Méthodes de regroupement et de découverte de motifs pour les données fonctionnelles

Mémoire

Pathe Conte

Maîtrise en biostatistique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Résumé

Ce mémoire propose des outils d'analyse des données fonctionnelles (ADF). Cette analyse s'applique aux données dont la structure peut être considérée comme une fonction. Dans ce travail, nous avons fait, avec des exemples pratiques sous R, un parcours sur une partie des méthodes existantes.

D'abord, nous avons commencé par une petite introducion générale sur l'ADF où nous avons parlé non seulement des différentes bases classiques (base de Fourier, base polynomiale, base spline) qui permettent de représenter les données fonctionelles en les rapprochant avec une combinaison linéaire de fonctions de base, mais aussi du lissage qui permet de convertir des données observées discrètes en courbes coutinues (par les moindres carrés ou par la pénalisation). Au vu du problème de variabilité en phase des données fonctionnelles, nous avons parlé des différentes méthodes d'alignement des courbes (par points de repère et alignement continu).

Ensuite, nous avons étudié le clustering de données fonctionnelles. Il s'agit du processus qui permet de regrouper en cluster des fonctions similaires. Plusieurs méthodes de clustering exitent, mais nous nous sommes focalisés principalement sur les méthodes basées sur la distance, plus particulièrement sur la méthode K-moyennes. Egalement, le K-moyennes combinant simultanément l'alignement de manière globale et le clustering des courbes a été présentée. Et à travers des exemples pratiques, nous avons pu trouver la structure sous-jacente de données (identification de certaines tendances des données).

Enfin, nous avons terminé par présenter le K-moyennes probabiliste avec alignement local qui prend en compte la forme locale des courbes, ce qui n'est pas le cas avec les méthodes susmentionnées qui traitent les courbes dans leur intégralité. Cette méthode, développée par Cremona et Chiaromonte, permet le regroupement local des courbes et la découverte de motifs fonctionnels ("formes" typiques) dans ces courbes. Sous R, cette méthode est fournit par le package probKMA.FMD (ProbKMA-based Functional Motif Discovery) qui est en construction. Des exemples pratiques ont été montrés afin de bien tester les codes et les fonctions du package.

Abstract

This thesis proposes tools for functional data analysis (FDA). FDA is used on data whose structure can be considered as a function. In this work, we have looked at some of the existing methods using practical examples with R.

First, we started with a short general introduction on FDA in which we discussed the different classical bases (Fourier basis, polynomial basis, Spline basis) which allow to represent functional data through a linear combination of basis functions. We also considered smoothing, which allows the conversion of discrete observed data into continuous curves (by least squares or by rougghness penalty). Regarding the problem of phase variability of functional data, we discussed different methods of curve alignment (by landmarks and continuous alignment).

Second, we studied the clustering of functional data, which permits to group similar functions into clusters. Several clustering methods exist, but we mainly focused on distance-based methods, particularly on the K-means. Moreover, we considered a K-means combining simultaneously global alignment and clustering of curves. Through practical examples, we were able to find the underlying structure of the data (identification of certain patterns in the data).

Finally, we presented the probabilistic K-means with local alignment which takes into account the local shape of the curves,as opposed to the previously mentioned methods which treat the curves in their entirety. This method, developed by Cremona and Chiaromonte, allows the local clustering of curves and the discovery of functional motifs ("shapes") in these curves. In R, this method is provided by the probKMA.FMD (ProbKMA-based Functional Motif Discovery) package which is under construction. Practical examples have been shown to test the codes and functions of the package.

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Mes premiers remerciements vont à l'endroit de mes directeurs de recherche, Monsieur M'Hamed Lajmi Lakhal Chaieb et Madame Marzia Angela Cremona. Je me réjouis d'avoir fait ma maîtrise sous votre direction. La réalisation de ce travail aurait pu être difficile sans un soutien constant de votre part. Votre encadrement a permis de faire avancer ma recherche par le biais de nos rencontres hebdomadaires au cours desquelles les échanges ont été très fructueux. Je vous remercie pour votre disponibilité et le suivi ponctuel à mon égard.

Mes remerciements vont également à l'endroit de tout le personnel et étudiants du département de mathématiques et statistique. Je témoigne aussi ma reconnaissance à ma maman, à ma très chère épouse et à tous mes compatriotes sénégalais vivant au Canada. Votre accompagnement depuis mon premier jour au Canada m'a été d'une grande utilité.

Merci à tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce projet.

Introduction

La révolution numérique, cause de la croissante inflation de la masse de données à traiter, est responsable de l'amélioration des techniques d'exploitation et d'analyse de données. Afin de demeurer toujours compétitives dans l'économie mondiale actuelle, les entreprises doivent tirer parti de tous les outils d'analyse disponibles. L'analyse de données fonctionnelles (ADF) est une technique très utilisée par les statisticiens afin de faciliter la prise de décision à l'ère numérique.

Généralement en statistique, les observations sont des réalisations de variables aléatoires réelles ou vectorielles, mais grâce aux progrès récents en matière de stockage et de traitement des donnée, nous rentrons dans le champ de la "très grande dimension". Les observations sont des fonctions aléatoires, des courbes continues ou encore des données de dimension infinie. Ces types de données sont souvent produits en biologie et en climatologie, par exemples. L'ADF nous permet de résumer via une modélisation fonctionnelle simple des données réelles de dimension infinie.

En ADF, l'on se repose sur l'hypothèse que les données à traiter possèdent une structure sous-jacente qui doit êre prise en compte afin d'étendre plus efficacement les techniques d'analyse de données traditionnelles. Dans ce mémoire, nous allons passer en revue l'ensemble des méthodes d'analyse fonctionnelle avec des exemples pratiques sous le logiciel libre R. Nous allons effectuer l'étude de concepts de base dans l'analyse des données fonctionnelles (lissage et alignement des courbes), du clustering global dans le cadre fonctionnel et d'une nouvelle méthode de clustering local et de recherche de motifs fonctionnel appelée "ProbKMA" (Cremona et Chiaromonte 2020). L'objectif principal est la mise en œuvre un package R pour cette nouvelle méthode, à partir du code R existant.

Ce document débutera par une introduction à l'analyse de données fonctionnelles au chapitre 1 où nous présenterons les concepts de base liés à l'ADF. Nous donnerons quelques exemples d'utilisation de bases classiques avec le package `fda` de R; étudierons avec des exemples pratiques les différentes méthodes de transformations des observations en données fonctionnelles appelées lissage (par moindres carrés et par pénalisation). Nous terminerons ce premier chapitre par l'étude de l'alignement des courbes à l'aide des points de repère (exemple pratique via la fonction `landmarkreg()`) et à l'aide d'un critère continu (exemple pratique via la fonction `register.fd()`).

Au chapitre 2, nous parlerons du clustering des données fonctionnelles et plus précisément, nous décrivons l'algorithme K-means (pour données multivariées et pour données fonctionnelles). Le choix du nombre adéquat de clusters et l'évaluation des clusters obtenus seront basés sur l'indice de silhouette. Sous R, la fonction `K-means()` permettra d'effectuer nos exemples pratiques.

Le chapitre 3 sera consacré à la méthode du "K-means alignment" qui permet d'effectuer simultanément l'alignement et le K-means des courbes de manière globale. Nous parlerons dans ce chapitre de l'alignement sur les K modèles, de l'algorithme "K-means alignment" et du package R `fdakma` via un exemple pratique.

Enfin, le chapitre 4 nous dévoilera la nouvelle méthode K-means probabiliste avec alignement local des courbes. Nous verrons le problème d'optimisation posé par la méthode; sa résolution; l'algorithme et l'évaluation de ProbKMA; et la découverte de motifs fonctionnels. Nous terminerons par la description des différentes fonctions du package ProbKMA, ainsi que leurs entrées et sorties et par des exemples pratiques via les données "Berkley Growth" et "Italian Covid-19".

Chapitre I

Introduction à l'analyse des données fonctionnelles

I. Introduction à l'analyse des données fonctionnelles

Le monde actuel est marqué par la multiplication des systèmes d'information, ce qui a pour conséquence directe, une croissance importante de la masse de données à traiter. Celle-ci, dévoilant les limites des techniques traditionnelles d'exploitation et d'analyse des données, nécessite une mise en oeuvre de nouvelles méthodes de traitement adaptées à cette forte hausse de données.

L'Analyse des Données Fonctionnelles (ADF) est une démarche statistique qui apporte d'importants éléments de réponses au problème sus-mentionné. L'ADF s'applique aux données dont la structure peut être naturellement considérée comme une fonction ou une courbe lisse (Ramsay and Silverman 2005). En d'autres termes, les variables observées sont à valeurs fonctionnelles et non réelles. Comme exemples basiques de données fonctionnelles, nous pouvons en citer:

- l'étude de la croissance physique: Considérons une étude de la croissance physique d'enfant (mesure de la taille par année). Chaque enregistrement donne des valeurs discrètes, mais ces valeurs reflètent une variation en taille lisse qui pourrait être évaluée. chaque enfant est donc représenté par une fonction qui, à l'âge, associe la taille (cf Ramsay et Silverman).
- les données en climatologie: Considérons une étude du climat dans un pays donné (variations de la température mensuelle moyenne au cours de l'année en différents lieux du pays). Chaque station météorologique ("individu") est décrit par une fonction qui associe la température mensuelle moyenne observée (cf Ramsay et Silverman).

De manière générale, l'ADF apporte des réponses aux problèmes telsque la visualisation simple des données, la quantité volumineuse des données (trop de temps de calcul), la structure des données (variabilité temporelle ou spatiale) et la redondance dans les observations (des observations fortement corrélées). Elle permet de résumer les données en remplaçant les observations réelles par une modélisation fonctionnelle simple. Selon Ramsay et Silverman, il est très classique d'approcher les fonctions qui décrivent chaque individu par une représentation graphique (projection sur une base de splines par exemple). Ainsi, non seulement le fait de pouvoir travailler sur les coordonnées du projeté à la place des données brutes réduit le volume de données, mais aussi, la possibilité d'utiliser une représentation périodique sous forme de séries de Fourier si les données sont périodiques (prise en compte de la structure).

1 Définitions

Une variable aléatoire, X, à valeurs dans un espace de dimension infinie, est dite variable aléatoire fonctionnelle. Une observation d'une variable fonctionnelle est appelée donnée fonctionnelle.

$$X = \{X_t : \ t \in T\}$$

Où T est un ensemble continu.

Si $T \in \mathbb{R}$, alors les données sont des courbes et on parle alors de l'analyse statistique de courbes ou, de manière plus pratique, l'analyse de données fonctionnelles.

En pratique, une donnée fonctionnelle est observée dans un nombre fini d'instants. En d'autres termes, les valeurs des données fonctionnelles ne sont jamais connues à tous les points $t \in T$; elles ne sont disponibles qu'à un certain nombre de points spécifiques $\{X_{t_0},\ X_{t_1},\ X_{t_2},\ ....X_{t_k}\}$.

2 Méthodes d'ADF

Dans toutes les méthodes de l'ADF, on est appelé à comparer des fonctions entre elles. Soit on compare une fonction observée (une donnée fonctionnelle) avec une fonction contrôlée par la méthode, soit on compare

des fonctions observées entre elles. La première semble plus simple à faire puisque les fonctions produites par les méthodes peuvent être bien représentées par les méthodes classiques d'approximation de fonction (approximation par les polynomes, par splines, par séries de Fourier,...).

2.1 Fonctions de base

Écrire une fonction comme combinaison linéaire de fonctions de base (fonctions élémentaires) permet de bien l'approximer puisque ces fonctions élémentaires ont de bonnes propriétés d'approximation. Selon Ramsay et Sylverman, un système de fonctions de base est un ensemble de fonctions connues ϕ_k qui sont mathématiquement indépendantes les unes des autres et qui ont la propriété de nous permettre de bien approximer arbitrairement toute fonction en prenant une combinaison linéaire d'un nombre suffisamment grand K de ces fonctions élémentaires Pour une fonction x donnée, sa représenation par le sytème de fonctions de base ϕ_k est une somme pondérée de K fonctions ϕ_k connues:

$$x(t) = \sum_{k=1}^{K} c_k \phi_k(t)$$

Pour approcher une fonction, non seulement les coefficients c_k doivent être estimés, mais aussi la dimension K de l'espace d'approximation doit être bien choisie en fonction des caractéristiques des données. Le choix de la base $(\phi_k)_k$ (choix déterminant) dépend des hypothèses *a priori* sur la variable à étudier (estimer ou non les dérivées de X, nature des données, ...). Par exemple, dans la pratique, la base de Fourier est choisie pour les données périodiques et la base B-spline pour les données non périodiques.

2.2 Quelques bases classiques

2.2.1 Système de base de Fourier Probablement, la base de Fourier, encore appelée *base trigonométrique*, est la plus connue des bases. Elle est la plus utilisée pour approcher des courbes ayant des comportements périodiques. Pour tout t, elle est définie ainsi:

$$\phi_0(t) = 1, \quad \phi_{2k-1}(t) = \sin(k\omega t) \quad et \quad \phi_{2k}(t) = \cos(k\omega t)$$

Le paramètre ω détermine la période $\frac{2\pi}{\omega}$. Les coefficients c_k du dèveloppement d'une fonction de cette base sont appelés *coefficients de Fourier*. L'algorithme dénommé Transformée de Fourier Rapide (*Fast Fourier transform (FFT)*) permet de calculer de manière extrêmement rapide les coefficients de Fourier d'une fonction. Les coefficients des dérivées s'obtiennent facilement puisque:

$$(\sin(k\omega t))' = k\omega \cos(k\omega t) \quad et \quad (\cos(k\omega t))' = -k\omega \sin(k\omega t)$$

Ainsi, si les coefficients de x sont

$$(0,\ c_1,\ c_2,\ c_3,\ ...)$$

alors ceux de $Dx = x'$ sont

$$(0,\ -\omega c_2,\ \omega c_1,\ -2\omega c_4,\ 2\omega c_3,\ ...)$$

et ceux de $D^2x = x''$ sont

$$(0,\ -\omega^2 c_1,\ -\omega^2 c_2,\ -4\omega^2 c_3,\ -4\omega^2 c_4,\ ...)$$

Les coefficients des dérivées supérieures s'obtienent en multipliant les coefficients individuels par les puissances appropriées de $k\omega$, avec des changements de signe et l'échange des coefficients de sinus et de cosinus selon le cas.

La série de Fourier est particulièrement utile pour les fonctions extrêmement régulières, de courbure à peu près semblable en tout point. Cependant, elle ne convient pas pour des fonctions reflétant des discontinuités dans la fonction elle-même ou dans les dérivés d'ordre inférieur.

2.2.2 Base polynomiale Pendant longtemps, la base des monômes (ou base monomiale) a été la référence pour l'approximation des fonctions non périodiques, en raison de la simplicité d'estimation des coefficients. Elle se définit ainsi:

$$\phi_k(t) = (t - \omega)^k, \quad k = 0, ..., K$$

où ω est le paramètre de décalage (ou de translation) qui est généralement choisi comme le centre de l'intervalle d'approximation T.

Cependant, du fait de la forte corrélation des valeurs des monômes lorsque le degré augmente, la base des monômes a été remplacée par la base des splines. Même si les polynômes ont tendance à bien se placer au centre des données,ils présentent un comportement plutôt peu attrayant dans les extrêmités ce qui fait que les bases polynomiales constituent généralement de mauvaises bases pour les extrapolations ou les prévisions (Ramsay and Silverman 2005).

2.2.3 Système de base spline Quant il s'agit d'approximer des données fonctionnelles non périodiques, le choix le plus courant de système d'approximation est la base spline. Cette base est composée de fonctions appelées *fonctions splines* ou simplement *splines*. Par définition, une fonction spline est une fonction polynomiale par morceaux, avec conditions de continuité sur la fonction et ses dérivées aux jointures. De manière plus détaillée, une spline sur un intervalle $T = [a,\ b]$ est une fonction ayant les propriétés suivantes:

- elle se caractérise par la donnée de L sous-intervalles de T délimités par des points appelés points de rupture (*breakpoints*) ou noeuds (*knots*) $\tau_0 = a < \tau_1 < \tau_2 < ... < \tau_L = b$ non forcément répartis régulièrement dans T;
- sur chaque sous-intervalle, la spline est un polynôme d'ordre m spécifié;
- aux points de ruptures, les différents polynômes s'y rejoignent (les valeurs des fonctions y sont égales), de même pour les dérivées jusqu'à l'ordre $m - 2$.

Dans une spline, plus le nombre de points de rupture augmente, plus on gagne en flexibilité et en précision. Et en général, pour approcher une fonction avec des splines, on aura tendance à placer plus de noeuds aux endroits de T où la fonction semble avoir le comportement le plus complexe et moins de noeuds là où elle n'est que légèrement non linéaire.

Ainsi, une base de spline d'ordre m et de séquence de noeuds τ est une famille de fonctions $\phi_k(t)$ vérifiant les propriétés suivantes:

- les fonctions de base $\phi_k(t)$ sont linéairement indépendantes,
- chaque fonction de base $\phi_k(t)$ est elle-même une fonction spline d'ordre m et de séquence de noeuds τ,
- toute combinaison linéaire de ces fonctions de base est encore une fonction spline,
- toute spline d'ordre m et de séquence de noeuds τ peut être exprimée comme combinaison linéaire de ces fonctions de base.

Comme exemple de base spline, nous pouvons citer la base plus populaire, développée par Carl De Boor, et appelée la base B-spline. Dans cette base, une fonction spline $S(t)$ avec des noeuds intérieurs discrets est définie comme:

$$S(t) = \sum_{k=1}^{m+L-1} c_k B_k(t, \tau)$$

Les fonctions de base du système B-spline de Boor, notées $B_k(t, \tau)$ au point t, avec τ comme séquence de noeuds, sont définies ainsi (Boor 1986):

- On part d'une partition des noeuds $t := t_i$ et on définit les B-splines d'ordre 1 comme étant les fonctions indicatrices dans cette partition, i.e:

$$B_1(t,i) = B_{i,1}(t) = X_i(t) = \begin{cases} 1 & \text{si } t \in [t_i,\ t_{i+1}[\\ 0 & \text{sinon} \end{cases}$$

de telle façon que $\sum_i B_{i,1}(t) = 1$ pour tout t.

- A partir de ces B-splines d'ordre 1, on obtient, par récurrence, les B-splines d'ordre supérieur:

$$B_{i,k}(t) = w_{i,k} B_{i,k-1}(t) + (1 - w_{i+1,k}) B_{i+1,k-1}(t)$$

avec

$$w_{i,k} = \begin{cases} \frac{t-t_i}{t_{i+k-1}-t_i} & \text{si } t_i \neq t_{i+k-1} \\ 0 & \text{sinon} \end{cases}$$

2.3 Utilisation du package `fda` de R

Sous le logiciel **R**, des objets de type fonctionnels sont créés à partir du package `fda`. Ces objets ont pour classe *"fd"*. Dans cette partie, nous allons montrer comment créer des objets fonctionnels et comment les utiliser via ce package.

Considérons une fonction x donnée avec comme représentation dans un système de fonctions de base ϕ_k:

$$x(t) = \sum_{k=1}^{K} c_k \phi_k(t)$$

La première étape consiste à construire la base classique. Pour cela, il faut faire appel aux fonctions du package `fda` dont le nom est de type `create.base_name.basis()` et qui génèrent des objets de classe *"basisfd"*. Ces fonctions génèrent des systèmes de fonctions classiques ayant pour nom `base_name`. Par exemple, si nous souhaitons générer les fonctions de la base de Fourier (respectivement la base polynomiale et la base B-spline), la fonction correspondante est `create.fourier.basis()` (respectivement `create.monomial.basis()` et `create.bspline.basis()`). Toutes ces fonctions ont comme premier argument `rangeval` qui permet de spécifier les extrémités de l'intervalle sur lequel on travaille. Les arguments suivants sont propres à chaque fonction. Entre autre, nous avons:

Table 1: Description des arguments des fonctions de création de base classique

Argument	Fonctions	Description
nbasis	create.bspline.basis create.monomial.basis create.fourier.basis	le nombre de fonctions de base
period	create.fourier.basis	le nombre de fonctions de base (impair)
norder	create.bspline.basis	l'ordre des splines (4 par défaut, pour des splines cubiques)
breaks	create.bspline.basis	liste des noeuds
rangeval	create.bspline.basis create.monomial.basis create.fourier.basis	les extrémités de l'intervalle sur lequel on travaille

Pour les B-splines, le nombre de fonctions de base doit vérifier :

nbasis = norder + length(breaks) - 2

En deuxième étape, nous avons la définition des coefficients c_k sous forme de vecteur ayant la même longueur que le nombre de fonctions de base créées. Si nous souhaitons avoir des valeurs aléatoires, nous pouvons faire appel aux fonctions aléatoires `rlois()` (où `lois` est le diminutif de la loi statistique) de **R** permettant de générer des valeurs aléatoires suivant une loi donnée comme par exemple `rnorm()` (loi normale), `rexp()` (loi exponentielle), `rpois()`(loi de Poisson), ...

La troisième étape est la création de l'objet fonctionnel à partir de la fonction `fd()` dont les deux principaux arguments sont `coeff=` pour le vecteur des coefficients et `basisobj` pour l'objet créer avec la fonction `create.base_name.basis()`.

la quatrième et dernière étape consiste à la représentation et à l'évaluation de l'objet fonctionnel via les fonctions `plot()` (pour la représentation de la fonction et/ou de ses dérivées) et `eval.fd()` (pour l'évaluation de la fonction et/ou de ses dérivées). L'argument `nx` permet de spécifier le nombre de points sur lesquels l'objet fonctionnel sera évalué (par défaut, `nx = 201`).

Les fonctions `plot()` et `eval.basis()` peuvent être appliquées à l'objet de classe *basisfd* (construit à la première étape) respectivement pour représenter et évaluer les fonctions de base et/ou leur dérivées en différents points comme le montre l'exemple de la figure ci-dessous.

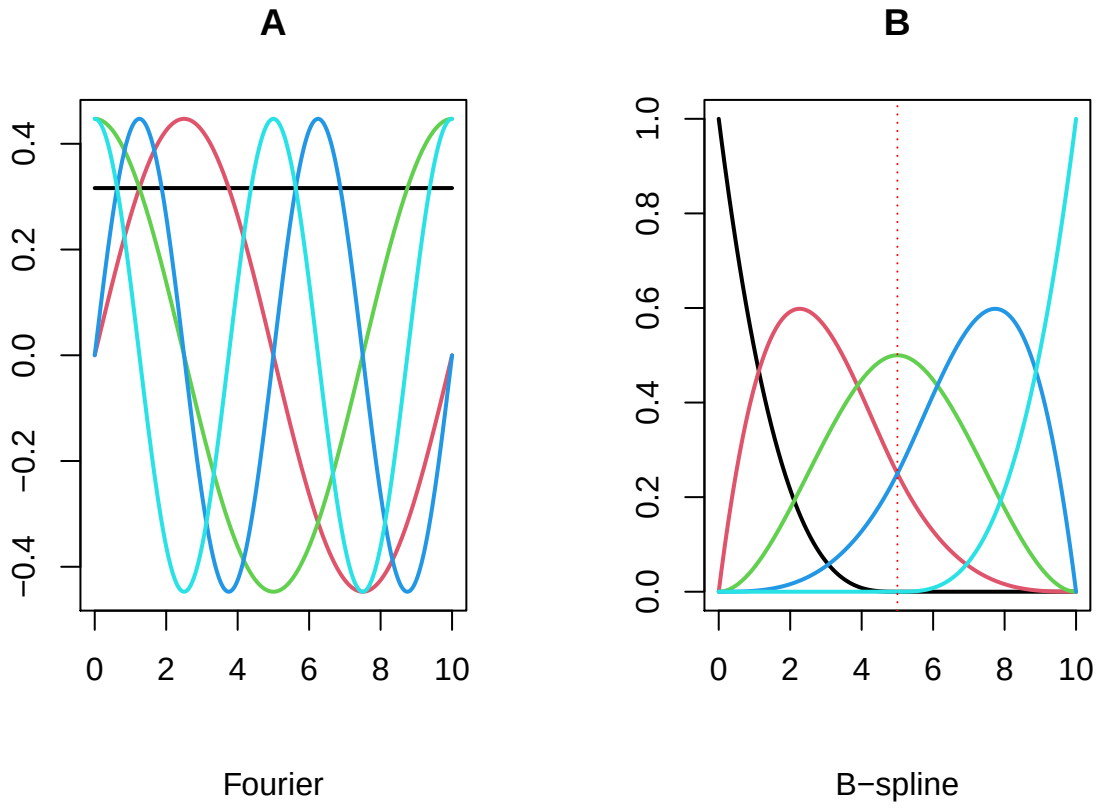


Figure 1: Les cinq premières fonctions de base des systèmes de Fourier (A) et B-spline (B)

Dans la figure I.1,le graphique à gauche montre les cinq premières fonctions de base de Fourier. La première fonction de la base est la fonction `constante`, puis il y'a les fonctions `sinus` et `cosinus` de période égale à la longueur de l'intervalle. Elles sont suivies par des fonctions sinus et cosinus de périodes décroissantes ou, de façon équivalente, de fréquences croissantes. Un nombre impair de fonctions de base est utilisé: la fonction constante et les couples sinus/cosinus. On représente au graphique de droite les fonctions des 5 premières

B-splines cubiques (d'ordre 4) avec noeuds équidistants tous disctincts sur tout l'intervalle $[0,\ 10]$ (Kokoszka and Reimherr 2017).

En guise d'exemple d'application, nous allons construire, étape par étape, une base de splines cubiques avec 21 noeuds équitablement répartis dans l'intervalle $[0,\ 2]$ et nous allons faire différents essais pour les tracer toutes (ou seulement certaines) et tracer également pour certaines fonctions, leurs dérivées jusqu'à l'ordre 3.

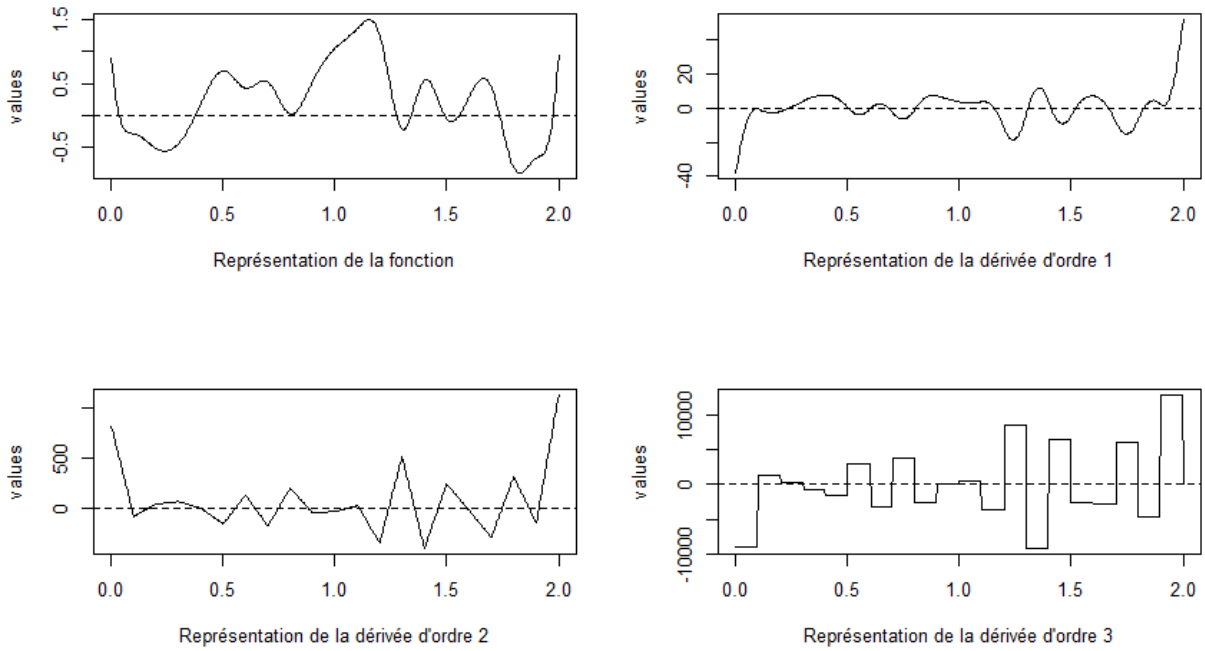


Figure 2: Représentations d'une fonction B-spline et de ses dérivées d'ordre $i = 1, ...,$

Un autre exemple est la courbe représentative des 10 premiers monômes sur l'intervalle $[-1,\ 1]$.

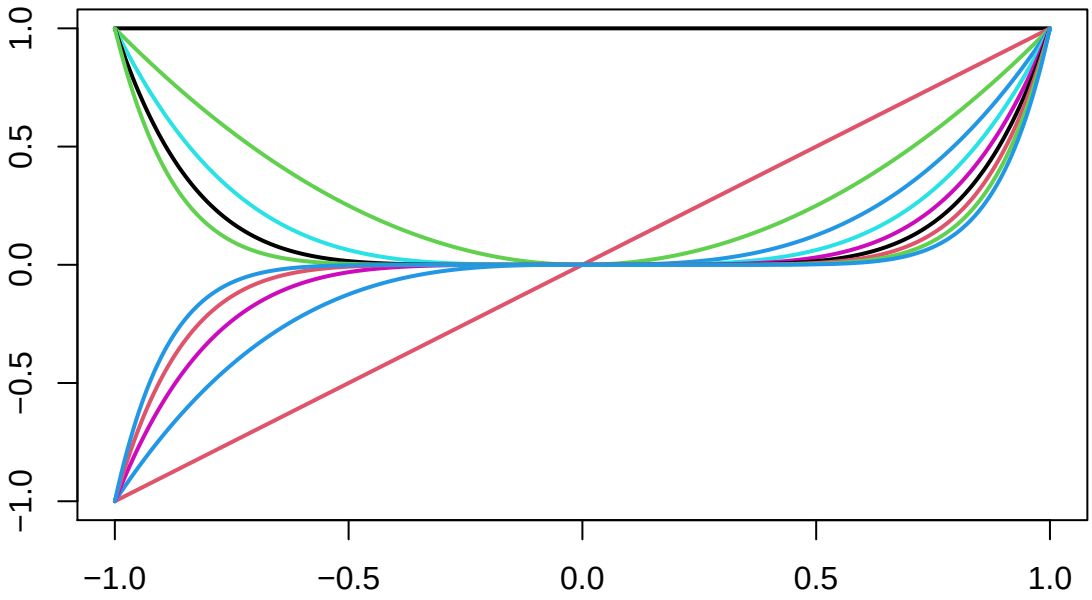


Figure 3: Représentation des 10 premiers monômes

3 Méthodes de lissage

3.1 Généralité

Généralement, les mesures d'un phénomène physique se traduisent par des points d'un graphique et chacun de ces points représente un état du phénomène. Ils sont présentés soit de manière discrète (c'est-à-dire isolés les uns des autres), soit de manière continue (c'est-à-dire groupés en un tracé). Ainsi, l'on se demande comment passer des données brutes sous forme discrétisées $y_j,\ j = 1, \ldots, n$ à des données sous leur forme continue (fonction ou tracé) $y_j = x(t_j) + \epsilon_j,\ j = 1, \ldots, n$. L'opération qui consiste en la conversion de mesures discrètes en une courbe continue est appelée lissage de points. La courbe ainsi tracée est nommée courbe de lissage et elle constitue la "meilleure représentation" possible du phénomène étudié.

