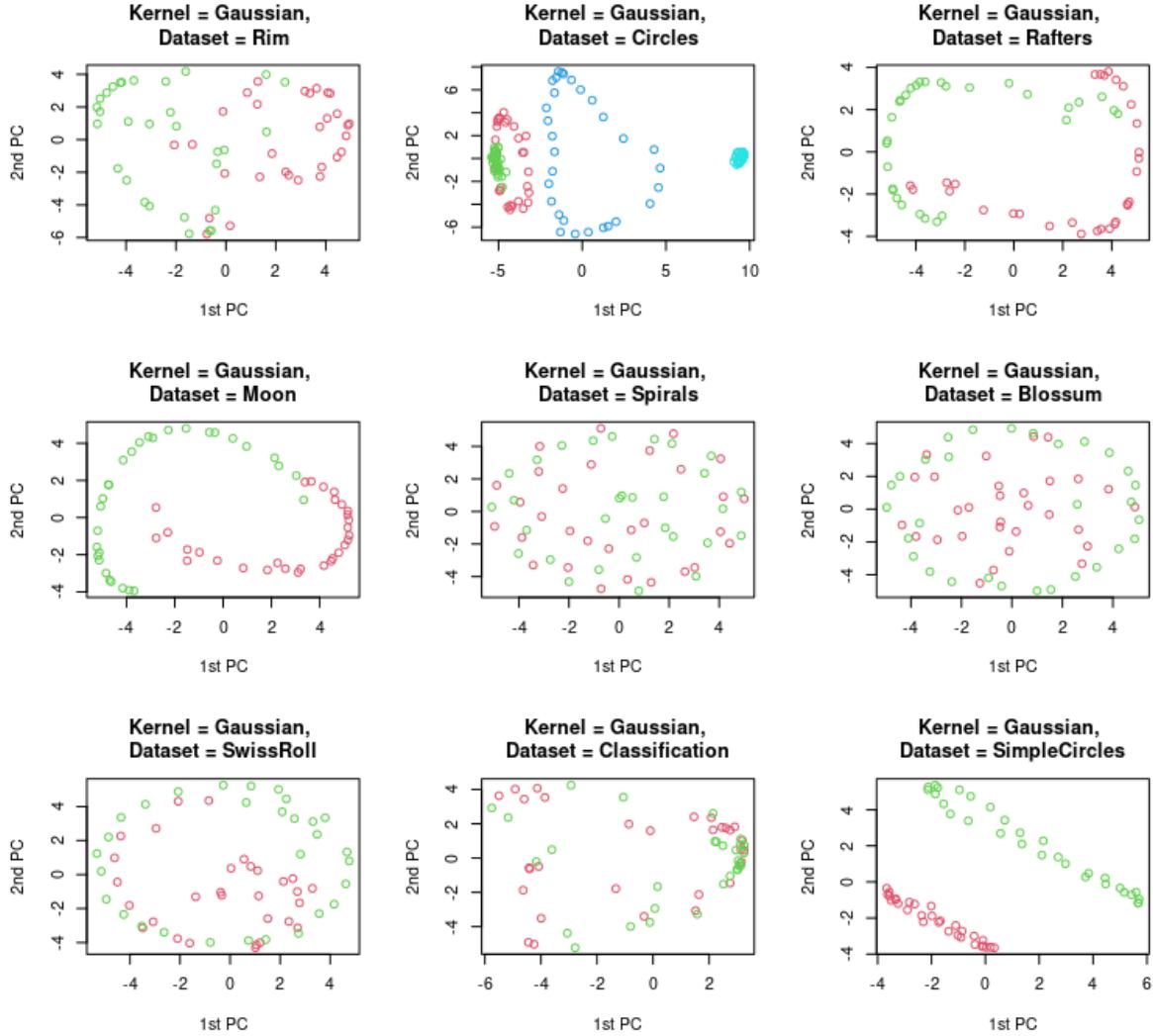


Figure 5: ACP à noyau Gaussien. Les individus sont colorés en fonction de leur catégorie. Le nombre d'individus pour ces jeux de données a été augmenté ( $n = 30$ ) afin de mieux rendre compte de l'effet de l'ACP à noyau.

Sur cette figure, il est possible de se rendre compte que pour certains jeux de données, l'ACP à noyau Gaussien sur les deux premiers axes permet de séparer les deux (ou plus) catégories simulées. En effet, le premier axe de la KPCA, pour les jeux de données "Rim" et "Circles", permet de séparer les différentes catégories. Pour le jeu de données "SimpleCircles", c'est le second axe qui semble le mieux séparer les deux catégories. En ce qui concerne les jeux de données "Rafters" et "Moon", les données sont partiellement séparées sur le premier axe, mais pas complètement. Nous pouvons tester l'hypothèse que l'hyper-paramètre  $\sigma$  peut être optimisé pour mieux séparer les deux catégories pour ces jeux de données.

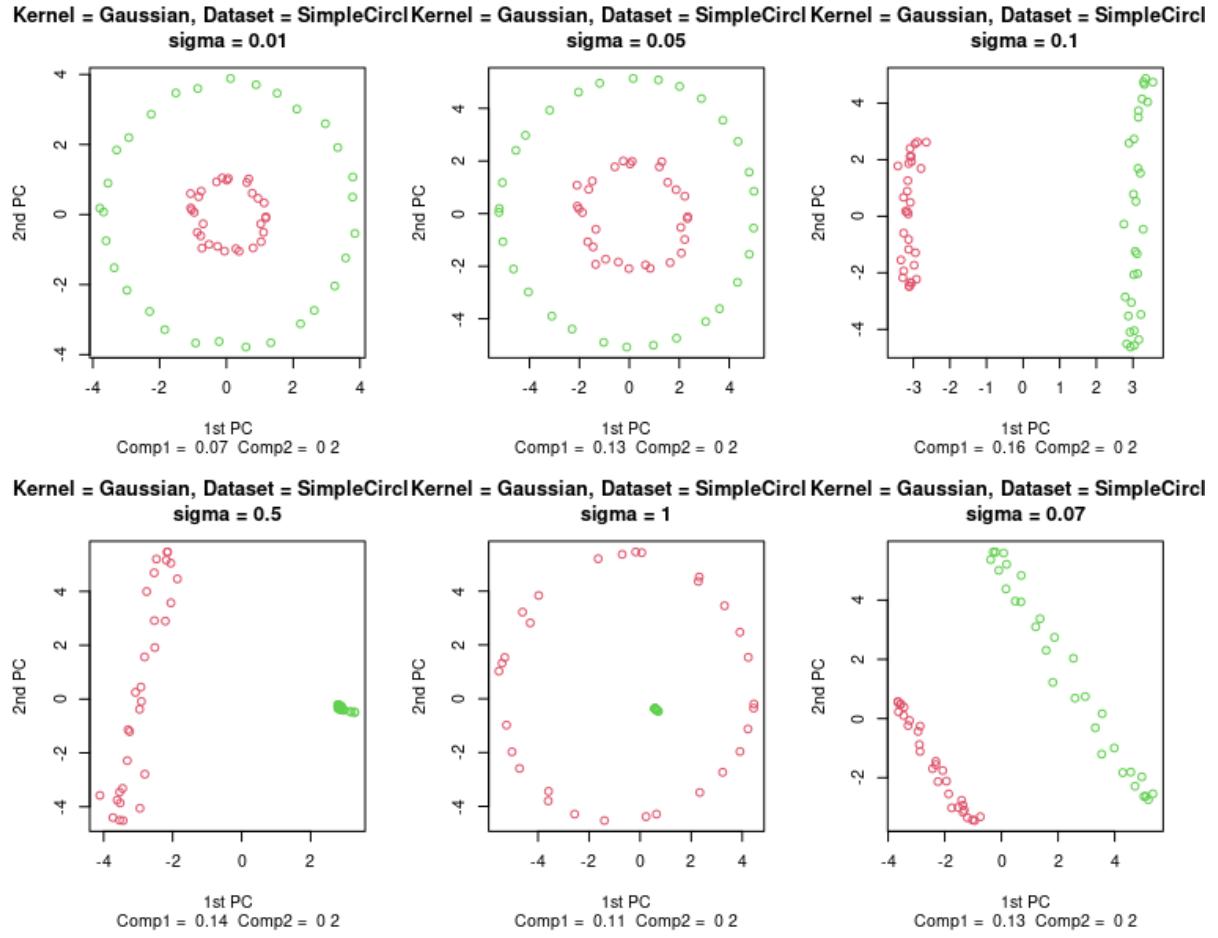


Figure 6: ACP à noyau Gaussien sur le jeu de données "SimpleCircles" pour lesquelles la valeur de l'hyper-paramètre  $\sigma$  varie.

Nous avons fait varier la valeur de l'hyper-paramètre  $\sigma$ . La figure 6 représente les ACP à noyau pour le jeu de données "SimpleCircles" avec des valeurs de  $\sigma$  qui varient de 0.01 à 1 et la valeur calculée par l'heuristique (Cf. 2.2.2). Ainsi, les valeurs de  $\sigma$  0.1, 0.5 ou calculée par l'heuristique permettent de séparer les données selon le premier axe de l'ACP à noyau.

## 2.5 K-means à noyaux

Dans le but d'identifier les catégories dans nos jeux de données, nous avons testé la méthode du "Kernel K-Means", ou partitionnement en k-moyennes à noyau en français. Le principe de cette méthode est la sélection de façon aléatoire d'un nombre pré-défini, en paramètre de la fonction, de centres initiaux correspondant au nombre de groupes que le jeu de données est supposé contenir. Puis, de manière récursive, la fonction minimise la somme des carrés des similarités calculées par les noyaux entre chaque individu et chaque centre et modifie leur position. En fin de compte, lorsque la position des centres est stabilisée, un vecteur est renvoyé, assignant un groupe à chaque individu.

La figure 7 représente les jeux de données simulés, sur un repère en deux dimensions

correspondant aux deux variables de chaque jeu de données, la couleur de chaque individu correspond au groupe auquel la fonction "Kernek K-Means" l'a assigné.

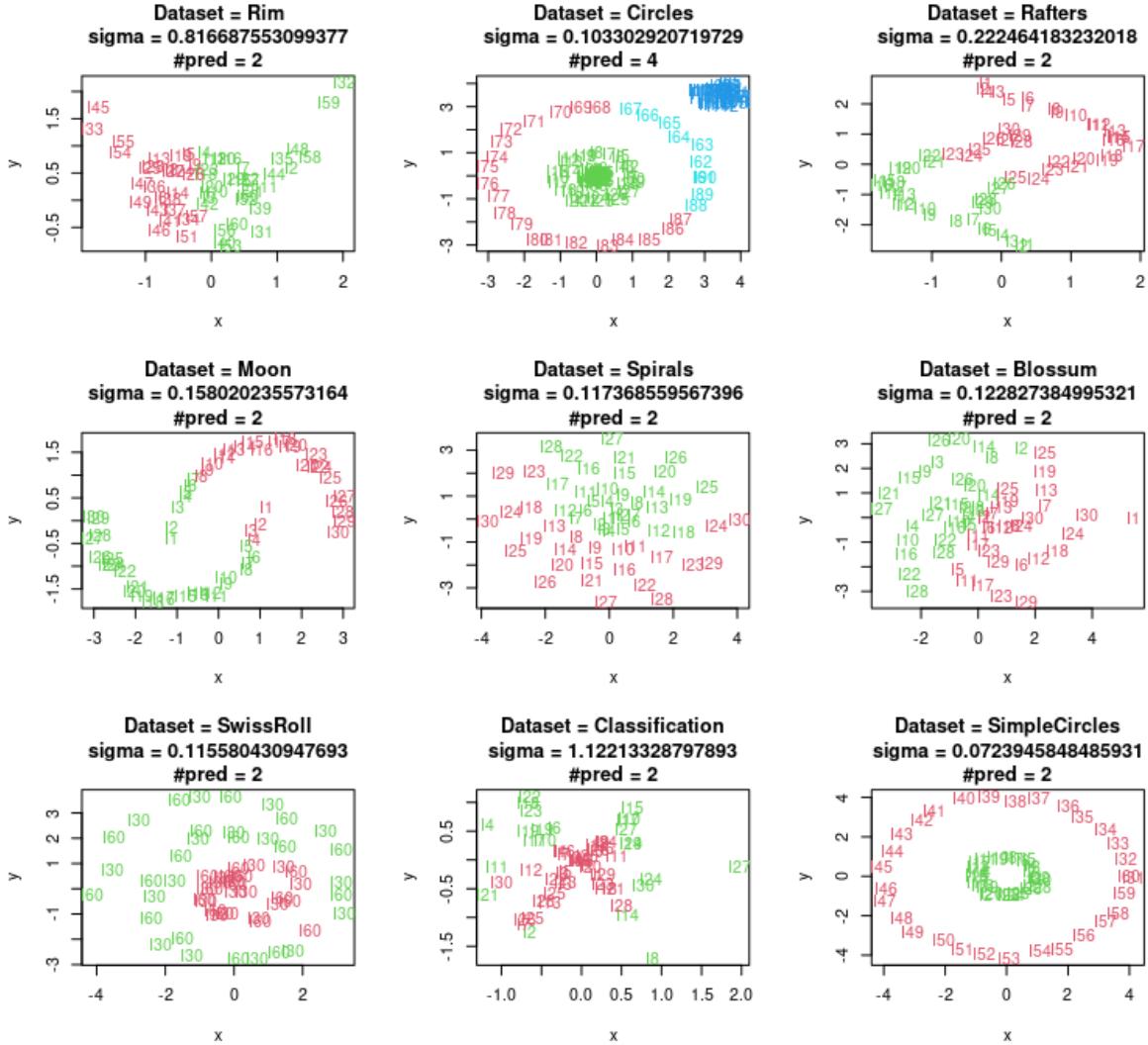


Figure 7: Kernel K-MEANS Noyau Gaussien.

A la lumière de cette représentation, il semble que le "Kernel K-Means" ne soit pas la meilleure façon de retrouver nos groupes initiaux. En effet, à l'exception des jeux "SimpleCircles" dont l'assignation des groupes est parfaite et "SwissRoll", dont l'assignation est quasi parfaite, pour les autres jeux, les groupes ne sont pas retrouvés. De plus, en raison de la sélection aléatoire des centres initiaux les résultats ne sont pas reproductibles. En effet, les groupes assignés sont différents à chaque lancement de la fonction.

Cette partie du travail nous a permis de comprendre ce que sont des noyaux, quelles peuvent être les possibilités qu'ils offrent dans l'analyse de jeux de données simples et simulés. Bien que notre exploration ne soit pas exhaustive, nous avons acquis une certaine "sensibilité"

à ces méthodes. Nous avons donc poursuivi ce travail par l'analyse de jeux de données de biologie.

### 3 Méthodes à noyaux pour l'analyse de données de génotypage simulées et réelles

Parmi les applications possibles des méthodes à noyaux en biologie, nous avons choisi de travailler sur des données de génotypage. Les données de génotypage sont des données relatives à une séquence nucléotidique, c'est à dire, une séquence dans un alphabet composé de 4 base : "A", "C", "G", et "T". De plus, chez un individu diploïde, à chaque marqueur, trois états sont possibles, l'état homozygote pour un allèle, homozygote pour l'autre et hétérozygote. Si l'on considère un individu comme référence, on peut coder l'ensemble de son génotype, son état de chacun des marqueurs, par la valeur "0". De cette manière, le génotype de cet individu référence est, au sens informatique du terme, un vecteur de "0" d'une longueur égale au nombre de marqueurs considérés. Ensuite, pour les autres individus, lorsque le marqueur est du même état que la référence, il est codé par un "0", s'il est hétérozygote il est codé "1" et enfin s'il est homozygote mais différent de la référence, il est codé "2". En fin de compte, il est possible de stocker ces données de génotypage sous la forme d'une matrice de dimension  $n \times m$ , avec  $n$  le nombre d'individus génotypés en ligne et  $m$  le nombre de marqueurs en colonne.

#### 3.1 Jeux de données de génotypage simulés

##### 3.1.1 Simulation de données de génotypage

De la même façon que nous avons travaillés sur des jeux de données simulés afin de mieux les maîtriser, nous avons choisi de développer une approche de simulation des données de génotypes dans un premier temps.

Pour cela, une fonction a été implémentée pour générer un nombre  $n$  d'individus avec un nombre  $m$  de marqueurs génétiques parmi un alphabet, passés en paramètres de la fonction. Dans un premier temps cette fonction crée une matrice nulle de dimensions  $n \times m$ . La première ligne de cette matrice est considérée comme l'individu référence, dont tous les marqueurs sont codés "0". Puis pour chacun des autres individus, c'est à dire les autres lignes de la matrice, le nombre de mutations est déterminé de manière aléatoire. La position de ces mutations est sélectionnée parmi un ensemble de positions possibles pour lesquelles une probabilité de mutation est assignée en amont. Enfin, lorsque les nombres et positions des mutations sont déterminés, la fonction assigne une valeur au marqueur différente de celle avant mutation parmi l'alphabet, en l'occurrence "0", "1" ou "2". Par conséquent, seules les valeurs "1" et "2" peuvent être assignées.

En fin de compte, une matrice de génotypage est générée de manière aléatoire mais dont le nombre, la position et la fréquence de mutations à chaque position possible sont contrôlés.

Le code R des fonctions de création de jeux de données de génotypage ainsi que les analyses sont disponibles en annexe 2.

### 3.1.2 Calcul de noyaux adaptés au données de génotypage

Dans l'optique de comprendre le fonctionnement des noyaux et d'appréhender la problématique de façon simple, nous avons choisi de réaliser un calcul de noyau simple. Nous avons pris comme exemple les noyaux implémentés dans le "package" SKAT. A partir des codes sources du "package" et de la documentation nous avons implanté trois des noyaux proposés, avec  $G$  la matrice de génotypage contenant  $m$  marqueurs,  $n$  individus et  $i \subset n$  :

- le noyau dit linéaire,

$$K(G_i, G_{i'}) = \sum_{j=1}^m G_{ij}G_{i'j}$$

,

- le noyau dit quadratique,

$$K(G_i, G_{i'}) = (1 + \sum_{j=1}^m G_{ij}G_{i'j})^2$$

,

- le noyau dit additif,

$$K(G_i, G_{i'}) = \sum_{j=1}^m (2 - |G_{ij} - G_{i'j}|)$$

.

### 3.1.3 "Benchmarking" des fonctions de calcul pour le noyau additif

Deux fonctions ont été codées pour effectuer le calcul du noyau additif. Une première fonction naïve, nommée "alternatif", qui comprend deux boucles `for` et une fonction optimisée, nommée "additif", qui tire parti des fonction `apply` de R. Les résultats en temps et espace de ces deux fonctions ont été calculés grâce à la fonction `system.time` de R et sont représentés sur la figure 8. Les codes de ces deux fonctions sont disponibles dans l'annexe 2.