Supposons une courbe $y(t)$ donnée par ses échantillons observables bruités $y(t_j) = y_j = x(t_j) + \epsilon_j$ pour tout $j = 1, ..., n$ où les $x(t_j)$ représentent les échantillons de la fonction idéale $x(t)$ sur l'intervalle $T = [a,\ b]$ et les $\epsilon(t_j)$ (variables non observées, i.i.d, centrées et admettant une variance inconnue) représentent les erreurs de mesure. La courbe $x(t)$ est estimée par la fonction $h(t)$, dérivable m fois sur l'ensemble T, qui minimise l'expression suivante:

$$E = \sum_{j=1}^{n} [y(t_j) - h(t_j)]^2 + \lambda \int_T [D^m h(t)]^2 dt$$

où $\lambda \geq 0$ est le paramètre de lissage.

Dans cette expression E, le premier terme, $\sum_{j=1}^{n}[y(t_j) - h(t_j)]^2$, minimise une distance de la courbe de lissage à l'ensemble des points et le deuxième terme, $PEN_m(h) = \lambda \int_T [D^m h(t)]^2 dt$, permet d'avoir une courbe régulière (c'est la contrainte de régularité). Les différentes méthodes de lissage différent par le choix de la fonction de minimisation $h(t)$ et par l'évaluation de la contrainte de régularité. Ici, pour le second terme de E, nous allons nous limiter uniquement aux mesures de la régularité ayant la forme classique ci-dessus ($PEN_m(h)$). Il en existe d'autres telles que

$$PEN_m(X) = \int_T [LX]^2(t)dt, \;\; LX = X^{(3)} + w^2 X'$$

où w est la période et l'opérateur L est appelé opérateur d'accélération harmonique.

Dans cette partie, nous allons présenter deux méthodes spécifiques de lissage de données fonctionnelles: le lissage par moindres carrés (cas où $\lambda = 0$) et le lissage par moindres carrés pénalisés (cas où $\lambda > 0$).

3.2 Lissage par les moindres carrés

Pour cette méthode le paramètre de lissage est nul ($\lambda = 0$). Ce qui fait que l'expression devient:

$$E = \sum_{j=1}^{n}[y(t_j) - h(t_j)]^2$$

Il s'agit ainsi d'un probléme de minimisation d'une erreur quadratique sans contrainte de régularité. Considérons la base de fonctions indépendantes $\phi_k(t),\ k = 1, \ldots, K$, telle que la fonction $h(t)$ s'écrive linéairement en fonction des $\phi_k(t)$:

$$h(t) = \sum_{k}^{K} c_k \phi_k(t) = C'\Phi$$

Alors le problème de minimisation devient:

$$E = \sum_{j=1}^{n}(y_j - \sum_{k}^{K} c_k \phi_k(t_j))^2 = (Y - \Phi C)'(Y - \Phi C) = \|Y - \Phi C\|^2$$

avec Φ une matrice à n lignes et K colonnes, $C = (c_1, \ldots, c_K)'$ et $Y = (y_1, \ldots, y_n)'$. Il s'agit d'une approximation par les moindres carrés dite simple ou ordinaire ou encore standarde (la matrice de pondération des résidus est la matrice identité $W = I$). Pour obtenir la solution du problème de minimisation, on annule la dérivée première de $\mathbf{E}$ par rapport à $\mathbf{C}$ et on résolve l'équation pour $\mathbf{C}$:

$$\frac{\partial E}{\partial C} = -2\Phi' Y + 2F'FC = 0 \;\Rightarrow\; \hat{C} = (\Phi'\Phi)^{-1}\Phi' Y$$

Ainsi, le vecteur des valeurs ajustées $\hat{Y} = (\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n})' = (\hat{x}(t_1), \hat{x}(t_2), \ldots, \hat{x}(t_n))'$ est donc:

$$\hat{Y} = \Phi\hat{C} = \Phi(\Phi'\Phi)^{-1}\Phi' Y$$

Selon Ramsay et Silverman, une approximation par les moindres carrés simples est appropriée dans les situations où l'on suppose que les résidus ϵ_j sont distribués de manière identique et indépendante avec une moyenne nulle et une variance constante σ^2.

Cependant, il se peut que les variances des ϵ_j varient en fonction des temps d'observation. Dans cette situation, la matrice de pondération (ou matrice de poids), différente de l'identité ($W \neq I$), est égale à la réciproque (*"l'inverse"*) de la matrice de variance-covariance $\sum_\epsilon$ (si elle est connue) des résidus: $W = \sum_\epsilon{}^{-1}$ (Ramsay et Silverman, 2005). On parle, alors, d'approximation par les moindres carrés pondérés. Ainsi,

si la matrice de poids est diagonale, l'expression de minimisation E et le vecteur de coefficient estimé $\hat{C}$ s'écrivent:

$$E = \sum_{j=1}^{n} w_j [y_j - \sum_{k}^{K} c_k \phi_k(t_j)]^2 = (Y - \Phi C)' W (Y - \Phi C) \text{et} \;\; \hat{C} = (\Phi' W \Phi)^{-1} \Phi' W Y$$

Lorsque l'on suppose bien les erreurs ϵ_j décorrélées et leurs variances connues, alors on peut choisir W comme étant les réciproques des variances des résidus. Sinon, la matrice de variance-covariance $\sum_\epsilon$ doit être estimée. Pour plus détails sur les différentes manières d'estimer $\sum_\epsilon$, nous suggérons une lecteure du livre de Ramsay et Silverman (2005).

3.2.1 Choix de la dimension de l'espace d'approximation

Que ce soit le lissage par les moindres carrés ordinaires ou par les moindres carrés pondérés, le choix de la dimension K (nombre de coefficients $\hat{c}_k$ estimés) de l'espace d'approximation reste crucial.

En effet, suivant les valeurs de K (grande ou petite), un compromis biais-variance (Voir Ramsay et Silverman pour plus de détails) s'impose:

- si la valeur de K est trop petite, la base de fonction n'est pas riche et le biais de l'estimation sera trop grand, ce qui donne peu de flexibilité: on parle alors de données sous-ajustées ("*underfitting*"). Les fonctions obtenues sont très lisses mais n'approximent pas trop les données.
- si la valeur de K est trop grande, la base est très riche. On aura beaucoup de flexibilité, ce qui peut causer une importante variabilité: on parle alors de données sur-ajustées ("*overfitting*"). Les fonctions obtenues ne sont pas très lisses (beaucoup d'oscillations)

Plusieurs méthodes, que nous ne détaillerons pas ici, ont été proposées pour le choix de K. Certaines d'entre elles fournissent l'idée de sélectionner les variables progressivement les unes après les autres, en vérifiant au fur et à mesure l'apport en terme de significativité de ces nouvelles fonctions ajoutées (on part d'une valeur petite de K). Par contre, d'autres méthodes, dites méthodes d'élagage de variables, commence avec une grande valeur de K et abandonnent (ou retranchent) au fur et à mesure des fonctions de bases qui ne semblent pas avoir une variation essentielle. La méthode de la validation croisée généralisée (qui sera détaillée dans la section suivante) nous offre, via le critère GCV, la possibilité de savoir la valeur optimale de K.

3.2.2 Codes R

Voyons maintenant, comment on calcule, de manière pratique, les coefficients $\hat{c}_k$ estimés. Supposons que l'on dispose des données $(t_j,\ y_j)_{j=1,\ldots,n}$ et que l'on a dèjà construit la matrice des fonctions de base évaluées au points t_j:

$$\Phi = (\phi_k(t_j)) \begin{array}{l} j = 1, \ldots, n \\ k = 1, \ldots, K \end{array}$$

Alors à l'aide de quelques fonctions **R**, nous pouvons calculer le vecteur des coefficients $\hat{C} = (\Phi'\Phi)^{-1}\Phi' Y$, construire l'objet fonctionnel et également, comparer les données brutes au lissage obtenu. Nous allons présenter ces fonctions **R**:

- fonction `crossprod()`: elle permet d'obtenir les valeurs $A = \Phi'\Phi$ et $b = \Phi' Y$. Si on lui fournit un seul argument ($x = \Phi$), elle renvoie A. Sinon pour deux arguments ($x = \Phi$ et $y = Y$), elle renvoie b.

- fonction `solve()`: elle permet de résoudre le système $A\hat{C} = b$. Elle prend comme arguments les objets ($a = A$ et $b = b$) obtenus par la fonction `crossprod()`.
- fonction `fd()`: elle permet de créer l'objet fonctionnel (objet de type fd) estimé $\hat{h}(t) = \sum_k^K \hat{c}_k \phi_k(t) = \hat{C}'\Phi$. Pour cela, il suffit de lui fournit les arguments $coeff = \hat{C}$ et $basisobj = \Phi$.
- fonction `plotfit.fd()`: elle permet de comparer graphiquement les données brutes aux données lisses.

Cependant, la fonction `smooth.basis()`, dont nous détaillerons plutard, est particulièrement adaptée au lissage de donnée.

3.2.3 Exemple pratique de lissage par moindres carrés Considérons les données `MontrealTemp` du package `fda`. Il s'agit des données sur les températures moyennes journalières en degrés Celsius à Montréal de 1961 à 1994.

Prenons l'exemple de l'année 1961 et représentons les données brutes que nous allons lisser en utilisant une base de Fourier avec $K = \{7, 9, 11, 15\}$.

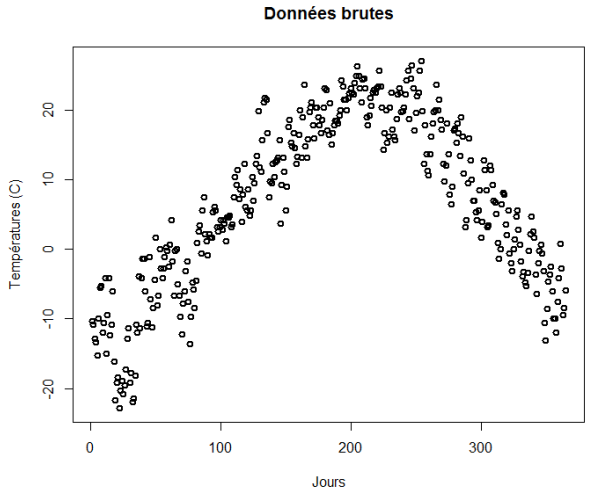


Figure 4: Températures moyennes journalières à Montréal en 1961 (du 01 janvier au 31 décembre)

Les graphes ci-dessous montrent le lissage des données avec différentes valeurs de K.

- Représentation avec les données brutes (figure 5):

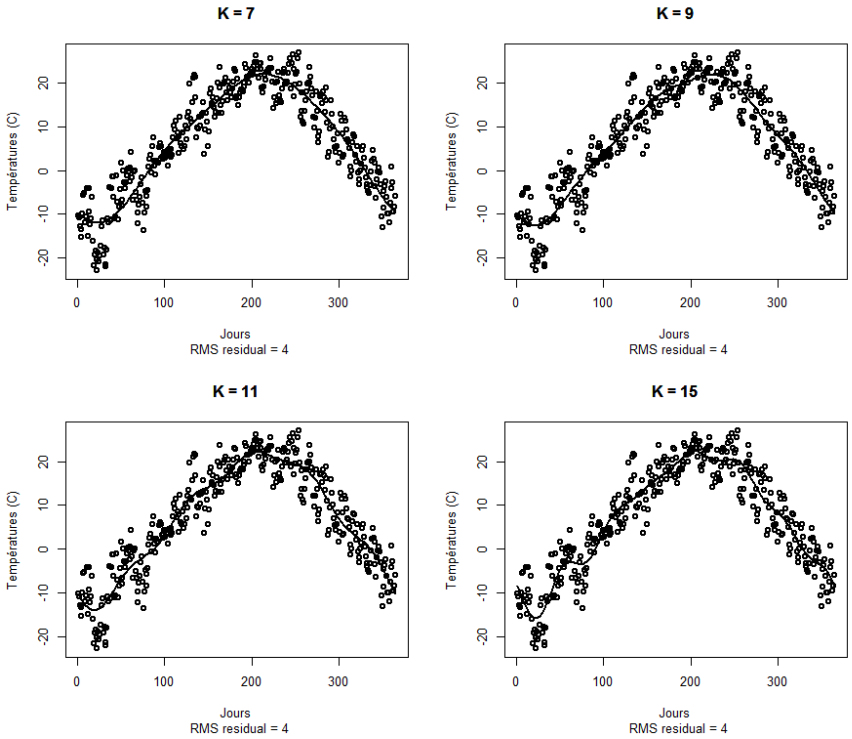


Figure 5: Lissage par moindres carrés (avec les données brutes)

- Représentation sans les données brutes (figure 6):

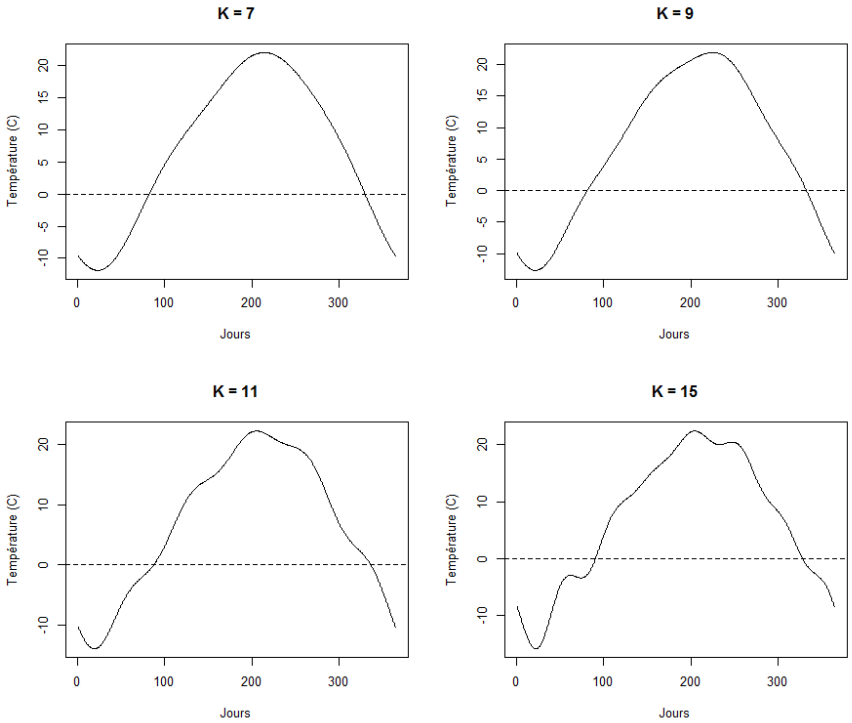


Figure 6: Lissage par moindres carrés (sans les données brutes)

Nous pouvons essayer de voir l'effet de la dimension de l'espace d'approximation en prenant différentes valeurs de K (par exemple, on peut prendre toutes les valeurs impaires de l'intervalle $[9,\ 24]$) et tracer pour l'année 1961 la courbe du critère de la validation croisée généralisée GCV (*Generalized Cross-Validation*) en fonction des valeurs de K. Nous allons dans ce cas utiliser la fonction `smooth.basis()` du package `fda`.

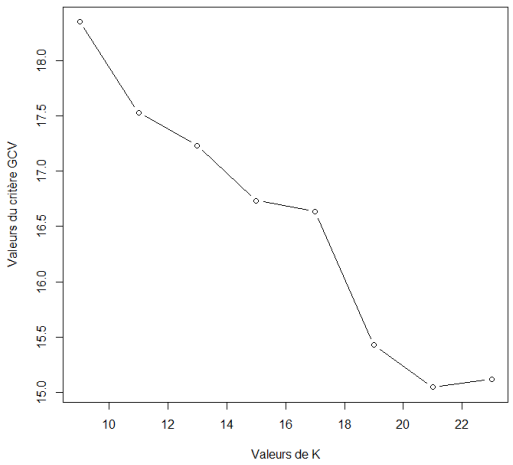


Figure 7: Représentation du critère GCV en fonction des valeurs de K pour l'année 1961

Selon ce critère, si on considère l'intervalle $[9,\ 24]$, la valeur optimale de K pour choisir la dimension de l'espace d'approximation pour l'année 1961, est $K = 21$ qui correspond à la valeur pour laquelle le GCV atteint son minimum.

Nous pouvons ainsi voir le graphique obtenu pour $K = 21$.

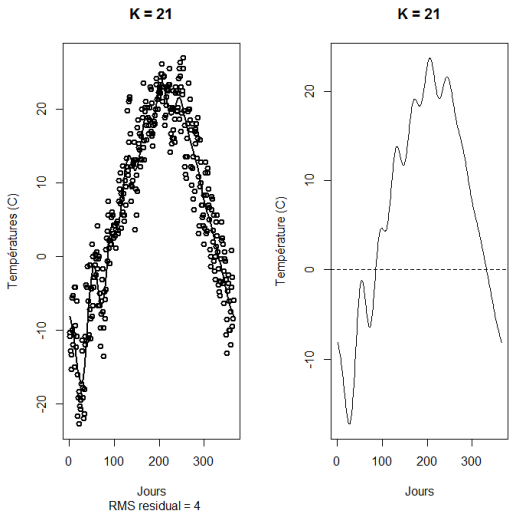


Figure 8: Lissage par moindres carrés pour K = 21

3.3 Lissage par la pénalisation

Précèdement, nous avons vu le cas où le paramètre de lissage ou de pénalité vaut zéro ($\lambda = 0$). Dans cette partie, nous allons considérer $\lambda > 0$. L'expression de minimisation E devient alors: (on l'appelle critère des moindres carrés pénalisés)

$$E = \sum_{j=1}^{n}[y(t_j) - h(t_j)]^2 + \lambda \int_T [D^m h(t)]^2 dt \text{ , avec } \lambda > 0$$

Dans ce problème d'optimisation, deux objectifs contradictoires naissent:

- avoir un bon ajustement aux données,
- mais que cet ajustement ne soit pas trop précis, sous peine d'obtenir trop de variabilité (sur-ajustement).

Par conséquent, il s'agit d'une minimisation d'un critère de type moindres carrés, mais sous la contrainte que la fonction à estimer soit régulière. C'est-à-dire que la fonction cible appartient à un espace de fonctions dites *"lisses"* (fonctions m fois dérivables et de dérivées $m^{\text{ièmes}}$ intégrables sur T).

L'unique solution $\hat{h} \in \arg\min_{h \in S^m(T)}(E)$ (où $S^m(T)$ est l'espace des fonctions m fois dérivables et de dérivées $m^{\text{ièmes}}$ intégrables sur T) de ce problème d'optimisation est appelée spline naturelle de lissage d'ordre m associée aux données $(t_j,\ y_j),\ j = 1, \ldots, n$. Selon Green et Silverman (1994), pour $T = [a,\ b]$, $\hat{h}$ y est défini comme suit:

- la restriction de $\hat{h}$ aux intervalles $[a,\ t_1]$ et $[t_n,\ b]$ est un polynôme de degré au plus $m-1$;
- la restriction de $\hat{h}$ aux intervalles $[t_j,\ t_{j+1}],\ j = 1, \ldots, n-1$ est un polynôme de degré au plus $2m-1$;
- la fonction $\hat{h}$ est de classe $C^{2m-2}(T)$

En particulier, $\hat{h}$ est une spline dont les noeuds sont les points t_j où il y a des données.

Considérons, de manière générale, la base de fonctions indépendantes $\phi_k(t),\ k = 1, \ldots, K$, telle que la fonction $h(t)$ s'écrive linéairement en fonction des $\phi_k(t)$ et W, la matrice des poids. Alors le problème de minimisation devient:

$E = \sum_{j=1}^{n} w_j[y_j - h(t_j)]^2 + \lambda \int_T [D^m h(t)]^2 dt$ or, nous avons: $h(t_j) = \sum_k^K c_k \phi_k(t_j)$

$E = \sum_{j=1}^{n} w_j[y_j - \sum_k^K c_k \phi_k(t_j)]^2 + \lambda \int_T [D^m(\sum_k^K c_k \phi_k(t))]^2 dt$

$E = (Y - \Phi C)' W (Y - \Phi C) + \lambda \int_T [D^m C' \Phi]^2$ écriture matricielle

$E = (Y - \Phi C)' W (Y - \Phi C) + \lambda C' [\int_T (D^m \Phi)(D^m \Phi')] C$

$E = (Y - \Phi C)' W (Y - \Phi C) + \lambda C' R C$ avec, $R = \int_T (D^m \Phi)(D^m \Phi')$

Comme nous l'avons fait à la partie précédente, nous allons annuler la dérivée première de E par rapport à C et résoudre l'équation obtenue pour C.

$$\frac{\partial E}{\partial C} = -2\Phi' W Y + 2F' W F C + \lambda R C = 0 \ \Rightarrow\ \hat{C} = (\Phi' W \Phi + \lambda R)^{-1} \Phi' W Y$$

Ainsi, l'expression du vecteur $\hat{Y}$ d'ajustement des données est:

$$\hat{Y} = \Phi \hat{C} = \Phi (\Phi' W \Phi + \lambda R)^{-1} \Phi' W Y$$

3.3.1 Choix du paramètre de lissage

Le choix de λ est inévitable. En effet, dans la pratique, suivant les valeurs de λ, nous avons le compromis suivant:

- si λ est petit, le poids de la pénalité dans la minimisation sera peu important et, dans ce cas, l'ajustement aux données est privilégié;
- par contre si λ est grand, le poids de la pénalité dans la minimisation sera important et, dans ce cas, le caractère lisse de la fonction choisie sera privilégié.

La méthode de la *validation croisée* (méthode CV) est la plus utilisée pour choisir λ. Le principe de base de cette méthode est de retirer une partie des données, appelée échantillon de validation, et d'ajuster le modèle au reste des données, appelées échantillon de formation. Ce qui permet de voir dans quelle mesure le modèle s'ajuste aux données non utilisées.

Plusieurs variantes de la validation croisée existent. Sans beaucoup détailler, nous pouvons en citer:

- la méthode de la validation croisée leave one-out (CV): l'échantillon de validation est composée d'une seule observation et on ajuste le modèle aux données restantes. On estime la valeur ajustée pour la valeur des données omises. Cette procédure sera répétée pour chaque observation (n fois) et on calcule à la fin le score ou critère de validation croisée défini, pour chaque $\lambda > 0$, par:

$$CV(\lambda) = \frac{1}{n}\sum_{j=1}^{n}[y(t_j) - \hat{h}_\lambda^{(-j)}(t_j)]^2$$

  avec $\hat{h}_\lambda^{(-j)} \in \arg\min_{h\in S^m(T)}(\sum_{j'\neq j}[y_{j'} - h(t_{j'})]^2 + \lambda \int_T [D^m h(t)]^2 dt)$.

- la méthode de la validation croisée généralisée (GCV): il s'agit d'une version améliorée du critère précédent. Elle se repose sur le critère suivant:

$$GCV(\lambda) = \frac{n \text{ trace}\{Y'[I - S_{\Phi,\lambda}]^{-2}Y\}}{\{\text{trace } [I - S_{\Phi,\lambda}]\}^2}$$

  avec $S_{\Phi,\lambda} = \Phi M(\lambda)^{-1}\Phi' W$ et $M(\lambda) = \Phi' W\Phi + \lambda R$.

La valeur de λ qui donne le minimum de ce critère (CV ou GCV) sera choisi comme paramètre de lissage.

3.3.2 Codes R

Comme nous l'avons fait avec la méthode de lissage précédente, voyons également, comment on calcule, de manière pratique, les coefficients $\hat{c}_k$ estimés. Supposons que l'on dispose des données $(t_j,\ y_j)_{j=1,\ldots,n}$ et que l'on a dèjà construit la matrice des fonctions de base évaluées au points t_j:

$$\Phi = (\phi_k(t_j)) \begin{array}{l} j = 1, \ldots, n \\ k = 1, \ldots, K \end{array}$$

Alors deux fonctions du package **fda** sont essentielles pour construire l'objet fonctionnel souhaité: `smooth.basis()` et `fdPar()`.

- la fonction `smooth.basis()`: elle permet de construire un objet fonctionnel en minimisant, sur un espace de fonctions donné, un critère des moindres carrés pénalisés ou non. Principalement, elle prend en entrée `argvals` (les valeurs des $(t_j)_{j=1,\dots,n}$), `y` (les données $yt_j)_{j=1,\dots,n}$) et `fdParObj` (la matrice Φ des fonctions de base). Elle retourne un objet de type `fdsmooth` qui contient plusieurs éléments dont les trois premiers nous intéressent: `$fg` qui est l'objet fonctionnel, `$df` qui contient les degrés de liberté utilisés pour définir la courbe ajustée et `$gcv` qui contient la valeur du critère GCV à minimiser pour choisir le paramètre λ optimal.
- la fonction `fdPar()`: lorsque l'on souhaite spécifier la pénalité de rugosité pour le lissage par les moindres carrés pénalisés, l'appel à la fonction `fdPar()` doit précédé celui de la fonction `smooth.basis()`. Dans ce cas, elle prend principalement comme entrée, `fdobj` (la matrice Φ des fonctions de base), `Lfdoj` (l'ordre m de la dérivée à utiliser si on choisit comme pénalité $PEN_m(h)$) et `lambda` (le paramètre λ devant la pénalité). Une fois ces éléments fournis, on fait appel à la fonction `smooth.basis()` en fournissant à l'argument `fdParObj` l'objet renvoyé par la fonction `fdPar()`.

3.3.3 Exemple pratique de lissage par moindres carrés pénalisés

Considérons l'année 1961 des données `MontrealTemp` du package `fda`. En guise d'exemple, nous allons effectuer le lissage des données en utilisant une base de Fourier avec $K = 7$, $m = 2$ et $\lambda = 1$.

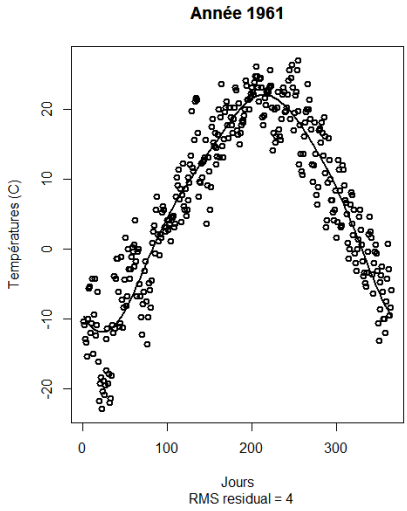


Figure 9: Représentation brute des Températures moyennes journalières avec la courbe lisse obtenue

Essayons d'augmenter la valeur du paramètre λ pour voir ce que nous obtenons. Prenons par exemples, $\lambda = (1, 15000, 500000, 1000000)$.

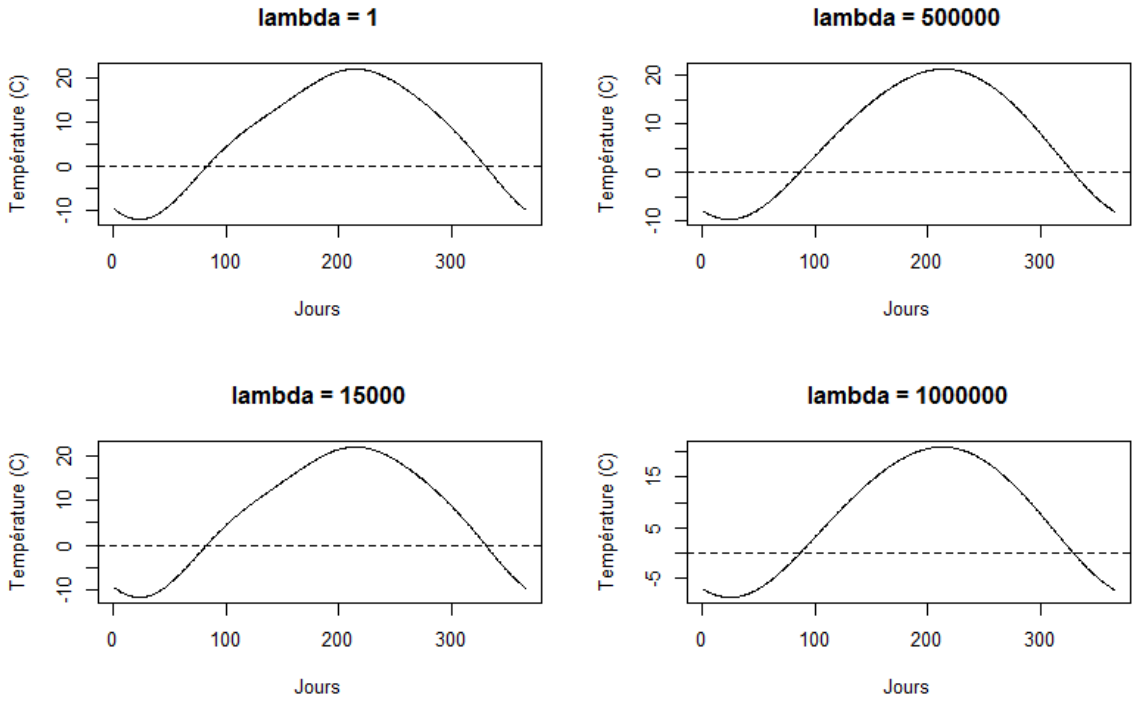


Figure 10: Représentation des courbes lisses suivant les valeurs de λ

D'après la figure 10, nous avons l'impression que les courbes deviennent plus lisses avec l'augmentation de la valeur du paramètre λ. Nous pouvez vérifier cela via le critère de validation croisée généralisée de chaque ajustement. Le tableau ci-dessous fournit les valeurs du critère GCV obtenues pour chaque ajustement.