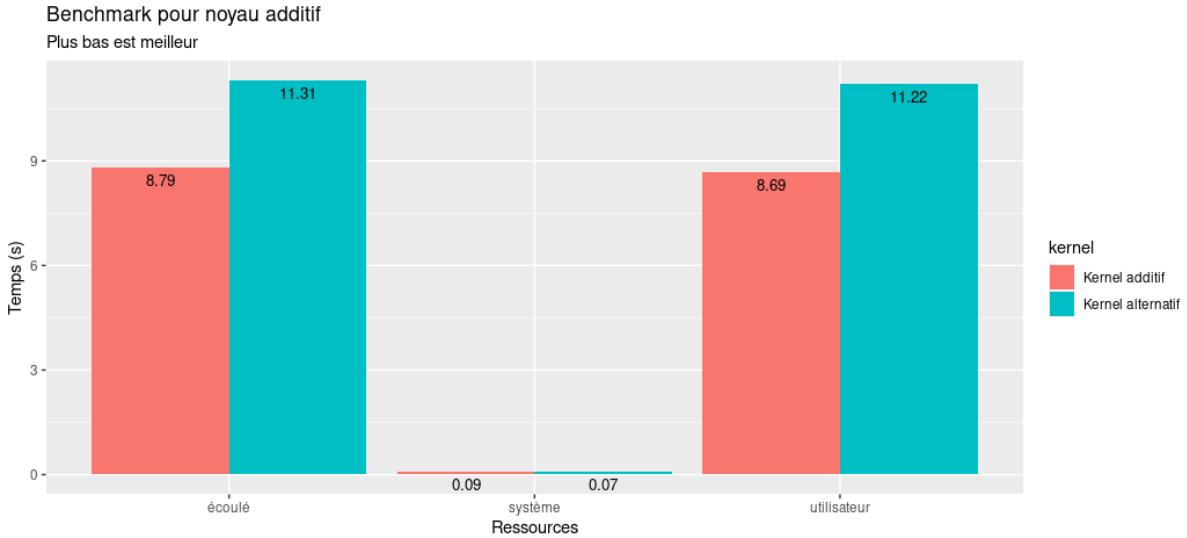


Figure 8: Résultats des temps et ressources nécessaire aux fonction nommées "additif" et "alternatif" pour le calcul du noyau de génotypage additif.

L'optimisation de cette fonction entraîne bien un gain en temps de calcul.

### 3.1.4 Représentation

Afin de représenter la matrice de génotypage et identifier les individus mutés, le nombre de mutations et le statut de ces mutations (homozygote ou hétérozygote), nous avons réalisé des corrélogrammes. Bien évidemment, il s'agit d'une utilisation détournée de ce type de représentation puisque ce sont des valeurs de génotype arbitraires qui sont représentés et non des valeurs de corrélations. Néanmoins, pour de petites matrices de génotypage telles que simulées pour ce travail, cela donne une image directe de la matrice (Figure 9, première vignette).

Puis, de la même façon que pour les jeux de données simulés, des Kernel PCA ont été réalisées pour chacun des noyaux. Le noyau linéaire (Figure 9, deuxième vignette), le noyau quadratique (Figure 9, troisième vignette) et le noyau additif (Figure 9, quatrième vignette) sont représentés. La répartition des individus est différentes sur chacune de ces représentations. Toutefois, il est difficile d’appréhender la spécificité de chacun des noyaux sur un jeu de données petit tel que celui ci.

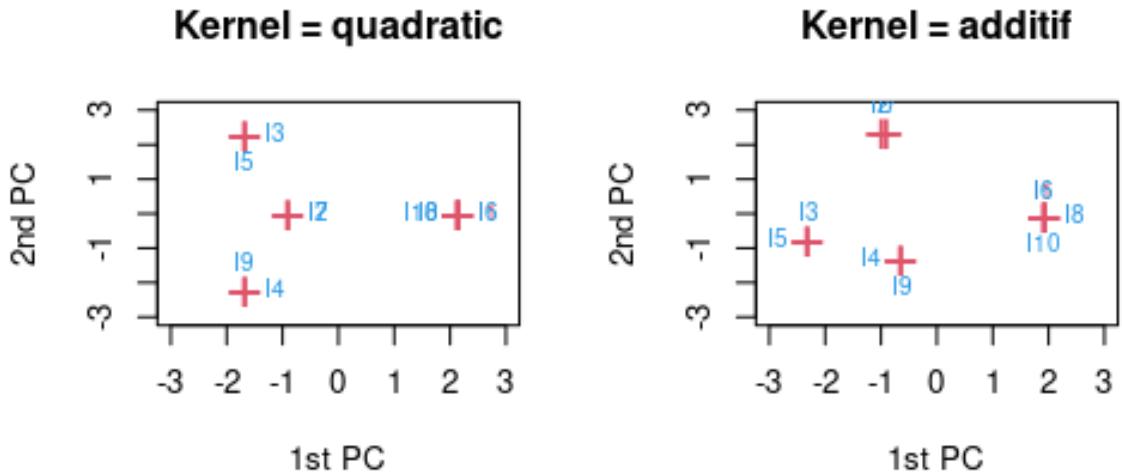
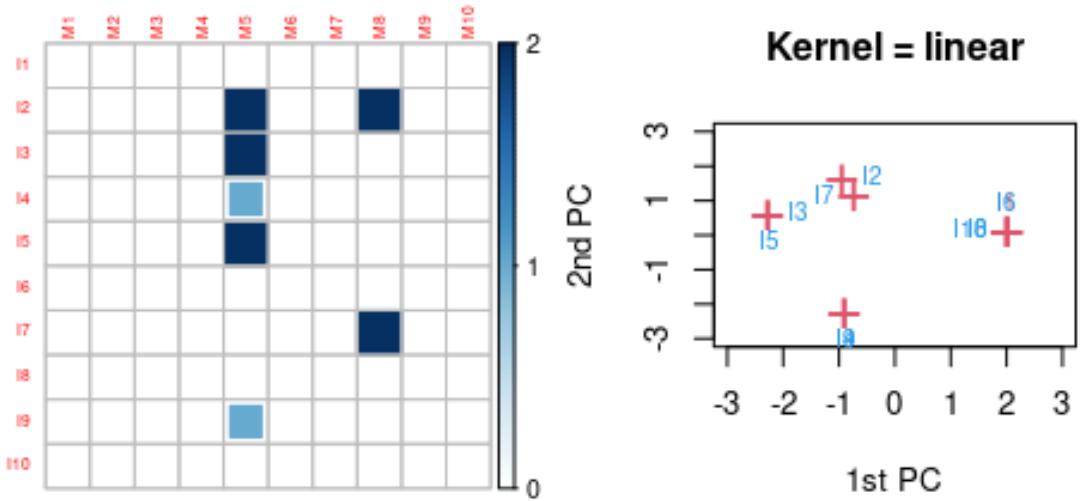


Figure 9: Corrélogramme et KPCA sur les données de génotypage simulées, noyaux linéaire, quadratique et additif.

Ces trois noyaux et les représentations adaptées nous permettent de réaliser une première analyse sur un jeu de données réel.

### 3.2 Jeu de données de génotypage réel

#### 3.2.1 Description

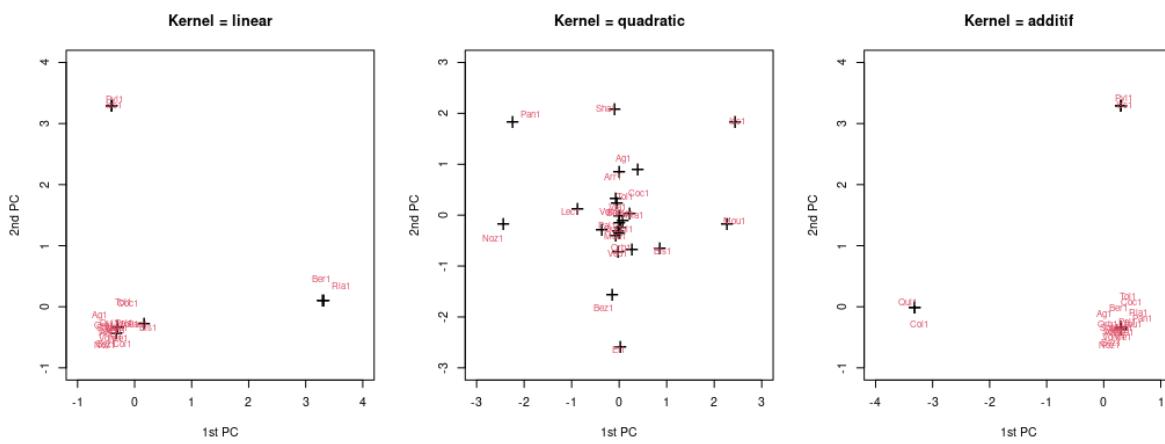
Le jeu de données réel à notre disposition provient des travaux de d'Harold Duruflé *et al.*, 2019 [9]. Il s'agit du génotypage de 22 écotypes d'*Arabidopsis thaliana* provenant du projet 1001 génome proche des Pyrénées comparés avec les références Col0 et Sha un écotype de haute altitude du Tadjikistan ; à l'aide de 338 marqueurs SNP ("Single Nucléotide polymorphism").

L'objet de cette étude, bien que cela ne soit pas le sujet de ce travail, était de déterminer quels sont les mécanismes d'adaptation d'*A. thaliana* aux températures d'altitude.

Les différents noyaux de génotypage ont été calculés sur la base de la matrice de génotypage de cette étude. La matrice étant déjà formatée telle que nous simulions les données de génotypage (Cf. 3.1.1), elle a pu être utilisée telle quel, à l'exception d'un entête de colonne qui contenait le caractère " $\circ$ " qui lève une exception.

### 3.2.2 Analyse

Des analyses en composantes principales à noyaux ont été effectuées (Figure 10), pour chacun des noyaux de génotypage. Nous pouvons voir que les résultats obtenus, c'est à dire la position des individus par rapport aux axes de la KPCA, sont différents pour les trois noyaux. Les noyaux linéaire et additif regroupent les individus en 3 catégories alors que le noyau quadratique semble plus sensible aux différences entre ces derniers. De plus, même si le nombre de groupes mis en évidence par les KPCA à noyau linéaire et additif est le même, la composition de ces derniers n'est pas équivalente.



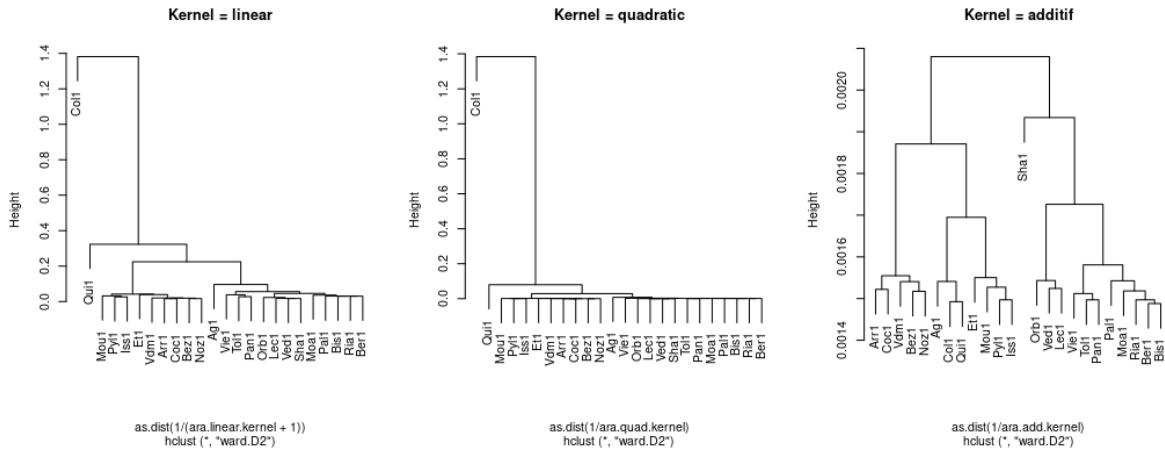


Figure 11: Dendrogrammes sur les données de génotypage d'*A. thaliana*, noyaux linéaire, quadratique et additif.

En l'état actuel du projet, nous ne pouvons tirer de conclusions sur ces résultats quant à leur interprétation au sens biologique. Toutefois, nous programmons une réunion avec Harold Duruflé, expert de ces données, afin de faire ressortir le sens de ces analyses. Nous pouvons toutefois anticiper le fait que chaque noyau, en raison de leurs spécificités telles que décrites par Wu *et al.*, 2011 [6], que les groupes sont formés différemment sur ces noyaux puisqu'ils mesurent des similarités différentes.

## 4 Discussion et conclusion

### 4.1 Conclusion

Ce travail a permis de se familiariser avec les méthodes à noyaux, d'un point de vue théorique dans un premier temps, en cherchant à comprendre et manipuler des noyaux simples sur des jeux de données simples, puis de mettre en oeuvre ces méthodes dans un contexte bioinformatique. En fin de compte, nous avons pu mettre en oeuvre une approche pour analyser des données de génotypage faisant ressortir différents schémas dans les données. L'interprétation biologique de ces résultats reste à être explorer afin de déterminer comment les variations génétiques des individus sont mises en avant dans un noyau ou dans un autre.

Par la suite, il sera intéressant de réaliser une étude de génétique d'association complète, c'est à dire en introduisant les données de phénotypage associées aux données de génotypage pour le jeu réel sur *A. thaliana*. Les résultats seront comparés à ceux obtenus par des méthodes de GWAS "classiques".

### 4.2 Discussion

Les méthodes à noyaux sont à la base des méthodes d'assignation de groupes du "machine learning" [10]. Nous avons donc cherché avec des noyaux de base à retrouver les groupes, ou catégories, de nos jeux de données. Il est à noter que tous les noyaux ne sont pas adaptés à tous les jeux de données. Ainsi, comme c'est le cas des méthodes linéaires que nous avons décrites dans le rapport de bibliographie (HMSN307), il est nécessaire de choisir une méthode adaptée à la structure des données à analyser. Pour résumer, il ne sert à rien d'utiliser un marteau pour planter des vis.

Parmi les perspectives de ce travail, il en est une particulièrement intéressante dans le contexte de la bioinformatique, l'analyse de données de types graphe ou réseau d'interactions. En effet, de nombreuses données de biologie sont structurées de cette manière. Nous pouvons citer les réseaux d'interactions de protéines par exemple.

Par la suite, les perspectives de ce travail seront de développer des approches supervisées à l'instar des analyses proposées dans le "package" `mixOmics`, ainsi que l'ensemble des méthodes de type `sparse` et `discriminant analysis`. L'idée sera alors de proposer un "package" permettant d'intégrer des données de biologie de natures différentes.

## References

- [1] Elad Noor, Sarah Cherkaoui, and Uwe Sauer. Biological insights through omics data integration. *Current Opinion in Systems Biology*, 15, 03 2019.
- [2] Florian Rohart, Benoît Gautier, Amrit Singh, and Kim-Anh Lê Cao. mixomics: An r package for 'omics feature selection and multiple data integration. *PLoS computational biology*, 13:11, 2017.
- [3] Jérôme Mariette. Apprentissage statistique pour l'intégration de données omiques. *online: <HTTP://TEL.ARCHIVES-OUVERTES.FR/TEL-01666744v3>*, 2018.
- [4] Nathalie Villa-Vialaneix Jérôme Mariette. Unsupervised multiple kernel learning for heterogeneous data integration. *Bioinformatics*, 34:1009–1015, 2018.
- [5] Yosuke Kawai Kazuharu Misawa Takahiro Mimori Masao Nagasaki Takanori Hasegawa, Kaname Kojima. Ap-skat: Highly-efficient genome-wide rare variant association test. *BMC Genomics*, 17(745):1–8, 2016.
- [6] Michael Wu, Seunggeun Lee, Tianxi Cai, Yun Li, Michael Boehnke, and Xihong Lin. Rare-variant association testing for sequencing data with the sequence kernel association test. *American journal of human genetics*, 89:82–93, 07 2011.
- [7] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.
- [8] Kenji Fukumizu, Arthur Gretton, Gert R. Lanckriet, Bernhard Schölkopf, and Bharath K. Sriperumbudur. Kernel choice and classifiability for rkhs embeddings of probability distributions. *Advances in Neural Information Processing Systems 22*, pages 1750–1758, 2009.
- [9] Harold Duruflé, Philippe Ranocha, Duchesse Lacour Mbadinga Mbadinga, Sébastien Déjean, Maxime Bonhomme, Hélène San Clemente, Sébastien Viudes, Ali Eljebbawi, Valérie Delorme-Hinoux, Julio Sáez-Vásquez, Jean-Philippe Reichheld, Nathalie Escaravage, Monique Burrus, and Christophe Dunand. Phenotypic trait variation as a response to altitude-related constraints in arabidopsis populations. *Frontiers in Plant Science*, 10:430, 2019.
- [10] Javier M.Moguerzab Anastasios Apsemidisa, Stelios Psarakisa. A review of machine learning kernel methods in statistical process monitoring. *Computers Industrial Engineering*, 142:1–12, 2020.

## **5 Annexes**

Annexe 1 : Synthetic Datasets

# Synthetic Datasets

Xavier Grand

25/06/2020

## Packages

```
list.of.packages <- c("ggplot2", "kernlab", "corrplot", "FactoMineR")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages() [, "Package"])]
if(length(new.packages)) install.packages(new.packages)

library(ggplot2)
library(kernlab)

##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##      alpha

library(corrplot)

## corrplot 0.84 loaded

library(FactoMineR)
```

## Datasets simulation

### Functions to create Datasets

Dataset rim :

```
rim <- function(n){
  x1 <- rnorm(n, sd=0.5)
  x2 <- rnorm(n)

  y1 <- runif(n)
  y2 <- -1 + (x2 / 1.2)^2 + runif(n)
```

```

x <- c(x1,x2)
y <- c(y1,y2)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I",1:length(x))
df$dataset <- rep("rim", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

Dataset Concentric circle:

```

circle <- function(n){

tmp1 <- seq(0,2*pi, l=n)

x1 <- cos(tmp1) + rnorm(n, sd=0.1)
y1 <- sin(tmp1) + rnorm(n, sd=0.1)

x2 <- 0.2*cos(tmp1) + rnorm(n, sd=0.1)
y2 <- 0.2*sin(tmp1) + rnorm(n, sd=0.1)

x3 <- 3*cos(tmp1) + rnorm(n, sd=0.1)
y3 <- 3*sin(tmp1) + rnorm(n, sd=0.1)

## plus petit cercle extérieur :

x4 <- 3.5 + 0.4*cos(tmp1) + rnorm(n, sd=0.1)
y4 <- 3.5 + 0.4*sin(tmp1) + rnorm(n, sd=0.1)

x <- c(x1,x2,x3,x4)
y <- c(y1,y2,y3,y4)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I",1:length(x))
df$dataset <- rep("circle", length(x))
df$predictor <- c(rep("A", n), rep("B", n), rep("C", n), rep("D", n))

df
}

```

Dataset rafters:

```

rafters <- function(n){

tmp1 <- seq(0,2*pi, l=n)

```

```

x1 <- cos(tmp1) + rnorm(n, sd=0.1) -0.75
y1 <- seq(-2, 2, along.with = x1) + rnorm(n, sd=0.1) -0.75

x2 <- -cos(tmp1) + rnorm(n, sd=0.1) + 0.75
y2 <- -seq(-2, 2, along.with = x2) + rnorm(n, sd=0.1) + 0.75

x <- c(x1, x2)
y <- c(y1, y2)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I", 1:n)
df$dataset <- rep("rafters", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

### Dataset Moon:

```

moon <- function(n){

tmp1 <- seq(-pi/2, pi/2, l = n)

x1 <- sin(tmp1)*2 + rnorm(n, sd=0.1) + 1
y1 <- cos(tmp1)*2 + rnorm(n, sd=0.1) - 0.25

x2 <- -sin(tmp1)*2 + rnorm(n, sd=0.1) - 1
y2 <- -cos(tmp1)*2 + rnorm(n, sd=0.1) + 0.25

x <- c(x1, x2)
y <- c(y1, y2)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I", 1:n)
df$dataset <- rep("moon", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

### Dataset spirals:

```

spirals <- function(n){

a <- seq(0.1, 4, l = n)
b <- 0.5

```

```

t1 <- seq(0, 5 * pi, l = n)
t2 <- seq(5 * pi, 10 * pi, l = n)

x1 <- a * exp(b ^ t1) * cos(t1) + rnorm(n, sd=0.1)
y1 <- a * exp(b ^ t1) * sin(t1) + rnorm(n, sd=0.1)

x2 <- a * exp(b ^ t2) * cos(t2) + rnorm(n, sd=0.1)
y2 <- a * exp(b ^ t2) * sin(t2) + rnorm(n, sd=0.1)

x <- c(x1, x2)
y <- c(y1, y2)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I", 1:n)
df$dataset <- rep("spirals", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

Dataset blossom:

```

blossum <- function(n){

a1 <- seq(0.1, 2, l = n)
a2 <- seq(2, 3.8, l = n)
b <- 0.5

t <- seq(0, 10 * pi, l = n)

x1 <- a1 * exp(b ^ t) * cos(t) + rnorm(n, sd=0.1)
y1 <- a1 * exp(b ^ t) * sin(t) + rnorm(n, sd=0.1)

x2 <- a2 * exp(b ^ t) * cos(t) + rnorm(n, sd=0.1)
y2 <- a2 * exp(b ^ t) * sin(t) + rnorm(n, sd=0.1)

x <- c(x1, x2)
y <- c(y1, y2)

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I", 1:n)
df$dataset <- rep("blossum", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

Dataset roll:

```
roll <- function(n){

  a <- seq(0, 4, l = 2*n)
  b <- 0.5
  t <- seq(0, 10 * pi/2, l = 2*n)

  x <- a * exp(b ^t) * cos(t) + rnorm(2*n, sd=0.2)
  y <- a * exp(b ^t) * sin(t) + rnorm(2*n, sd=0.2)

  df <- cbind(x, y)
  df <- as.data.frame(df)
  df$name <- paste0("I",1:2*n)
  df$dataset <- rep("swiss-roll", length(x))
  df$predictor <- c(rep("A", n), rep("B", n))

  df
}
```

Dataset Biohazard :

```
hazard <- function(n){

  tmp1 <- seq(0,2*pi, l=n)

  x1 <- 0.9 + cos(tmp1) + rnorm(n, sd=0.1)
  y1 <- -0.9 + sin(tmp1) + rnorm(n, sd=0.1)

  x2 <- -0.9 + cos(tmp1) + rnorm(n, sd=0.1)
  y2 <- -0.9 + sin(tmp1) + rnorm(n, sd=0.1)

  x3 <- cos(tmp1) + rnorm(n, sd=0.1)
  y3 <- 0.9 + sin(tmp1) + rnorm(n, sd=0.1)

  x4 <- cos(tmp1) + rnorm(n, sd=0.1)
  y4 <- sin(tmp1) + rnorm(n, sd=0.1)

  x5 <- runif(n, min = -3, max = 3)
  y5 <- -2 + rnorm(n, sd=0.1)

  x6 <- runif(n, min = -3, max = 0)
  y6 <- (5/3) * x6 + 3 + rnorm(n, sd=0.1)

  x7 <- runif(n, min = 0, max = 3)
  y7 <- (-5/3) * x7 + 3 + rnorm(n, sd=0.1)

  x <- c(x1,x2,x3,x4,x5,x6,x7)
  y <- c(y1,y2,y3,y4,y5,y6,y7)

  df <- cbind(x, y)
```

```

df <- as.data.frame(df)
df$name <- paste0("I",1:7*n)
df$dataset <- rep("hazard", length(x))
df$predictor <- c(rep("A", n), rep("B", n), rep("C", n), rep("D", n),
rep("E", n), rep("F", n), rep("G", n))

df
}

```

### Dataset Classification:

```

classif <- function(n){

x1 <- abs(rnorm(n, sd = 1)) - 0.75
y1 <- x1 * runif(n, min = -2, max = 2)

x2 <- -abs(rnorm(n, sd = 1)) + 0.75
y2 <- x2 * runif(n, min = -2, max = 2)

x <- c(x1,x2)
y <- c(y1,y2)

# plot(x, y, col=rep(1:2, each=n))

df <- cbind(x, y)
df <- as.data.frame(df)
df$name <- paste0("I",1:n)
df$dataset <- rep("classification", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

### Dataset Concentric circles 2:

```

circles2 <- function(n){

tmp1 <- seq(0,2*pi, l=n)

x1 <- cos(tmp1) + rnorm(n, sd=0.1)
y1 <- sin(tmp1) + rnorm(n, sd=0.1)

x2 <- 4*cos(tmp1) + rnorm(n, sd=0.1)
y2 <- 4*sin(tmp1) + rnorm(n, sd=0.1)

x <- c(x1,x2)
y <- c(y1,y2)

df <- cbind(x, y)
df <- as.data.frame(df)

```

```

df$name <- paste0("I", 1:length(x))
df$dataset <- rep("SimpleCircles", length(x))
df$predictor <- c(rep("A", n), rep("B", n))

df
}

```

## Datasets creation

Data points to simulate:

```
n = 30
```

Datasets creation *sensus stricto*:

```

jante <- rim(n)
cercle <- circle(n)
chevrons <- rafters(n)
lunes <- moon(n)
spirales <- spirals(n)
fleur <- blossom(n)
swissRoll <- roll(n)
bioHazard <- hazard(n)
classificationDataset <- classif(n)
simpleCircles <- circles2(n)

completeDataSet <- rbind(jante, cercle, chevrons, lunes, spirales, fleur,
                           swissRoll, classificationDataset, simpleCircles)

datasets <- list(simpleCircles)
names.df <- c("SimpleCircles")

```

## Plot Datasets

Plot all-datasets

```
table(completeDataSet$dataset)
```

```

##
##      blossom          circle  classification          moon          rafters
##            60             120                 60             60             60
##      rim  SimpleCircles          spirals    swiss-roll
##            60               60                 60               60

```

```

str(completeDataSet)

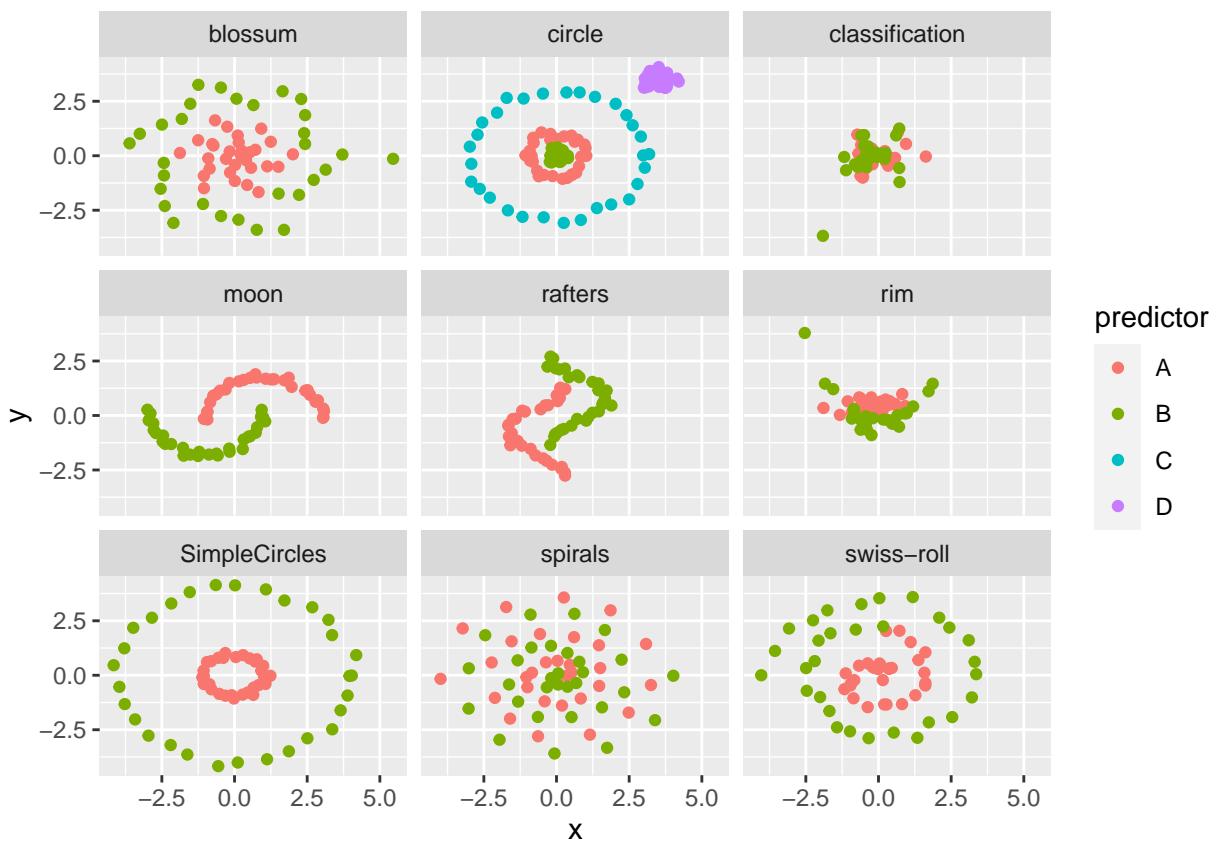
## 'data.frame':   600 obs. of  5 variables:
## $ x      : num  -0.2831 -0.1106 -0.3078 -0.2493 0.0742 ...
## $ y      : num  0.4877 0.0847 0.5847 0.8317 0.5638 ...
## $ name   : chr  "I1" "I2" "I3" "I4" ...
## $ dataset: chr  "rim" "rim" "rim" "rim" ...
## $ predictor: chr  "A" "A" "A" "A" ...

```

```

qplot(x, y, data = completeDataSet) +
  facet_wrap(~ dataset) +
  aes(color = predictor)

```



Fonction d'affichage du dataset original :

```

originalPlot <- function(df) {

  plot(df[,1:2],
       main = paste("Dataset = ", names(df)[i], sep = ""),
       type = "n",
       col = 1 + as.numeric(as.factor(df$predictor)))
  text(df[,1:2], label = df$name, adj = c(0,0),

```

```

    col = 1 + as.numeric(as.factor(df$predictor)))
}

```

## Visualisation des transformations en noyaux

```

transf.gauss <- function(x, sig)
{
  exp(-sig*x**2)
}

transf.poly <- function(x, echel, decal, deg)
{
  (echel*(x+decal)**deg
}

```

### Visualisation des transformations en noyaux polynomial :

Cette représentation n'est pas bonne, puisqu'il n'existe pas un lien direct de cette manière entre la distance Euclidienne et la fonction du noyau polynomial.

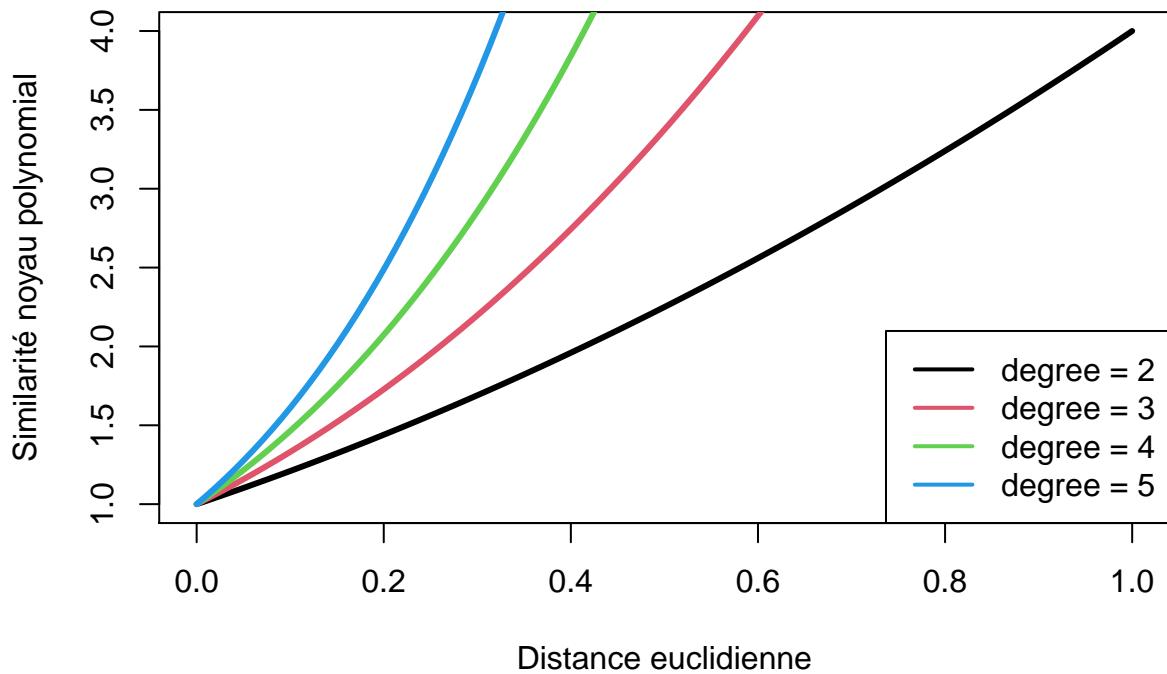
```

x <- seq(0, 1, length.out = 100)
plot(x, transf.poly(x, 1, 1, 2), type="l", lwd=3,
      xlab="Distance euclidienne", ylab="Similarité noyau polynomial",
      main = "Noyau polynomial")

points(x, transf.poly(x, 1, 1, 3), type="l", lwd=3, col=2)
points(x, transf.poly(x, 1, 1, 4), type="l", lwd=3, col=3)
points(x, transf.poly(x, 1, 1, 5), type="l", lwd=3, col=4)
legend("bottomright",
      legend = paste("degree =", c(2, 3, 4, 5)),
      lty=1,
      lwd=2,
      col=1:4)

```

## Noyau polynomial

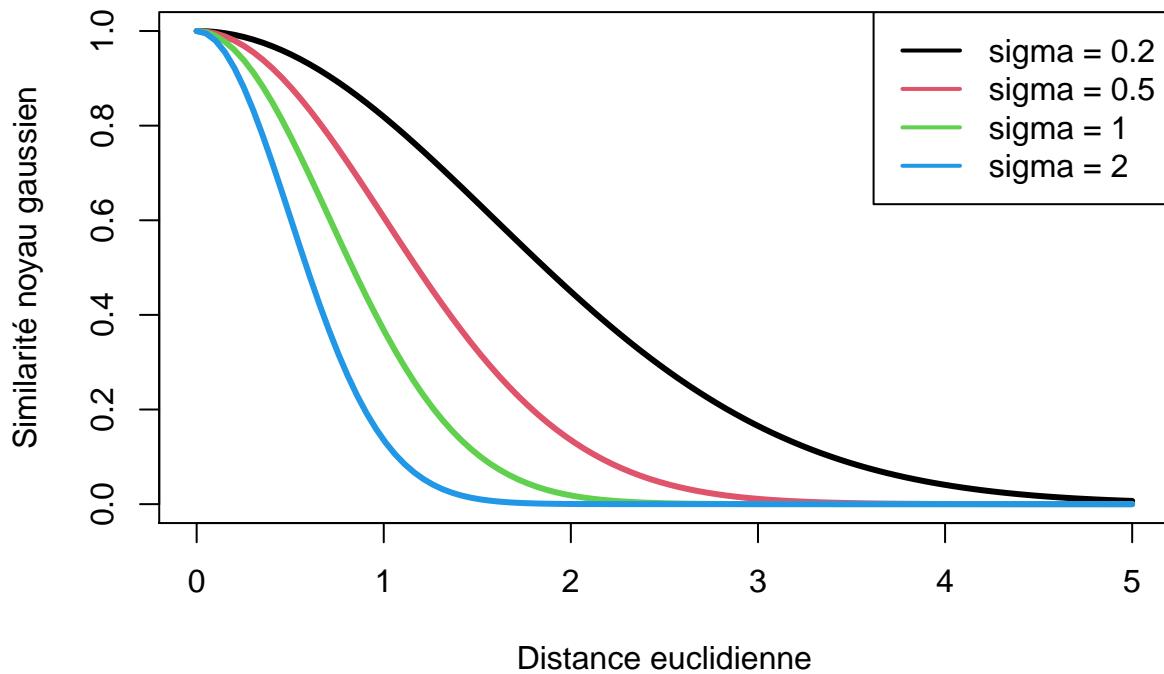


Visualisation des transformations en noyaux gaussiens :

```
x <- seq(0, 5, length.out = 100)
plot(x, transf.gauss(x, 0.2), type="l", lwd=3, ylim=c(0,1),
      xlab="Distance euclidienne", ylab="Similarité noyau gaussien",
      main = "Noyau gaussien")

points(x, transf.gauss(x, 0.5), type="l", lwd=3, col=2)
points(x, transf.gauss(x, 1), type="l", lwd=3, col=3)
points(x, transf.gauss(x, 2), type="l", lwd=3, col=4)
legend("topright",
       legend = paste("sigma =", c(0.2, 0.5, 1, 2)),
       lty=1,
       lwd=2,
       col=1:4)
```

## Noyau gaussien



## Heuristique Sigma

Le paramètre sigma peut être obtenu par une heuristique telle que :