Table 2: Valeur de GCV en fonction du paramètre

Paramètre λ	Critère $GCV(\lambda)$
1	18.51682
15000	18.49802
500000	19.1433
1000000	20.12329

La plus petite valeur de GCV est obtenu pour $\lambda = 15000$. Ce qui veut dire que par rapport aux autres valeurs, la valeur $\lambda = 15000$ est plus optimale pour l'ajustement.
Nous pouvons également prendre une plage de valeurs de λ et tracer la courbe $GCV(\lambda)$ afin de voir la valeur optimale du paramètre de lissage. Pour ce faire, nous allons considérer des plages de valeurs pour λ et à l'aide des fonctions `smooth.basis()` et `fdPar()`, nous obtenons la figure 11 suivante:

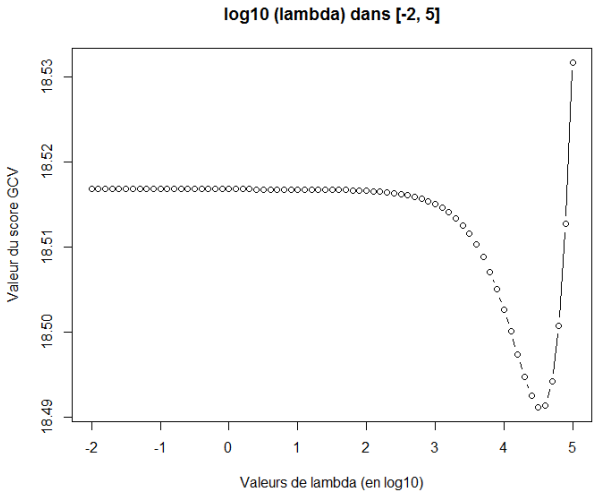


Figure 11: Représentation du critère GCV en fonction des valeurs de $\log_{10}(\lambda)$

Ici, nous considérons la plage de valeurs $\lambda \in [10^{-2},\ 10^{5}]$. Une simple lecture du graphique 11 montre que la valeur minimale du score GCV est obtenue pour $\log_{10}(\lambda) = 4.5 \Rightarrow \lambda = 10^{4.5} \approx 32000$. Donc $\lambda = 32000$ est la valeur optimale du paramètre de lissage au sens de la validation croisée généralisée (GCV).
La figure 12 montre le grapique obtenu pour cette valeur de λ.

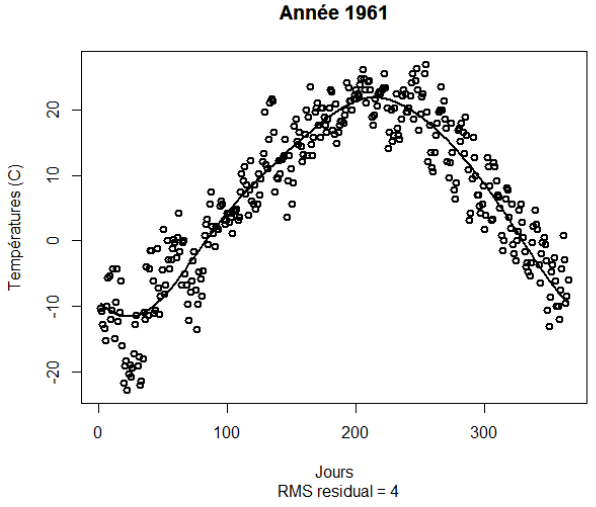
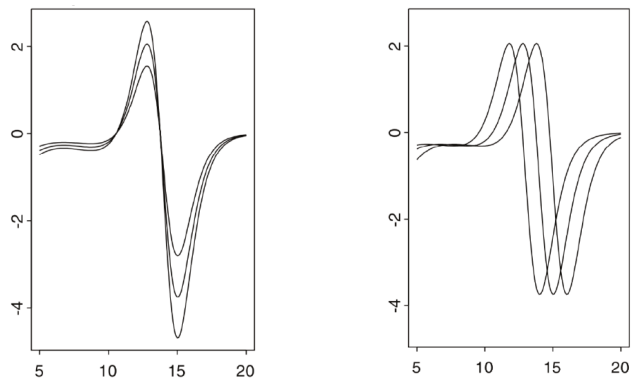


Figure 12: Représentation de la courbe obtenue pour $\log_{10}(\lambda) = 4.5$

4 Alignement des courbes

Nous venons de transformer nos observations en formes fonctionnelles (lissage): $x(t) = \sum_k^K c_k \phi_k(t)$ et maintenant, nous voulons les analyser via des statistiques sommaires telles que les fonctions de moyenne, de covariance, ... Cependant, le problème de *variabilités des données fonctionnelles* se pose (Figure 12). Il s'agit de la variabilité en amplitude et de la variabilité en phase entre les observations fonctionnelles (les courbes ne sont pas alignées). Par conséquent, le calcul direct de la moyenne pour produire une moyenne transversale sur des courbes non alignées produira de mauvais estimateurs des vraies valeurs des fonctions de moyenne et de covariance. Ainsi, l'alignement ou l'enregistrement (*curve registration*) des données fonctionnelles qui implique des transformations de l'argument t plutôt que des valeurs $x(t)$, permet d'y remédier (Ramsay and Silverman 2005).



Variation d'amplitude *Variation de phase*

Figure 13: Représentation des deux types de variabilités des données fonctionnelles

Tandis que la variation d'amplitude (ou variation verticale) est due au fait que deux ou plusieurs valeurs des fonctions $\{x_i(t),\ i = 1, \ldots, n\}$ diffèrent aux moments t de comparaison, mais présentent les mêmes caractéristiques de forme à ce même moment; la variation de phase est, quant à elle, due au fait que deux ou plusieurs valeurs des fonctions $\{x_i(t),\ i = 1, \ldots, n\}$ ne peuvent pas être comparées au même moment t parce qu'elles n'y présentent pas le même comportement. Ce qui fait que, pour comparer ces fonctions, l'échelle de temps doit être transformée (Ramsay and Li 1998).
Qu'est-ce qu'une fonction de déformation ? Comment aligner des courbes ? Quels outils utilise-t-on pour l'analyse descriptive des données fonctionnelles ? La réponse de ces trois questions fera l'objet de cette section que nous terminerons par des exemples pratiques d'alignement de courbes.

4.1 Fonction de déformation

Afin de remédier au problème de variabilité des données fonctionnelles susmentionné, nous pouvons déformer l'argument de temps. En effet, tout se fait à l'aide de fonctions dites *fonctions de déformation temporelle* $h(t)$ qui consistent à effectuer des transformations de l'échelle temporelle de chaque courbe de façon à ce que les extrêmums et les passages par zéro se produisent au même moment. Pour une courbe $x(t)$ donnée, sa courbe enregistrée est notée $x^*(t) = (x \circ h)(t) = x[h(t)]$.
L'objectif principal est l'estimation des fonctions de déformation $h(t)$ qui doivent toutes répondre obligatoirement à trois conditions (Ramsay, Hooker, and Graves 2009). D'abord, il y'a la condition d'invariabilité au niveau des bornes. Les fonctions $h(t)$ sont telles que les débuts (t_d) et les fins (t_f) des courbes ne changent pas: $h(t_d) = t_d$ et $h(t_f) = t_f$. Ensuite, il y'a la condition de monotonicité. Les événements dans les courbes alignées se produisent dans le même ordre que ceux des courbes non alignées, c'est-à-dire que les fonctions de déformation $h(t)$ doivent toutes être monotones et croissantes: $h(t_i) > h(t_j) \Longleftrightarrow t_i > t_j$ pour tout $i \neq j$. Enfin, la condition de proportionnalité exige que les courbes alignées possèdent toutes la même forme que les courbes non alignées sans affecter les amplitudes, c'est-à-dire que les pics et les vallées se produisent en même temps: si x_1 et x_2 sont proportionnelles, alors $x_1 \circ h$ et $x_2 \circ h$ le sont aussi, i.e $(x_1 \circ h)(t) = \alpha(x_2 \circ h)(t)$, avec $\alpha > 0$, les courbes alignées doivent être proportionnelles entre elles.

4.2 Alignement par points de repère

Il s'agit de la procédure d'alignement la plus simple. Elle utilise les points de repère qui se définissent comme étant une certaine caractéristique des courbes avec des emplacements clairement identifiables telsque les minima, les maxima ou les passages à zéro. L'alignement se fait en transformant t pour chaque courbe de telle sorte que les emplacements des points de repère restent inchangés pour toutes les courbes (Ramsay, Hooker, and Graves 2009). En d'autres termes, les points de repère des courbes alignées se produisent au même moment que ceux des courbes non alignées.
Sous **R**, la fonction `landmarkreg()` permet de faire l'alignement par points de repère. Le tableau ci-dessous fournit une description des arguments les plus importants de cette fonction.

Table 4: Description des arguments de la fonction landmarkreg()

Arguments	Description
$fdobj$	objet de données fonctionnelles pour les courbes à enregistrer
$ximarks$	une matrice $N \times NL$ contenant les temps ou coordonnées des points de repère des courbes à enregistrer. Le nombre de lignes N est égal au nombre d'observations et le nombre de colonnes NL est égal au nombre de points de repère.
$x0marks$	vecteur de longueur NL des temps de repères pour la courbe cible. Par défaut, la moyenne des heures de repère en ximarks est utilisée: x0marks = xmeanmarks
$WfdPar$	objet de paramètre fonctionnel définissant une fonction de déformation
$monwrd$	valeur logique: si VRAI, la fonction de déformation est estimée à l'aide d'une méthode de lissage monotone; sinon, une méthode de lissage régulière est utilisée ce qui ne garantit pas des fonctions de déformation strictement monotones
$ylambda$	paramètre de lissage à utiliser dans le calcul des fonctions à enregistrer. Pour les bases de grande dimension, des ondulations locales peuvent être trouvées dans les fonctions enregistrées ou leurs dérivés qui ne sont pas visibles dans les fonctions non enregistrées. Dans ce cas, ce paramètre doit être augmenté.

Principalement, la fonction `landmarkreg()` renvoie comme résultat plus intéressant, deux objets fonctionnels: `fdreg` qui contient des données pour les courbes enregistrées et `warpfd` qui contient des données pour les fonctions de déformation.
Pour cette méthode d'alignement, il est essentiel que l'emplacement de chaque point de repère soit clairement défini dans chacune des courbes cibles. Si tel n'est pas le cas, alors on pourra faire recours à une autre procédure d'alignement dite alignement continu.

4.3 Alignement continu

Plutôt que de se limiter uniquement à quelques points de la courbe, cette procédure utilise entièrement toute une courbe pour l'aligner. Pour des courbes cibles données, elle aligne leurs caractéristiques saillantes telles que les pics, les creux et les franchissements de seuils fixes, en transformant ou en déformant le domaine d'argument de chaque courbe cible d'une manière qui est, non seulement non linéaire, mais aussi qui préserve strictement l'ordre. Selon Ramsay et Hooker, l'idée derrière cette méthode d'alignement est que si une courbe enregistrée via $x[h(t)]$ et une courbe cible $x(t)$ différent uniquement en termes de variation d'amplitude, alors elles sont proportionnelles l'une de l'autre suivant les valeurs de l'argument t.
Mathématiquement, cette procédure, souvent dite *"Procrustes method"*, se résume par l'algorithme suivant:
,

- **1.** Exprimer les données sous leur forme focntionnelle (lissage): $x(t) = \sum_{k=1}^{K} c_k \phi_k(t)$
- **2.** Calculer la courbe moyenne des fonctions: $\bar{\mu}_n(t) = \frac{1}{n} \sum_{i=1}^{n} x_i(t)$
- **3.** Calculer, pour chaque observation i, la fonction de déformation temporelle $h_i(t)$ qui minimise la fonction objective : $J = \sum_{i=1}^{n} \int_T [x_i(h_i(t)) - \bar{\mu}_n(t)]^2 dt$. On dit alors qu'on aligne les courbes avec la moyenne des observations.
- **4.** On répète les étapes 2 et 3 jusqu'à convergence. Selon Ramsay et Silvermann, ce processus converge généralement en une ou deux itérations.

Sous **R**, la fonction `register.fd()` permet de faire l'alignement des courbes à l'aide d'un critère continu. Le tableau ci-dessous fournit une description des principaux arguments de la fonction `register.fd()`.

Table 5: Description des arguments de la fonction register.fd()

Arguments	Description
$y0fd$	Objet de données fonctionnelles pour la fonction cible. Il peut contenir soit une seule courbe, soit le même nombre de courbes contenues dans l'argument yfd.
yfd	Objet de données fonctionnelles pour les fonctions à enregistrer sur $y0fd$.
$WfdParobj$	un objet de paramètre fonctionnel contenant une seule fonction ou le même nombre de fonctions que celui contenu dans yfd. Les coefficients fournissent les valeurs initiales dans l'estimation d'une fonction $W(t)$ qui définit les fonctions de déformation $h(t)$. $WfdParobj$ définit également la pénalité de rugosité et le paramètre de lissage utilisés pour contrôler la rugosité de $h(t)$.
$conv$	critère de convergence des itérations.
$iterlim$	une limite sur le nombre d'itérations.
$dbglev$	Entier $\{0, 1, 2\}$ qui donne le niveau de sortie de l'historique des calculs. 0 signifie aucune sortie, 1 niveau de sortie intermédiaire et 2 sortie complète
$periodic$	Valeur logique. Si VRAI, alors les courbes sont périodiques et un paramètre de décalage est ajusté. L'option périodique doit uniquement être utilisée avec une base de Fourier pour la fonction cible $y0fd$ et les fonctions à enregistrer yfd.
crit	Entier $\{1, 2\}$ qui indique la nature du critère d'enregistrement continu qui est utilisé: 1 pour les moindres carrés et 2 pour le log du rapport des valeurs propres

La fonction `register.fd()` renvoie une liste de quatre éléments à savoir: l'objet `regfd` qui contient les fonctions enregistrées, l'objet `warpfd` qui contient les fonctions de déformation $h(t)$, l'objet `wfd` qui contient les fonctions $W(t)$ qui permettent de définir les fonctions $h(t)$ et l'objet `shift` qui contient le vecteur de décalage pour les courbes périodiques.

4.4 Outils d'exploration

Nous allons fournir des versions fonctionnelles des outils de statistique sommaire tels que la moyenne, la variance, la covariance et la corrélation entre deux variables fonctionnelles et pour chaque outils, nous donnons la fonction R qui permet de le calculer. Pour cela, on suppose n observations de variables aléatoires fonctionnelles $X_1, \ldots, X_n$, i.i.d.
Le tableau ci-dessous donne quelques formules de ces outils statistiques et des fonctions R du package `fda` qui permettent de les calculer.

Table 6: Outils d'exploration et fonctions R

Outils	Formules	Fonctions R
Moyenne	$\bar{X}_n(t) = \frac{1}{n}\sum_{i=1}^{n} X_i(t)$	*mean.fd()*
Variance	$Var_n(t) = \frac{1}{n}\sum_{i=1}^{n}[X_i(t) - \bar{X}_n(t)]^2$	*sd.fd()* *std.fd()*
Covariance	$C_n(t_1,\ t_2) = \frac{1}{n}\sum_{i=1}^{n}[X_i(t_1) - \bar{X}_n(t_1)][X_i(t_2) - \bar{X}_n(t_2)]$	*var.fd()*
Corrélation	$Corr_n(t_1,\ t_2) = \frac{C_n(t_1,\ t_2)}{\sqrt{Var_n(t_1)Var_n(t_2)}}$	*cor.fd()*

4.5 Exemples pratiques

Considérons encore les données météorologiques du Montréal `MontrealTemp` du package `fda`. Les 34 courbes correspondent aux températures moyennes journalières des 34 années, de 1961 à 1994. Sur chacune des courbes dans ces exemples, la courbe moyenne des fonctions est représentée par la ligne pointillée en noire.

- **Lissage des données**: Nous avons effectué le lissage par moindres carrés pénalisés des données avec paramètre de pénalisation $\lambda = 10^4$ (valeur optimal obtenue à la figure 14), en utilisant 7 fonctions de la base de Fourier.

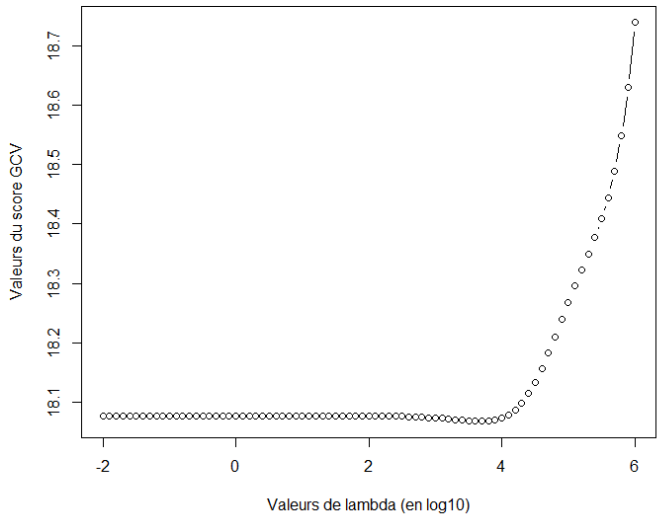


Figure 14: Représentation des valeurs GCV moyennes des 34 courbes.

La figure 15 montre les 34 courbes non alignées obtenues après le lissage. En observant les températures moyennes, nous constatons, par exemple, que le moments des journées les plus chaudes ou les plus froides varient d'une année à l'autre. Au même moment t, les courbes n'y présentent pas le même comportement: c'est la variation de phase. Par conséquent, les courbes ne peuvent pas être comparées au même moment t. D'où la nécessité de faire l'alignement.

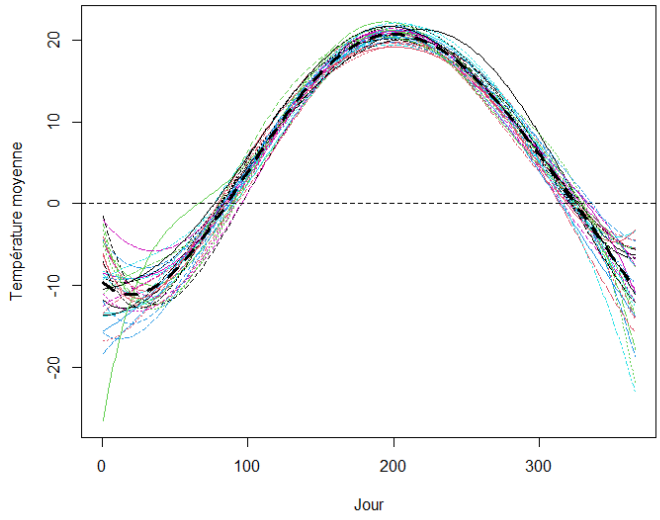


Figure 15: Lissage des données MontrealTemp par MCP.

- **Alignement par points de repère**: Comme points de repère, nous choisissons, pour chaque année, les températures minimales, maximales et nulles. Graphiquement, les températures nulles correspon-

dent au passage à zéro des courbes (à gauche et à droite sur l'axe $y = 0$). Nous avons utilisé la fonction `landmarkreg()` pour effectuer cet alignement.

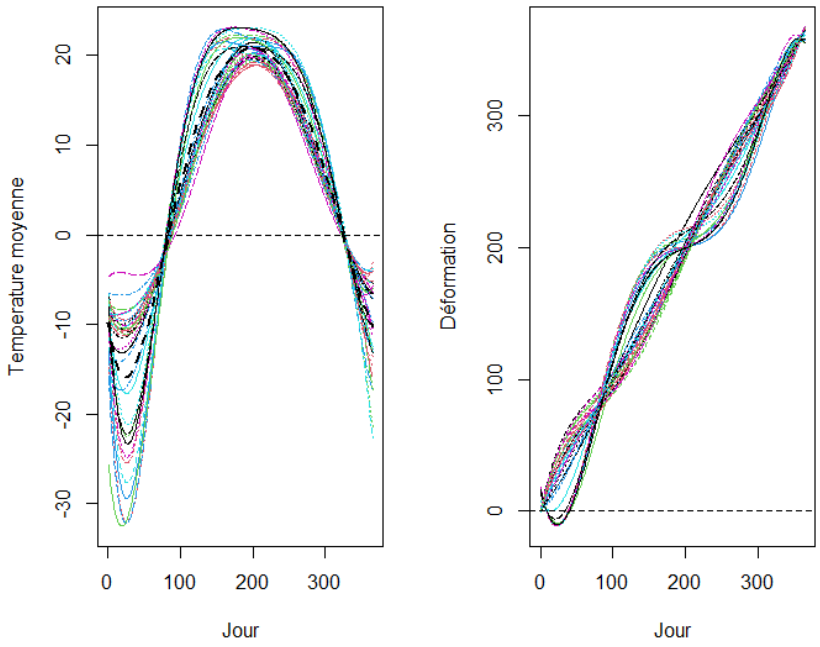


Figure 16: Alignement des courbes par points de repère via la fonction 'landmarkreg()'.
On constate que les emplacements des points de repère restent inchangés pour toutes les courbes.

Sur le graphique de droite de la figure 16, nous pouvons voir les fonctions de déformation $h_i(t)$ utilisées pour l'alignement par points de repères des courbes (panneau de gauche). Ici, la ligne en pointillées noires représente la courbe de la fonction linéaire $f(t) = t$ pour $t = 0.5, 1.5, \ldots, 364.5$.
A un moment t donné, si $h(t) > t$, alors notre processus observé est en retard (Ramsay and Silverman 2005). Ici, cela signifie que la température atteint le minimum ou les zéros plus tard que la moyenne en ce moment. De même, si $h(t) < t$, notre processus est en avance. Dans ce cas, la température atteint le pic plus tôt que la moyenne.

- **Alignement continu**: En utilisant la fonction `register.fd()`, nous avons effectué l'alignement des courbes de température par la méthode d'enregistrement continu.

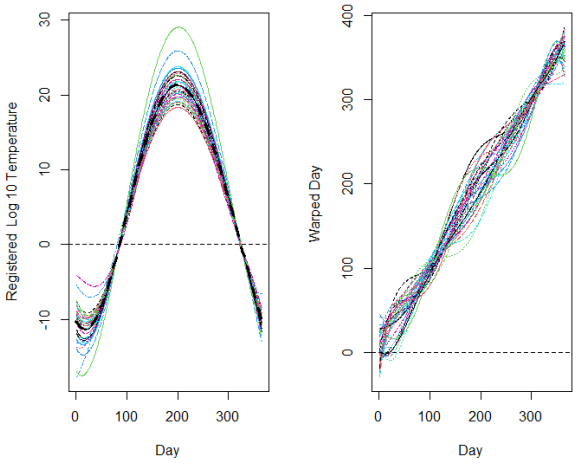


Figure 17: Alignement continu des courbes via la fonction 'register.fd()'.

Dans la figure 17, le panneau de gauche montre que les courbes de température enregistrées en continu sont maintenant alignées sur l'ensemble des valeurs des températures; le panneau de droite montre les fonctions de déformation du temps $h(t)$ de toutes les années.
A l'aide de la fonction `plotreg.fd()`, nous obtenons, de manière détaillée, une série de tracés pour chaque courbe.

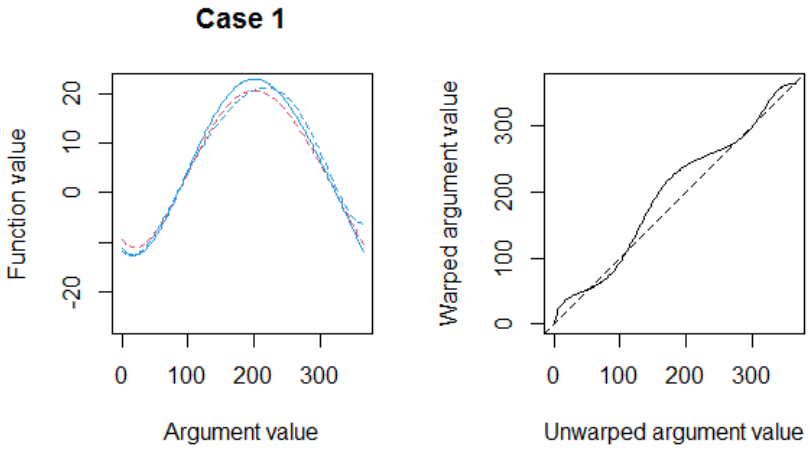


Figure 18: Tracés pour l'année 1961.

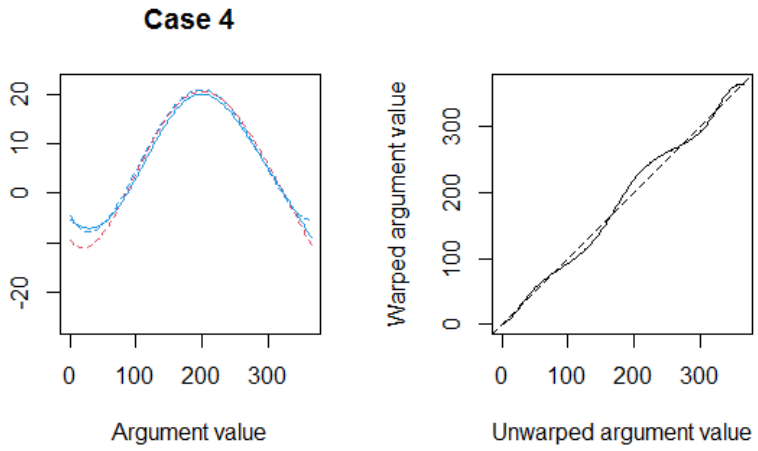


Figure 19: Tracés pour l'année 1964.

Chacun de ces tracés contient deux panneaux côte à côte. Le panneau de gauche contient la courbe non enregistrée (ligne bleue en pointillées), la cible (ligne rouge en pointillées) et la courbe enregistrée (ligne bleue continue). Le panneau de droite contient la fonction de déformation $h_i(t)$ et la fonction linéaire correspondante. Nous trouvons, dans les figures 18 et 19, ces tracés pour les années 1961 et 1964.

Chapitre II

CLUSTERING DE DONNEES FONCTIONNELLES

II. Clustering de données fonctionnelles

1 Généralité

En analyse de données, deux principales catégories d'approches existent:

- **les approches supervisées**: Elles cherchent à expliquer ou à prédire une variable dite variable réponse (notée y) en fonction d'autres variables dites variables explicatives (notées $X = (x_1, \ldots, x_p)$) qui peuvent être qualitatives ou quantitatives.On peut citer, par exemple, la regression (pour y quantitative) et la classification supervisée ou méthode de discrimination (pour y qualitative).
- **les approches non-supervisées**: Par opposition aux approches supervisées, elles n'ont pas de variables à expliquer. Elles cherchent, à partir uniquement des observations brutes, la structure sous-jacente ("vraie structure") des données. On peut citer comme exemple, la classification non-supervisée (*clustering*, en anglais), qui est l'objet de cette section et qui consiste au regroupement de données ayant des traits communs dans des *clusters*; ce qui est utile pour identifier des tendances dans les données.

Le clustering (ou encore classification automatique) est le processus qui cherche à partitionner un ensemble de données en clusters (groupes) de façon à regrouper les observations similaires et à séparer les observations non similaires. Les individus d'un même cluster sont le *"plus ressemblants"* possible et ceux de deux clusters différents sont le *"plus dissemblables"* possible. En d'autres termes, disposant d'un ensemble d'observations, on se fixe l'objectif d'identifier des sous-classes ε_i telsque pour tout $i \neq j$, le cluster ε_i est bien différent du cluster ε_j et tous les éléments de ε_i sont similaires. Le clustering consiste donc à grouper des observations similaires. Cependant, ces observations peuvent appartenir à un ou plusieurs clusters.
Suivant les informations dont on dispose sur les observations et/ou suivant les objectifs qu'on se fixe pour la classification, différentes approches de clustering sont proposées. Généralement, la distinction entre les différentes approches de clustering peut se faire au niveau de plusieurs points: leur objectif, le critère de regroupement, la nature des données à regrouper, la nature des clusters, la représentation des données, ...
En ce qui concerne les données fonctionnelles, le schéma ci-dessous (Figure II.1), proposé par Jacques et Preda (Jacques and Preda 2014), illustre une segmentation des différentes approches de clustering en quatre groupes. Voici, une description globale de ces quatre groupes d'approches de clustering:

- **Approches de clustering de données brutes**: Les méthodes de données brutes consistent à regrouper directement les courbes sur la base de leurs points d'évaluation. Pour ces méthodes, il n'est pas nécessaire de reconstruire la forme fonctionnelle des données puisque les fonctions sont, généralement, déjà observées à certains points d'observation discrets. De ce fait, ces approches ne prennent pas en compte la fonctionnalité des données (continuité, dérivées, etc.), ce qui peut conduire à une perte d'information. Différentes techniques de clustering reposant sur ces approches de données brutes sont proposées par Bouveyron et Brunet (Bouveyron and Brunet-Saumard 2014) et par Boullé (Boullé 2012).
- **Méthodes de filtrage**: ces méthodes approchent, dans un premier temps, les courbes en une base finie de fonctions (l'étape de filtrage) et effectuent, dans un second temps, un clustering en utilisant les coefficients d'expansion de base (les coefficients sont les paramètres fixes qui identifient les courbes). Pour la réduction de dimension, les bases classiques telles que les B-splines (Abraham et al. 2003) oubien l'analyse des composantes principales fonctionnelles (Peng and Müller 2008) peuvent être utilisées. Pour ces méthodes, les courbes sont représentées sous leur forme fonctionnelle.
- **Méthodes adaptatives**: Les méthodes adaptatives fonctionnent comme les méthodes de filtrage sauf qu'au lieu de traiter les coefficients d'expansion de base comme des paramètres fixes d'identification, ils sont considérés comme des variables aléatoires ayant une distribution de probabilité spécifique au cluster. ces méthodes considèrent que la représentation fonctionnelle des observations dépend des clusters et effectuent simultanément une réduction de dimension et un clustering. Ainsi, en fonction de son appartenance aux clusters, une courbe pourrait avoir différentes représentations. La plupart des techniques de regroupement adaptatif utilise la modélisation des coefficients d'expansion de base (James and Sugar 2003) ou a modélisation des scores de l'analyse des composantes principales fonctionnelles (Delaigle and Hall 2010).

- **Méthodes basées sur la distance**: Ce sont les méthodes les plus utilisées en clustering de courbes fonctionnelles. Elles reposent sur des algorithmes de clustering basés sur des distances spécifiques pour les données fonctionnelles. Les méthodes, diffèrentes par le choix de la distance d utilisée, peuvent être liées à des données brutes ou à des méthodes de filtrage. Pour deux courbes données, les mesures de distance utilisées par ces méthodes ont pour forme générale:

$$d_\ell(x_i, x_{i'}) = [\int_{\mathcal{T}} (x_i^{(\ell)}(t) - x_{i'}^{(\ell)}(t))^2 dt]^{\frac{1}{2}}$$

, où $x_i^{(\ell)}$ est le ℓ^{ieme} dérivé de x. Pour $\ell = 0$, on aura la distance L_2. Parmi les algorithmes les plus populaires de ces approches, nous pouvons citer le classification hiérarchique pour les données fonctionnelles (Ferraty and Vieu 2006) et l'algorithme des K-moyennes (Ieva et al. 2013).