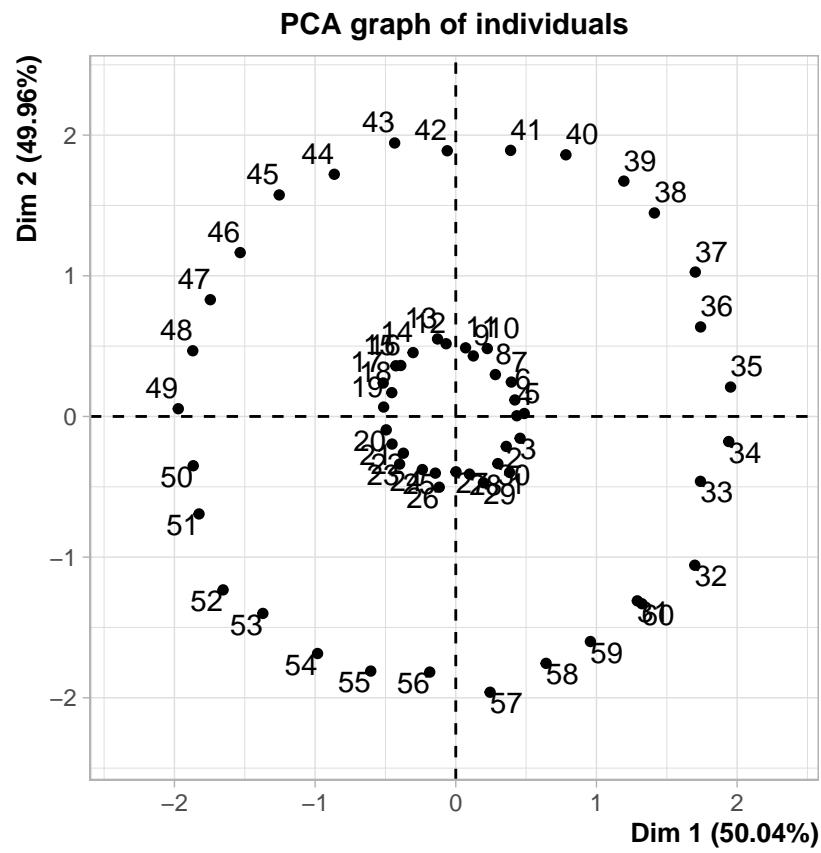
```
sh <- function(df) {  
  sh <- median(1/(dist(df)**2))  
}
```

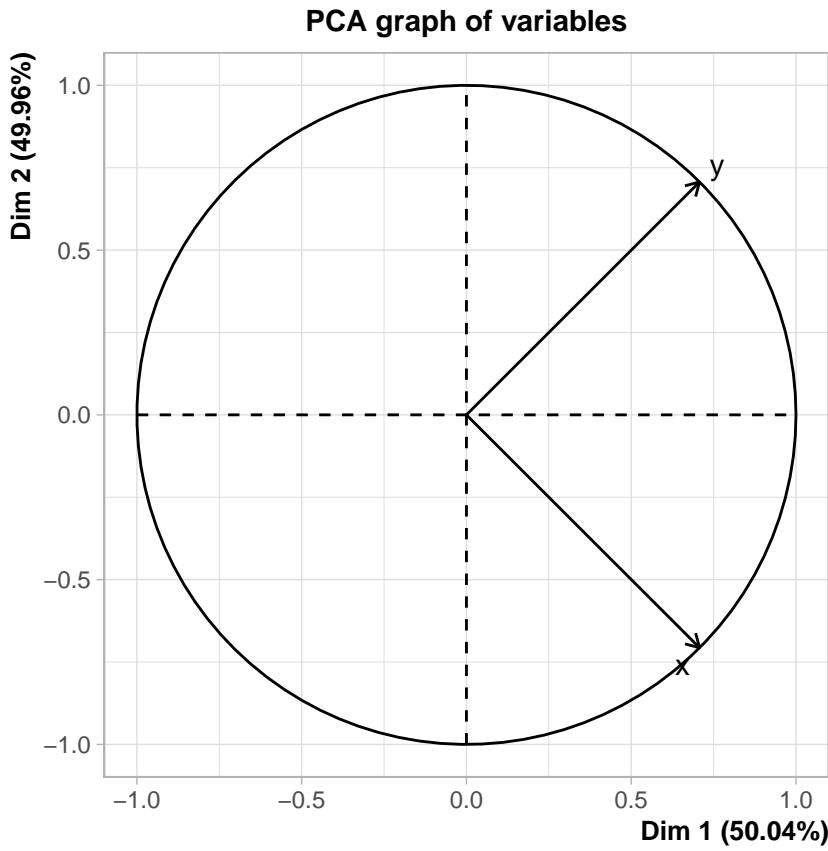
## ANALYSIS

### PCA

```
for (i in 1:length(names.df)) {  
  
  print(i)  
  df <- datasets[[i]]  
  print(str(df))  
  print(names.df[i])  
  
  acp_classique <- PCA(as.matrix(df[,1:2]), ncp = 2)
```

```
    acp_classique$eig  
}
```





## KPCA

Noyau Linéaire : 1 composante

```

for (i in 1:length(names.df)) {

  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])

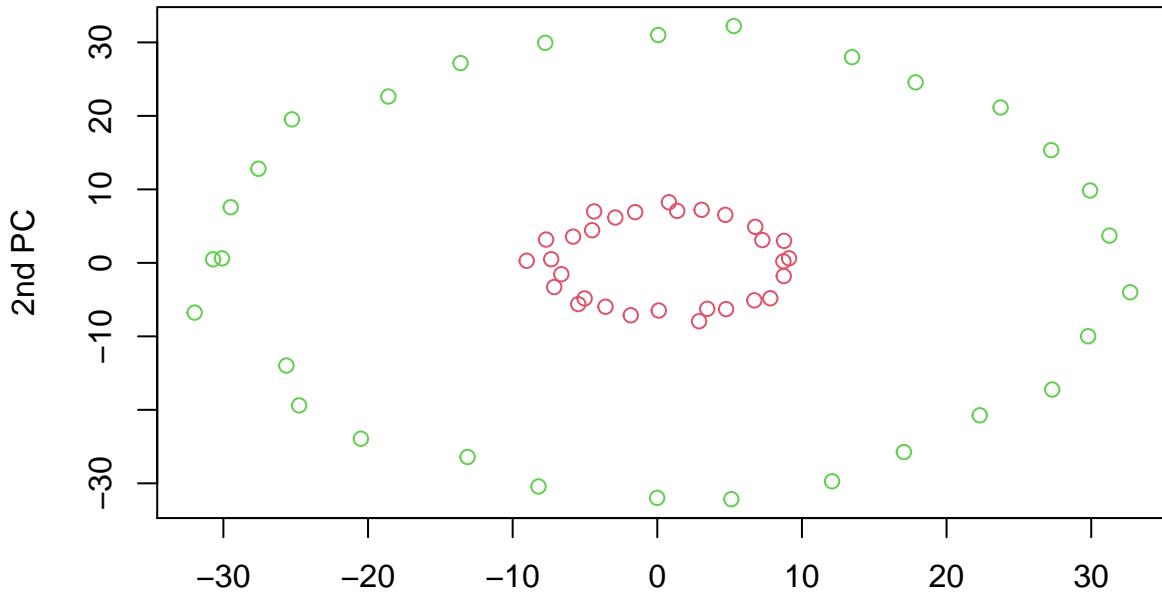
  kpc.linear <- kpca(as.matrix(df[,1:2]), kernel="vanilladot",
                      kpar=list(), features=2)

  pcv(kpc.linear)
  eig(kpc.linear)

  plot(rotated(kpc.linear),
       xlab="1st PC", ylab="2nd PC",
       main = paste("Kernel = Linear, Dataset = ", names.df[i], sep = ""),
       sub = paste("eigen : Comp1 = ", eig(kpc.linear)[1],
                  " Comp2 = ", eig(kpc.linear)[2]),
       col = 1 + as.numeric(as.factor(df$predictor)))
}

```

## Kernel = Linear, Dataset = SimpleCircles



1st PC

eigen : Comp1 = 4.46335611252915 Comp2 = 4.1711416898666

### Noyau Linéaire : 2 composantes

```
for (i in 1:length(names.df)) {

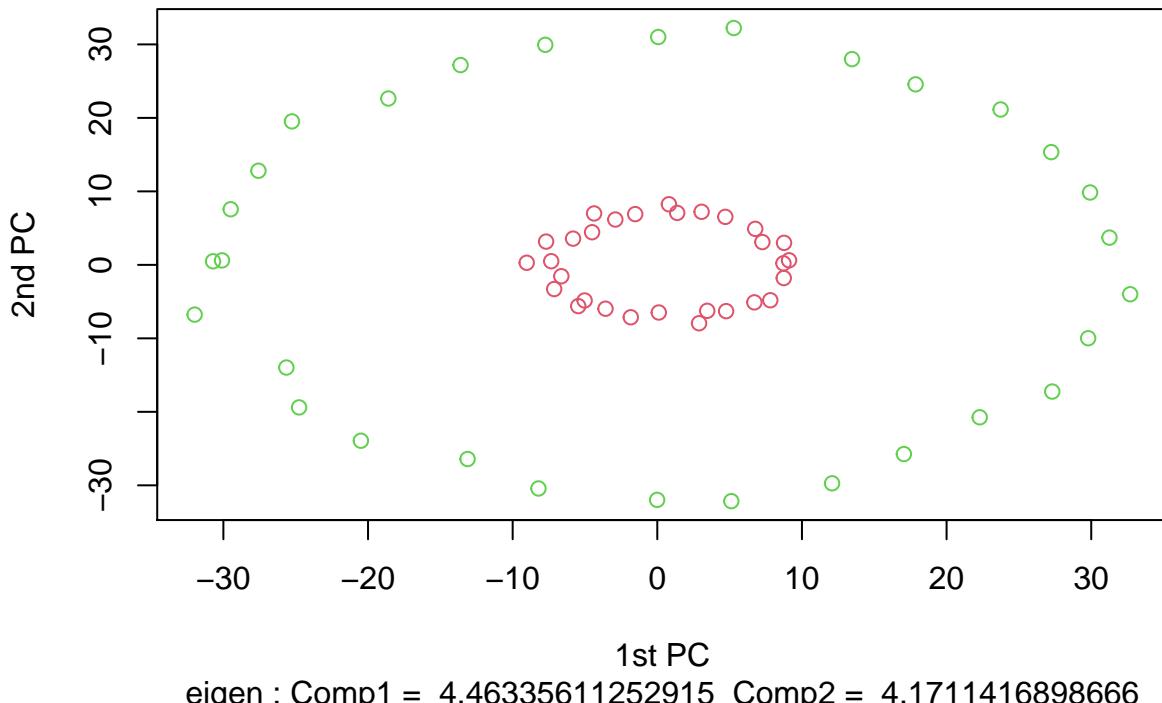
  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])

  kpc.linear <- kpca(as.matrix(df[,1:2]), kernel="vanilladot",
                      kpar=list(), features=2)

  pcv(kpc.linear)
  eig(kpc.linear)

  plot(rotated(kpc.linear),
       xlab="1st PC", ylab="2nd PC",
       main = paste("Kernel = Linear, Dataset = ", names.df[i], sep = ""),
       sub = paste("eigen : Comp1 = ", eig(kpc.linear)[1],
                  " Comp2 = ", eig(kpc.linear)[2]),
       col = 1 + as.numeric(as.factor(df$predictor)))
}
```

## Kernel = Linear, Dataset = SimpleCircles



Noyau Gaussien : composantes

```
par(mfrow = c(2, 3))

for (i in 1:length(names.df)) {

  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])
  sig <- sh(df[, 1:2])

  # originalPlot(df)

  sigma <- list(0.01, 0.05, 0.1, 0.5, 1, sig)

  for (s in sigma) {
    print(s)
    kpc.gaussian <- kpca(as.matrix(df[,1:2]), kernel="rbfdot",
                           kpar=list(sigma=s), features=2)

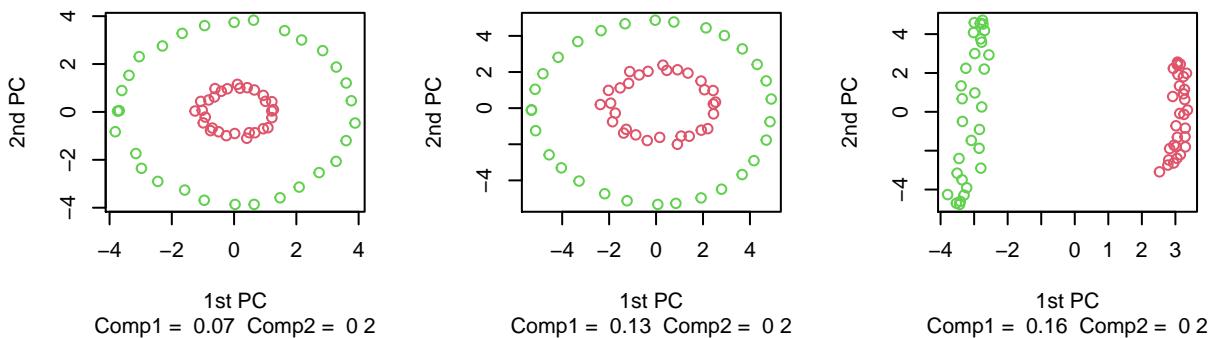
    pcv(kpc.gaussian)
    eig(kpc.gaussian)
```

```

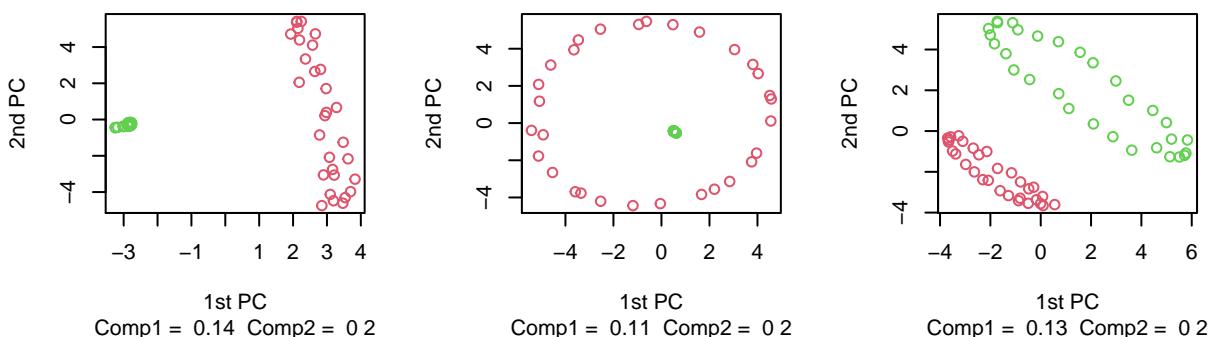
plot(rotated(kpc.gaussian),
      xlab="1st PC", ylab="2nd PC",
      main = paste("Kernel = Gaussian, Dataset = ", names.df[i],
                  ", \nsigma = ", round(s, 2), sep = ""),
      sub = paste("Comp1 = ", round(eig(kpc.gaussian)[1], 2),
                  " Comp2 = ", round(eig(kpc.gaussian)[2]), 2),
      col = 1 + as.numeric(as.factor(df$predictor)))
}
}

```

**nel = Gaussian, Dataset = Simple**  
**nel = Gaussian, Dataset = Simple**  
**nel = Gaussian, Dataset = Simple**



**nel = Gaussian, Dataset = Simple**  
**nel = Gaussian, Dataset = Simple**  
**nel = Gaussian, Dataset = Simple**



1 jeu de données, plusieurs noyaux :

```

names.nx <- c("Linear", "Gaussian", "Polynomial")

for (i in 1:length(names.df)) {
  df <- datasets[[i]]
  noyaux <- list(vanilladot(), rbfddot(sigma = sh(df[,1:2])),
                 polydot(degree = 2, scale = 0.5, offset = 5))

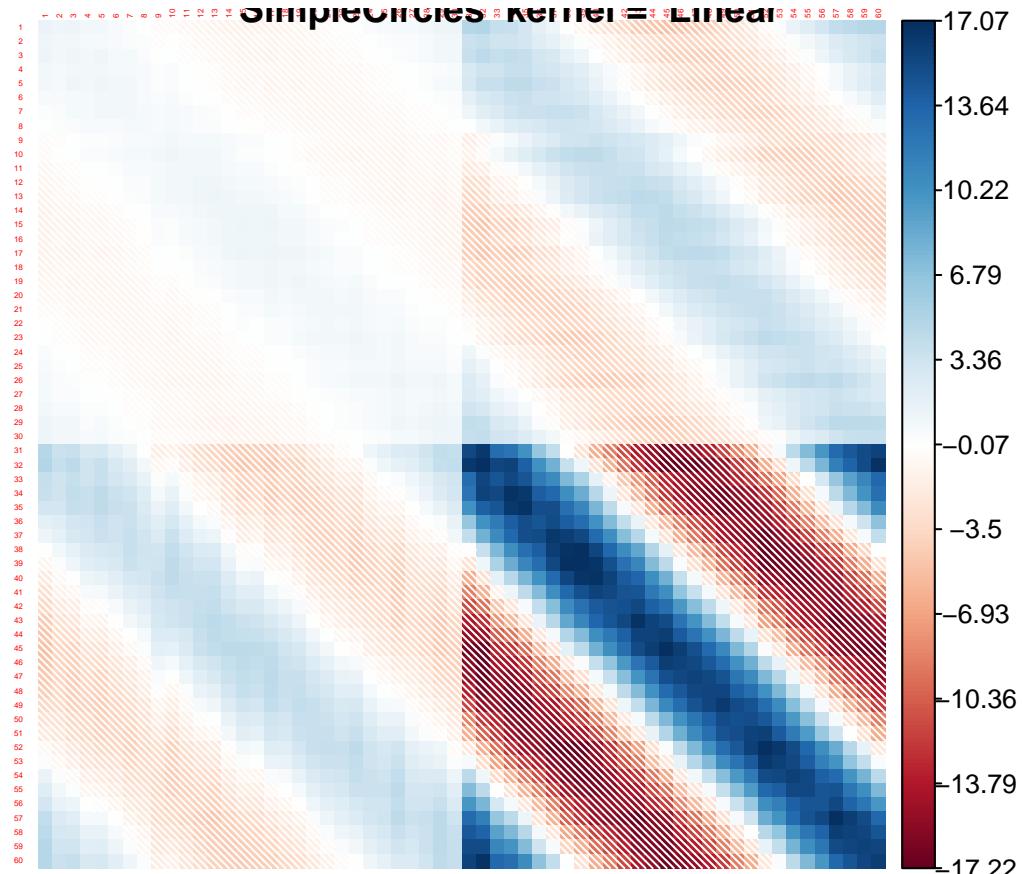
  for(j in 1:length(noyaux)) {
    nx <- noyaux[[j]]
    dist.nx <- kernelMatrix(nx, as.matrix(df[,1:2]))
    corrplot(as.matrix(dist.nx), is.corr=FALSE, tl.cex = 0.25,

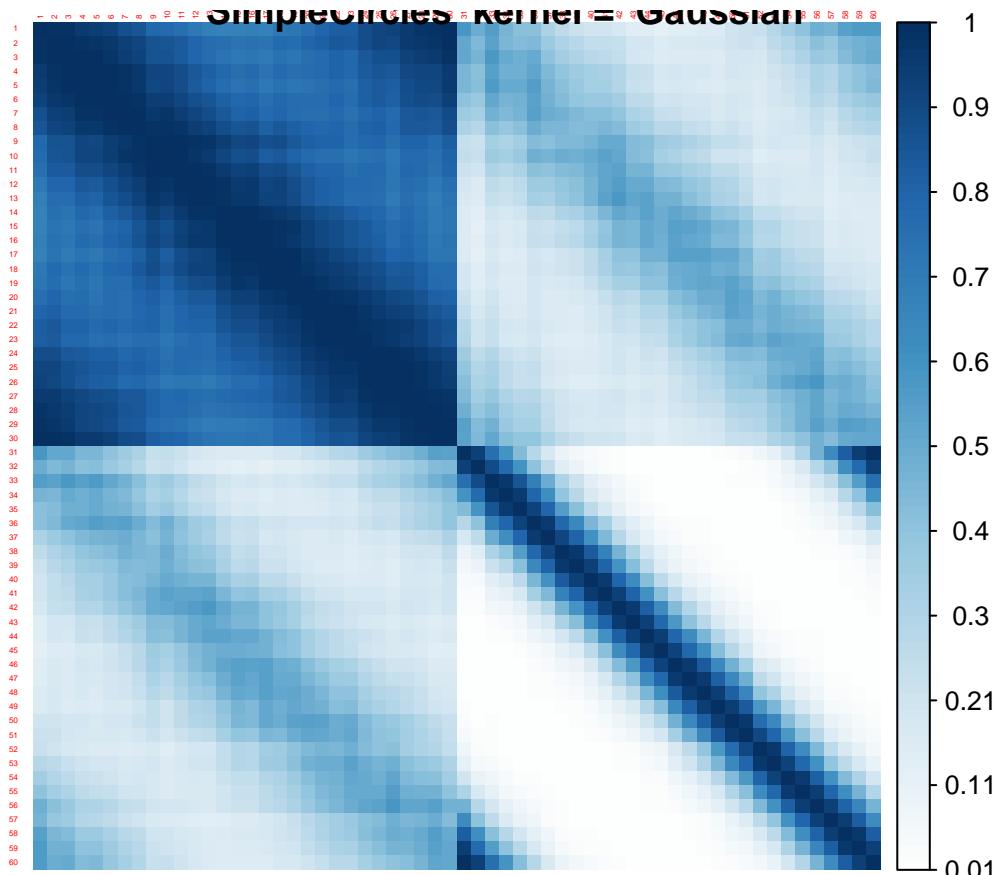
```

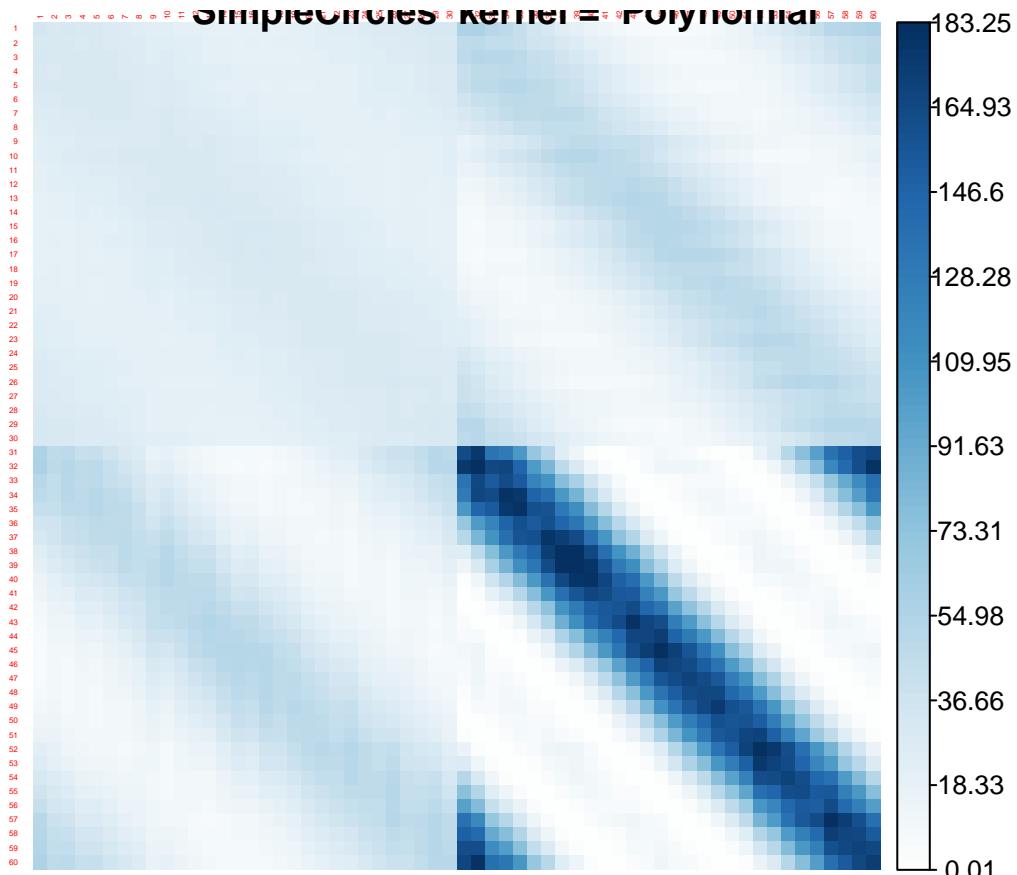
```

        method = "shade",
        title = paste(names.df[i], " kernel = ", names.nx[j]))
    }
}

```







Datasets simpleCircles Noyaux gaussien, variation de sigma

```

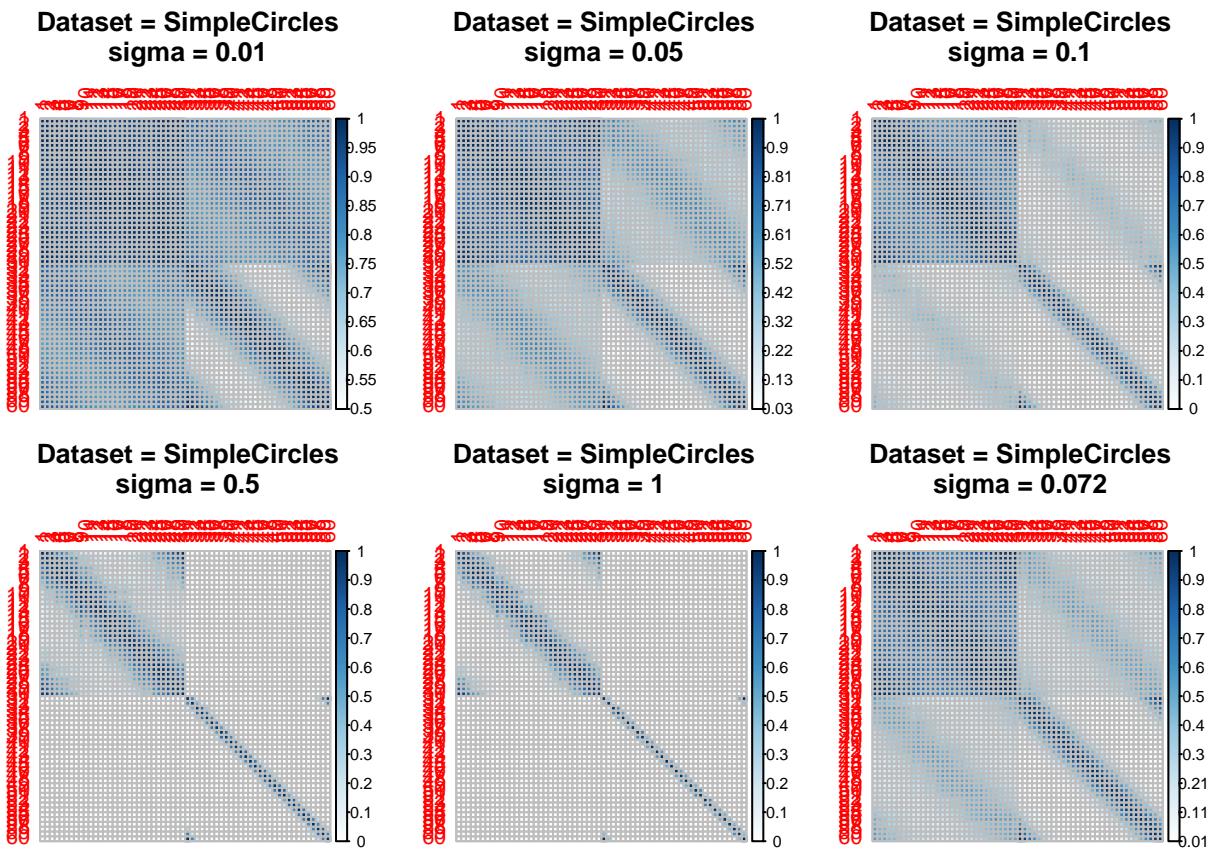
par(mfrow = c(2,3))

df <- simpleCircles

sigma <- list(0.01, 0.05, 0.1, 0.5, 1, sh(df[,1:2]))

for (s in sigma) {
  noyau.gaussien <- rbfDot(sigma = s)
  dist.gauss <- kernelMatrix(noyau.gaussien, as.matrix(df[,1:2]))
  corrplot(as.matrix(dist.gauss), is.corr=FALSE, tl.cex = 1,
           title = paste("Dataset = ",names.df[i],
                         "\nsigma = ", round(s,3),
                         sep = ""),
           method = c("square"),
           # mpg = c(3, 1, 0)
           mar = c(0,0,4,0)
  )
}

```



## KK-means

Kernel k-means : Linear

```
par(mfrow = c(3,3))

for (i in 1:length(names.df)) {
  df <- datasets[[i]]

  # originalPlot(df)

  # sigma <- sh(df[,1:2]) # list(0.001, 1, 1.5, 5, sh(df[,1:2]))

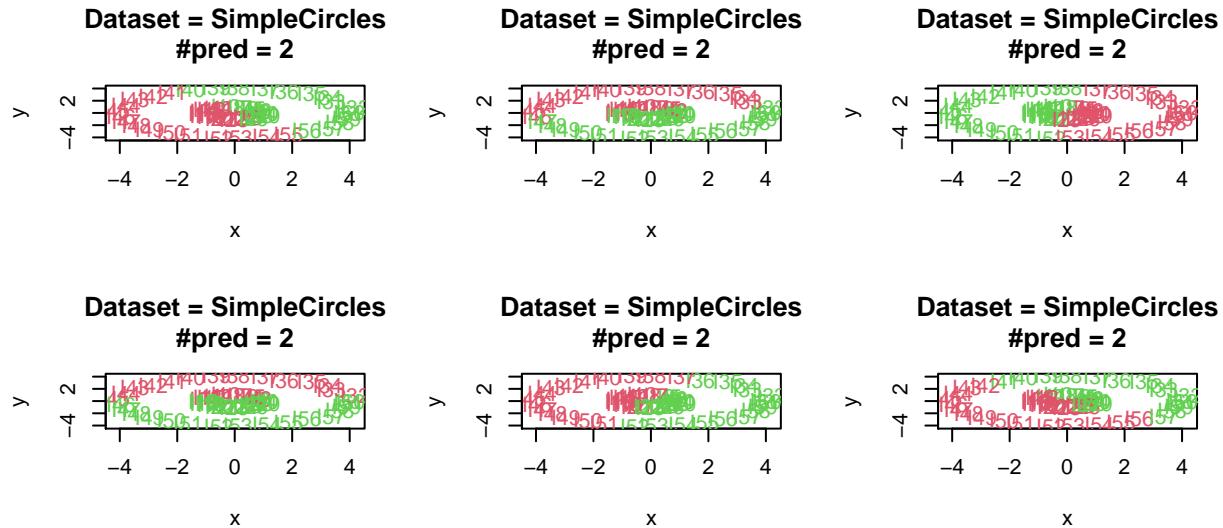
  for (s in sigma) {
    KMclusters <- kkmeans(as.matrix(df[,1:2]),
                           centers = length(unique(df$predictor)),
                           kernel = "vanilladot",
                           kpar = list(),
                           alg="kkmeans",
                           p=0)

    plot(df[,1:2],
         main = paste("Dataset = ",names.df[i],
                     "\n#pred = ", length(unique(df$predictor)), sep = ""))
  }
}
```

```

        type = "n",
        col = 1 + KMclusters@.Data)
text(df[,1:2], label = df$name,
     col = 1 + KMclusters@.Data)
}
}

```



Kernel k-means : Gaussian.

```

par(mfrow = c(3,3))

for (i in 1:length(names.df)) {
  df <- datasets[[i]]

  # originalPlot(df)

  sigma <- sh(df[,1:2]) # list(0.001, 1, 1.5, 5, sh(df[,1:2]))

  for (s in sigma) {
    KMclusters <- kkmeans(as.matrix(df[,1:2]),
                           centers = length(unique(df$predictor)),
                           kernel = "rbfdot",
                           kpar = list(sigma = s),

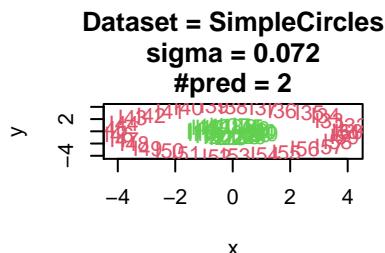
```

```

        alg="kkmeans",
        p=0)

plot(df[,1:2],
      main = paste("Dataset = ",names.df[i], "\nsigma = ", round(s, 3),
                  "\n#pred = ", length(unique(df$predictor)), sep = ""),
      type = "n",
      col = 1 + KMclusters@.Data)
text(df[,1:2], label = df$name,
      col = 1 + KMclusters@.Data)
}
}

```



## Kernel k-means : polynomial. A RETRAVAILLER

```

par(mfrow = c(3,3))

for (i in 1:length(names.df)) {
  df <- datasets[[i]]

  ## Affichage du dataset d'origine en x,y
  # originalPlot(df)

```

```

KMclusters <- kkmeans(as.matrix(df[,1:2]), 2, kernel = "polydot",
                      kpar = "automatic", # list(3, 4, 1),
                      #list(degree, scale, offset)

                      alg="kkmeans", p=1)

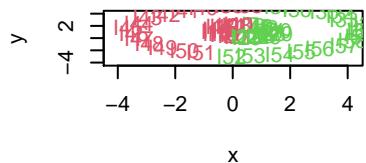
plot(df[,1:2],
      main = paste("Kernel = polynomial, \nDataset = ",names.df[i], sep = ""),
      type = "n",
      col = 1 + KMclusters)

text(df[,1:2], label = df$name, adj = c(0,0),
      col = 1 + KMclusters)
}

KMclusters

```

**Kernel = polynomial,  
Dataset = SimpleCircles**



All datasets :

```

datasets <- list(jante, cercle, chevrons, lunes, spirales, fleur, swissRoll,
                  classificationDataset, simpleCircles)

```

```
names.df <- c("Rim", "Circles", "Rafters", "Moon", "Spirals", "Blossum",
            "SwissRoll", "Classification", "SimpleCircles")
```

## ANALYSIS

### KPCA

Noyau Linéaire :

```
par(mfrow = c(3,3))

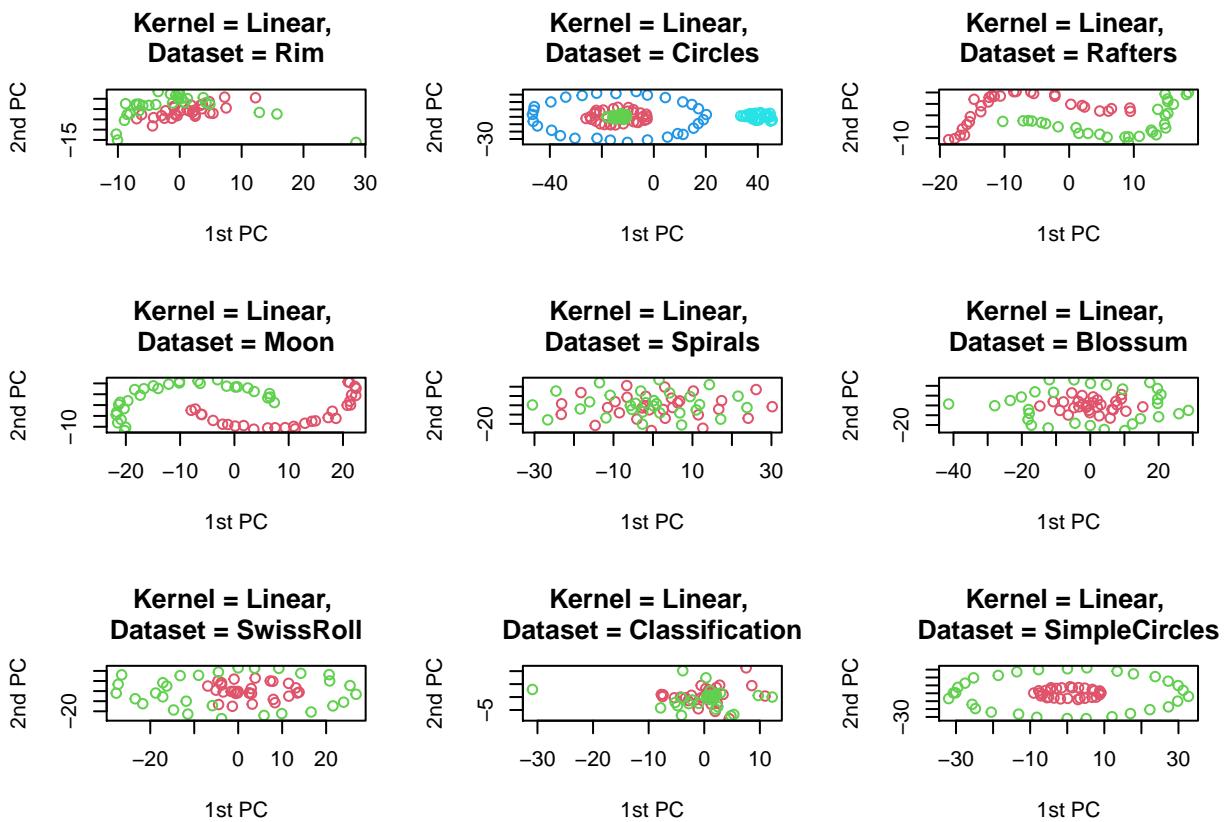
for (i in 1:length(names.df)) {

  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])

  kpc.linear <- kpca(as.matrix(df[,1:2]), kernel="vanilladot",
                      kpar=list(), features=2)

  pcv(kpc.linear)
  eig(kpc.linear)

  plot(rotated(kpc.linear),
       xlab="1st PC", ylab="2nd PC",
       main = paste("Kernel = Linear, \nDataset = ", names.df[i], sep = ""),
       # sub = paste("eigen : Comp1 = ", round(eig(kpc.linear)[1], 2), " Comp2 = ", round(eig(kpc.linear)[2], 2)),
       col = 1 + as.numeric(as.factor(df$predictor)))
}
```