Schéma proposé par Jacques et Préda

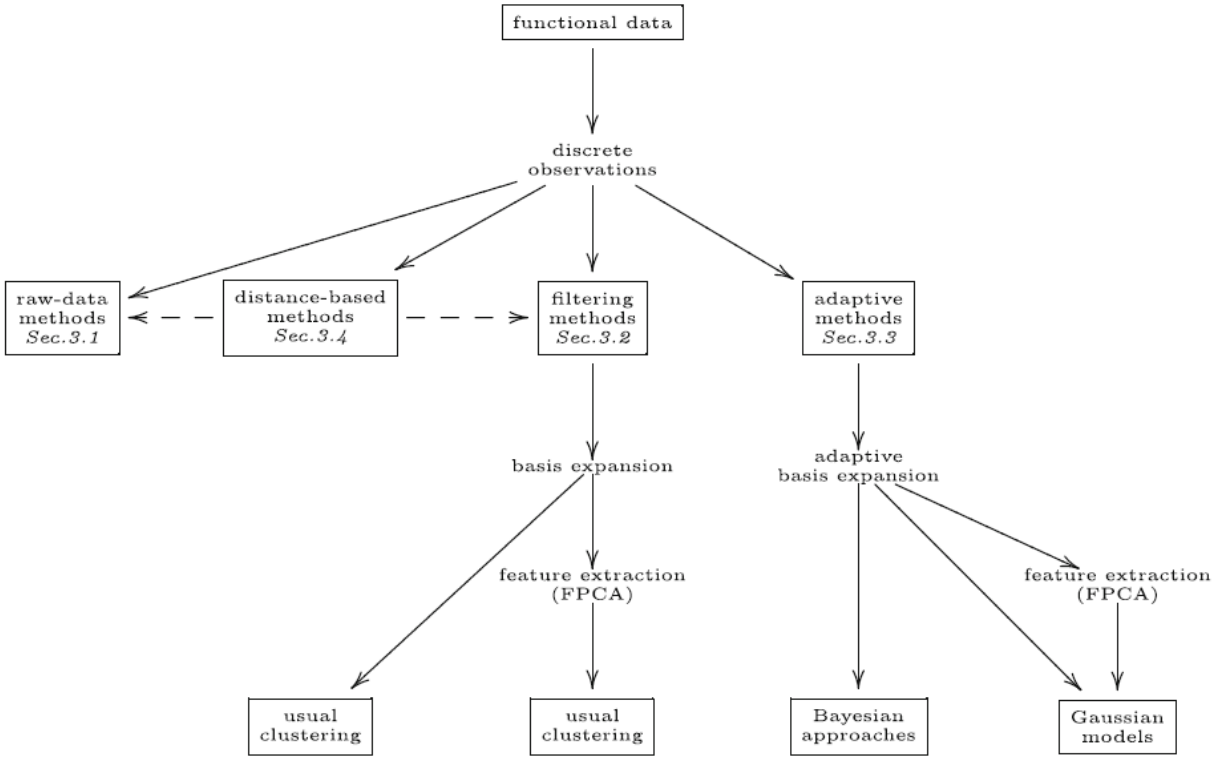


Figure 20: Segmentation des différentes méthodes de clustering des données fonctionnelles

Dans ce qui suit, nous allons présenter en détail une des plus populaires des méthodes basées sur la distance, à savoir la méthode des K-moyennes (*K-means* en anglais).

2 Algorithme K-moyennes

2.1 K-means pour données multivariées

Soient $\mathcal{E}$, l'ensemble des n observations, d une distance définie sur $\mathbb{R}^p$ et $X = (x_1, x_2, \ldots, x_p)$, l'ensemble des p variables décrivant ces données. Chaque observation est un vecteur de dimension p. Les données peuvent donc être représentées par une matrice de taille $n \times p$. L'objectif est de regrouper ces données en K groupes disjoints $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_K$ par la méthode K-means où chaque cluster est représenté par le centre ou la moyenne des points de données appartenant au cluster.
L'idée de base, derrière le clustering K-means, consiste à définir des clusters de sorte que la somme des inerties intra-cluster (ou encore, variation totale intra-cluster) soit minimisée:

$$\phi_{\mathcal{E}}(\mathcal{C}) = \sum_{j=1}^{K} \sum_{x_i \in \mathcal{E}_k} d^2(x_i, c_j)$$

avec $\mathcal{C} = \{c_j, 1 \leq j \leq K\}$ est l'ensemble des centres des K groupes. Plus la valeur de $\phi_{\mathcal{E}}(\mathcal{C})$ est petite, plus les groupes sont "compacts" autour de leurs centres, ce qui donc une meilleure qualité du partitionnement.

2.2 K-means pour données fonctionnelles

Considérons un ensemble de fonctions ou courbes $\mathcal{F}$, $f(s) : \mathbb{R} \longmapsto \mathbb{R}^d$. Pour pouvoir regrouper ces courbes en K cluster distincts $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$, l'algorithme K-mean a besoin d'un moyen de comparer le degré de similarité entre les différentes courbes. Chaque cluster doit être constitué de courbes similaires (ou encore de courbes proches). Suivant l'indice de similarité défini par Sangalli et all (2010), deux courbes qui se ressemblent (courbes proches), auront un indice de similarité proche de 1, alors que suivant la distance L_2 entre fonctions ou entre dérivées, deux courbes qui se ressemblent auront une distance proche de 0. Cet indice de similarité et ces distances L_2 peuvent être définis comme étant :

$$\begin{aligned}
\rho(\cdot,\cdot) &: \mathcal{F} \times \mathcal{F} \longmapsto \mathbb{R} \\
\rho(f_1,\ f_2) &= \frac{1}{d} \sum_{p=1}^{d} \frac{\int_{\mathbb{R}} f'_{1p}(s) f'_{2p}(s) ds}{\sqrt{\int_{\mathbb{R}} f'_{1p}(s)^2 ds} \sqrt{\int_{\mathbb{R}} f'_{2p}(s)^2 ds}} \quad \text{indice de Sangali et all.} \\
d_0(f_1,\ f_2) &= \sqrt{\frac{1}{d} \sum_{p=1}^{d} \int_{\mathbb{R}} \left(f_{1p}(s) - f_{2p}(s)\right)^2 ds} \quad \text{distance } L_2 \text{ entre fonctions} \\
d_1(f_1,\ f_2) &= \sqrt{\frac{1}{d} \sum_{p=1}^{d} \int_{\mathbb{R}} \left(f'_{1p}(s) - f'_{2p}(s)\right)^2 ds} \quad \text{distance } L_2 \text{ entre dérivées}
\end{aligned}$$

f_{ip} représente la $p^{ième}$ composante de $f_i = (f_{i1}, \ldots, f_{id})$ et f' indique la courbe dérivée de f. Suivant l'indice de Sangali et all, si $\rho(f_1,\ f_2) \approx 1$, alors les deux courbes f_1 et f_2 sont dites similaires. Et suivant la distance d_0 (respectivement d_1), les courbes f_1 et f_2 sont dites similaires si $d_0(f_1,\ f_2) \approx 0$ (respectivement $d_1(f_1,\ f_2) \approx 0$)
Appliquer l'algorithme K-means sur cet ensemble de fonctions, revient à les regrouper dans des clusters de telle sorte que la variation totale inter-cluster $\phi_{\mathcal{F}}(\mathcal{C})$ soit minimisée:

$$\begin{aligned}
\phi_{\mathcal{F}}(\mathcal{C}) &= \sum_{j=1}^{K} \sum_{f_i \in \mathcal{C}_k} \left[1 - \rho(f_i, c_j)\right] \quad \text{suivant l'indice de similarité} \\
\phi_{\mathcal{F}}(\mathcal{C}) &= \sum_{j=1}^{K} \sum_{f_i \in \mathcal{C}_k} \left[d_0(f_i, c_j)\right]^2 \quad \text{suivant la distance } L_2 \text{ entre fonctions} \\
\phi_{\mathcal{F}}(\mathcal{C}) &= \sum_{j=1}^{K} \sum_{f_i \in \mathcal{C}_k} \left[d_1(f_i, c_j)\right]^2 \quad \text{suivant la distance } L_2 \text{ entre dérivées premières}
\end{aligned}$$

2.3 Description de l'algorithme

Il existe plusieurs algorithmes K-means permettant de trouver le minimum de $\phi_{\mathcal{E}}(\mathcal{C})$. En prenant les exemples d'illustration du guide *Analytics Vidhya* ("K Means Clustering | K Means Clustering Algorithm in Python" 2019), nous pouvons décrire, de manière générale, l'algorithme K-means comme suit:

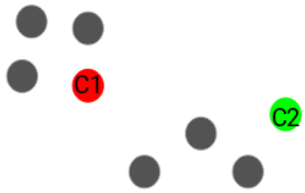
Considérons ces 8 points et appliquons l'algorithme K-means, ...

- **1. Initialisation du nombre K de clusters**: on commence par spécifier (choisir) le nombre K de clusters qui seront générés dans la solution finale

..., choisissons: k=2. C'est-à-dire, nous voulons avoir deux clusters, ...

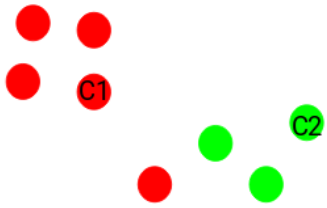
- **2. Sélection des centres**: l'algorithme sélectionne au hasard, dans l'ensemble des données (ou des courbes), k objets qui vont servir de centres initiaux pour les clusters

..., on choisit aléatoirement 2 centres de gravité pour les 2 clusters: les cercles rouge C1 et vert C2, ...



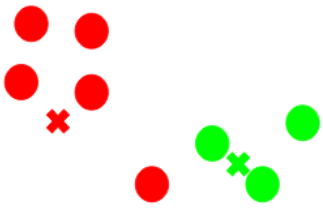
- **3. Attribution de cluster**: l'algorithme attribue à chacune des $N - k$ objets restants, le centre le plus proche; c'est-à-dire, en fonction de la distance d utilisée entre l'objet et le centre du cluster

..., nous pouvons voir ici que les points les plus proches du point rouge C1 sont affectés au cluster rouge alors que les points qui sont plus proches du point vert C2 sont affectés au cluster vert, ...



- **4. Mise à jour des centres**: l'algorithme calcule la nouvelle valeur moyenne de chaque cluster (le nouveau centre) et il vérifie à nouveau, pour chaque objet, s'il pourrait être plus proche d'un autre cluster. Ainsi, en utilisant les moyennes des clusters mis à jour, tous les objets sont réaffectés à un nouveau groupe.

..., les croix rouge et verte représentent les nouveaux centres, ...



- **5. Stabilisation des centres**: l'algorithme répète de manière itérative les étapes 3 et 4, jusqu'à ce que les attributions de cluster cessent de changer (atteinte de la convergence) ou encore, jusqu'à ce que les objets restent dans le même cluster. Cela revient à dire qu'on réduit itérativement la somme des inerties intra-cluster.

..., les deux clusters finaux obtenus.

En résumé, pour n courbes données $f_1(s), \ldots, f_n(s)$, l'objectif de l'algorithme K-means consiste à trouver K objets ($\mathcal{C} = c_1, \ldots, c_K$) appelés centres qui résument le mieux possible l'information contenue dans l'ensemble $\{f_i(s)\}_{i=1,\ldots,n}$. La détermination des K centres repose sur une minimisation de la quantité suivante (Serban and Wasserman 2005):

$$W_n(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} \min_{j=1,\ldots,K} \|f_i(s) - c_j\|^2$$

3 Choix du nombre de clusters

En termes de rappel, l'idée de base derrière le clustering, est de définir des clusters de telle sorte que la variation intra-cluster totale soit minimisée. Cette dernière mesure la compacité du clustering et nous voulons qu'elle soit aussi petite que possible. Pour certaines méthodes de clustering, telles que l'algorithme K-moyennes, il est exigé de spécifier le nombre de clusters à générer. Alors, l'on se demande comment choisir le nombre optimal de clusters ("le bon nombre") qui permettent cette minimisation de la variation intra-cluster totale. Plusieurs méthodes reposant sur l'optimisation d'un critère sur la variation intra-cluster totale, sont proposées. Parmi ces méthodes, figure une méthode, dite *du coude* et qui peut être décrite, pour un algorithme de clustering donné, comme suit:

- **Etape 1:** On applique l'algorithme de clustering pour différentes valeurs de K. Par exemple, on peut prendre une plage de valeurs de K.
- **Etape 2:** On calcule, pour chaque valeur de K, la valeur minimale de la variation intra-cluster totale, $\phi_\varepsilon(\mathcal{C})$.
- **Etape 3:** On trace la courbe des valeurs minimales atteintes par $\phi_\varepsilon(\mathcal{C})$ en fonction du nombre de clusters K : $\phi_\varepsilon(\mathcal{C}) = f(k)$.
- **Etape 4:** On cherhce alors l'emplacement d'un coude (genou) dans le graphique. Ce coude est généralement considéré comme un indicateur du nombre approprié de grappes.

Notons toutefois que si la plage de valeurs de K explorées est suffisamment large, il est fréquent d'obtenir plusieurs coudes sur le graphique. C'est ce qui fait la limite de cette méthode du coude.
Une autre méthode fréquemment utilisée pour le choix de ce nombre optimal de clusters est la méthode de la silhouette. Elle permet de sélectionner le K qui donne le clustering ayant le meilleur coefficient de silhouette. Ce coefficient se calcule d'abord à un point i d'un cluster donné, ensuite pour un cluster $\mathcal{E}_k$ donné et enfin pour l'ensemble du clustering $\mathcal{E}$.
Pour un point i donné, le coefficient de silhouette $s(i)$ permet d'évaluer si ce point appartient ou non au bon cluster. Il est défini comme étant la différence entre la distance moyenne avec les points du même cluster que i (la cohésion) et la distance moyenne avec les points des autres clusters voisins (la séparation). Si cette différence est négative, le point i est en moyenne plus proche du cluster voisin que du sien : il est donc mal classé. A l'inverse, si cette différence est positive, le point i est en moyenne plus proche de son groupe que du groupe voisin : il est donc bien classé.

$$s(i) = \frac{b(i) - a(i)}{\max\left(a(i),\ b(i)\right)}$$

où $a(i) = \frac{1}{|\mathcal{E}_k|-1} \sum_{j \in \mathcal{E}_k, j \neq i} d(j,i)$ est la distance moyenne du point i à tous les autres points du cluster $\mathcal{E}_k$ auquel il appartient;

$b(i) = \min_{l \neq k} \frac{1}{|\mathcal{E}_l|} \sum_{j \in \mathcal{E}_l} d(j,i)$ est la distance moyenne du point i au cluster le plus proche: autrement dit, c'est la plus petite valeur que pourrait prendre $a(i)$, si i était affecté à un autre cluster.
Ce coefficient $s(i)$ est donc compris entre -1 et 1. Plus $s(i)$ est proche de 1, plus l'assignation de i à son cluster $\mathcal{E}_k$ est satisfaisante. En d'autres termes, selon cette méthode:

- les points i avec un coefficient de Silhouette $s(i)$ proche de 1 sont bien regroupés,
- les points i avec un coefficient de Silhouette $s(i)$ qui tourne autour de 0, sont sur (ou sont très proche de) la limite de décision entre deux clusters voisins,
- les points i avec un coefficient de Silhouette $s(i)$ proche de -1 sont mal regroupés.

Pour un cluster $\mathcal{E}_k$ donné, le coefficient de Silhouette est la moyenne des coefficients de l'ensemble des points du cluster:

$$S_{\mathcal{E}_k} = \frac{1}{|\mathcal{E}_k|} \sum_{i \in \mathcal{E}_k} s(i)$$

Pour évaluer un clustering, on peut calculer son coefficient de silhouette moyen qui est la moyenne des $S_{\mathcal{E}_k}$ (donc la moyenne des coefficients de Silhouette de tous les points). Il est définit ainsi:

$$S_{\mathcal{E}} = \frac{1}{K} \sum_{k=1}^{K} S_{\mathcal{E}_k} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\mathcal{E}_k|} \sum_{i \in \mathcal{E}_k} s(i)$$

4 Codes R pour K-means

Sous **R**, la fonction standard dédiée à l'algorithme K-means est `kmeans()` du package `stats`. Ses principaux arguments peuvent être décrits comme suit:

Table 7: Description des arguments de la fonction kmeans

Arguments	Description
`x`	Jeu de données
`centers`	Le nombre K de clusters ou une liste de centres
`iter.max`	Le nombre maximum d'itérations autorisées. La valeur par défaut est de 10
`nstart`	Le nombre d'ensembles aléatoires qu'il faut choisir lorsque `centers` est un nombre.

Pour l'argument `centers`, si un nombre est fourni, alors un ensemble d'objet (distincts) dans `x` est choisi aléatoirement comme centres initiaux. Afin d'obtenir un résultat plus stable, il est recommandé de choisir de grandes valeurs pour l'argument `nstart` (entre 25 et 50). Par exemple, pour `nstart = 25`, l'agorithme fera 25 assignations aléatoires différentes et ensuite sélectionnera les meilleurs résultats correspondant à celui qui a la plus faible variation au sein du cluster.
Les données fournies dans `x` sont partitionnées en groupes de telle sorte que la somme des carrés des points aux centres de cluster assignés soit minimisée. La fonction `kmeans()` renvoie comme valeur un objet de classe `kmeans`. Il s'agit d'une liste d'éléments dont les principaux sont:

Table 9: Description des éléments renvoyés par la fonction kmeans

Arguments	Description
`cluster`	Le vecteur d'entiers $(1, \ldots, K)$ indiquant le cluster auquel chaque objet est alloué
`centers`	La matrice des centres de clusters
`totss`	La somme totale des carrés
`withinss`	Le vecteur de la somme des carrés intra-cluster, un composant par cluster.

Arguments	Description
`tot.withinss`	Somme totale des carrés intra-cluster, i.e $\sum$`(withinss)`
`betweenss`	La somme des carrés entre les grappes, i.e `totss - tot.withinss`
`size`	Le nombre d'objets que contient chaque cluster

Pour la représentation graphique et l'estimation du nombre optimal de clusters, nous ferons recours à quelques fonctions du package `factoextra` en utilisant l'objet renvoyé par la fonction `kmeans()`. Principalement, nous utiliserons:

- La fonction `fviz_cluster()` qui permet de visualiser facilement les clusters. Comme arguements principaux, elle prend les résultats de k-means (dans l'argument `object`) et les données qui ont été utilisées pour le clustering (dans l'argument `data`).
- La fonction `fviz_nbclust()` qui fournit une solution pratique pour estimer le nombre optimal de clusters. Elle détermine et visualise le nombre optimal de clusters en utilisant différentes méthodes qui peuvent être spécifier lors de son appel. Par exemples, pour la méthode du coude, on a `method ="wss"` et pour la méthode de Silhouette, `method ="silhouette"`.

Egalement, nous pouvons utiliser la fonction `kma.compare()` du package `fdakma` pour le K-means fonctionnel. La section 4 du chapitre III fournit une description détaillée de cette fonction.

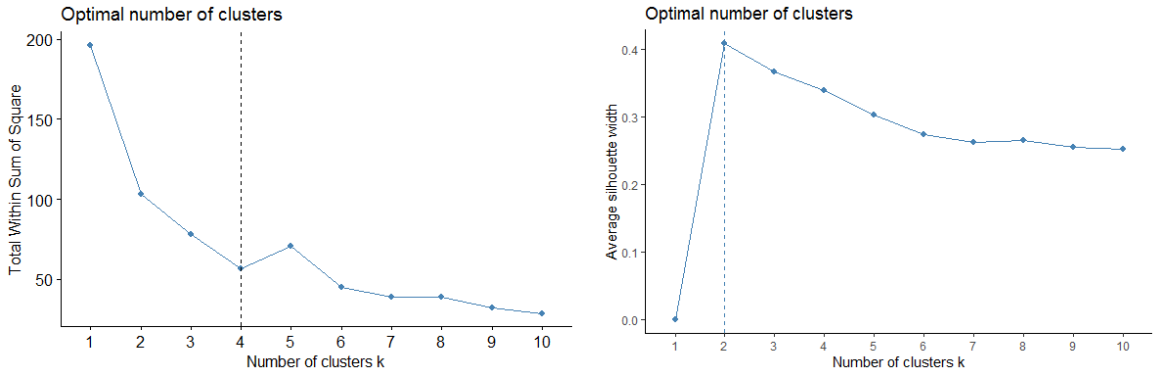
5 Exemples pratiques

5.1 Exemple sur données multivariées

Considérons l'ensemble de données `USArrests` dans **R**. Il contient des statistiques sur les arrestations pour 100 000 habitants pour voies de fait, meurtres et viols dans chacun des 50 États américains en 1973. Le pourcentage de la population vivant dans les zones urbaines est également indiqué. Ainsi, nous avons une matrice 50×4 ($n = 50$ et $p = 4$).
Pour effectuer un clustering K-means sous R, en général, les données doivent être préparées comme suit:

- Les lignes représentent les observations et les colonnes, les variables.
- Toutes les valeurs manquantes dans les données doivent être supprimées ou estimées.
- Les données doivent être normalisées (c'est-à-dire mises à l'échelle) pour rendre les variables comparables. La fonction `scale()` sera utilisée.

Après la préparation des données, nous pouvons choisir le nombre de cluster souhaité. Pour cela, nous ferons recours à la fonction `fviz_nbclust()`. Le graphe à droite de la figure ci-dessous suggère $K = 4$ comme nombre optimal de clusters puisqu'il semble être la courbure du coude. Par contre la méthode Silhouette représentée par le graphe à gauche suggère de prendre $K = 2$. Par la suite, nous optons pour $K = 4$.



Méthode du coude *Méthode Silhouette*

Figure 21: Détermination du nombre optimal de clusters.

Nous pouvons maintenant exécuter la fonction `kmeans()` avec $K = 4$ clusters. L'image 1 dans l'annexe montre l'impression des résultats obtenus suite à l'exécution de cette fonction. D'abord, nous pouvons voir que le regroupement s'est fait en quatre groupes ayant pour tailles respectives 8, 13, 16 et 13. Ensuite, nous voyons les centres de cluster (les moyennes) pour les quatres groupes à travers les quatres variables (Murder, Assault, UrbanPop, Rape). Enfin, nous obtenons l'affectation à un cluster de chaque observation. Par exemple, l'Alabama a été affecté au cluster 1, l'Alaska au cluster 2, le Delaware au cluster 3, l'Iowa au cluster 4, etc.
Comme nous l'avons mentionné dans la partie précédente, les résultats peuvent être visualisés à l'aide de la foncion `fviz_cluster()`. La figure ci-dessous fournit une belle visualisation des clusters.

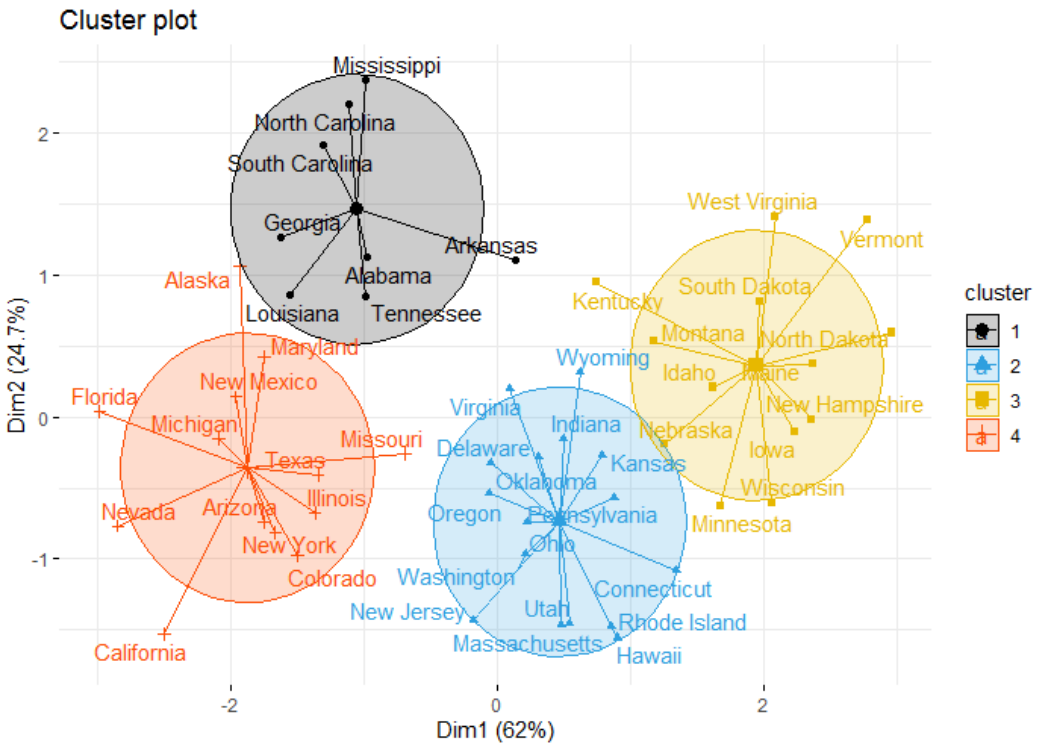


Figure 22: Visualisation des clusters

5.2 Exemple sur données fonctionnelles

Considèrons les données `kma.data` du package `fdakma`. Ces données sont constituées de trois éléments:

- `x`: valeurs d'abscisses où chaque fonction est évaluée
- `y0`: évaluations des fonctions sur l'axe des abscisses
- `y1`: évaluations des dérivées premières des fonctions sur l'axe des abscisses

Le graphique 23 nous montre l'ensemble des 30 courbes de cette base.

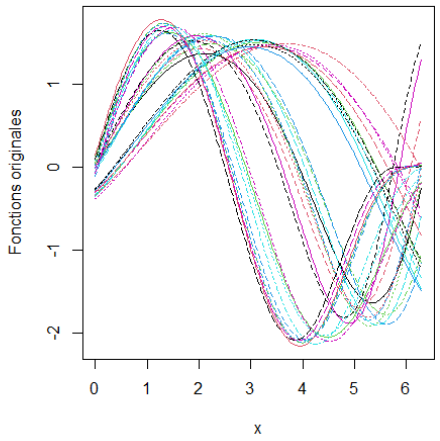


Figure 23: Ensemble des 30 courbes.

A l'aide de la fonction `kma.compare()` du package `fdakma` (voir chapitre III, section 4), nous avons effectué le clustering avec $K = \{2,\ 3\}$.
La figure 24 montre les similarités moyennes correspondantes (points oranges). On note qu'un alignement pour $k = 2$ entraine une forte augmentation des similarités par rapport aux similarités des courbes initiales. Notons que les similarités obtenues pour $k = 2$ sont déjà très importantes et que l'utilisation de $k = 3$ groupe d'amplitude n'est pas récompensée par un gain supplémentaire de similarité. Ainsi, la procédure de regroupement suggère que $k = 2$ groupes d'amplitude est largement suffisant pour saisir la similarité maximale des courbes.

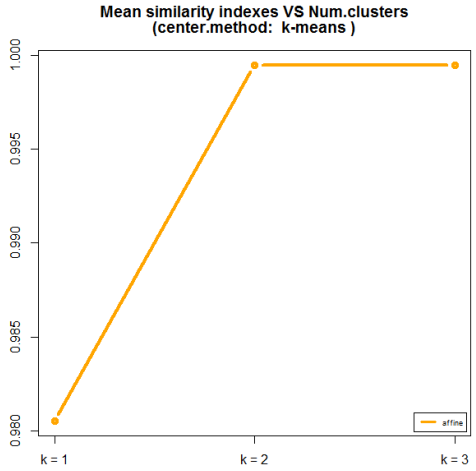
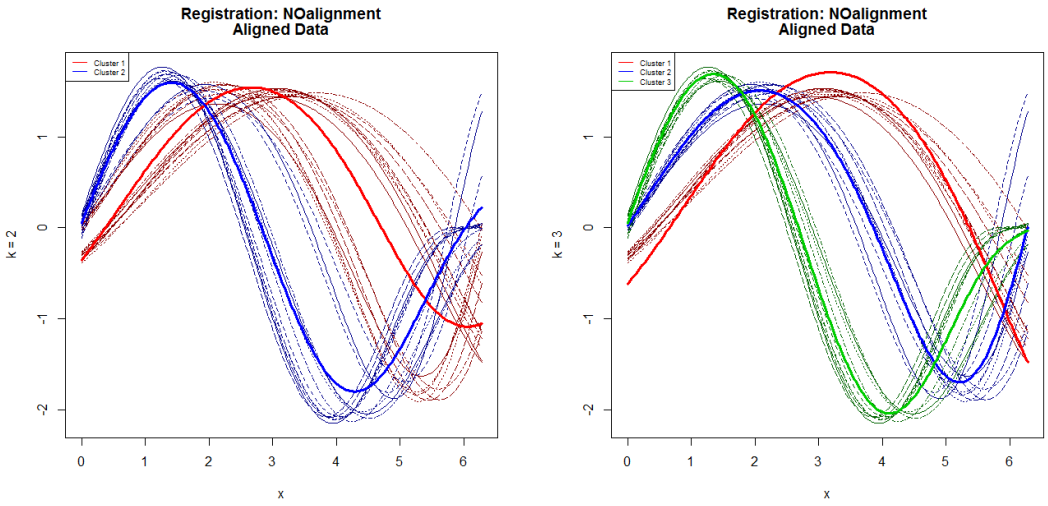


Figure 24: Moyennes des indices de similarité en fonction du nombre de clusters.

Nous obtenons le graphique de la figure 25. Chaque courbe est colorée suivant la couleur du cluster auquel elle appartient (par exemple, toutes les courbes rouges appartiennent au cluster rouge).



Clustering avec $k = 2$ *Clustering avec $k = 3$*

Figure 25: Clustering des courbes

Chapitre III

K-MOYENNES AVEC ALIGNEMENT GLOBAL DES COURBES

III. K-Moyennes avec alignement global des courbes

Comme nous l'avons souligné à la section I.4, le problème de variabilité des données fonctionnelles est souvent rencontré dans les études d'analyse des données fonctionnelles (FDA). Afin d'éliminer (ou de réduire considérablement) la variabilité de phase tout en préservant la forme et l'amplitude des courbes, on fait recours à l'alignement des courbes.
De nombreuses méthodes d'alignement des courbes ont été poposées. Nous pouvons en citer, par exemples, l'alignement par des points de repère (landmark), l'alignement continu, ... Dans cette section, nous allons décrire une procédure d'alignement nommée *"k-mean alignment"* (K-moyenne avec alignement) qui effectue simultanément l'alignement et le regroupement des courbes en k groupes (Sangalli et al. 2010).