Noyau Gaussien :

```
par(mfrow = c(3,3))

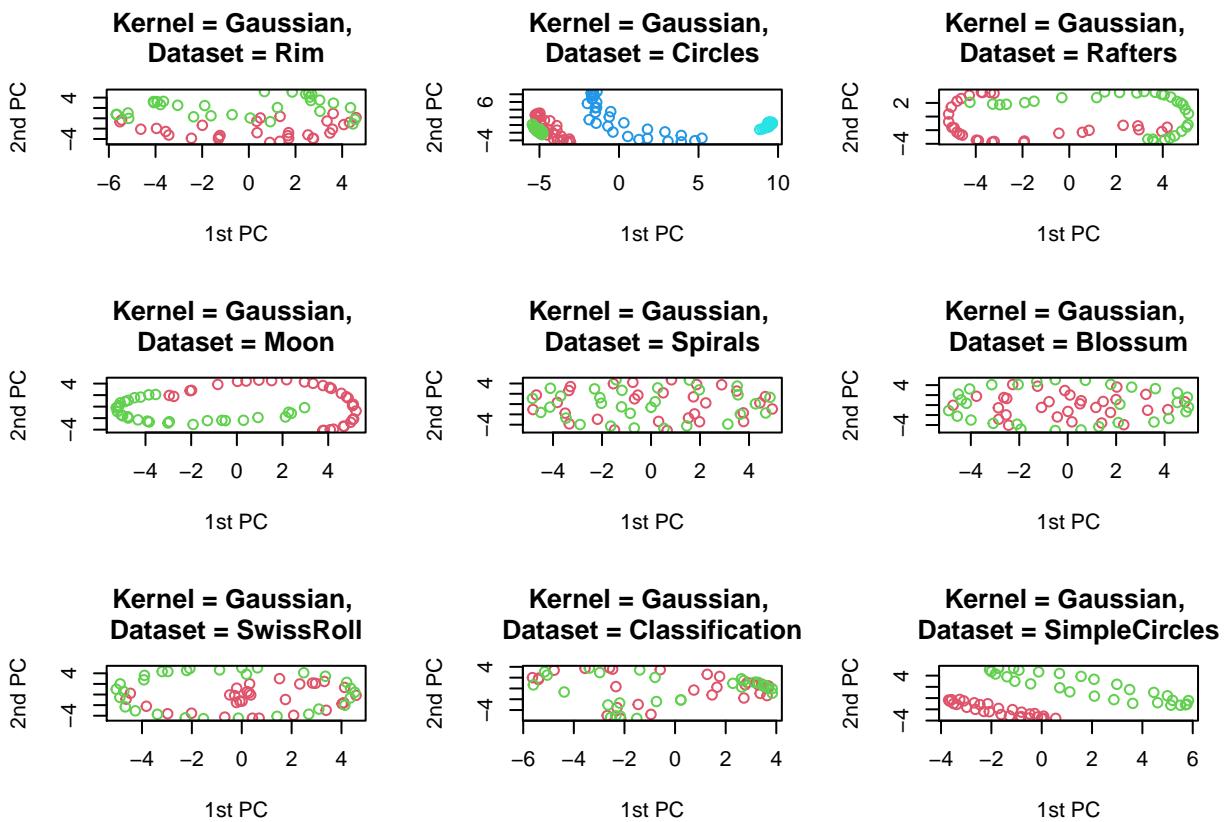
for (i in 1:length(names.df)) {

  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])

  kpc.linear <- kPCA(as.matrix(df[,1:2]), kernel="rbfdot",
                      kpar=list(sigma = sh(df[,1:2])),
                      features=2)

  pcv(kpc.linear)
  eig(kpc.linear)

  plot(rotated(kpc.linear),
       xlab="1st PC", ylab="2nd PC",
       main = paste("Kernel = Gaussian, \nDataset = ", names.df[i], sep = ""),
       # sub = paste("eigen : Comp1 = ", round(eig(kpc.linear)[1], 2), " Comp2 = ", r
       col = 1 + as.numeric(as.factor(df$predictor)))
}
```



Noyau polynomial :

```
par(mfrow = c(3,3))

for (i in 1:length(names.df)) {

  print(i)
  df <- datasets[[i]]
  print(str(df))
  print(names.df[i])

  kpc.linear <- kPCA(as.matrix(df[,1:2]), kernel = "polydot",
                      kpar=list(scale = 1, offset = 0,
                                degree = 2),
                      features=2)

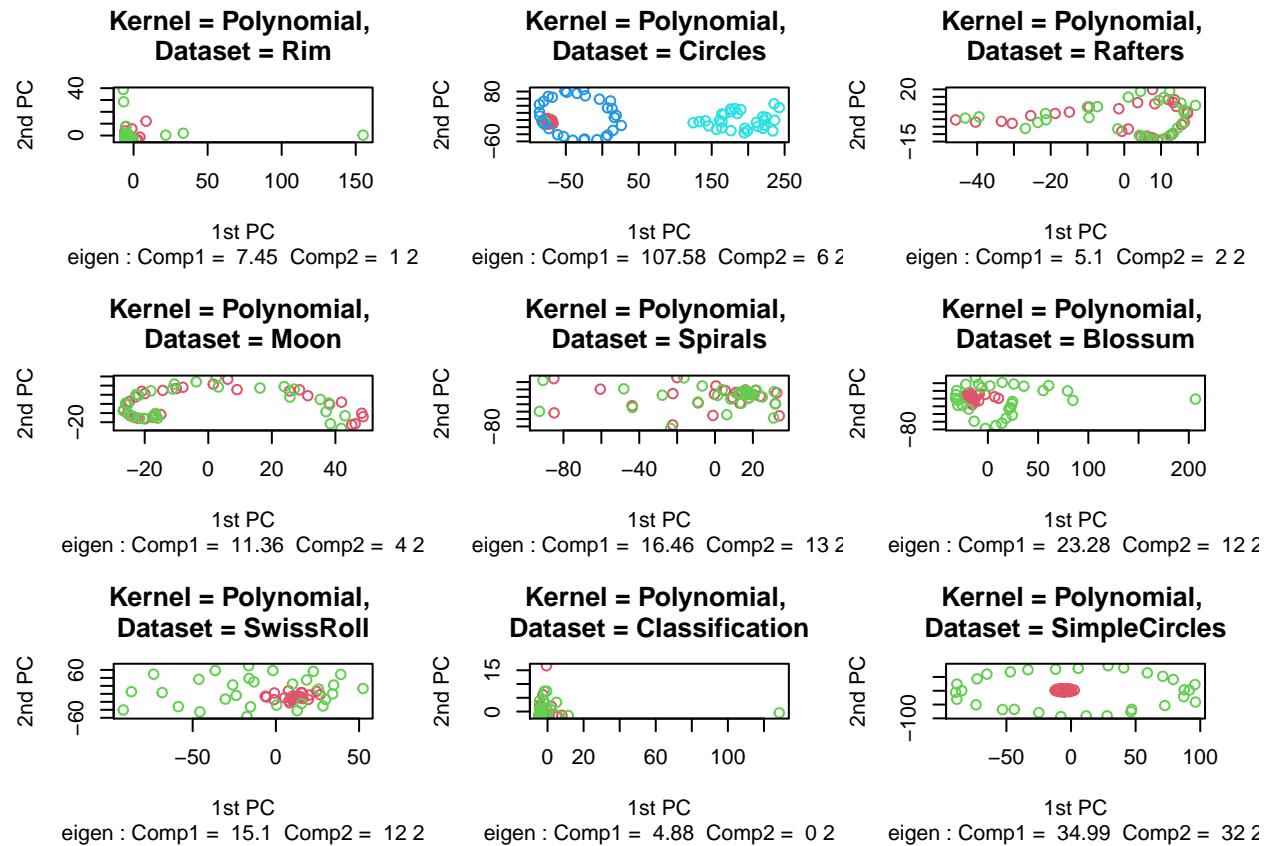
  pcv(kpc.linear)
  eig(kpc.linear)

  plot(rotated(kpc.linear),
       xlab="1st PC", ylab="2nd PC",
       main = paste("Kernel = Polynomial, \nDataset = ", names.df[i], sep = ""),
       sub = paste("eigen : Comp1 = ",
                  round(eig(kpc.linear)[1], 2),
```

```

    " Comp2 = ", round(eig(kpc.linear)[2]), 2),
  col = 1 + as.numeric(as.factor(df$predictor)))
}

```



## Dissimilarity matrix

Noyau linéaire

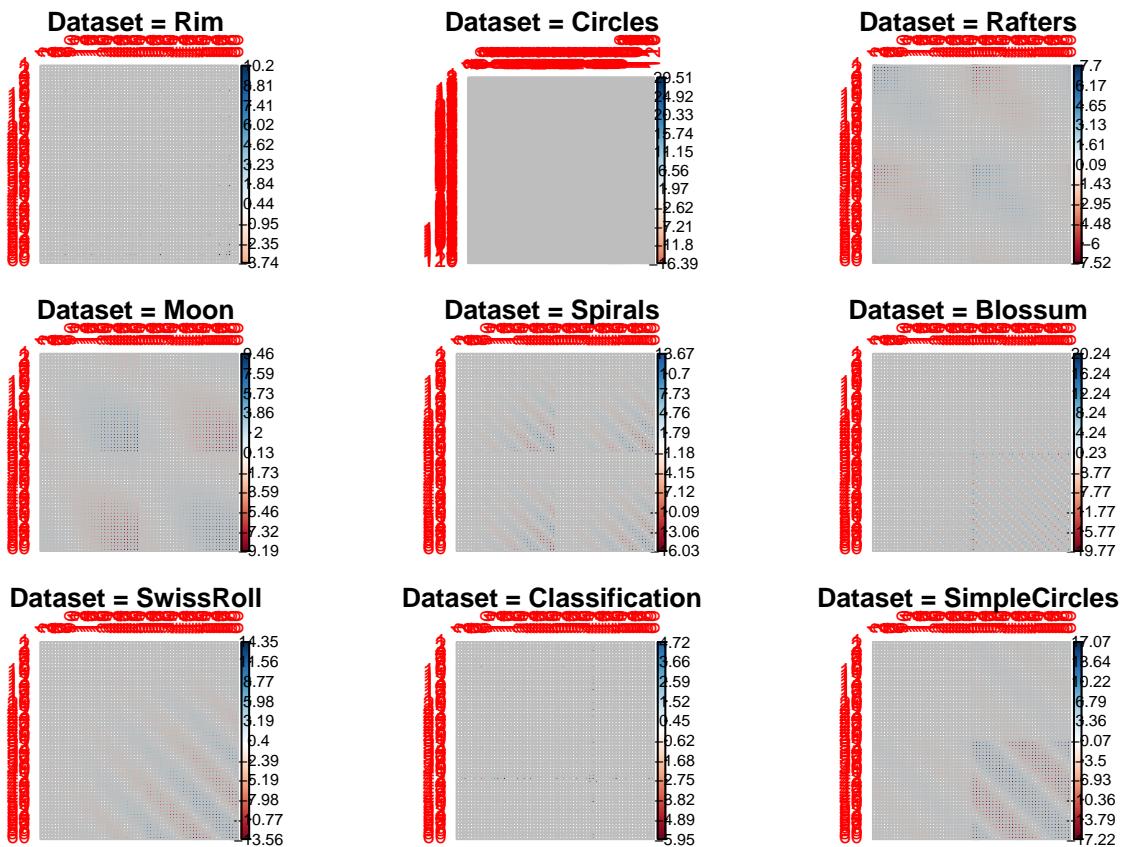
```

par(mfrow = c(3,3))

for (i in 1:length(names.df)) {

  df <- datasets[[i]]
  noyau.lin <- vanilladot()
  dist.lin <- kernelMatrix(noyau.lin, as.matrix(df[,1:2]))
  corrplot(as.matrix(dist.lin), is.corr=FALSE, tl.cex = 1,
           title = paste("Dataset = ",names.df[i],sep = ""),
           method = c("square"),
           mar = c(1,1,1,1))
}

```



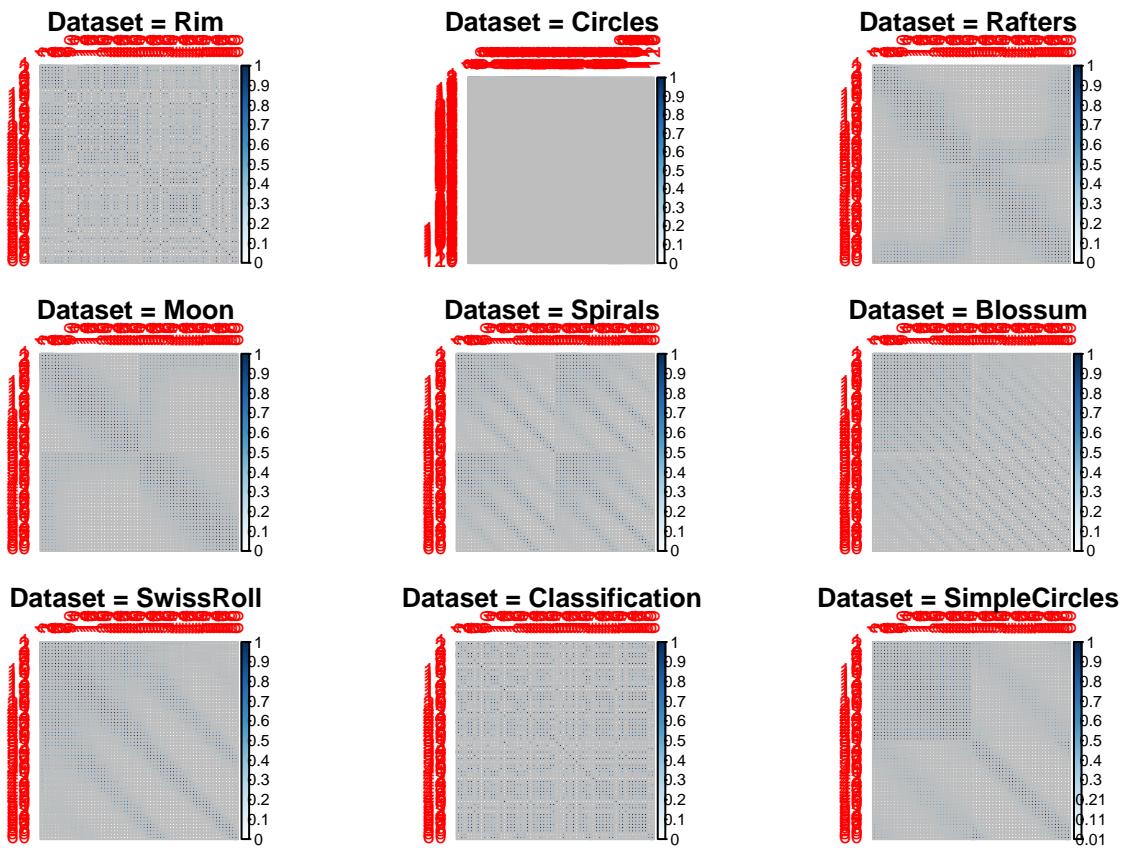
Noyau gaussien (paramètre sigma = sh(df[,1:2]))

```
par(mfrow = c(3,3))

for (i in 1:length(names.df)) {

  df <- datasets[[i]]

  noyau.gaussien <- rbfddot(sigma = sh(df[,1:2]))
  dist.gauss <- kernelMatrix(noyau.gaussien, as.matrix(df[,1:2]))
  corrplot(as.matrix(dist.gauss), is.corr=FALSE, tl.cex = 1,
           title = paste("Dataset = ",names.df[i],sep = ""),
           method = c("square"),
           mar = c(1,1,1,1)
         )
}
```



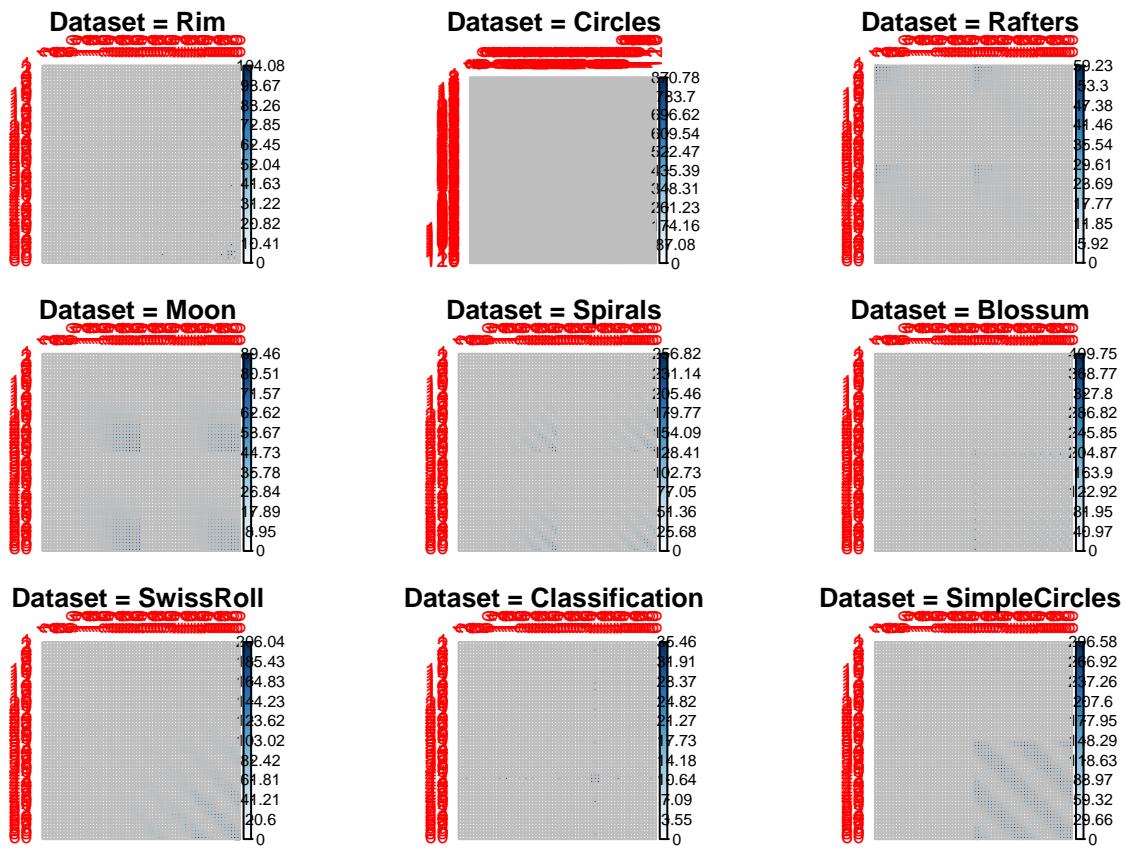
Noyau polynomial (paramètres degree = 2, scale=0.5, offset=5)

```
par(mfrow = c(3,3))

for (i in 1:length(names.df)) {

  df <- datasets[[i]]

  noyau.poly <- polydot(degree = 2, scale = 1, offset = 0)
  dist.poly <- kernelMatrix(noyau.poly, as.matrix(df[,1:2]))
  corrplot(as.matrix(dist.poly), is.corr=FALSE, tl.cex = 1,
           title = paste("Dataset = ",names.df[i],sep = ""),
           method = c("square"),
           mar = c(1,1,1,1)
         )
}
```



## Annexe 2 : Genotyping Dataset

# Genotyping Dataset

Xavier Grand

25/06/2020

## Packages

```
list.of.packages <- c("ggplot2", "kernlab", "corrplot", "FactoMineR", "SKAT", "FactoMineR", "mixKernel")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages() [, "Package"])]
if(length(new.packages)) install.packages(new.packages, dependencies = T)
library(ggplot2)
library(kernlab)

## 
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
## 
##     alpha

library(corrplot)

## corrplot 0.84 loaded

library(SKAT)

## Loading required package: Matrix

## Loading required package: SPAtest

library(FactoMineR)
library(mixKernel)

## Loading required package: mixOmics

## Loading required package: MASS

## Loading required package: lattice
```

```

## 
## Loaded mixOmics 6.12.1
## Thank you for using mixOmics!
## Tutorials: http://mixomics.org
## Bookdown vignette: https://mixomicsteam.github.io/Bookdown
## Questions, issues: Follow the prompts at http://mixomics.org/contact-us
## Cite us: citation('mixOmics')

## Loading required package: reticulate

## 
## mixKernel will soon be included in the mixOmics package
##
## Visit http://www.mixOmics.org for a tutorial to use our method.
## Any bug reports or comments? Notify us at jerome.mariette@inrae.fr or https://bitbucket.org/klecao/p

```

## Utilities :

### Checks whether a matrix is a kernel :

Fonction is.kernel : renvoie T ou F si la matrice testée est un noyau.

```

is.kernel <- function (K, test.pos.semidef = FALSE) {

  if (!is.matrix(K)) {
    K <- as.matrix(K)
  }

  if (test.pos.semidef) {
    eval <- eigen(K, only.values = TRUE, symmetric = TRUE)$values
    tolerance <- 10^(-10)

    if (is.complex( eval )) {
      stop("Kernel matrix has complex eigenvalues")
    }

    if (isSymmetric(K)) {
      return(all(eval[which(eval > tolerance)] > 0))
    } else {
      return(FALSE)
    }
  } else {
    return(isSymmetric(K))
  }
}

```

## Datasets simulation

### Datasets function

```
geno.ds <- function(alphabet, indiv, mk) {  
  matrice.indiv <- matrix(rep(0,(indiv * mk)), ncol=mk, nrow=indiv)  
  rownames(matrice.indiv) <- paste0("I",1:indiv)  
  colnames(matrice.indiv) <- paste0("M",1:mk)  
  
  # vec.bases <- sample(alphabet, size = mk, replace = TRUE) # pour le 1er individu  
  #  
  # for (i in 1:indiv)  
  # {  
  #   matrice.indiv[i,] <- vec.bases  
  # }  
  # construit une matrice avec 30 lignes identiques  
  
  nb.mut <- sample(x=0:2, size = indiv - 1, replace=TRUE)  
  # donne le nombre de mutations (entre 0 et 5) pour 29 individus (le premier restera inchangé)  
  
  for (i in 2:indiv)  
    # on démarre à l'individu 2  
  {  
    if (nb.mut[i-1] > 0){  
      indice.a.changer <- sample(x=c(5, 8),  
                                size = nb.mut[i-1],  
                                prob = c(0.75, 0.25))  
      bases.a.changer <- matrice.indiv[i, indice.a.changer]  
      # ERROR : indice hors limites  
      # c'est un objet de type list  
      nouvelles.bases <- sapply(bases.a.changer,  
                                function(x){sample(setdiff(alphabet, x), 1)})  
      matrice.indiv[i, indice.a.changer] <- nouvelles.bases  
    }  
  }  
  matrice.indiv  
}
```

### Datasets creation

```
nuc <- c(0,1,2)  
indiv = 1000  
mk = 100  
  
matrice.geno <- geno.ds(nuc, indiv, mk)
```

## Calcul des noyaux : From SKAT sources (Kernel.R)

```

K1_Help= function(x,y){
  # Helper function for 2 way interaction kernel
  p = length(x)
  a = x*y
  b = cumsum(a)
  return(sum(a[-1]*b[-p]))
}

lskmTest.GetKernel = function(Z, kernel, weights,n,m){
  if (kernel == "quadratic") {
    K = (Z%*%t(Z)+1)**2
  }
  if (kernel == "IBS.weighted_OLD") {
    #K = matrix(nrow = n, ncol = n)
    if (is.null(weights)) {
      qs = apply(Z, 2, mean)/(2)
      weights = 1/sqrt(qs)
    } else {
      weights <- weights^2
    }
    K1 = matrix(nrow =n, ncol = n)
    for (i in 1:n) {
      K1[i,] = apply(abs(t(Z)-Z[i,])*weights,2, sum)
    }
    K= 1-(K1)/(2*sum(weights))
  }

  if (kernel == "IBS_OLD") {
    K1=matrix(nrow=n,ncol=n)
    for (i in 1:n) {
      K1[i,] = apply(abs(t(Z)-Z[i,]),2, sum)
    }
    K = (2*m-K1)/(2*m)
  }
  if (kernel == "2wayIX_OLD") {
    K = 1+Z%*%t(Z)
    N1= matrix(nrow = n, ncol = n)
    for (i in 1:n){
      for (j in i:n){
        N1[j,i] = N1[i,j] = K1_Help(Z[i,], Z[j,])
      }
    }
    K = K+N1
  }
  return(K)
}

```

## essai de la fonction kernel de SKAT

```
# Kernel = quadratic

res <- lskmTest.GetKernel(matrice.geno, "quadratic", weights = NULL, indiv, mk)
print(is.kernel(res))

# Kernel = IBS.weighted_OLD

res <- lskmTest.GetKernel(matrice.geno, "IBS.weighted_OLD", weights = NULL, indiv, mk)
print(is.kernel(res))

# Kernel = "IBS_OLD"

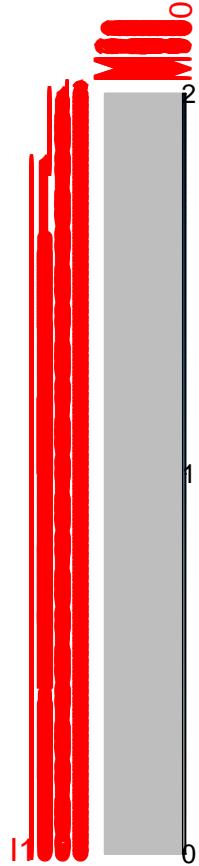
res <- lskmTest.GetKernel(matrice.geno, "IBS_OLD", weights = NULL, indiv, mk)
print(is.kernel(res))

# Kernel = "2wayIX_OLD"

res <- lskmTest.GetKernel(matrice.geno, "2wayIX_OLD", weights = NULL, indiv, mk)
print(is.kernel(res))
```

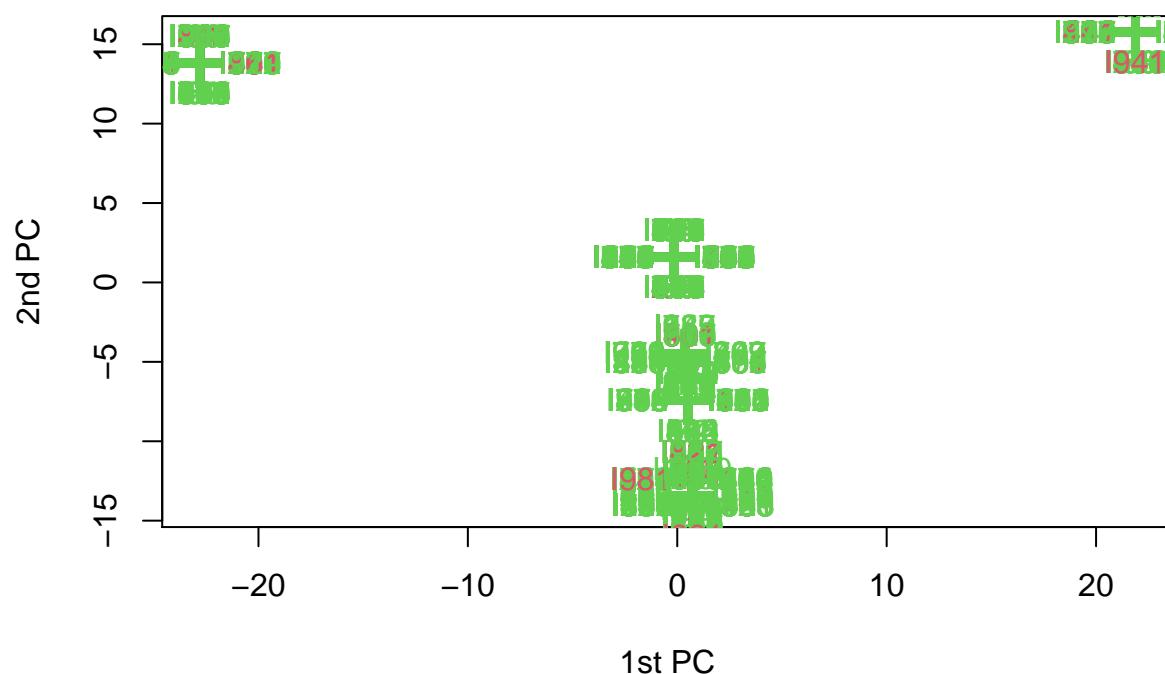
## Visualisation des résultats : PCA

```
# Visualisation de la matrice geno :
corrplot(matrice.geno, is.corr = F, cl.length = 3)
```



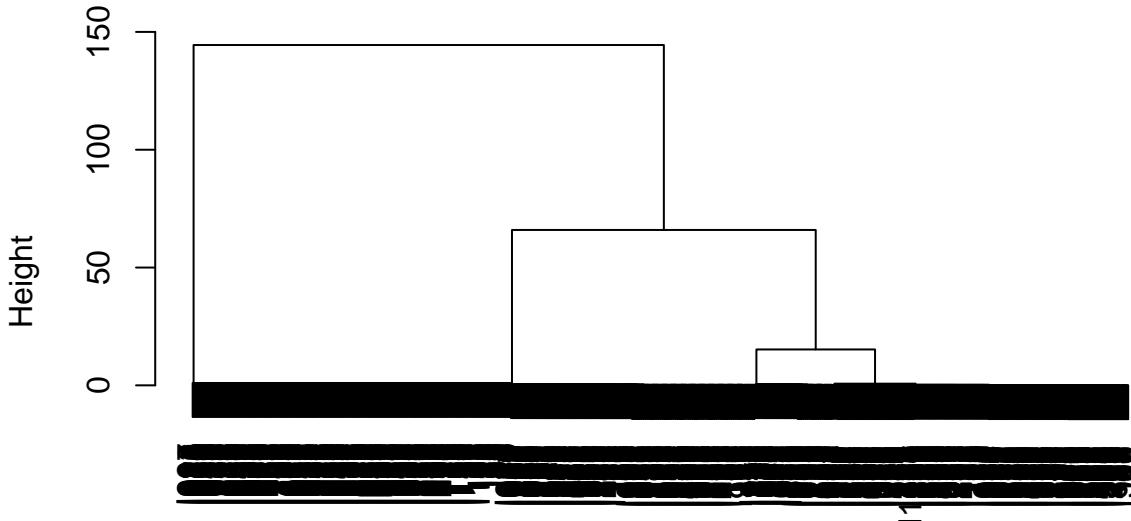
```
# KPCA :  
K.pca <- kPCA(res, features = 3)  
  
# Plot la KPCA  
mon.jitter <- rnorm(40, sd = 0.2)  
  
plot(rotated(K.pca)[,2:3],  
      xlab="1st PC", ylab="2nd PC",  
      main = "Kernel = IBS",  
      pch = "+", cex = 3,  
      col = c(2, rep(3, 19)))  
text(rotated(K.pca)[,2:3], label = rownames(matrice.geno),  
     col = c(2, rep(3, 19)), pos = sample(1:4, size = indiv, replace = T))
```

## **Kernel = IBS**



```
# kernel clustering : "Bricolé"
hc <- hclust(as.dist(1/res), method = "ward.D")
plot(hc)
```

## Cluster Dendrogram



```
as.dist(1/res)  
hclust (*, "ward.D")
```

```
toy.geno <- matrix(c(0,0,0,0,1,0,0,2,1), ncol = 3)  
  
quad.kernel.toy <- lskmTest.GetKernel(toy.geno, "quadratic", weights = NULL, 3, 3)  
  
IBS.kernel.toy <- lskmTest.GetKernel(toy.geno, "IBS_OLD", weights = NULL, 3, 3)
```

### Calcul des noyaux : Recodage

J'ai recodé les fonction noyau du papier sans prendre en compte le poids assigné à chaque marqueur. Il s'agit donc des fonctions noyau dans leurs états les plus simples.

```
SKAT.GetKernel = function(Z, kernel, n, m){  
  
  # Noyau linéaire : Modèle linéaire  
  if (kernel == "linear") {  
    K <- (Z%*%t(Z))  
  }  
  
  # Noyau quadratique : Effet majeur  
  if (kernel == "quadratic") {  
    K <- (Z%*%t(Z)+1)**2  
  }  
  
  # Noyau effet additif :
```

```

if (kernel == "additif") {
  K <- matrix(nrow=n, ncol=n)
  for (i in 1:n) {
    K[i,] <- apply(2-abs(t(Z)-Z[i,]), 2, sum)
  }
  rownames(K) <- rownames(Z)
  colnames(K) <- rownames(Z)
}

# alternative code add :
if (kernel == "alternatif") {
  K <- matrix(nrow=n, ncol=n)
  for (i in 1:n) {
    for (j in 1:n) {
      K[i,j] <- sum(2-abs(Z[i,]-Z[j,]))
    }
  }
  rownames(K) <- rownames(Z)
  colnames(K) <- rownames(Z)
}

return(K)
}

```

test des kernel recodés sur la toy matrice...

```

toy.geno <- matrix(c(0,0,0,0,1,2,0,0,1,1,1,1), ncol = 3)
rownames(toy.geno) <- paste0("I", 1:dim(toy.geno)[1])
colnames(toy.geno) <- paste0("M", 1:dim(toy.geno)[2])
print(toy.geno)

linear.kernel.toy <- SKAT.GetKernel(toy.geno, "linear", dim(toy.geno)[1], dim(toy.geno)[2])
print(linear.kernel.toy)
print(is.kernel(linear.kernel.toy))

quad.kernel.toy <- SKAT.GetKernel(toy.geno, "quadratic", dim(toy.geno)[1], dim(toy.geno)[2])
print(quad.kernel.toy)
print(is.kernel(quad.kernel.toy))

add.kernel.toy <- SKAT.GetKernel(toy.geno, "additif", dim(toy.geno)[1], dim(toy.geno)[2])
print(add.kernel.toy)
print(is.kernel(add.kernel.toy))

alt.kernel.toy <- SKAT.GetKernel(toy.geno, "alternatif", dim(toy.geno)[1], dim(toy.geno)[2])
print(alt.kernel.toy)
print(is.kernel(alt.kernel.toy))

par(mfrow = c(2,2))

# Visualisation de la matrice geno :
corrplot(toy.geno, is.corr= F, cl.length = 3, tl.cex = 0.5)

```

```

# KPCA :
K.pca.linear <- kpca(linear.kernel.toy, features = 2)
K.pca.quad <- kpca(quad.kernel.toy, features = 2)
K.pca.add <- kpca(add.kernel.toy, features = 2)

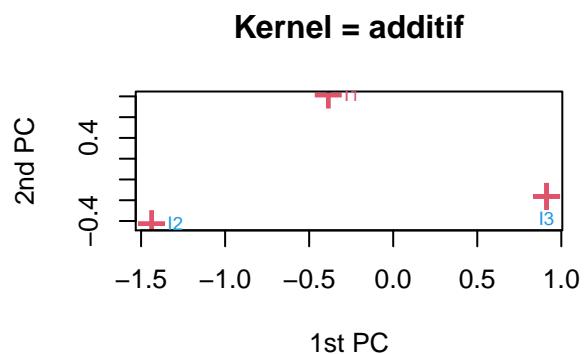
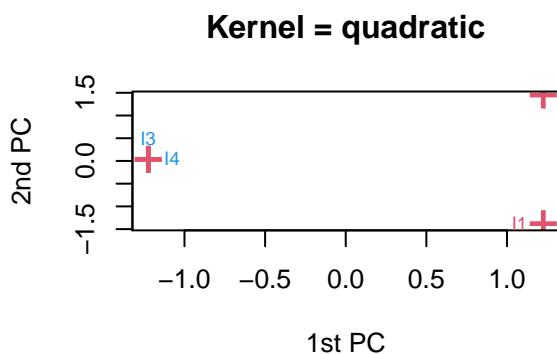
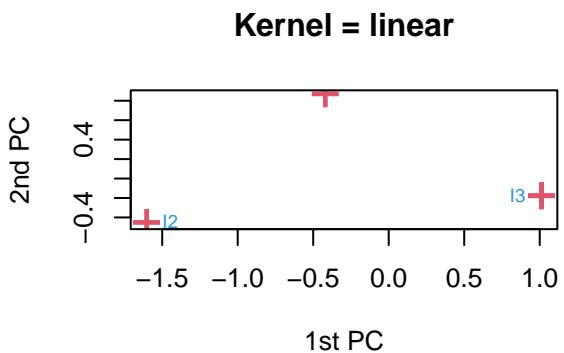
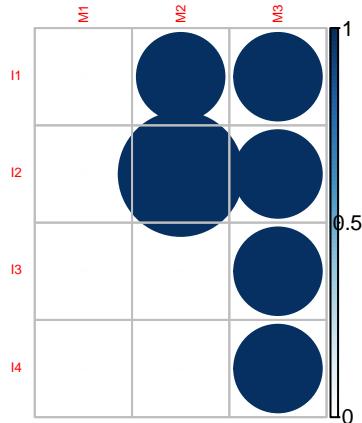
# Plot la KPCA

plot(rotated(K.pca.linear),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = linear",
      pch = "+", cex = 2,
      col = 2)
text(rotated(K.pca.linear), label = rownames(toy.geno), col = c(2, rep(4, dim(toy.geno)[1]-1)), cex = 0.75)

plot(rotated(K.pca.quad),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = quadratic",
      pch = "+", cex = 2,
      col = 2)
text(rotated(K.pca.quad), label = rownames(toy.geno), col = c(2, rep(4, dim(toy.geno)[1]-1)), cex = 0.75)

plot(rotated(K.pca.add),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = additif",
      pch = "+", cex = 2,
      col = 2)
text(rotated(K.pca.add), label = rownames(toy.geno), col = c(2, rep(4, dim(toy.geno)[1]-1)), cex = 0.75)