1 Généralité

D'une manière générale, la méthode *k-mean alignment* peut être considérée comme étant un alignement continu de courbes avec k courbes modèles; ou encore, d'une manière équivalente, comme un clustering suivant l'algorithme des K-means qui autorise la déformation temporelle. Si le nombre de clusters est égal à l'unité ($k = 1$), alors la méthode se résume par un alignement selon la méthode des Procrustres (*Section I.4.1*). D'un autre côté, si aucun alignement n'est autorisé (pas de déformation temporelle), elle se résume en un clustering K-means pour données fonctionnelles (*Section II.2*).
Considérons un ensemble de courbes $\mathcal{C}$ (éventuellement multidimensionnelles) $c(s) : \mathbb{R} \longmapsto \mathbb{R}^d$. Aligner $c_1 \in \mathcal{C}$ sur $c_2 \in \mathcal{C}$, revient à chercher une fonction de déformation $h(s) : \mathbb{R} \longmapsto \mathbb{R}$ du paramètre d'abscisse s qui fait que les deux courbes $(c_1 \circ h)$ et c_2 soient les plus semblables possibles. D'où la nécessité de spécifier, non seulement un indice de similarité $\rho(.,.) : \mathcal{C} \times \mathcal{C} \longmapsto \mathbb{R}$ qui traduit la similarité entre deux courbes, mais aussi une classe $\mathcal{H}$ de fonctions de déformation h (tellesque $c \circ h \in \mathcal{C}$ pour tout $c \in \mathcal{C}$ et $h \in \mathcal{H}$) qui indique les transformations autorisées sur l'abscisse s.
Ainsi, aligner $c_1 \in \mathcal{C}$ sur $c_2 \in \mathcal{C}$ selon le couple $(\rho, \mathcal{H})$, signifie chercher $h^* \in \mathcal{H}$ qui maximise $\rho(c_1 \circ h,\ c_2)$. En guise d'éxigences minimales de connaissance, ce couple $(\rho, \mathcal{H})$ doit vérifier les propriétés suivantes:

- ρ est borné avec une valeur maximale qui vaut 1 et est:
  - *réflexive:* $\rho(c, c) = 1,\ \forall\ c \in \mathcal{C}$
  - *symétrique:* $\rho(c_1, c_2) = \rho(c_2, c_1),\ \forall\ c_1, c_2 \in \mathcal{C}$
  - *transitive:* $[\rho(c_1, c_2) = 1 \text{ et } \rho(c_2, c_3) = 1] \Longrightarrow \rho(c_1, c_3) = 1,\ \forall\ c_1, c_2, c_3 \in \mathcal{C}$
- la classe $\mathcal{H}$ des fonctions de déformation est un espace vectoriel convexe et possède une structure de groupe pour la composition des fonctions $\circ$.
- ρ et $\mathcal{H}$ sont cohérents au sens que si deux courbes c_1 et c_2 sont déformées simultanément selon la même fonction $h \in \mathcal{H}$, alors leur indice de similarité ne change pas: $\rho(c_1, c_2) = \rho(c_1 \circ h,\ c_2 \circ h),\ \forall\ h \in \mathcal{H}$
- La similarité qui peut être obtenue en alignant c_1 sur c_2 est la même que celle qui peut être obtenue en alignant c_2 sur c_1: pour tout h_1 et $h_2 \in \mathcal{H}$,

$$\rho(c_1 \circ h_1,\ c_2 \circ h_2) = \rho(c_1 \circ h_1 \circ h_2^{-1},\ c_2) = \rho(c_1,\ c_2 \circ h_2 \circ h_1^{-1})$$

- l'indice de similarité entre deux courbes n'est pas affecté par des transformations affines strictement croissantes d'une ou de plusieurs composantes des courbes: $\rho(t_1(c_1),\ t_2(c_2)) = \rho(c_1,\ c_2)$

L'indice de similarité choisi par Sangali et all. est défini par:

$$\rho(c_1,\ c_2) = \frac{1}{d} \sum_{p=1}^{d} \frac{\int_{\mathbb{R}} c'_{1p}(s) c'_{2p}(s) ds}{\sqrt{\int_{\mathbb{R}} c'_{1p}(s)^2 ds} \sqrt{\int_{\mathbb{R}} c'_{2p}(s)^2 ds}}$$

avec c_{ip} la $p^{ième}$ composante de $c_i = (c_{i1}, \ldots, c_{id})$. Si $\rho(c_1,\ c_2) = 1$, alors les deux courbes c_1 et c_2 sont dites similaires. Le choix de ρ s'accompagne du choix de la classe $\mathcal{H}$ des fonctions de déformation suivant:

$$\mathcal{H} = \{h : h(s) = ms + q, \quad (m,\ q) \in \mathbb{R}^+ \times \mathbb{R}\}$$

2 Alignement sur les k modèles

Considérons un ensemble de N courbes $\{c_1, \ldots, c_N\} \subset \mathcal{C}$ mal alignées. Comment regrouper et aligner ces courbes sur un ensemble de K courbes modèles $\underline{\varphi} = \{\varphi_1, \ldots, \varphi_K\} \subset \mathcal{C}$?
Pour répondre à cette question, Sangali et all. définissent, dans un premier temps, un domaine d'attraction pour chaque courbe modèle $\varphi_j \in \underline{\varphi}$:

$$\Delta_j(\underline{\varphi}) = \{c \in \mathcal{C} : \sup_{h \in \mathcal{H}} \rho(\varphi_j,\ c \circ h) \geq \sup_{h \in \mathcal{H}} \rho(\varphi_r,\ c \circ h),\ \forall\ r \neq j\}, \quad j = 1, \ldots, K$$

Et dans un second temps, ils définissent la fonction d'étiquetage (*"labeling function"*) qui indique le modèle pour lequel une courbe peut être le mieux alignée et le cluster auquel elle doit être affectée:

$$\lambda(\underline{\varphi}, c) = \min\{r : c \in \Delta_r(\underline{\varphi})\}$$

Ainsi, $\lambda(\underline{\varphi}, c) = j$ signifie que l'indice de similarité obtenu en alignant la courbe c sur la courbe modèle φ_j est meilleur (plus grand) que tout autre indice de similarité obtenu en alignant c sur toute autre courbe modèle $\varphi_{j'}$ avec $j' \neq j$. Egalement, la courbe c doit être assignée au $j^{ième}$ cluster.
Donc, si les K courbes modèles $\underline{\varphi} = \{\varphi_1, \ldots, \varphi_K\}$ sont connues, alors l'alignement et le regroupement des N courbes $\{c_1, \ldots, c_N\}$ par rapport aux courbes modèles $\underline{\varphi}$ signifie simplement une attribution de la courbe c_i, au cluster $\lambda(\underline{\varphi}, c_i)$ et son alignement sur la courbe modèle correspondante $\varphi_{\lambda(\underline{\varphi}, c_i)}$, pour tout $i = 1, \ldots, N$.
Dans le cas où les courbes modèles ne sont pas connues, les auteurs proposent de les estimer en résolvant le problème d'optimisation suivant:

- (i) Trouver $\underline{\varphi} = \{\varphi_1, \ldots, \varphi_K\} \subset \mathcal{C}$ et $\underline{h} = \{h_1, \ldots, h_N\} \subset \mathcal{H}$ telsque:

$$\frac{1}{N} \sum_{i=1}^{N} \rho(\varphi_{\lambda(\underline{\varphi}, c_i)},\ c_i \circ h_i) \geq \frac{1}{N} \sum_{i=1}^{N} \rho(\psi_{\lambda(\underline{\psi}, c_i)},\ c_i \circ f_i),$$

  pour tout autre ensemble de courbes modèles $\underline{\psi} = \{\psi_1, \ldots, \psi_K\} \subset \mathcal{C}$ et tout autre ensemble de fonctions de déformation $\underline{f} = \{f_1, \ldots, f_N\} \subset \mathcal{H}$.

- (ii) Pour tout $i = 1, \ldots, N$, assigner la courbe c_i au cluster $\lambda(\underline{\varphi}, c_i)$ et l'aligner sur la courbe modèle correspondante $\varphi_{\lambda(\underline{\varphi}, c_i)}$ en utilisant la fonction de déformation h_i.

Vu que le problème d'optimisation non linéaire en (i) n'est pas analytiquement résoluble dans son une généralité complète, les auteurs ont proposé de traiter simultanément les points (i) et (ii) au moyen d'un algorithme *k-mean alignment*.

3 Algorithme k-mean alignment

L'algorithme *k-mean alignment* itère les étapes d'identification des courbes modèles et les étapes d'alignement et de regroupement. En effet, après initialisation de l'algorithme avec des courbes modèles initiales, on a:

- dans l'étape d'identification, les K courbes modèles seront estimées à partir des K groupes identifiés lors de l'étape d'alignement et de regroupement précédente;
- dans l'étape d'alignement et de regroupement, les N courbes seront alignées sur les K modèles obtenus lors de l'étape d'identification et chacune des courbes sera affectée à un des K clusters.

De manière plus abstraite, soient $\underline{\varphi}_{[q-1]} = \{\varphi_{1[q-1]}, \ldots, \varphi_{K[q-1]}\}$ l'ensemble des K courbes modèles obtenues après l'itération $q-1$ et $\{c_{1[q-1]}, \ldots, c_{N[q-1]}\}$ les N courbes alignées et groupées suivant $\underline{\varphi}_{[q-1]}$. Alors, à la $q^{ième}$ itération, l'algorithme effectue les étapes suivantes:

- Identification des modèles: Pour $j = 1, \ldots, K$, la courbe modèle $\varphi_{j[q]}$ est estimée en utilisant toutes les courbes affectées au $j^{ième}$ cluster lors de l'itération $q-1$. Il s'agit de toutes les courbes $c_{i[q-1]}$ qui vérifient $\lambda(\underline{\varphi}_{[q-1]},\ c_{i[q-1]}) = j$. Idéalement, cette courbe $\varphi_{j[q]}$ doit être estimée comme étant la courbe $\varphi \in \mathcal{C}$ qui maximise la similarité totale:

$$\sum_{i:\lambda(\underline{\varphi}_{[q-1]},\ c_{i[q]})=j} \rho(\varphi, c_{i[q]})$$

- Alignement et affectation: L'ensemble des courbes $\{c_{1[q-1]}, \ldots, c_{N[q-1]}\}$ sont alignées et affctées suivant les courbes modèles $\underline{\varphi}_{[q]} = \{\varphi_{1[q]}, \ldots, \varphi_{K[q]}\}$. C'est-à-dire, pour $i = 1, \ldots, N$, la $i^{ième}$ courbe $c_{i[q-1]}$ est alignée sur $\varphi_{\lambda(\underline{\varphi}_{[q]},\ c_{i[q-1]})}$ et la courbe alignée $\tilde{c}_{i[q]} = c_{i[q-1]} \circ h_{i[q]}$ est affectée au cluster $\lambda(\underline{\varphi}_{[q]},\ c_{i[q-1]} \circ h_{i[q]})$
- Normalisation: Pour $j = 1, \ldots, K$, toutes les $N_{j[q]}$ courbes $\tilde{c}_{i[q]}$ affectées au cluster j sont déformées via la focntion de déformation $(\bar{h}_{j[q]})^{-1}$. Ce qui fait que: $c_{i[q]} = \tilde{c}_{i[q]} \circ (\bar{h}_{j[q]})^{-1} = c_{i[q-1]} \circ h_{i[q]} \circ (\bar{h}_{j[q]})^{-1}$ avec:

  $$\bar{h}_{j[q]} = \frac{1}{N_{j[q]}} \sum_{i:\lambda(\underline{\varphi}_{[q]},\ \tilde{c}_{i[q]})=j} h_{i[q]}$$

  Ainsi, à chaque itération, la déformation moyenne subie par toutes les courbes affectées au cluster j est la transformation identité $h(s) = s$.

Cette étape de normalisation permet de sélectionner, parmi toutes les solutions candidates au problème d'optimisation, celle qui laisse les emplacements moyens des clusters inchangés, évitant ainsi la dérive des clusters ou la dérive globale de l'ensemble des courbes. Elle préserve la structure de clustering choisie lors de l'étape d'affectation et d'alignement, c'est-à-dire: $\lambda(\underline{\varphi}_{[q]},\ \tilde{c}_{i[q]}) = \lambda(\underline{\varphi}_{[q]},\ c_{i[q]})$, pour tout $i = 1, \ldots, N$ (Sangalli et al. 2010).

4 Codes R

Sous **R**, le package `fdakma` contient des fonctions qui permettent d'effectuer conjointement le clustering et l'alignement des courbes. Parmi ces fonctions, nous avons la fonction `kma` qui s'effectue en précisant le nombre de clusters et la méthode d'alignement. La fonction `kma.compare` permet d'exécuter `kma` avec différents nombres de clusters et différentes méthodes d'alignement.
La fonction `kma.compare` prend principalement comme arguments:

- `x` : Valeurs d'abscisse où chaque fonction est évaluée
- `y0` : évaluations de l'ensemble des fonctions sur l'axe des abscisses `x`. Cet argument `y0` est obligatoire si `similarity.method` concerne des fonctions d'origines.
- `y1` : évaluations des dérivées premières des fonctions sur l'axe des abscisses `x`. Cet argument `y1` est obligatoire si `similarity.method` concerne les dérivées premières des fonctions d'origines.
- `n.clust` : vecteur contenant les valeurs `K` du nombre de clusters
- `warping.method` : vecteur contenant les différentes méthodes d'alignement (`NOalignment`, `affine`, `shift`, `dilation`)
- `similarity.method` : caractère indiquant la distance choisie pour la mesure de similarité (`d0.pearson`, `d1.pearson`, `d0.L2`, `d1.L2`, `d0.centered`, `d1.centered`)
- `center.method` : caractère indiquant la méthode pour le clustering (`k-means`, `k-medoids`)

La fonction `kma.compare` renvoie une liste contenant le résultat (`Result.NOalignment`, `Result.affine`, `Result.shift` et `Result.dilation`) et les indices de similarité moyens (`mean.similarity.NOalignment`, `mean.similarity.affine`, `mean.similarity.shift` et `mean.similarity.dilation`) pour toutes les méthodes d'alignement choisies. Pour les méthodes non choisies, le résultat sera `NULL`.

5 Exemple pratique

Reprenons les données `kma.data` du package `fdakma`. Cet ensemble de données fonctionnelles affiche à la fois l'amplitude et la variabilité de phase.
La figure 26 montre une représentation des 30 courbes initiales que nous allons regrouper et aligner suivant l'algorithme *K-means alignment*. L'idée est de choisir le bon nombre de clusters k suivant les similarités entre les différentes courbes. En d'autres termes, on cherche le k qui permet d'obtenir la similarité maximale des courbes.

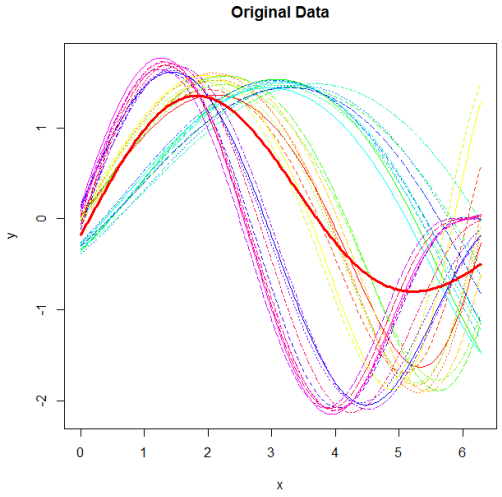
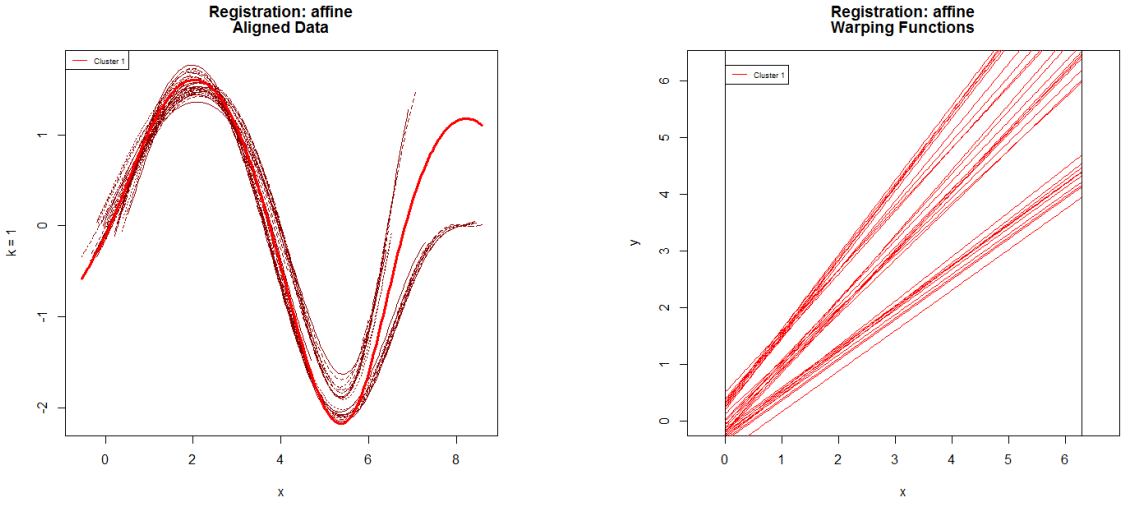


Figure 26: Fonctions initiales.

La figure 27 montre les courbes alignées et les fonctions de déformation résultantes de l'alignement k-means des courbes avec $k = \{1, 2, 3\}$.
L'alignement pour $k = 1$ ne donne pas une image claire d'un seul bloc recherché dans la variabilité d'amplitude, puisque les courbes alignées semblent toujours être séparées en deux groupes. En plus, on constate deux ou trois blocs dans la variabilité de phase (graphique des fonctions de déformation). Egalement, pour $k = 3$, l'alignement fournit clairement les trois groupes d'amplitude; par contre, dans la phase, il semble y avoir trois groupes différents.
Une meilleure image est donnée par l'alignement pour $k = 2$ avec deux groupes d'amplitude visiblement bien séparés et aucun regroupement en phase.



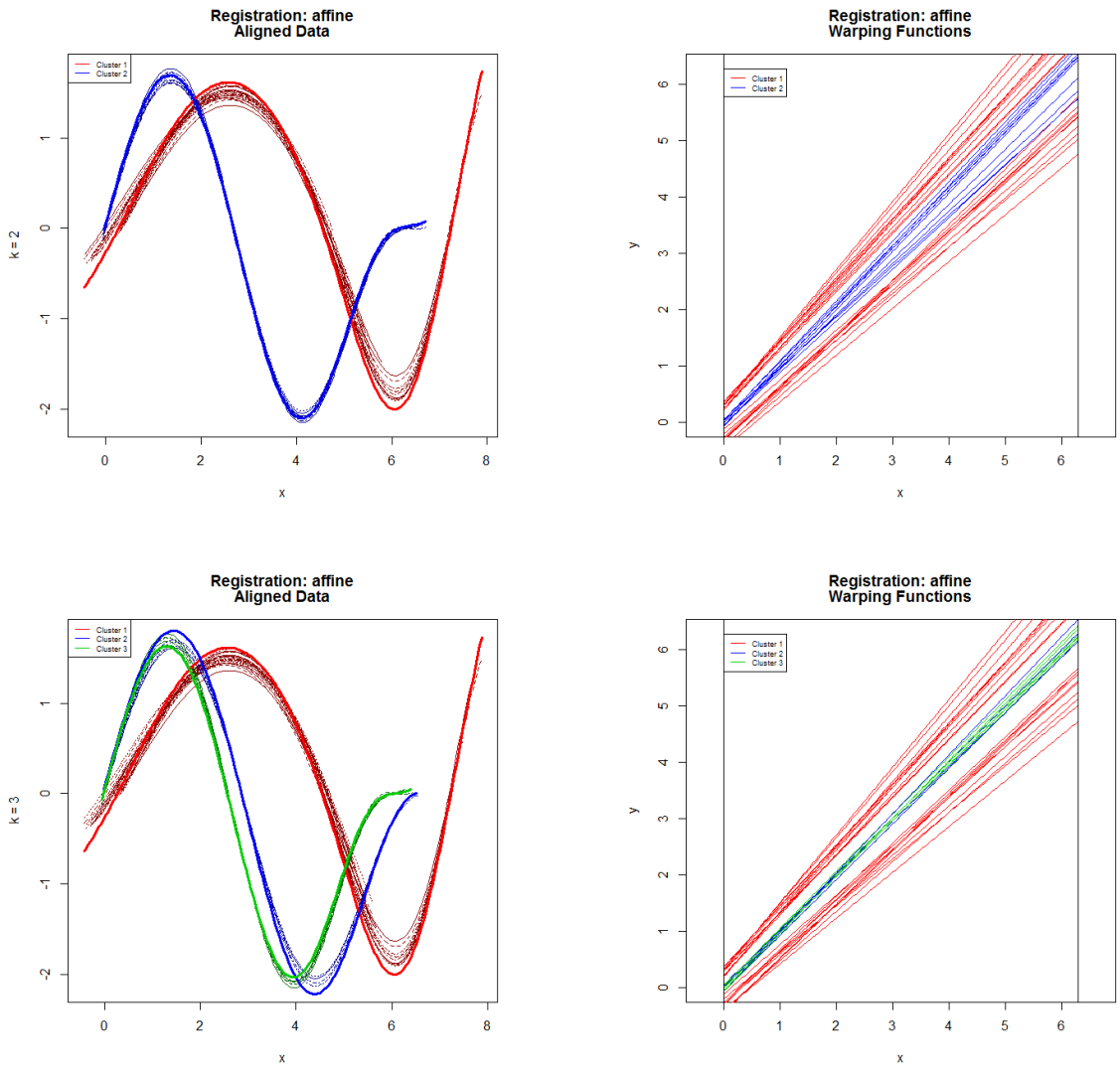


Figure 27: K-means alignés et fonctions de déformation correspondantes

Egalement, nous observons que, suivant le nombre de clusters, l'algorithme permet de bien distinguer la variabilité de l'amplitude et de la phase.

Chapitre IV

K-Moyennes probabiliste avec alignement local

IV. K-Moyennes probabiliste avec alignement local

1 Généralité

Dans les chapitres précédents, nous avons vu, de manière générale, différentes méthodes d'analyse de données fonctionnelles. D'abord, étant donné un ensemble de données brutes (sous forme discrétisées), nous avons vu comment les transformer à des données sous leur forme continue (c'est le lissage de données) en utilisant différentes méthodes: lissages par moindres carrés avec et sans pénalité. Ensuite, voulant analyser ces données fonctionnelles via des statistiques sommaires telles que la moyenne ou la covariance, nous avons effectué, afin de contourner le problème potentiel de variabilités (en amplitude et en phase) que peuvent poser ces données fonctionnelles, des transformations de l'argument t selon deux différentes méthodes d'alignement ou d'enregistrement: alignement par point de repère et alignement continu. Enfin, cherchant à touver la structure sous-jacente des données, le clustering, c'est-à-dire le regroupement des données ayant les mèmes caractéristiques dans des clusters, a été effectué afin d'identifier certaines tendances dans ces données. L'algorithme K-means a été utilisé pour ce regroupement.Etant donné que les courbes sont souvent mal alignées, nous avons également vu, dans le chapitre 3, l'algorithme *"K-means alignment"* qui effectue simultanément l'alignement et le clustering des courbes.

Cependant, toutes ces méthodes d'analyse de données fonctionnelles traitent les courbes dans leur intégralité. Aucune d'entre elles ne considèrent les courbes localement. En d'autres termes, la structure locale des courbes n'est pas tenue en compte dans ces différentes méthodes. C'est ainsi que Cremona et Chiaromonte ont développé une nouvelle méthode d'analyse des données fonctionnelles qui tient compte de la structure locale de courbes. Une méthode qui permet de regrouper localement les courbes et de découvrir des motifs fonctionnels, c'est-à-dire des "formes" typiques, dans les courbes comme le montre la figure ci-dessous. Dans les lignes qui vont suivre, nous allons décrire cette nouvelle procédure appelée ProbKMA *"Probabilistic K-mean with local alignment for clustering and motif discovery in functional data"* (Cremona and Chiaromonte 2020).

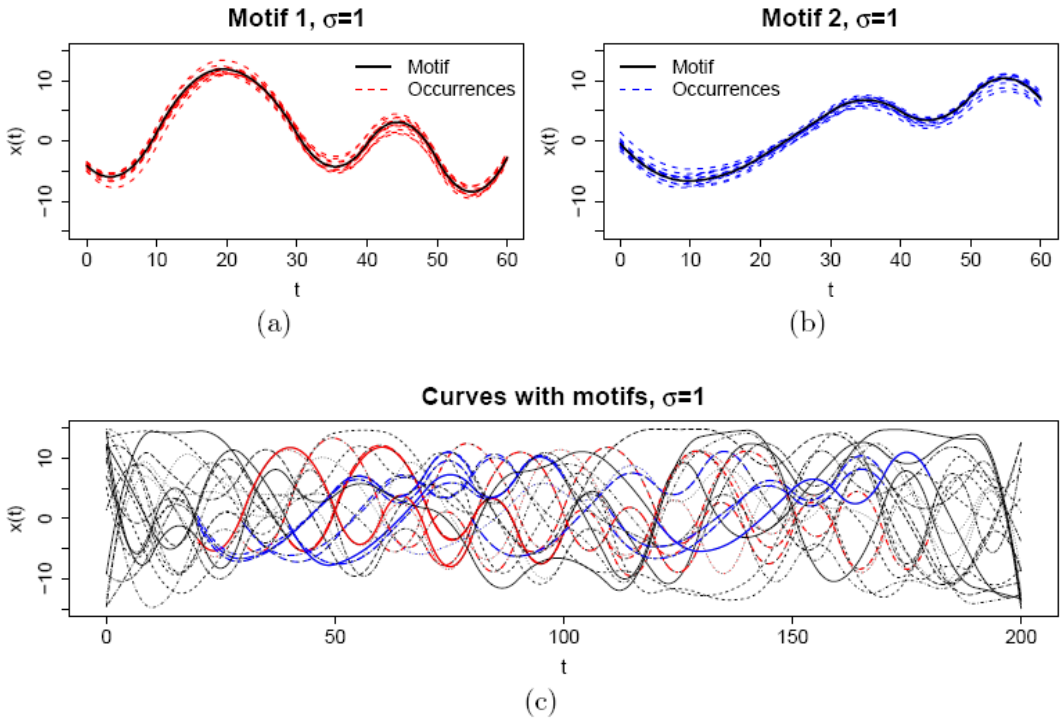


Figure 28: Découverte de motifs fonctionnels

Dans cette figure, (a) et (b) montrent deux motifs fonctionnels représentés par les lignes continues noires.

Pour chaque motif, les occurences alignées sont représentées par les pointillées (rouges et bleues). Le graphique (c) montre 20 courbes intégrant les occurrences des deux motifs (parties rouges et bleues).

2 Problème d'optimisation

Etant donné un ensemble de N courbes, $X_i : \mathbb{R} \longrightarrow \mathbb{R}^d$, $i = 1, \ldots, N$, nous cherchons à identifier des motifs qui font que les courbes soient localement très similaires par rapport à une distance $d(\cdot, \cdot)$ donnée. En termes mathématiques, il s'agit d'identifier K centres de clusters de longueurs potentiellement différentes $c_1, \ldots, c_K$, définis dans les intervalles $[0,\ c_k]$, $k = 1, \ldots, K$:

$$v_k : [0,\ c_k] \longrightarrow \mathbb{R}^d$$

Ainsi, chaque courbe X_i est alignée sur chaque centre de cluster v_k afin de minimiser leur distance. Cet alignement est effectué par une composition de chaque courbe X_i avec la fonction de déformation $h_{k,i} : \mathbb{R} \longrightarrow \mathbb{R}$ de la classe des décalages:

$$W = \{h : t \longmapsto t + s,\ s \in \mathbb{R}\}$$

De ce fait, une courbe peut appartenir à plusieurs clusters, c'est-à-dire différentes parties d'une courbe peuvent être similaires à des parties d'autres courbes. Par conséquent, cela revient à attribuer à chaque courbe X_i, une probabilité $p_{k,i}$ d'appartenance au cluster k. Donc pour chaque cluster $k = 1, \ldots, K$, nous définissons une fonction d'appartenance p_k par:

$$\begin{array}{rcl} p_k : \{X_1, \ldots, X_N\} & \longrightarrow & [0;\ 1] \\ X_i & \longmapsto & p_{k,i} \end{array}$$

de telle sorte que pour tout $i = 1, \ldots, N$, nous avons $\sum_{k=1}^{K} p_{k,i} = 1$, c'est-à-dire chaque courbe appartient à un ou plusieurs clusters et pour tout $k = 1, \ldots, K$, nous avons $\sum_{i=1}^{N} p_{k,i} > 0$, c'est-à-dire chaque cluster contient au moins une courbe.
Chaque probabilité d'appartenance $p_{k,i}$ correspond à un décalage particulier $s_{k,i}$ de la courbe X_i qui minimise la distance $d(X_i, v_k)$ compte tenu de toutes les contraintes.
Ainsi, pour des tailles de centres de clusters $c_1, \ldots, c_K$ fixées, ProbKMA est équivalent au problème d'optimisation suivant:
trouver les K centres de clusters $V = (v_1, \ldots, v_K)$, les probabilités d'appartenance $P = [p_{k,i}]$ et les décalages $S = [s_{k,i}]$ qui minimisent la fonction des moindres carrés généralisés

$$J_m(P, S, V) = \sum_{i=1}^{N} \sum_{k=1}^{K} (p_{k,i})^m d^2(\tilde{X}_{i,s_{k,i}}\ ;\ v_k)$$

sous les contraintes suivantes: pour tout $k = 1, \ldots, K$ et $i = 1, \ldots, N$

$$\begin{cases} p_{k,i} \in [0;\ 1], & \forall\ i,\ k \quad \text{(ce sont des probabilités d'appartenance)} \\ \sum_{k=1}^{K} p_{k,i} = 1, & \forall\ i \quad \text{(chaque courbe appartient à au moins un cluster)} \\ \sum_{i=1}^{N} p_{k,i} > 0, & \forall\ k \quad \text{(chaque cluster contient au moins une courbe)} \end{cases}$$

avec $m > 1$, le paramètre de pondération et $\tilde{X}_{i,s_{k,i}} = X \circ h_{k,i}$, les courbes alignées.

3 Résolution du problème

La condition nécessaire pour qu'un triplet $(\hat{P}, \hat{S}, \hat{V})$ soit un minimiseur local de ce problème d'optimisation est que chacun des $\hat{P}$, $\hat{S}$ et $\hat{V}$ minimise J_m lors de la fixation des deux autres. En d'autres termes, pour résoudre le problème, on fixe deux variables et on minimise par rapport à la variable non fixée. Par exemple, pour trouver $\hat{P}$, un minimiseur local de J_m, on fixe (S, V) et on minimise par rapport à P.

Thèoriquement, Cremona et Chiaromonte ont proposé deux résultats clés pour les minimiseurs $\hat{P}$ et $\hat{V}$:

- **Proposition 1:** *Les probabilités d'appartenance* $\hat{P}$. Considèrons fixes la matrice de décalage $\hat{S}$, les K centres de clusters $\hat{V} = (\hat{v}_1, \ldots, \hat{v}_K)$ et l'ensemble des courbes ayant une distance non nulle (positive) avec tous les centres de cluster $R = \{1 \leq i \leq N : d(\tilde{X}_{i,\hat{s}_{k,i}}, \hat{v}_k) > 0, \forall k\}$ avec $|R| \geq K$. Alors $\hat{P} = [\hat{p}_{k,i}]$ est un minimiseur global de $J_m(., \hat{S}, \hat{V})$ si et seulement si:

$$\text{pour tout } i \in R, \quad \hat{p}_{k,i} = \left[\sum_{l=1}^{K} \left(\frac{d^2(\tilde{X}_{i,\hat{s}_{k,i}}, \hat{v}_k)}{d^2(\tilde{X}_{i,\hat{s}_{l,i}}, \hat{v}_l)} \right)^{\frac{1}{m-1}} \right]^{-1}, \quad k = 1, \ldots, K$$

$$\text{pour tout } i \notin R, \quad \hat{p}_{k,i} = \begin{cases} 0, & \text{pour tout } k \text{ telque } d(\tilde{X}_{i,\hat{s}_{k,i}}, \hat{v}_k) > 0 \\ \in [0,1], & \text{pour tout } k \text{ telque } d(\tilde{X}_{i,\hat{s}_{k,i}}, \hat{v}_k) = 0 \end{cases}, \text{ avec } \sum_{k=1}^{K} \hat{p}_{k,i} = 1$$

Cette solution est valable pour n'importe quelle distance utilisée.

Note: ProbKMA ne sait pas distinguer le cas d'une courbe X_{i_1} qui correspond à tous les K centres de clusters (c'est-à-dire $d(X_{i_1}, v_k) = \epsilon \approx 0, \forall k$) de celui d'une courbe X_{i_2} qui ne correspond à auncun des K centres de clusters (c'est-à-dire $d(X_{i_2}, v_k) = M \gg 0, \forall k$). Dans les deux cas, nous aurons les mêmes probabilités d'appartenance pour toutes ces courbes dites des cas extrêmes:

$$\forall i \in R, \quad p_{k,i_1} = p_{k,i_2} = \frac{1}{k}, \quad k = 1, \ldots, K$$

Pour résoudre ce problème, une étape de nettoyage de cluster, lorsque l'algorithme est proche de la convergence, est effectuée. La matrice P des probabilités d'appartenance est dichotomisée suivant le quantile $q_{\frac{1}{K}}$ d'ordre $\frac{1}{K}$ de toutes les distances $d(X_i, v_k)$:

$$\hat{p}_{k,i} = \begin{cases} 1, & \text{pour des distances inférieures à } q_{\frac{1}{K}} \\ 0, & \text{pour des distances supérieures à } q_{\frac{1}{K}} \end{cases}$$

- **Proposition 2:** *Les centres de clusters* $\hat{V}$. Considèrons fixes les matrices de décalage $\hat{S}$ et de probabilités d'appartenance $\hat{P}$. Pour n'importe quelle distance définie par

$$d_\alpha^2(X, V) = \sum_{j=1}^{d} \frac{w_j}{d} \left[\frac{1-\alpha}{c} \int_0^c (x^{(j)}(t) - v^{(j)}(t))^2 dt + \frac{\alpha}{c} \int_0^c (x'^{(j)}(t) - v'^{(j)}(t))^2 dt \right]$$

où $w_j > 0$ est le poids de la $j^{ième}$ composante d'une courbe et $\alpha \in [0,1]$ est le paramètre qui définit le poids relatif des niveaux et des dérivés de la courbe, alors $\hat{V}$ est un minimiseur global de $J_m(\hat{P}, \hat{S}, .)$ si et seulement si

$$\hat{v}_k = \frac{\sum_{i=1}^{N} (\hat{p}_{k,i})^m \tilde{X}_{i,\hat{s}_{k,i}}}{\sum_{i=1}^{N} (\hat{p}_{k,i})^m} \qquad \text{p.p sur } [0, c_k], \quad \forall k$$

Pour $\alpha = 1$, cette formule reste vraie et $\hat{v}_k$ y est défini à une constante additive près.

Note:Afin de traiter les grands écarts dans les courbes, c'est-à-dire les parties manquantes des courbes qui ne peuvent pas être imputées de manière significative par lissage, ProbKMA permet à chaque courbe X d'avoir un domaine $D \subset \mathbb{R}$. De ce fait, la contribution d'une courbe dépendra de son domaine, en plus de sa probabilité d'appartenir à un cluster. Ainsi, la distance $d_\alpha(.,.)$ est généralisée comme suit:

$$d_\alpha^2(X,V) = \sum_{j=1}^{d} \frac{w_j}{d} \left[\frac{1-\alpha}{|[0,c]\cap D|} \int_{[0,c]\cap D} (x^{(j)}(t) - v^{(j)}(t))^2 dt + \frac{\alpha}{|[0,c]\cap D|} \int_{[0,c]\cap D} (x'^{(j)}(t) - v'^{(j)}(t))^2 dt \right]$$

Se basant sur cette distance généralisée, le $k^{ième}$ centre de cluster devient:

$$\hat{v}_k = \frac{\sum_{i=1}^{N} \frac{(\hat{p}_{k,i})^m}{|[0,c_k]\cap \tilde{D}_{i,s_{k,i}}|} \mathbb{I}_{[0,c_k]\cap \tilde{D}_{i,s_{k,i}}} \tilde{X}_{i,\hat{s}_{k,i}}}{\sum_{i=1}^{N} \frac{(\hat{p}_{k,i})^m}{|[0,c_k]\cap \tilde{D}_{i,s_{k,i}}|} \mathbb{I}_{[0,c_k]\cap \tilde{D}_{i,s_{k,i}}}}, \forall k$$

où $\tilde{D}_{i,s} = D_i - s$ est le domaine de la courbe décalée $\tilde{X}_{i,s_\cdot}$.

4 Algorithme et évaluation de ProbKMA

4.1 Algorithme

Numériquement, les auteurs ont proposé un algorithme de minimisation pour ProbKMA qui se fait itérativement en alternant les étapes d'identification des centres des clusters, d'alignement des courbes et de calcul des probabilités d'appartenance. L'algorithme est le suivant:

- **Initialisation**: On fixe le nombre de clusters K et la taille des centres des clusters; on se donne une matrice de décalage initiale $S^{(0)}$ et une matrice de probabilités d'appartenance initiale $P^{(0)}$ vérifiant la non-dégénérescence des courbes

$$\begin{cases} \sum_{k=1}^{K} p_{k,i} = 1, & \forall\ i \\ \sum_{i=1}^{N} p_{k,i} > 0, & \forall\ k \end{cases}$$

- **Itération**: Pour $it = 1, 2, \ldots$, on répète les étapes suivantes jusqu'à la convergence:

Etape 1: *identification des centres des clusters.* Pour chaque $k = 1, \ldots, K$, on calcule le $k^{ième}$ centre de cluster $v_k^{(it)}$ en utilisant la formule de $\hat{v}_k$ de la proposition 2 et en considérant $S_k^{(it-1)}$ et $P_k^{(it-1)}$ obtenus lors de l'itération précédente;
Etape 2: *alignement de la courbe.* Pour chaque $i = 1, \ldots, N$ et $k = 1, \ldots, K$, on aligne la courbe X_i avec le nouveau centre de cluster $v_k^{(it)}$ en prenant $S_k^{(it)}$ qui minimise leur distance $d(\tilde{X}_{i,\hat{s}}, v_k^{(it)})$;
Etape 3: *calcul des probabilités d'appartenance.* On calcule $P^{(it)}$ via les formules de $\hat{p}_{k,i}$ de la proposition 1 et en utilisant $v_k^{(it)}$ et $S_k^{(it)}$;

- **Critère d'arrêt**: A chaque itération, la convergence est évaluée au moyen de la distance de Bhattacharyya entre les matrices d'appartenance $P^{(it)}$ et $P^{(it-1)}$. Pour un cluster k donné, la distance de Bhattacharyya est définie par:

$$BC_k = -\log\left(\sum_{i=1}^{N} \sqrt{p_{k,i}^{(it)} p_{k,i}^{(it-1)}} \right)$$

On calcule une distance BC globale (soit la moyenne, le maximum ou le quantile d'ordre q des BC_k) pour tous les clusters. On répète les étapes 1, 2 et 3 jusqu'à ce que la distance globale Bhattacharyya BC atteint une tolérance donnée.

4.2 Evaluation

Dans ProbKMA, un indice de silhouette généralisé, défini pour des portions de courbes, a été développé afin d'évaluer le clustering obtenu. Cet indice mesure l'adéquation de chaque portion dans le cluster auquel elle a été assignée. Pour chaque cluster, les mesures de qualité sont obtenues en agrégeant toutes les portions

attribuées au même cluster. Et pour la qualité globale du clustering, elle est obtenue en agrégeant sur l'ensemble des K clusters. Aisni, il sera possible de comparer les résultats correspondant à différents choix pour le nombre K de clusters et les longueurs $c_1, \ldots, c_K$ des centres des clusters.
Cet indice est obtenu suivant ces étapes:

- Calcul de la matrice des probabilités d'appartenance propres (dichotomisées) et identification des centres de clusters,
- Extraction, pour chaque courbe X_i, des portions qui correspondent à au moins un parmi les K clusters (c'est-à-dire, clusters pour lesquels $p_{k,i} = 1$),
- Calcul, pour chaque portion $j = 1, \ldots, J$ extraite du cluster k, de la distance moyenne $d_j(k)$ définie comme étant la moyenne des distances entre la portion j et toutes les autres portions attribuées au clusters k,
- Définission de la distance *intra-cluster* a_j comme étant la distance moyenne de la portion j au cluster k_j auquel elle appartient: i.e, $a_j = d_j(k_j)$,
- Définission de la distance *inter-cluster* b_j comme étant le minimum des distances moyennes de la portion j aux autres clusters $k \neq k_j$: i.e, $b_j = \min_{k \neq k_j} d_j(k)$,
- Définission de s_j l'indice de silhouette généralisé pour la portion j comme étant:

$$s_j = \frac{b_j - a_j}{\max(b_j, a_j)} \in [-1, 1]$$

Les grandes valeurs de s_j (valeurs proches de 1) indiquent que la portion j est bien affecté à son cluster. Par contre, les valeurs faibles indiquent de mauvaises affectations. En particulier, les valeurs négatives indiquent que la portion j est plus proche d'un cluster autre que celui auquel elle a été affectée.
Pour mesurer la qualité d'un cluster k, on calcule son indice de silhouette moyen S_k comme étant l'indice de silhouette moyen sur toutes les portions qui lui sont attribuées. L'indice de silhouette moyen global S, c'est-à-dire la moyenne de tous S_k, permet de mesurer la qualité globale du clustering.

4.3 Découvertes de motifs fonctionnels

Comme nous l'avons vu précédemment, le principal objectif de ProbKMA est la recherche d'un minimum local de J_m et sa sortie dépend fortement de l'initialisation. Si nous souhaitons uniquement regrouper les courbes en K clusters en fonction de la similarité locale, alors nous devons répéter l'algorithme avec différentes initialisations et nous sélectionnons la solution ayant la plus petite valeur de J_m. Par contre, si notre objectif est la découverte de motifs fonctionnels, alors l'algorithme doit être exécuté plusieurs fois avec différentes initialisations, différents nombres de clusters et différentes longueurs initiales de motifs. La reunion des solutions résultantes permet d'obtenir l'ensemble des motifs candidats. Ainsi, nous pouvons entamer la recherche de "vrais motifs" comme suit:
D'abord, l'ensemble des motifs candidats sera "nettoyé" en utilisant des indices de silhouette généralisés et le nombre d'occurrences de chaque motif. Ensuite, tous les motifs candidats qui sont très similaires les uns aux autres seront regroupés puisqu'ils peuvent en fait correspondre au même "vrai motif" identifié par plusieurs passages de l'algorithme. Pour cela, il faut comparer leur distance par paire avec l'ensemble des distances entre tous les motifs candidats et toutes les courbes. Enfin, étant donné l'ensemble des "vrais motifs" découverts, nous utilisons un algorithme de recherche de motifs afin de localiser toutes leurs occurrences dans les courbes d'entrée.
De manière détaillée, étant donné l'ensemble de motifs candidats, cette recherche de vrais motifs fonctionnels se fait suivant ces étapes illustrées dans la figure ci-dessous:

1. Calcul de la distance entre chaque paire de motifs candidats en utilisant la même distance que ProbKMA;
2. Regroupement hiérarchique avec une liaison moyenne des motifs candidats sur la base de leurs distances par paires;

3. Détermination d'un rayon global R_{all} basé sur les distances minimales entre tous les motifs candidats et toutes les courbes;

4. Découpage du dendrogramme du regroupement hiérarchique à la hauteur $2R_{all}$ afin d'obtenir M groupes de motifs similaires;

5. Pour chaque groupe $m = 1, \ldots, M$,

   - Détermination d'un rayon R_m spécifique au groupe en se basant sur les distances minimales entre les motifs du groupe m et toutes les courbes;
   - Pour chaque motif du groupe m,
     - Recherche des courbes contenant le motif, c'est-à-dire les courbes dont la distance par rapport au motif est $\leq R_m$,
     - Calcul du nombre d'occurrences dans les courbes (portions de courbes avec une distance $\leq R_m$ du motif), en comptant le nombre de courbes contenant le motif ;
     - Approximation de la distance moyenne à l'intérieur du motif (distance moyenne entre le motif et ses occurrences dans les courbes) avec la distance moyenne entre le motif et les courbes qui le contiennent ;
   - Sélection d'un petit nombre de motifs sur la base du nombre approximatif d'occurrences, du rayon approximatif et de la longueur du motif ;

6. Recherche de toutes les occurrences des motifs sélectionnés (portions de courbes avec une distance $\leq R_m$ du motif).

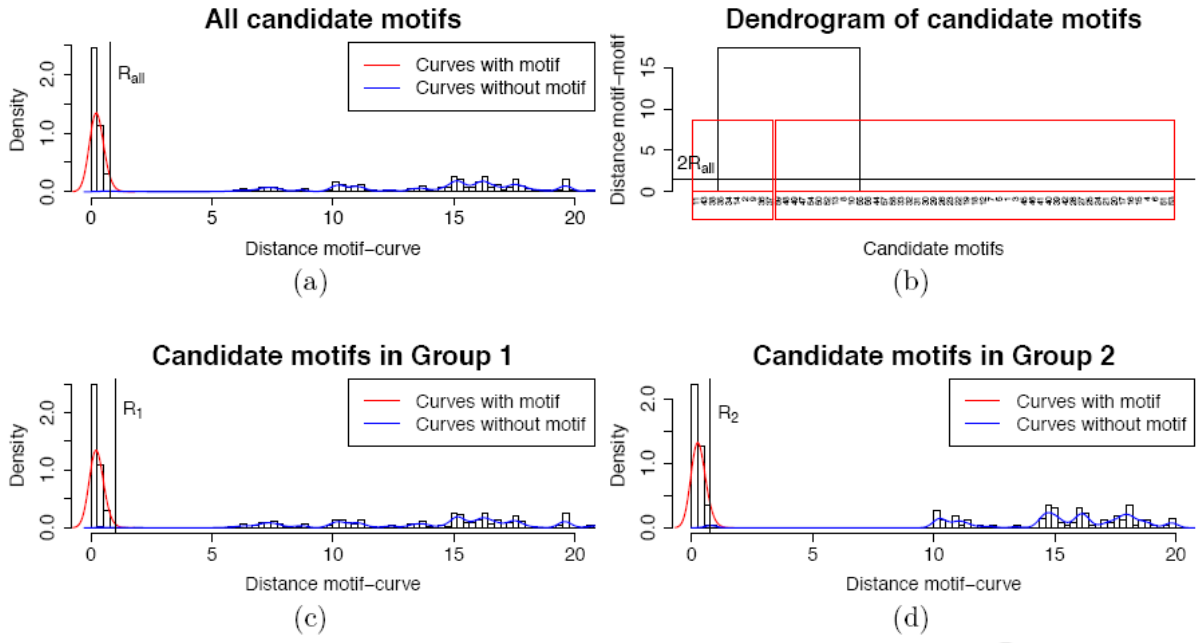


Figure 29: Recherche de vrais motifs

Dans cette figure, (a) montre la sélection d'un rayon global R_{all} (étape 3) ; (b) fournit le dendrogramme de regroupement hiérarchique coupé à la hauteur de $2R_{all}$ (étape 4) ; (c) et (d) montrent la sélection d'un rayon R_m spécifique au groupe m (étape 5a).

5 Implémentation du package ProbKMA

5.1 Description

Dans cette partie, nous allons fournir la description de ProbKMA et des principales fonctions qu'elle utilise pour son exécution.

ProbKMA

Il s'agit de la fonction principale du package. Elle permet de faire le K-mean probabiliste avec alignement local pour trouver des motifs candidats. Comme arguments, elle possède:

- `Y0` : il s'agit d'une liste de N vecteurs (respectivement, matrices à d colonnes) pour les courbes univariées (respectivement, pour des courbes à d dimensions) `y_i(x)`. Toutes les courbes doivent être évaluées sur une grille uniforme. Lorsque `y_j(x)=NA` dans la dimension j, alors `y_j(x)=NA` dans toutes les dimensions.
- `Y1` : il s'agit d'une liste de N vecteurs (respectivement, matrices à d colonnes) pour les courbes dérivées univariées (respectivement, pour des courbes dérivées à d dimensions) `y'_i(x)` avec l'évaluation des dérivées des courbes (toutes les courbes doivent être évaluées sur une grille uniforme). Lorsque `y'_j(x)=NA` dans la dimension j, alors `y'_j(x)=NA` dans toutes les dimensions. Cet argument doit être fourni obligatoirement lorsque `diss='d1_L2'` ou `diss='d0_d1_L2'`.
- `standardize` : si TRUE, chaque dimension est normalisée (score Z sur l'ensemble de toutes les régions).
- `K` : nombre de motifs.
- `c` : longueur minimale des motifs. Peut être un entier (ou un vecteur de K entiers).
- `c_max` : longueur maximale des motifs. Peut être un nombre entier (ou un vecteur de K nombres entiers).
- `P0` : matrice d'appartenance initiale, avec N lignes et K colonnes (si NULL, une `P0` aléatoire est choisie).
- `S0` : matrice initiale de déformation par décalage, avec N lignes et K colonnes (si NULL, une `S0` aléatoire est choisie).
- `diss` : dissimilarité. Les choix possibles sont: `'d0_L2'` (distance L_2 entre fonctions), `'d1_L2'`(distance L_2 entre dérivées) et `'d0_d1_L2'`(distance de Sobolev qui combine les distances L_2 entre fonctions et entre dérivées).
- `alpha` : lorsque `diss = 'd0_d1_L2'`, il représente le coefficient de pondération entre `d0_L2` et `d1_L2` (`alpha = 0` signifie `d0_L2`, `alpha = 1` signifie `d1_L2`).
- `w` : vecteur des poids pour l'indice de dissimilarité dans les différentes dimensions ($w > 0$).
- `m > 1` : exposant de pondération dans la fonctionnelle des moindres carrés.
- `iter_max` : le nombre maximum d'itérations autorisées.
- `stop_criterion` : critère pour arrêter l'itération, basé sur la distance de Bhattacharyya entre les appartenances dans les itérations suivantes. Les choix possibles sont : '`max`' pour le maximum des distances dans les différents motifs ; '`mean`' pour la moyenne des distances dans les différents motifs ; '`quantile`' pour le quantile des distances dans les différents motifs (dans ce cas, le quantile doit être fourni).
- `quantile` : probabilité dans $[0, 1]$ à utiliser si `stop_criterion='quantile'`.
- `tol` : tolérance de la méthode (la méthode s'arrête si le critère stop `stop_criterion` $<$ `tol`).

- `iter4elong` : l'élongation des motifs est effectuée à chaque `iter4elong` itérations (si `iter4elong` > `iter_max`, aucune élongation n'est effectuée). Par exemple pour `iter4elong = 4`, alors l'élongation sera faite aux itérations 4, 8, 12, ...
- `tol4elong` : tolérance sur la distance de Bhattacharyya (avec le choix fait dans `stop_criterion`) pour effectuer l'élongation des motifs.
- `max_elong` : élongation maximale autorisée en une seule itération, en pourcentage de la longueur du motif.
- `trials_elong` : nombre d'essais d'élongation (équidistants) de chaque côté du motif en une seule itération.
- `deltaJk_elong` : augmentation relative maximale de la fonction objectif autorisée dans chaque élongation du motif (pour les écarts et chaque côté).
- `max_gap` : écart maximal autorisé dans chaque alignement (pourcentage de la longueur du motif).
- `iter4clean` : le nettoyage du motif est effectué à chaque `iter4clean` itérations (si `iter4clean` > `iter_max`, aucun nettoyage n'est effectué).
- `tol4clean` : tolérance sur la distance de Bhattacharyya (avec le choix fait dans `stop_criterion`) pour effectuer le nettoyage des motifs.
- `quantile4clean` : quantile de dissimilarité à utiliser pour le nettoyage des motifs.
- `return_options` : TRUE ou FALSE. Si TRUE, les options `K`, `c`, `diss`, `w`, et `m` sont retournées par la fonction.
- `return_init` : TRUE ou FALSE. Si TRUE, `P0` et `S0` sont retournés par la fonction.
- `worker_number` : nombre de cœurs de CPU à utiliser pour la parallélisation. Si `worker_number=1`, la fonction est exécutée séquentiellement. Par défaut, il prend la valeur $n-1$ (où n est la nombre de cœurs de CPU disponibles calculé par la fonction).

Comme sortie, la fonction `ProbKMA` retourne une liste contenant les éléments suivants:

- quelques initialisations et options de `ProbKMA`: `P0` et `S0` (si `return_init = TRUE`) et `Y0`, `Y1`, `standardize`, `K`, `c`, `c_max`, `diss`, `alpha`, `w`, `m`, `iter4elong`, `tol4elong`, `max_elong`, `trials_elong`, `deltaJk_elong`, `max_gap`, `iter4clean` et `tol4clean` (si `return_options = TRUE`)
- les nouvelles matrices calculées: `P` (matrice de probabilités d'appartenance de taille $N \times K$), `P_clean` (matrice de probabilités d'appartenance dichotomisée suivant le quantile d'ordre $\frac{1}{K}$), `S` (matrice de déformation par décalage de taille $N \times K$), `S_clean` (matrice de déformation par décalage correspondant à `P_clean`), `D` (matrice de dissimilarité de taille $N \times K$) et `D_clean` (matrice de dissimilarité calculée à partir des motifs nettoyés)
- `iter` (nombre d'itérations effectuées), `J_iter` (minimum de la fonction des moindres carrés généralisés obtenue à la dernière itération), `BC_dist` (distance de Bhattacharyya globale selon le critère choisi) et `BC_dist_iter` (distance de Bhattacharyya obtenue à la dernière itération)
- `V0` (liste contenant les centres de clusters, c'est-à-dire motifs candidats), `V1` (liste contenant les dérivées des centres de clusters, c'est-à-dire des motifs candidats), `V0_clean` (liste contenant les centres de clusters après nettoyage, c'est-à-dire motifs candidats nettoyés), et `V1_clean` (liste contenant les dérivées des centres de clusters après nettoyage, c'est-à-dire des motifs candidats nettoyés)

La fonction ProbKMA_plot

Elle permet de tracer les résultats de probKMA. Comme arguments, elle prend:

- `probKMA_results` : la sortie de la fonction `probKMA`.
- `ylab` : un vecteur de longueur d comportant les titres de l'axe des abscisses.
- `cleaned` : TRUE ou FALSE. Tracer ou non les motifs après nettoyage.

La fonction ProbKMA_silhouette

Elle calcule l'indice de silhouette adapté sur les résultats de probKMA. Ses principaux arguments sont:

- `probKMA_results` : La sortie de la fonction `probKMA` (avec l'option `return_options=TRUE`).
- `align` : TRUE ou FALSE. Si TRUE, la fonction va essayer tous les alignements possibles entre les morceaux de courbes (correspondant au même motif ou à des motifs différents).
- `plot` : TRUE ou FALSE. Tracer ou non la silhouette.

Comme sortie, la fonction `ProbKMA_silhouette` renvoie une liste ayant les éléments suivants:

- `silhouette` (vecteur contenant l'indice de silhouette obtenu pour chaque morceau de courbe appartenant à un cluster);
- `motifs` (vecteur contenant les numéros des motifs trouvés), `curves` (vecteur contenant les numéros des courbes ayant des motifs);
- `silhouette_average` (vecteur contenant l'indice de silhouette moyen par cluster).

La fonction find_candidate_motifs

Elle permet d'exécuter plusieurs fois la fonction `probKMA` avec différents K, c et initialisations, dans le but de trouver un ensemble de motifs candidats. Si le dossier `nom_KK_cc` existe déjà et que n fichiers de résultats y sont présents, alors la fonction va les charger et continuer avec les exécutions `n_init-n`. Elle a pour arguments:

- `Y0` : il s'agit d'une liste de N vecteurs (respectivement, matrices à d colonnes) pour les courbes univariées (respectivement, pour des courbes à d dimensions) `y_i(x)`. Toutes les courbes doivent être évaluées sur une grille uniforme. Lorsque `y_j(x)=NA` dans la dimension j, alors `y_j(x)=NA` dans toutes les dimensions.
- `Y1` : il s'agit d'une liste de N vecteurs (respectivement, matrices à d colonnes) pour les courbes dérivées univariées (respectivement, pour des courbes dérivées à d dimensions) `y'_i(x)` avec l'évaluation des dérivées des courbes (toutes les courbes doivent être évaluées sur une grille uniforme). Lorsque `y'_j(x)=NA` dans la dimension j, alors `y'_j(x)=NA` dans toutes les dimensions. Cet argument doit être fourni obligatoirement lorsque `diss='d1_L2'` ou `diss='d0_d1_L2'`.
- `K` : vecteur comportant les nombres de motifs à tester.
- `c` : vecteur comportant les longueurs minimales des motifs qui doivent être testés.
- `n_init` : nombre d'initialisation aléatoire pour chaque combinaison de K et c.
- `name` : nom du dossier dans lequel les résultats sont sauvegardés.
- `names_var` : vecteur de longueur d, avec les noms des variables dans les différentes dimensions.
- `probKMA_options` : liste avec les options pour probKMA (voir l'aide de probKMA).
- `plot` : TRUE ou FALSE. Si TRUE, les graphiques de synthèse sont dessinés.
- `worker_number` : nombre de cœurs de CPU à utiliser pour la parallélisation. Si `worker_number=1`, la fonction est exécutée séquentiellement. Par défaut, il prend la valeur $n-1$ (où n est la nombre de cœurs de CPU disponibles calculé par la fonction).

Comme sortie, la fonction `find_candidate_motifs` renvoie une liste comportant les éléments suivants:

- quelques unes de ses entrées: `K`, `c`, `n_init` et `name`;
- `times` (liste contenant les temps d'exécution de la fonction `ProbKMA` pour chaque combinaison de `K`, `c` et `n_init`) et `silhouette_average_sd` (liste contenant des listes des matrices à `n_init` ligne(s) contenant les deux colonnes `silhouette_average` et `silhouette_sd`qui représentent respectivement la moyenne et l'écart-type des indices de silhouette pour chaque exécution de la fonction `ProbKMA`).

La fonction filter_candidate_motifs

Elle filtre les motifs candidats sur la base de seuils sur l'indice de silhouette moyen et sur la taille des courbes dans le motif.

- `find_candidate_motifs_results` : sortie de la fonction `find_candidate_motif`.
- `sil_threshold` : seuil sur l'indice de silhouette moyen.
- `size_threshold` : seuil sur la taille du motif (nombre de courbes dans le cluster).
- `K` : vecteur comportant les nombres de motifs qui doivent être considérés (doit être un sous-ensemble de `find_candidate_motifs_results$K`).
- `c` : vecteur avec les longueurs minimales des motifs qui doivent être considérés (doit être un sous-ensemble de `find_candidate_motifs_results$c`).

La fonction `filter_candidate_motifs` a comme sortie une liste contenant les éléments:

- quelques entrées de la fonction `ProbKMA`: `Y0`, `Y1`, `diss`, `alpha`, `w`, `c`, `K` et `max_gap`;

`V0_clean` (vecteur contenant les motifs candidats après filtrage), `V1_clean` (vecteur contenant les dérivées des motifs candidats après filtrage), `D_clean` (matrice contenant les distances des motifs candidats après filtrage par rapport aux courbes) et `P_clean` (matrice contenant les probabilités d'appartenance des motifs candidats après filtrage).

La fonction cluster_candidate_motifs

Dans un premier temps, elle permet de déterminer un rayon global R_{all} (basé sur les distances entre tous les motifs et toutes les courbes), et de regrouper les motifs candidats en fonction de leur distance, en exigeant que les groupes soient séparés par une dissimilarité de plus de $2 \times R_{all}$. Et dans un second temps, elle permet, pour chaque groupe $m = 1, ..., M$, de déterminer un rayon R_m spécifique au groupe (basé sur les distances entre les motifs du même groupe et toutes les courbes). Ses principaux arguments sont:

- `filter_candidate_motifs_results` : sortie de la fonction `filter_candidate_motifs`.
- `motif_overlap` : recouvrement minimum requis entre les motifs candidats dans leur calcul de distance, en % du motif le plus court.
- `k_knn` : nombre de voisins à utiliser dans les k plus proches voisins, afin de déterminer R_{all} et R_m lorsque les deux groupes (courbes avec/sans motif) ne sont pas séparés.
- `votes_knn_Rall` : seuil sur le pourcentage de votes pour la classe 1 (courbe avec le motif) dans les k plus proches voisins, afin de déterminer R_{all}.
- `votes_knn_Rm` : seuil sur le pourcentage de votes pour la classe 1 (la courbe a le motif) dans les k plus proches voisins, afin de déterminer R_m.
- `plot` : TRUE ou FALSE. Si TRUE, les histohrammes et le dendrogramme sont dessinés.
- `ask` : TRUE ou FALSE. Si TRUE, l'utilisateur est consulté avant qu'un nouveau graphique ne soit dessiné.

- `worker_number` : nombre de cœurs de CPU à utiliser pour la parallélisation (par défaut NULL). Si `worker_number=1`, la fonction est exécutée séquentiellement.

Comme sortie, la fonction `cluster_candidate_motifs` renvoie une liste contenant les éléments suivants:

- quelques unes de ses entrées: `k_knn`, `votes_knn_Rall` et `votes_knn_Rm`;
- les éléments renvoyés par la fonction `filter_candidate_motifs`;
- `VV_D` (matrice des distances entre les motifs candidats obtenus après filtrage), `VV_S` (matrice des déformations par décalage des motifs candidats obtenus après filtrage), `R_all` (rayon global utilisé pour couper le dendrogramme) et `hclust_res` (liste contenant le résultat du clustering hiérarchique basé sur les distances entre les motifs candidats obtenus après filtrage).

La fonction cluster_candidate_motifs_plot

Elle permet de tracer les résultats de la fonction `cluster_candidate_motifs`.

- `cluster_candidate_motifs_results` : sortie de la fonction `cluster_candidate_motifs`.
- `R_all` : rayon global, utilisé pour couper le dendrogramme (exigeant que les groupes soient séparés de plus de $2 \times R_{all}$).
- `R_m` : vecteur comportant les rayons spécifiques aux groupes. La longueur du vecteur doit correspondre au nombre de clusters obtenus en coupant le dendrogramme à une hauteur de $2 \times R_{all}$. Si `R_m = NULL`, alors il sera déterminé dans chaque groupe (basé sur les distances entre les motifs du même groupe et toutes les courbes).
- `ask` : TRUE ou FALSE. Si TRUE, l'utilisateur est consulté avant qu'un nouveau graphique ne soit dessiné.