```



```
# kernel clustering : "Bricolé"
# hc <- hclust(as.dist(linear.kernel), method = "single")
# plot(hc)
# hc <- hclust(as.dist(quad.kernel), method = "single")
# plot(hc)
# hc <- hclust(as.dist(add.kernel), method = "single")
# plot(hc)
# hc <- hclust(as.dist(alt.kernel), method = "single")
# plot(hc)
```

test des kernel recodés sur la matrice.geno simulée :

```
linear.kernel <- SKAT.GetKernel(matrice.geno, "linear", indiv, mk)
# print(linear.kernel)

quad.kernel <- SKAT.GetKernel(matrice.geno, "quadratic", indiv, mk)
# print(quad.kernel)

add.kernel <- SKAT.GetKernel(matrice.geno, "additif", indiv, mk)
# print(add.kernel)

alt.kernel <- SKAT.GetKernel(matrice.geno, "alternatif", indiv, mk)
# print(alt.kernel)
```

## Visualisation des résultats : PCA

```
par(mfrow = c(2,2))

# Visualisation de la matrice geno :
corrplot(matrice.geno, is.corr= F, cl.length = 3, method = "square", tl.cex = 0.5)

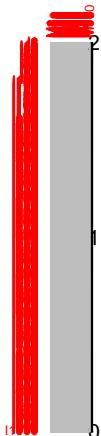
# KPCA :
K.pca.linear <- kpca(linear.kernel, features = 2)
K.pca.quad <- kpca(quadratic.kernel, features = 2)
K.pca.add <- kpca(additive.kernel, features = 2)

# Plot la KPCA

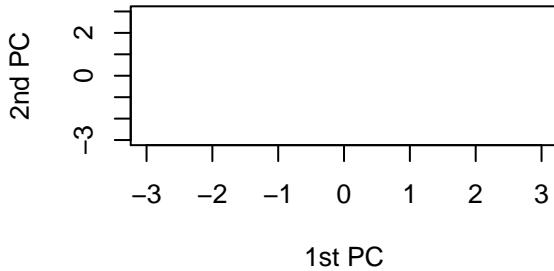
plot(rotated(K.pca.linear),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = linear",
      pch = "+", cex = 2,
      col = 2,
      xlim = c(-3, 3), ylim = c(-3, 3))
text(rotated(K.pca.linear), label = rownames(matrice.geno), col = c(2, rep(4, indiv-1)), cex = 0.75, pos = 1)

plot(rotated(K.pca.quad),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = quadratic",
      pch = "+", cex = 2,
      col = 2,
      xlim = c(-3, 3), ylim = c(-3, 3))
text(rotated(K.pca.quad), label = rownames(matrice.geno), col = c(2, rep(4, indiv-1)), cex = 0.75, pos = 1)

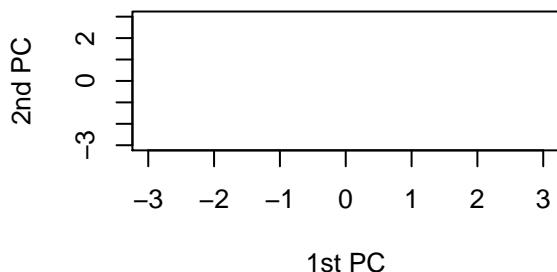
plot(rotated(K.pca.add),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = additif",
      pch = "+", cex = 2,
      col = 2,
      xlim = c(-3, 3), ylim = c(-3, 3))
text(rotated(K.pca.add), label = rownames(matrice.geno), col = c(2, rep(4, indiv-1)), cex = 0.75, pos = 1)
```



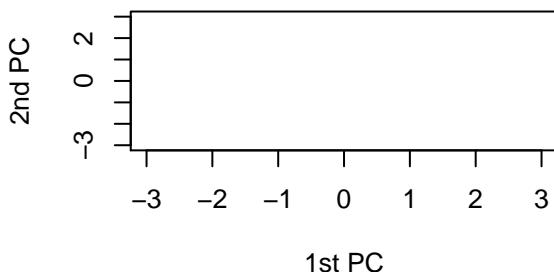
**Kernel = linear**



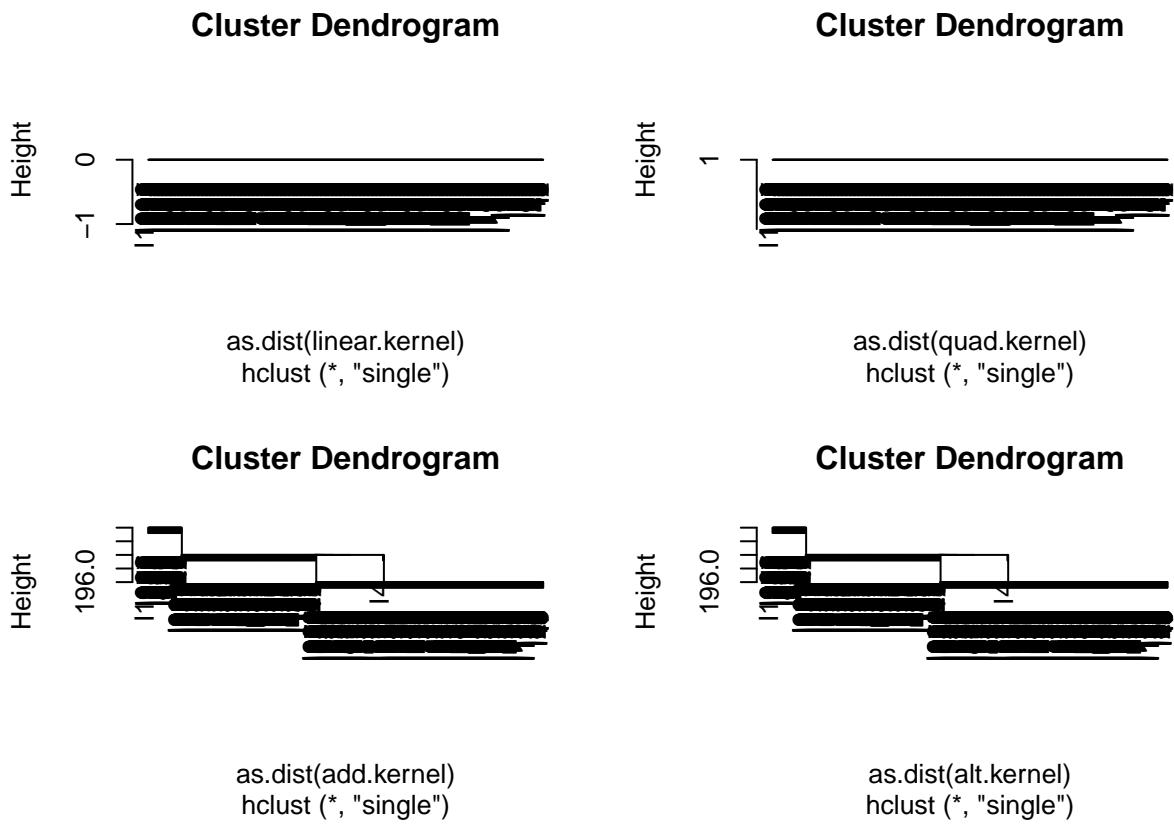
**Kernel = quadratic**



**Kernel = additif**



```
# kernel clustering : "Bricolé"
hc <- hclust(as.dist(linear.kernel), method = "single")
plot(hc)
hc <- hclust(as.dist(quad.kernel), method = "single")
plot(hc)
hc <- hclust(as.dist(add.kernel), method = "single")
plot(hc)
hc <- hclust(as.dist(alt.kernel), method = "single")
plot(hc)
```



## Jeu de données réel *Arabidopsis thaliana* from Duruflé H.

Attention, pour pouvoir charger le fichier sans risquer de problème par la suite, il m'a fallu modifier l'entête de la deuxième colonne qui contenait un “°” en “numero”.

```
araData <- read.csv2(file = "/home/xiav/Bureau/Kernel/kernel-cca/Sources/chimere.snp.csv")
uniqNamesPop <- paste0(araData[,1], araData[,2])
araData.clean <- as.matrix(data.frame(araData[,3:dim(araData)[2]], row.names = uniqNamesPop))

# print(araData.clean)

# res <- apply(araData.clean, 1, sum)
```

test des kernel recodés sur les data arabidopsis :

```
ara.linear.kernel <- SKAT.GetKernel(araData.clean, "linear", dim(araData.clean)[1], dim(araData.clean)[2])

ara.quad.kernel <- SKAT.GetKernel(araData.clean, "quadratic", dim(araData.clean)[1], dim(araData.clean)[2])

ara.add.kernel <- SKAT.GetKernel(araData.clean, "additif", dim(araData.clean)[1], dim(araData.clean)[2])
```

## Visualisation des résultats : PCA

```
par(mfrow = c(1,3))

# Visualisation de la matrice geno : Trop de points... Illisible
# corrplot(araData.clean, is.corr= F, cl.length = 3, method = "square", tl.cex = 0.25)

# KPCA :
K.pca.linear <- kpca(ara.linear.kernel, features = 2)
K.pca.quad <- kpca(ara.quad.kernel, features = 2)
K.pca.add <- kpca(ara.add.kernel, features = 2)

# Plot la KPCA
mon.jitter <- rnorm(24 * 2, sd = 0.2)

plot(rotated(K.pca.linear),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = linear",
      pch = "+", cex = 2,
      xlim = c(-1, 4), ylim = c(-1, 4))
text(rotated(K.pca.linear) + mon.jitter, label = rownames(araData.clean), col = 2, cex = 0.75)

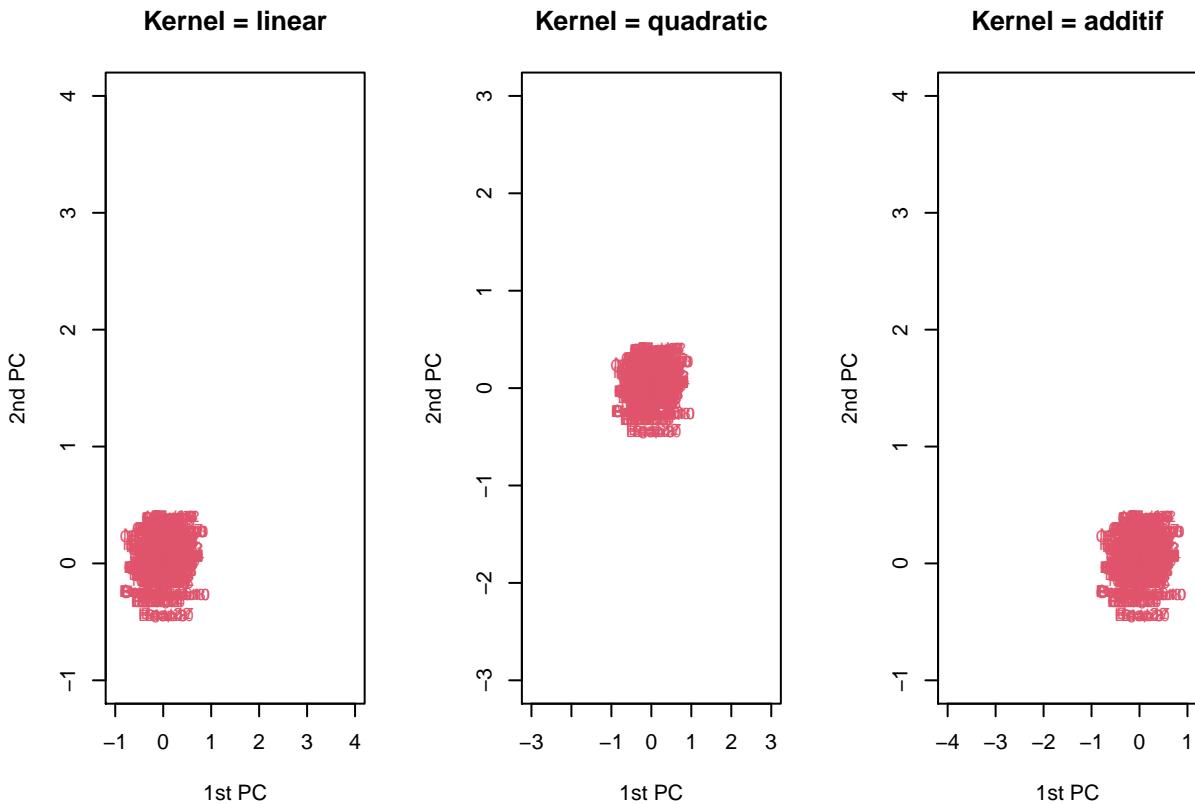
## Warning in rotated(K.pca.linear) + mon.jitter: la taille d'un objet plus long
## n'est pas multiple de la taille d'un objet plus court

plot(rotated(K.pca.quad),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = quadratic",
      pch = "+", cex = 2,
      xlim = c(-3, 3), ylim = c(-3, 3))
text(rotated(K.pca.quad) + mon.jitter, label = rownames(araData.clean), col = 2, cex = 0.75)

## Warning in rotated(K.pca.quad) + mon.jitter: la taille d'un objet plus long
## n'est pas multiple de la taille d'un objet plus court

plot(rotated(K.pca.add),
      xlab="1st PC", ylab="2nd PC",
      main = "Kernel = additif",
      pch = "+", cex = 2,
      xlim = c(-4, 1), ylim = c(-1, 4))
text(rotated(K.pca.add) + mon.jitter, label = rownames(araData.clean), col = 2, cex = 0.75)

## Warning in rotated(K.pca.add) + mon.jitter: la taille d'un objet plus long n'est
## pas multiple de la taille d'un objet plus court
```

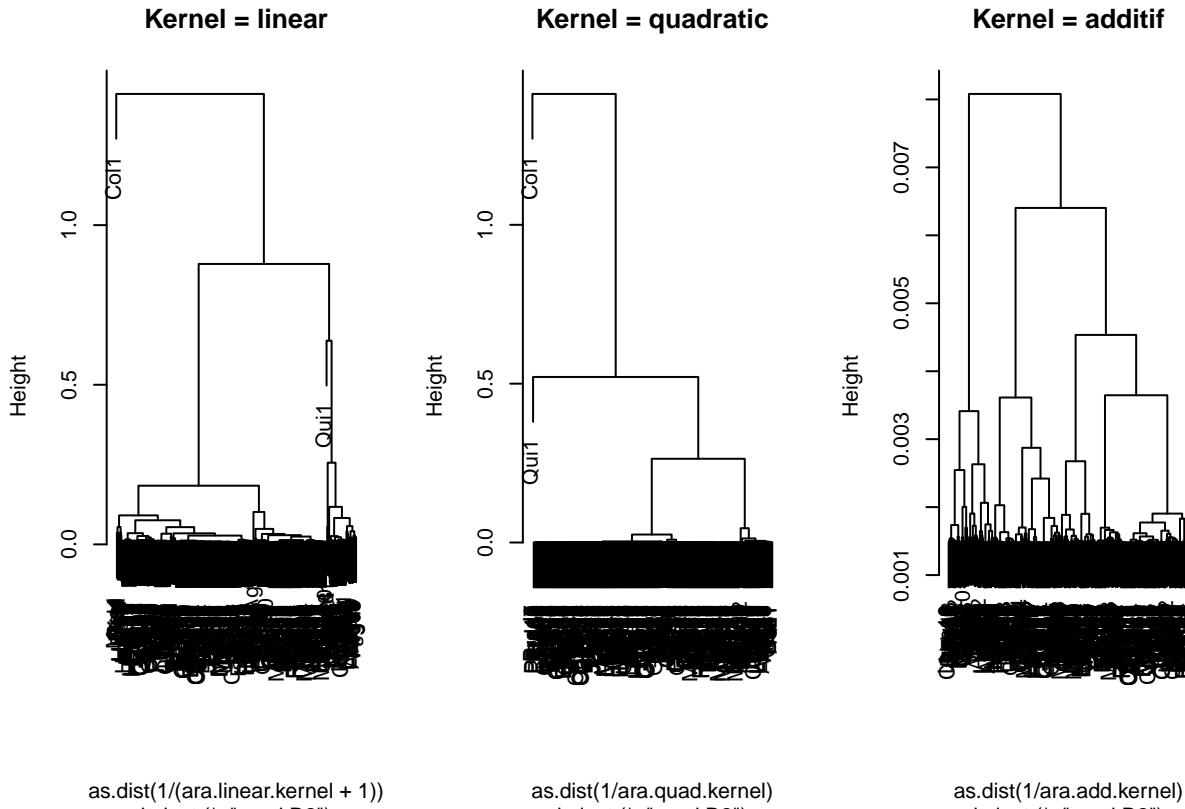


```

# ggplot(as.data.frame(rotated(K.pca.linear))) +
#   aes(V1, V2, label = rownames(araData.clean)) +
#   geom_point(color = "red") +
#   geom_text(position = position_jitter(width=0.2, height=0.2))
#
# ggplot(as.data.frame(rotated(K.pca.quad))) +
#   aes(V1, V2, label = rownames(araData.clean)) +
#   geom_point(color = "red") +
#   geom_text(position = position_jitter(width=0.2, height=0.2))
#
# ggplot(as.data.frame(rotated(K.pca.add))) +
#   aes(V1, V2, label = rownames(araData.clean)) +
#   geom_point(color = "red") +
#   geom_text(position = position_jitter(width=0.2, height=0.2))

# kernel clustering : "Bricolé"
hc <- hclust(as.dist(1/(ara.linear.kernel+1)), method = "ward.D2")
plot(hc, cex = 1, main = "Kernel = linear")
hc <- hclust(as.dist(1/ara.quad.kernel), method = "ward.D2")
plot(hc, cex = 1, main = "Kernel = quadratic")
hc <- hclust(as.dist(1/ara.add.kernel), method = "ward.D2")
plot(hc, cex = 1, main = "Kernel = additif")

```



Comparaison des fonctions additif et alternatif : Benchmarking :

#Sur les données *Arabidopsis* :

```
add.bench <- system.time(SKAT.GetKernel(araData.clean, "additif", dim(araData.clean)[1], dim(araData.cl
alt.bench <- system.time(SKAT.GetKernel(araData.clean, "alternatif", dim(araData.clean)[1], dim(araData
```

#sur des données simulées :

```
add.bench <- system.time(SKAT.GetKernel(matrice.geno, "additif", dim(matrice.geno)[1], dim(matrice.geno
alt.bench <- system.time(SKAT.GetKernel(matrice.geno, "alternatif", dim(matrice.geno)[1], dim(matrice.g
```

```
df2 <- c(add.bench[1:3], alt.bench[1:3])
df2 <- as.data.frame(df2)
df3 <- c("utilisateur", "système", "écoulé", "utilisateur", "système", "écoulé")
df4 <- cbind(df3, df2)
df5 <- c("Kernel additif", "Kernel additif", "Kernel additif", "Kernel alternatif", "Kernel alternatif")
df6 <- cbind(df5, df4)
colnames(df6) <- c("kernel", "ressource", "temps")

ggplot(data=df6, aes(x=ressource, y=temps, fill=kernel)) +
  geom_bar(stat="identity", position=position_dodge())+
  geom_text(aes(label=round(temps, 2)), vjust=1.6, color="black",
```

```
position = position_dodge(0.9), size=3.5)+  
labs(x="Ressources", y = "Temps (s)") +  
ggtitle("Benchmark pour noyau additif", subtitle = "Plus bas est meilleur")
```