La fonction motifs_search

Elle permet de trouver les occurrences des motifs candidats dans les courbes et les trier en fonction de leur fréquence et de leur rayon. Dans chaque groupe (défini en coupant le dendrogramme à une hauteur de $2 \times R_{all}$), nous choisissons le motif ayant la fréquence la plus élevée et la dissimilarité moyenne la plus faible (celui qui est le mieux classé dans les deux dimensions). Des motifs supplémentaires peuvent être choisis dans un groupe, si leurs longueurs diffèrent suffisamment de la longueur du premier motif choisi. Un motif candidat correspond à un morceau de courbe si leur dissimilarité est inférieure au R_m correspondant. Elle a pour principaux arguments:

- `cluster_candidate_motifs_results` : sortie de la fonction `cluster_candidate_motifs`.
- `R_all` : rayon global, utilisé pour couper le dendrogramme (exigeant que les groupes soient séparés de plus de $2 \times R_{all}$).
- `R_m` : vecteur comportant les rayons spécifiques aux groupes, utilisé pour trouver des occurrences de motifs. La longueur du vecteur doit correspondre au nombre de clusters obtenus en coupant le dendrogramme à la hauteur $2 \times R_{all}$. Si `R_m = NULL`, alors il sera déterminé dans chaque groupe (basé sur les distances entre les motifs du même groupe et toutes les courbes).
- `use_real_occurrences` : si TRUE, trouve les occurrences pour tous les motifs candidats et utilise la fréquence réelle et la dissimilarité moyenne pour choisir les motifs dans les groupes (plus précis, mais prend du temps). Sinon, utilise la fréquence approximative et la dissimilarité moyenne (par défaut).
- `length_diff` : différence minimale de longueur entre les motifs d'un même groupe, requise en ordre pour garder plus d'un motif, en % du motif le plus fréquent.

- `worker_number` : nombre de cœurs de CPU à utiliser pour la parallélisation. Si `worker_number = 1`, la fonction est exécutée séquentiellement. Par défaut, il prend la valeur $n-1$ (où n est la nombre de cœurs de CPU disponibles calculé par la fonction).

Comme sortie, la fonction `motifs_search` renvoie une liste contenant les éléments suivants:

- `V0` (liste contenant les motifs trouvés), `V1` (liste contenant les dérivées des motifs trouvés), `V_length` (vecteur contenant les longueurs réelles des motifs trouvés), `V_occurrences` (vecteur contenant les occurences réelles des motifs trouvés), `V_frequencies` (vecteur contenant les fréquences réelles des motifs trouvés), `V_mean_diss` (vecteurs contenant les dissimilarités moyennes réelles entre les motifs trouvés) et `R_motifs` (vecteur contenant les rayons des motifs trouvés);
- quelques éléments renvoyés par la fonction `cluster_candidate_motifs`:`Y0` et `Y1`.

La fonction motifs_search_plot

Elle permet de tracer les résultats de la fonction `motifs_search`. Comme arguments principaux, nous avons:

- `motifs_search_results` : sortie de la fonction `motifs_search`.
- `ylab` : un vecteur de longueur d comportant les titres pour l'axe des abscisses pour chaque dimension.
- `freq_threshold` : tracer seulement les motifs avec une fréquence au moins égale à `freq_threshold`.
- `top_n` : si `"all"`, alors tous les motifs trouvés seront tracés. Si `top_n` est un nombre entier, alors les `top_n` premiers motifs seront tracés.
- `plot_curves` : TRUE ou FALSE. Si TRUE, alors toutes les courbes seront tracées, avec les motifs trouvés colorés.

5.2 Quelques simulations

5.3.1 Application au données "BERKLEY GROWTH" : `Growth` est un des datasets du package `fda`. C'est une liste contenant les tailles de 39 garçons et 54 filles âgés de 1 à 18 ans et les âges auxquels les tailles ont été mesurées. Comme éléments, cette liste comprend:

- `hgtm`: une matrice numérique de taille 31×39 donnant les tailles en centimètres de 39 garçons à 31 ans.
- `hgtf`: une matrice numérique de taille 31×54 donnant les tailles en centimètres de 54 filles à 31 âges.
- `age` : un vecteur numérique de longueur 31 donnant les âges auxquels les tailles ont été mesurées

Lissage des données

Les données ont été lissées en utilisant une base B-spline monotone d'ordre 6, avec des nœuds aux âges observés, une pénalité de rugosité sur la troisième dérivée ($m = 3$) et un paramètre de lissage $\lambda = \frac{1}{\sqrt{10}}$ (Ramsay et al., 2009).

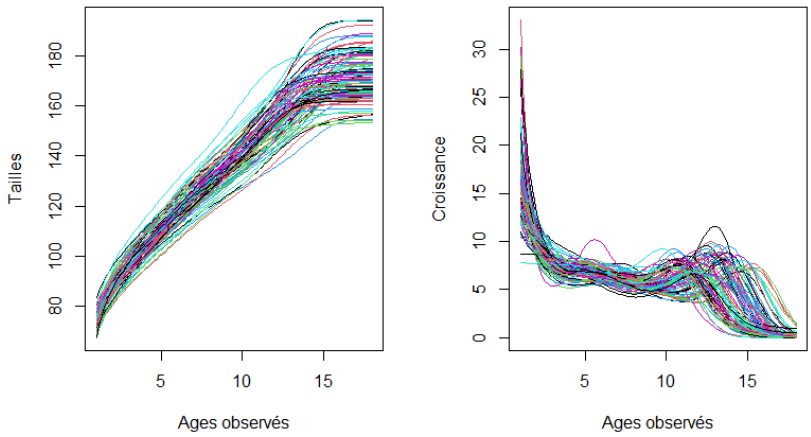


Figure 30: Les 93 courbes lisses

La figure 30 montre les courbes obtenues après lissage.

Clustering global probabiliste

Un clustering global probabiliste est effectué en exécutant la fonction `probKMA()` avec $K = 2$ clusters et L_2 comme pseudo-distance d_1 entre les courbes entières (aucun alignement n'est autorisé).

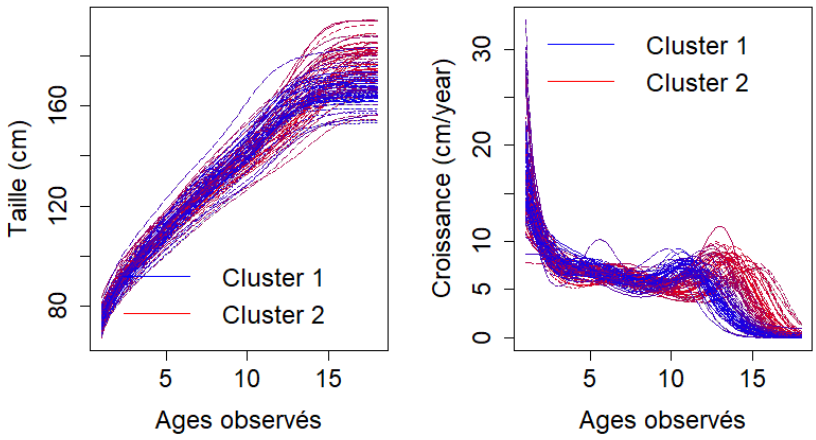


Figure 31: Clustering global probabiliste

Ainsi exécutée, la fonction `probKMA()` attribue à chaque courbe une probabilité d'appartenance aux deux clusters (bleu et rouge). En d'autres termes, si on prend une des 93 courbes, `probKMA()` lui attribue deux probabilités: une pour son appartenance au cluster 1 (de couleur bleue) et une autre pour son appartenance au cluster 2 (de couleur rouge). De ce fait, chaque courbe est assignée au cluster où elle a une plus grande probabilité d'appartenance.
\begin{center}

Table 11: Classification 1

Label	C1	C2
Garçon	37	02
Fille	09	45

Comme résultat, nous avons obtenu deux clusters qui diffèrent sur le moment principal de la poussée de croissance pubertaire et qui correspondent approximativement aux deux groupes de garçons et de filles (les garçons grandissent moins vite que les filles). Nous avons eu 11 enfants mal classés (2 garçons et 9 filles) avec un taux d'erreur de classification de 0.21.

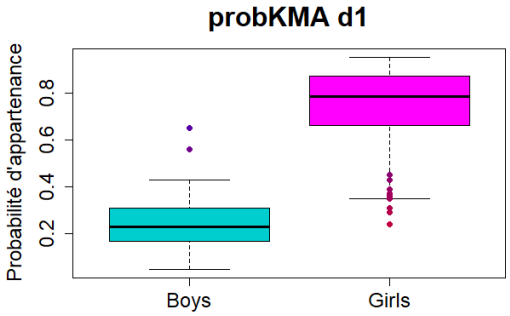


Figure 32: Classification

En termes plus détaillés, nous avons obtenu deux clusters. Le cluster 1 (respectivement le cluster 2) est composé de 46 enfants (respectivement 47 enfants) dont 37 garçons (respectivement 2) et 9 filles (respectivement 45).

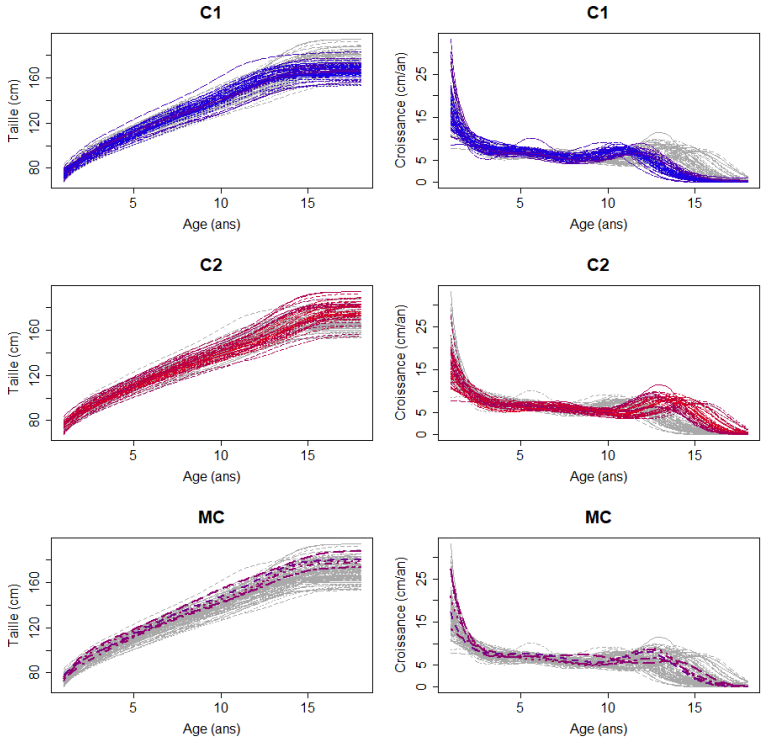


Figure 33: Représentation par cluster

Dans la figure 33 nous avons la représentation détaillée des clusters (contenu de chaque cluster C1 et C2). Les graphiques MC représentent les courbes malclassées. Nous pouvons également avoir un aperçu global (dans un même graphique) du clusterign avec les mal classés (voir figure 34)

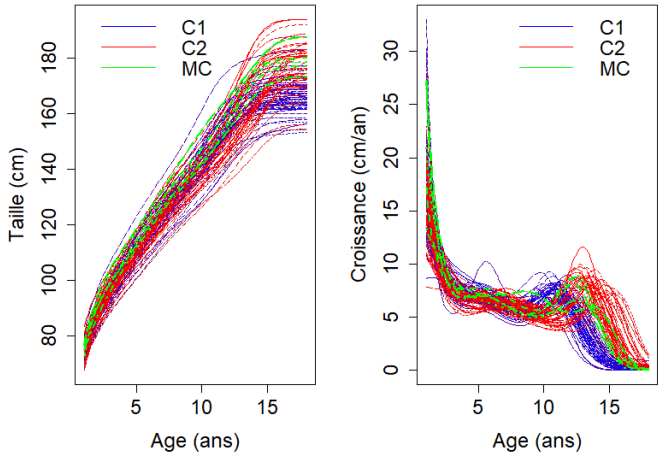


Figure 34: Représentation global (Clusters et les mal classés)

Clustering local probabiliste

Pour effectuer un clustering local probabiliste, nous exécutons la fonction `probKMA()` avec $K = 2$ et une pseudo-distance d_1 de type L_2 entre les portions de courbes de longueur $c = 51$, correspondant à 8.5 ans (décalage maximal de 8.5 ans).
Pour ce faire, nous allons dichotomiser les probabilités d'appartenance aux clusters en se basant sur la distance entre la portion de courbe et le centre du cluster. Nous fixons ces probabilités à 1 lorsque cette distance est inférieure à la distance médiane (voir figure 35). La barre verticale rouge indique la distance médiane utilisée pour la dicotomisation des probabilités.

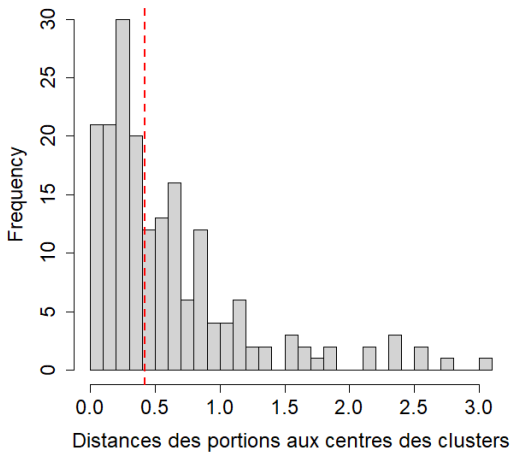


Figure 35: Dichotomisation

Aprés dichotomisation, nous obtenons deux clusters de 50 et 43 portions chacun (voir figure 36).

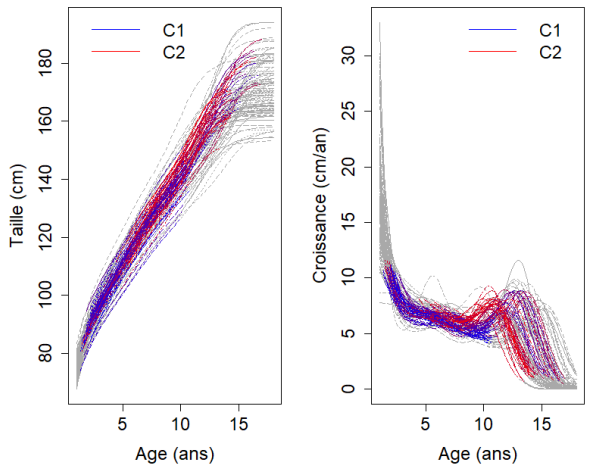


Figure 36: Clustering local probabiliste

Le tableau de classification nous montre que 18 courbes appartiennent aux deux clusters (c'est-à-dire qu'elles comprennent les deux motifs dans différentes parties de leurs domaines), que 32 courbes (respectivement 25 courbes) appartiennent uniquement au cluster 1 (respectivement au cluster 2) et que 18 courbes n'appartiennent à aucun cluster.
Le cluster 2 capture une forme particulière pour la poussée de croissance pubertaire, tandis que le cluster 1 capture la diminution de la vitesse de croissance qui est typique chez les enfants entre 2 et 3 ans.
\begin{center}

Table 13: Classification 2

Label	C1 et C2	C1	C2	Aucun
Garçon	08	18	06	07
Fille	10	14	19	11

De manière détaillée, nous pouvons visualiser, sur la figure 37, la composition de chaque cluster.

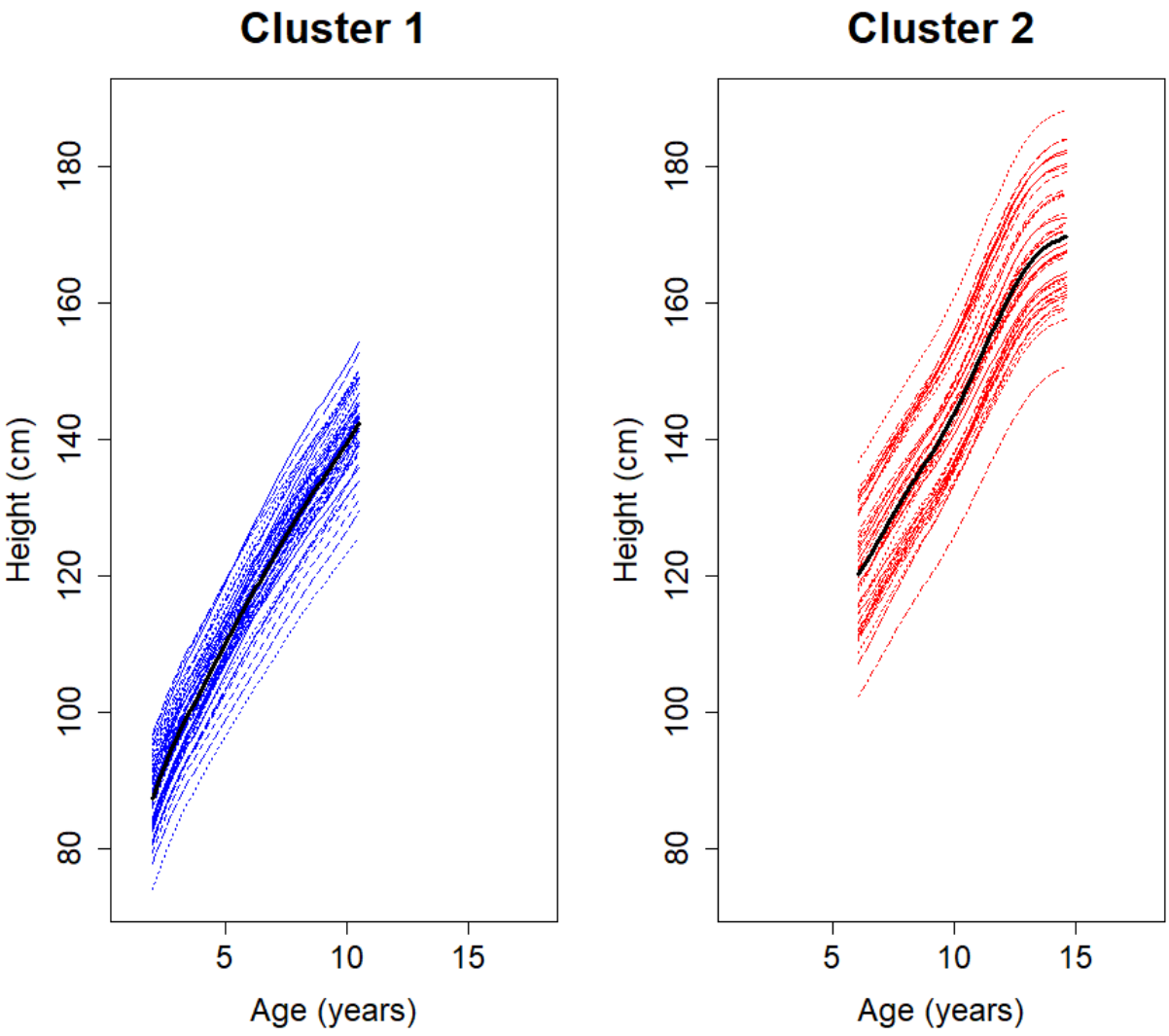


Figure 37: Cluster 1 et Cluster 2

Le graphique 38 nous donne l'ensemble.

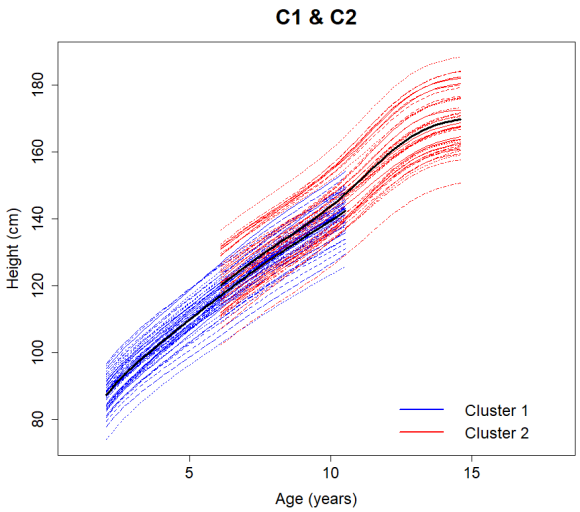


Figure 38: Les deux clusters

Nous pouvons faire les mêmes représentations pour la vitesse de croissance.

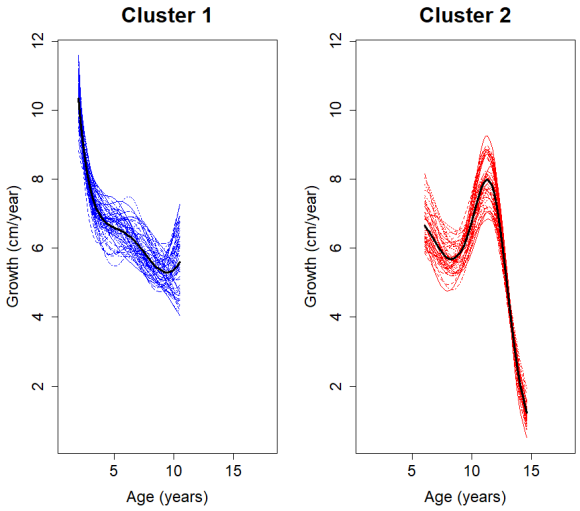


Figure 39: Courbes de vitesse de croissance pour chaque cluster

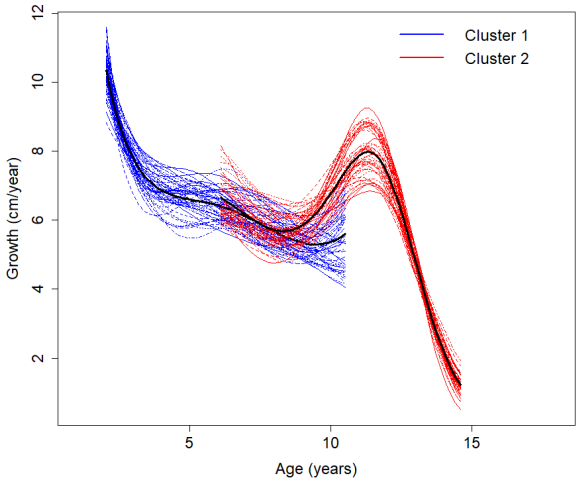


Figure 40: Courbes de vitesse de croissance

La ligne noire représente la courbe moyenne pour chaque cluster. Nous pouvons également les représenter (Voir figure 41).

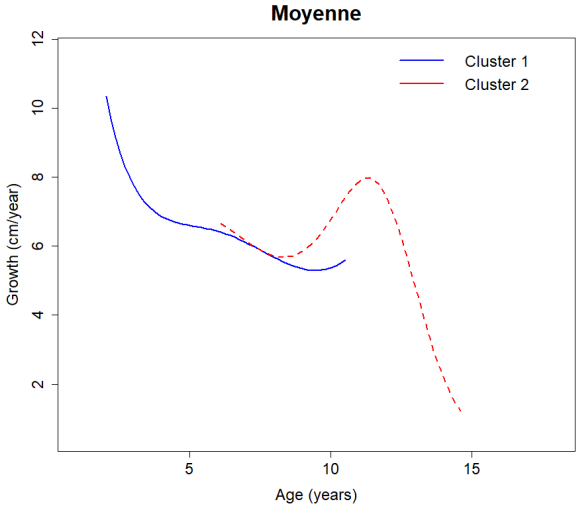


Figure 41: Courbes moyenne de vitesse de croissance

5.3.2 Application au données "Italian Covid-19": Nous considérerons la base `regions` qui contient le nombre quotidien de décès (maximum de la mortalité différentielle entre les données officielles de l'ISTAT et de la DPC) liés à la pandémie covid-19, pour 20 régions italiennes du 25 février 2020 au 18 juin 2021 (un total de 150 jours).
La base `regions` est une liste de trois éléments:

- `wave0`: Mortalité différentielle pour la période du 25 février 2020 au 18 Juin 2021
- `wave1`: Mortalité différentielle de la 1ere vague (du 25 février au 04 mai 2020)
- `wave2` : Mortalité différentielle de la 2eme vague (du 1er octobre 2020 au 27 février 2021)

Lissage des données

Les données ont été lissées en utilisant une B-spline (splines cubiques, nœuds à chaque jour, pénalité de rugosité sur la dérivée seconde et paramètre de lissage choisi par validation croisée généralisée moyenne). Les courbes lisses obtenues sont présentées à la figure 42.

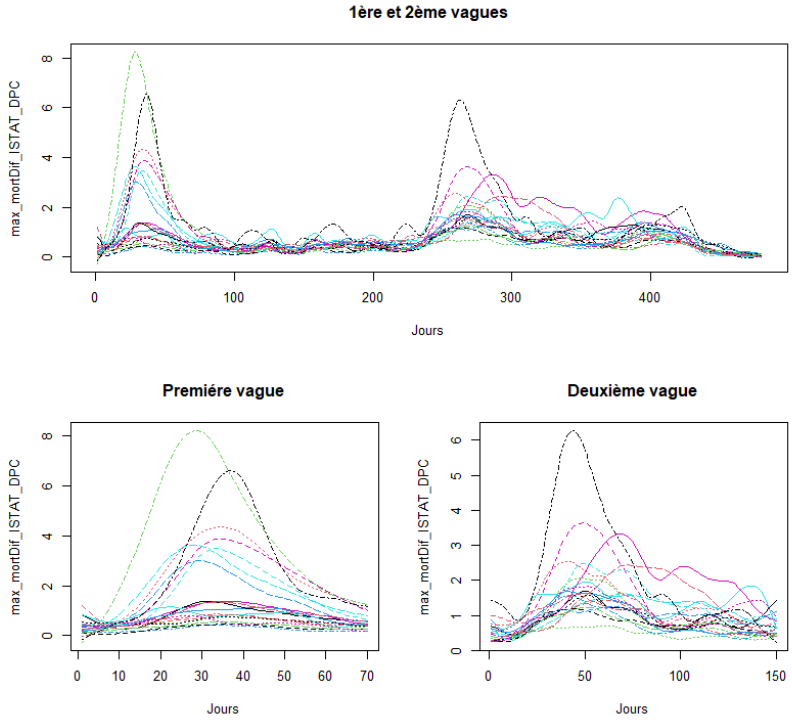


Figure 42: Les 20 courbes lisses obtenues

Clustering local probabiliste

Pour le clustering local, nous allons représenter les résultats ProbKMA obtenus avec une longueur de $c = 65$ jours pour les données de la 1ère vague et $c = 120$ jours pour celles de la deuxième vague. Comme nombres de clusters, nous avons choisis $K = 2$ et $K = 3$ (uniquement pour la deuxième vague).
Pour chaque vague, nous aurons les trois graphiques suivants:

- 1. Les courbes lisses obtenues
- 2. Le centre de cluster (courbe noire) avec des portions de courbes alignées en cluster;
- 3. L'alignement entre les portions de courbes au sein de chaque cluster (jour de début de chaque portion).

Une évaluation des clusters, via l'indice de silhouette généralisé, sera faite pour chaque clustering.

Données de la 1ère vague avec c = 65 jours

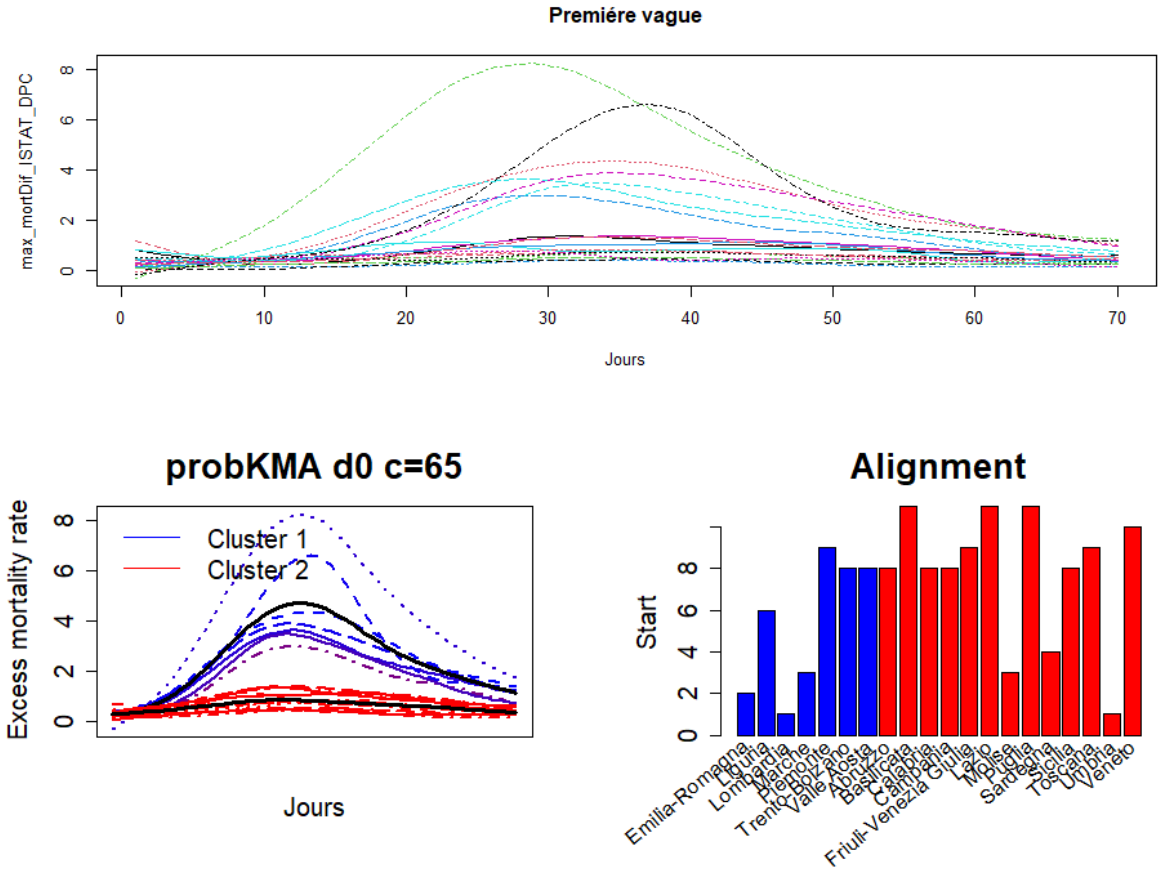


Figure 43: 1ere vague avec c= 65

Evaluation du clustering

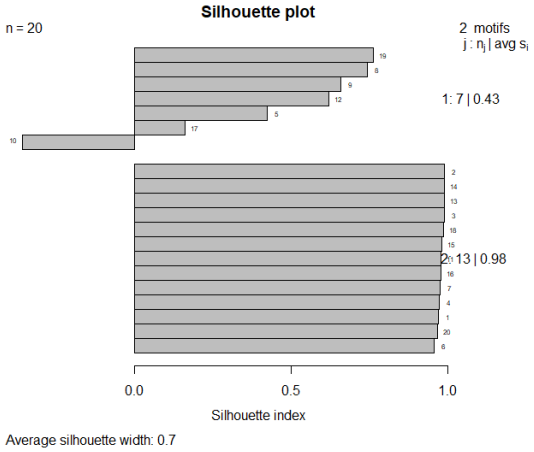


Figure 44: Graphique Silhouette du clustering avec c = 65

Par rapport aux 7 portions du cluster 1, les 13 portions du cluster 2 sont les mieux classées selon l'indice de silhouette (0.43 pour le cluster 1 et 0.98 pour le cluster 2). Toutes les portions du cluster 2 sont très bien classées (elles ont des $s_j \approx 1$). Par contre, certaines portions du cluster 1 ne le sont pas. C'est le cas, par exemple, de la portion 10 qui a un indice de silhouette négatif.

Clustering local avec les données de la 2eme vague

Données de la 2eme vague avec c = 120 jours et K = 2

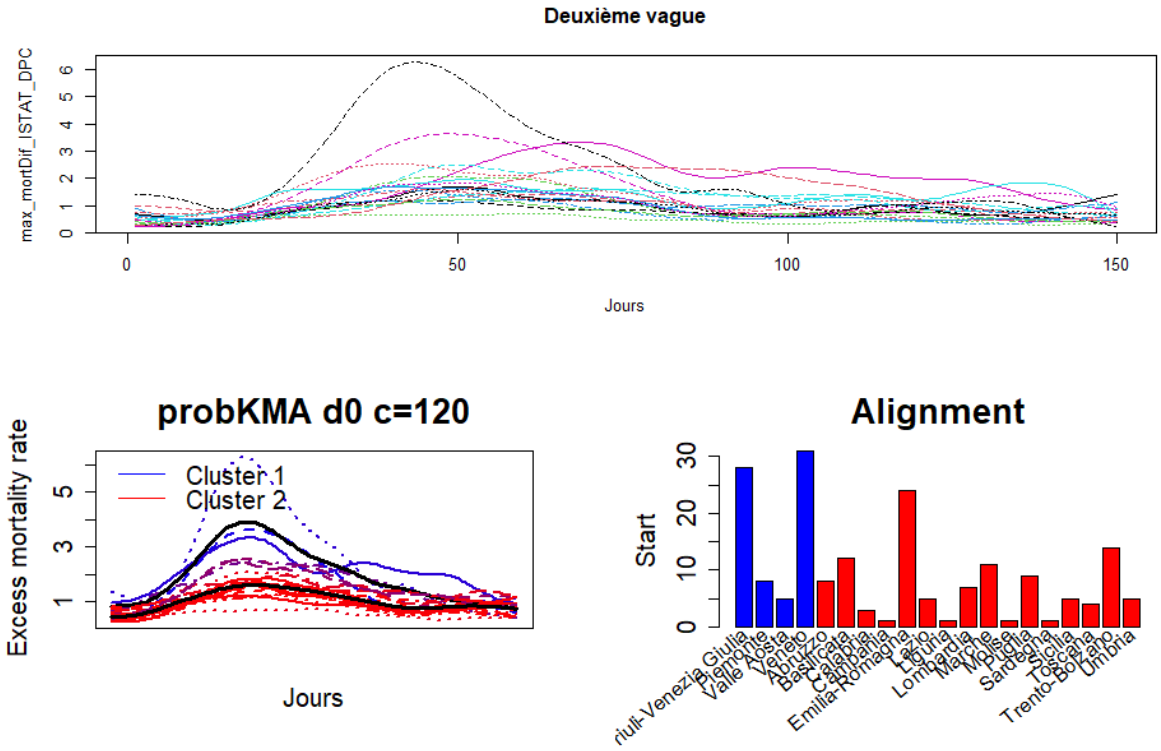


Figure 45: 2eme vague avec c= 120 et K = 2

Evaluation du clustering

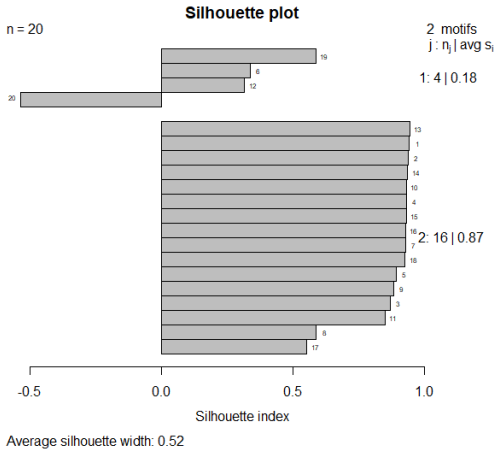


Figure 46: Graphique Silhouette du clustering avec c = 120 et K = 2

Avec un indice de silhouette moyen global de $S = 0.52$, la qualité globale du clustering est acceptable. La portion 20 (cluster 1), ayant un indice négatif, est mal classée. 14 parmi les 16 portions du cluster 2, sont très bien classées (avec des indices ≈ 1).

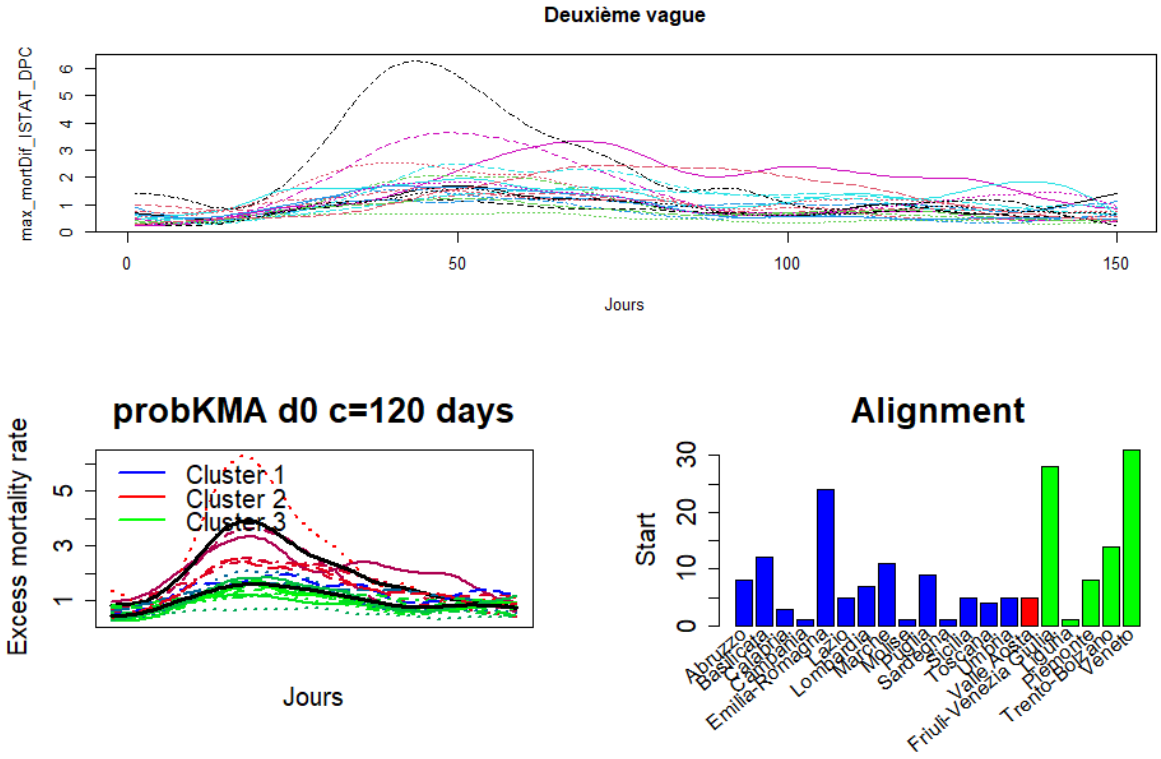


Figure 47: 2eme vague avec c= 120 et K = 3

Evaluation du clustering

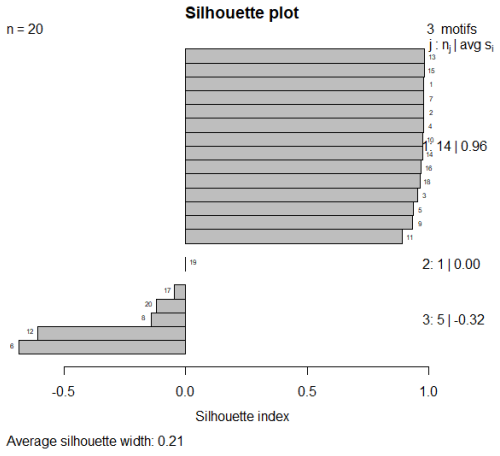


Figure 48: Graphique Silhouette du clustering avec c = 120 et K = 3

La qualité globale du clustering est discutable selon l'indice de silhouette qui vaut $S = 0.21$. Toutes les 14 portions du cluster 1 sont très bien classées. Par contre, celles des clusters 2 et 3 ne le sont pas, elles ont toutes des indices négatifs.

Recherche de motifs avec les données des deux vagues

Nous allons considérer les données de l'ensemble des deux vagues:

- Données de la première vague: du 25 février 2020 au 04 mai 2020.
- Données de la deuxième vague: du 01 octobre 2020 au 27 février 2021.

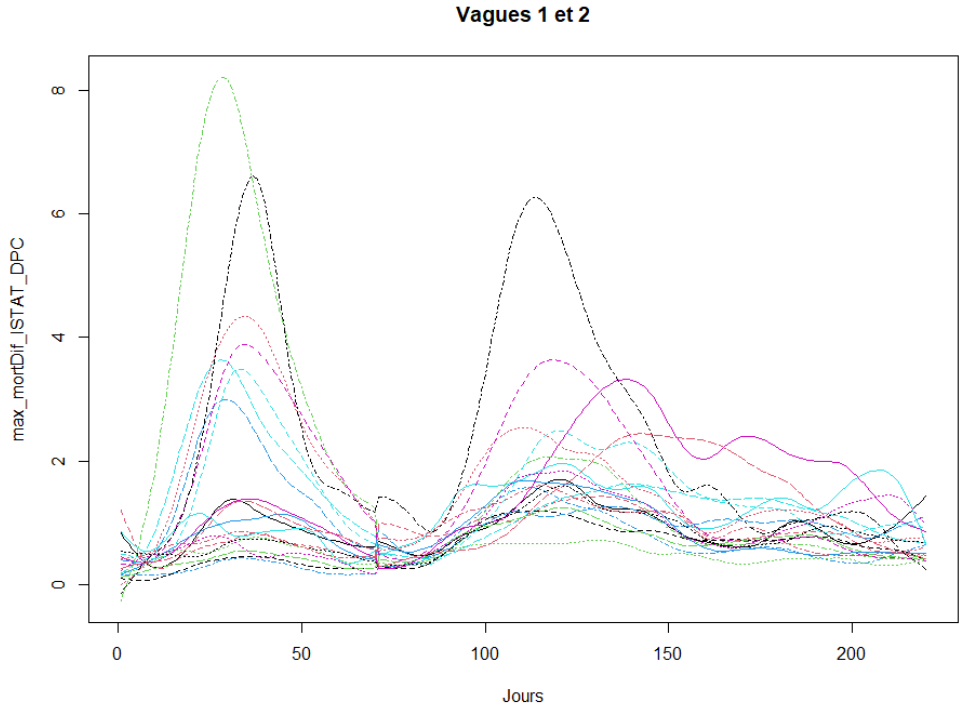


Figure 49: Données des deux vagues Covid-19 dans 20 régions d'Italie

Après plusieurs initialisations pour la recherche et le filtre de motifs candidats (avec 10 initialisations aléatoires à essayer, $K = c(2, 3)$ et $c = c(70, 65, 60)$, ce qui fait $2 \times 3 \times 10 = 60$ éxécutions de ProbKMA; avec un seuil de 0.3 sur l'indice de silhouette, correspondant à une probabilité de 90%; avec un seuil de 5 morceaux de courbes dans chaque motif), nous avons décidé de couper le dendrogramme obtenu à la distance motif-motif de `2*R_all = 2*0.007 = 0.014`. Ce qui nous permet d'avoir au total 6 motifs (résultat plus intéressant).

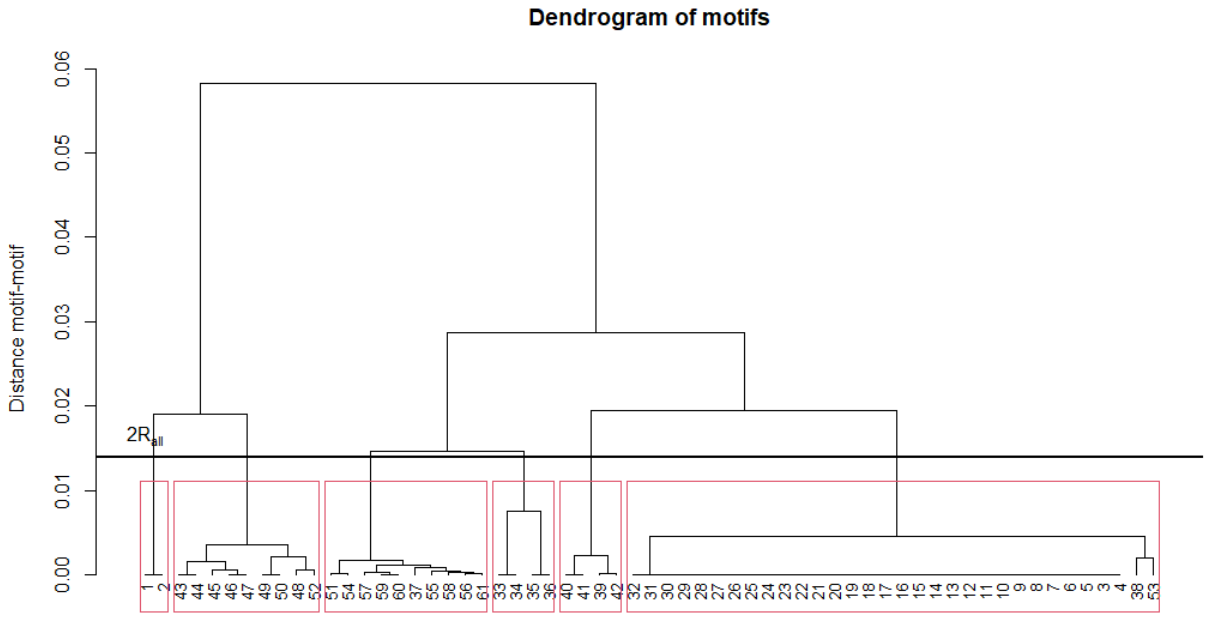


Figure 50: Dendrogramme des motifs

Les 6 motifs obtenus (visualisés sur le graphique 51) sont de longueurs différentes. Pour rappel, nous avions opté au niveau des paramètres, une longueur minimale de 60 jours (2 mois) pour les motifs. Certains des motifs obtenus sont de grande taille. C'est le cas des motifs 4 et 6, par exemples, qui ont des tailles de plus de 200 jours. Ceci s'explique par l'élongation faite par l'algorithme.

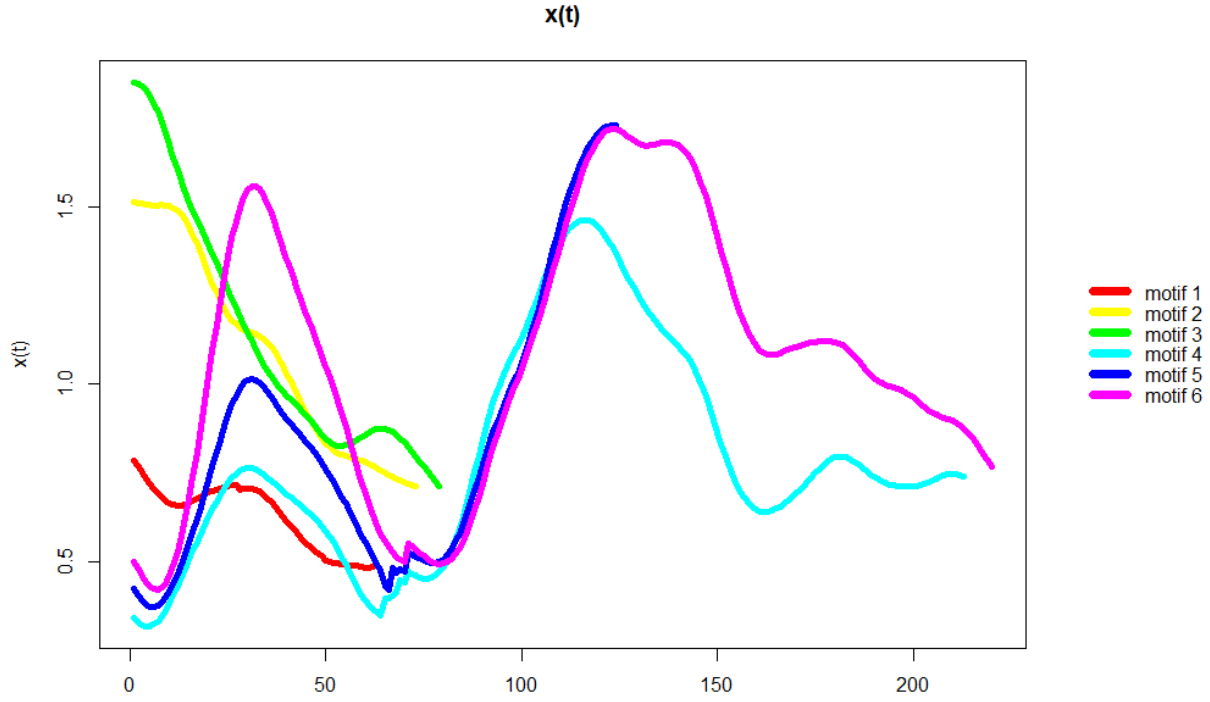


Figure 51: Les 6 motifs obtenus

Nous pouvons voir les détails via la liste donnée par `V_occurrences`. Par exemple, pour le motif 5, nous avons:

Table 15: Motif 5

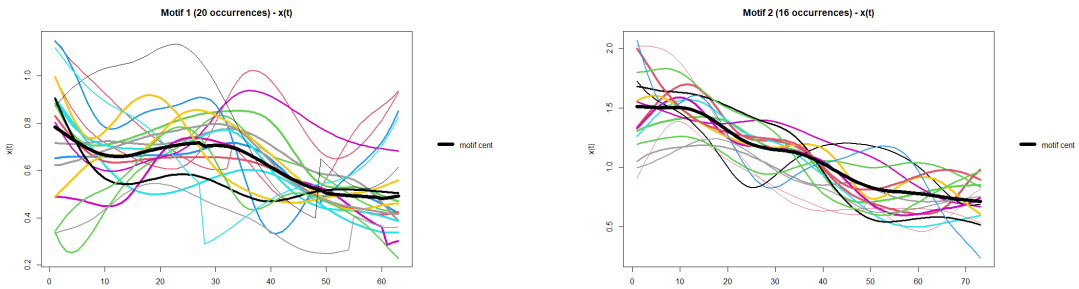
Région	Jour
Abruzzo	28-02-2020
Basilicata	29-02-2020
Puglia	29-02-2020
Sardegna	25-02-2020
Toscana	25-02-2020
Umbria	25-02-2020
Veneto	02-03-2020

Ici, nous constatons que le motif 5 concerne uniquement la première vague. Parmi ces 7 régions présentant ce motif, une seule se situe au Nord d'Italie (Veneto). La lecture du tableau nous montre, par exemple, sur la courbe de la région Puglia, nous y avons le motif 5 qui commence en date du 29 février 2020. Le fait que la date de début du motif 5 diffère d'une région à une autre (motif non aligné), prouve que la pandémie n'a pas commencé au même moment dans toutes les régions d'Italie.
Prenons le motif 3. Nous avons le tableau suivant:

Table 17: Motif 3

Région	Jour
Abruzzo	13-11-2020
Basilicata	20-11-2020
Liguria	29-11-2020
Puglia	13-11-2020
Sardegna	09-11-2020
Toscana	11-11-2020
Trento-Bolzano	11-12-2020
Umbria	14-11-2020

Nous remarquons que 8 des 20 courbes contiennent le motif 3 qui concerne uniquemet la deuxième vague (prinicipalement au mi-novembre 2020). A part la région Toscana qui se trouve au centre d'Italie, toutes les autres régions se situent au Sud d'Italie. Nous voyons, par exemple, sur la courbe représentant la région Abruzzo, le motif 3 à partir du 13 novembre 2020. Cette même courbe contient aussi le motif 5 à partir du 28 février 2020 (voir tableau précédent).
Le graphique 52 nous fourni une visualisation de chacun des 6 motifs avec leur occurrence (nombre de courbes sur lesquelles apparait le motif). Les courbes noires (respectivement, courbes colorées) représentent les motifs obtenus (respectivement les occurrences du motif). La largeur de ligne est proportionnelle à la similarité entre l'ocurrence et le motif.



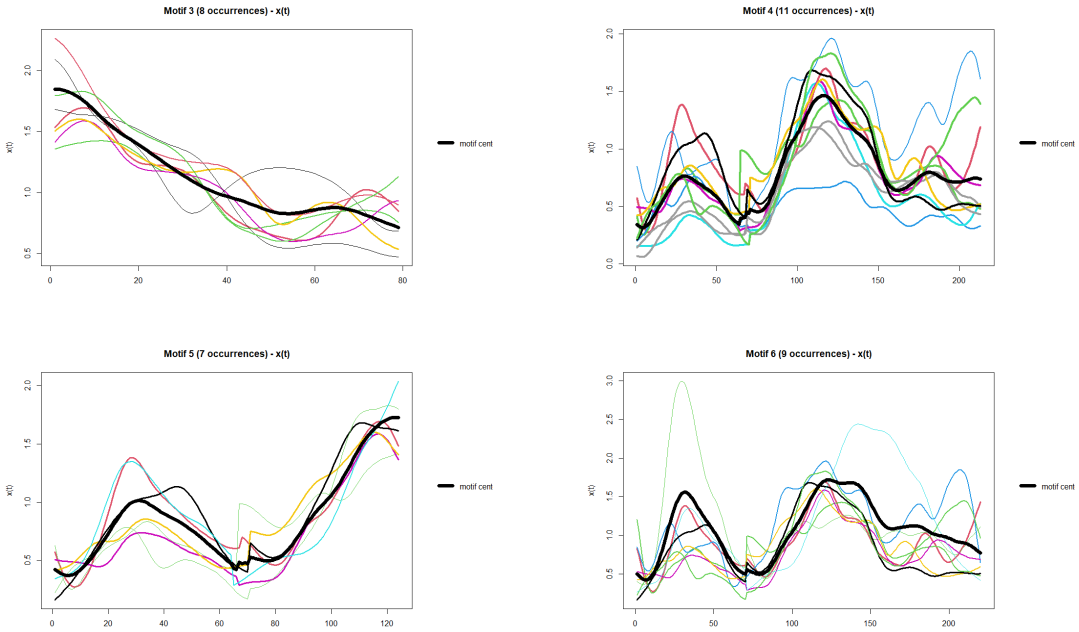


Figure 52: Motifs et occurrences

Conclusion

Devenue un moyen trés utilisé à l'ère numérique, l'analyse des données fonctionnelles (ADF) permet une meilleure aide à la prise de décision lorsque nous avons des données continues à notre disposition. Dans ce mémoire, nous avons vu des différentes méthodes d'analyse de données fonctionnelles (lissage, alignement, clustering par K-moyennes et clustering par K-moyennes avec alignement global). En suite, nous avons présenté une nouvelle approche de clustering des données fonctionnelles appelée ProbKMA (problème d'optimisation, résolution, algorithme, évaluation, description des fonctions et exemples pratiques) qui prend en compte la forme locale des courbes.
La méthode probKMA, se basant sur des idées d'ADF, est proposée pour faire un clustering local des courbes et pour découvrir des motifs candidats qui apparaissent plusieurs fois dans un ensemble de courbes. De plus, un indice de silhouette généralisé a été mis en oeuvre afin d'évaluer les résultats de ProbKMA; de même une étape de post-traitement pour fusionner les motifs candidats qui ont été decouverts plusieurs fois et rechercher les occurrences de motifs le long des courbes. Sous le logiciel libre R, un package nommé probKMA.FMD permettant d'effectuer cette méthode est actuellement en construction.
Pour chaque méthode d'ADF présentée, nous avons effectué une étude pratique via le logiciel libre R en présentant les différentes codes d'éxécution et les bases de données utilisées. Principalement, l'exemple pratique via les données "Italian Covid-19" nous a permis de bien comprendre les différentes fonctions du package probKMA.FMD qui est actuellement en construction. A travers cet exemple pratique, nous avons constaté que lorsque la longueur minimale du motif est proche de la longueur des courbes considérées, ProbKMA devient une version probabiliste de K-moyennes avec alignement global des courbes.
Malgré une longue durée recherche de motifs (environ 1h de temps) principalement dû à l'élongation que nous avons faite (nous avons commencé par des motifs de longueur 60 jours et à la fin nous avons, par exemple, des motifs de longueur de plus de 200 jours), les résultats obtenus sont satisfaisants. Ils prouvent que les fonctions de probKMA.FMD sont efficaces. Ainsi, en terme de perspectives, une amélioration et optimisation des codes des fonctions seront importantes pour plus d'efficacité sur le plan informatique.

Annexes

```
Console ~/
> print(kmeans(x = df, centers = 4, nstart = 25))
K-means clustering with 4 clusters of sizes 8, 13, 16, 13

Cluster means:
      Murder    Assault   UrbanPop        Rape
1  1.4118898  0.8743346 -0.8145211  0.01927104
2  0.6950701  1.0394414  0.7226370  1.27693964
3 -0.4894375 -0.3826001  0.5758298 -0.26165379
4 -0.9615407 -1.1066010 -0.9301069 -0.96676331

Clustering vector:
       Alabama         Alaska        Arizona       Arkansas     California
             1              2              2              1              2
      Colorado    Connecticut       Delaware        Florida        Georgia
             2              3              3              2              1
        Hawaii          Idaho       Illinois        Indiana           Iowa
             3              4              2              3              4
        Kansas       Kentucky      Louisiana          Maine       Maryland
             3              4              1              4              2
 Massachusetts       Michigan      Minnesota    Mississippi       Missouri
             3              2              4              1              2
       Montana       Nebraska         Nevada  New Hampshire     New Jersey
             4              4              2              4              3
    New Mexico       New York North Carolina   North Dakota           Ohio
             2              2              1              4              3
      Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
             3              3              3              3              1
  South Dakota      Tennessee          Texas           Utah        Vermont
             4              1              2              3              4
      Virginia     Washington  West Virginia      Wisconsin        Wyoming
             3              3              4              4              3

Within cluster sum of squares by cluster:
[1]  8.316061 19.922437 16.212213 11.952463
 (between_SS / total_SS =  71.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> |
```

Image 1: Résultats obtenus suite à l'exécution de kmeans

Références

Abraham, C., P. A. Cornillon, E. Matzner-Løber, and N. Molinari. 2003. "Unsupervised Curve Clustering Using B-Splines." *Scandinavian Journal of Statistics* 30 (3): 581–95. https://doi.org/10.1111/1467-9469.00350.

Boor, C. 1986. "B(Asic)-Spline Basics." *Undefined*. /paper/B(asic)-Spline-Basics.-Boor/982063b839314c85ddc0e8be2e53ce3fd6ba1e5e.

Boullé, Marc. 2012. "Functional Data Clustering via Piecewise Constant Nonparametric Density Estimation." *Pattern Recognition* 45 (12): 4389–4401. https://doi.org/10.1016/j.patcog.2012.05.016.

Bouveyron, Charles, and Camille Brunet-Saumard. 2014. "Model-Based Clustering of High-Dimensional Data: A Review." *Computational Statistics & Data Analysis* 71 (March): 52–78. https://doi.org/10.1016/j.csda.2012.12.008.

Cremona, Marzia A., and Francesca Chiaromonte. 2020. "Probabilistic K-Mean with Local Alignment for Clustering and Motif Discovery in Functional Data." *arXiv:1808.04773 [Stat]*, July. http://arxiv.org/abs/1808.04773.

Delaigle, Aurore, and Peter Hall. 2010. "Defining Probability Density for a Distribution of Random Functions." *Ann. Statiste.* 38 (2): 1171–93. https://doi.org/10.1214/09-AOS741.

Ferraty, Frédéric, and Philippe Vieu. 2006. *Analyse des données fonctionnelles non paramétriques, Texte d'origine: Théorie et pratique*. Série Springer en statistiques. New York: Springer-Verlag. https://doi.org/10.1007/0-387-36620-2.

Ieva, Francesca, Davide Pigoli, Anna Paganoni, and Valeria Vitelli. 2013. "Multivariate Functional Clustering for the Morphological Analysis of ECG Curves." *Journal of the Royal Statistical Society Series C Applied Statistics* 62 (April): 401–18. https://doi.org/10.1111/j.1467-9876.2012.01062.x.

Jacques, Julien, and Cristian Preda. 2014. "Functional Data Clustering: A Survey." *Adv Data Anal Classif* 8 (3): 231–55. https://doi.org/10.1007/s11634-013-0158-y.

James, Gareth M., and Catherine A. Sugar. 2003. "Clustering for Sparsely Sampled Functional Data." *Journal of the American Statistical Association* 98 (462): 397–408. https://doi.org/10.1198/016214503000189.

"K Means Clustering | K Means Clustering Algorithm in Python." 2019. *Analytics Vidhya*.

Kokoszka, P., and Matthew Reimherr. 2017. *Introduction to Functional Data Analysis*. Texts in Statistical Science Series. Boca Raton: CRC Press, Taylor & Francis Group.

Peng, Jie, and Hans-Georg Müller. 2008. "Distance-Based Clustering of Sparsely Observed Stochastic Processes, with Applications to Online Auctions." *Ann. Appl. Stat.* 2 (3): 1056–77. https://doi.org/10.1214/08-AOAS172.

Ramsay, J. O., Giles Hooker, and Spencer Graves. 2009. *Functional Data Analysis with R and MATLAB*. Use R! Dordrecht ; New York: Springer.

Ramsay, J. O., and Xiaochun Li. 1998. "Curve registration." *Journal of the Royal Statistical Society Series B* 60 (2): 351–63.

Ramsay, J. O., and B. W. Silverman. 2005. *Functional Data Analysis*. 2nd ed. Springer Series in Statistics. New York: Springer.

Sangalli, Laura M., Piercesare Secchi, Simone Vantini, and Valeria Vitelli. 2010. "K-Mean Alignment for Curve Clustering." *Computational Statistics & Data Analysis* 54 (5): 1219–33.

Serban, N., and Larry Wasserman. 2005. "CATS: Clustering After Transformation and Smoothing." *Journal of the American Statistical Association* 100 (February): 990–99. https://doi.org/10.2307/27590629.