INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

The University Of Sheffield.

Rapport de stage de 4ème année

Génie Mathématiques Appliquées

Année scolaire 2021 - 2022

Etudiante : Alexia GRENOUILLAT,
grenouil@etud.insa-toulouse.fr

Maître de stage : M.Arthur LEROY,
arthur.leroy.pro@gmail.com

Date de début de stage : 6 Juin 2022
Date de fin de stage : 23 Septembre 2022

Lieu : Université de Sheffield

Contents

1 Présentation générale

J'ai choisi de réaliser mon stage de fin de 4ème année au Royaume-Uni au sein de l'Université de Sheffield, sous la tutelle d'Arthur Leroy, post-doctorant en statistiques et Machine Learning et chercheur associé à l'Université de Manchester. Plus précisément, j'ai travaillé dans le département de computer science et ai été en contact avec les doctorants du laboratoire. J'ai notamment pu échanger avec certains d'entre eux sur leurs travaux en cours et ai assisté à un séminaire sur la data science à Sheffield.

Ayant pour objectif de travailler dans les sciences du sport une fois mon diplôme INSA obtenu, le stage proposé par M. Leroy s'est présenté comme une parfaite opportunité pour mettre un pied dans ce milieu et savoir s'il correspondait à mes aspirations. En particulier, j'ai travaillé avec Hugo Lelièvre sur une base de données fournie par la Fédération Française de Natation (FFN), que M. Leroy avait lui-même exploitée pour développer son package R MagmaClustR durant sa thèse. La problématique apportée par la FFN en 2017 était la suivante : identifier les jeunes nageurs prometteurs pour le sport de haut niveau grâce à des méthodes statistiques et / ou des algorithmes de Machine Learning, et développer un outil d'aide à la décision pour la fédération. Cette base de données a constitué un fil rouge illustratif tout au long de mon stage et a soulevé un problème qui constitue le coeur de l'études des données fonctionnelles (FDA) : l'étude des multiples séries temporelles supposées partager de l'information commune, et généralement observées à pas de temps irréguliers.

Les travaux actuels de M. Leroy reposent en majeure partie sur les processus gaussiens (GPs) et les algorithmes de Machine Learning. Les objectifs de ce stage étaient donc multiples :

- enrichir mes connaissances (encore très rudimentaires) en GPs. Sheffield étant l'un des berceaux de ces derniers, le laboratoire dans lequel j'ai effectué mon stage est donc l'endroit idéal pour apprendre à les maîtriser ;
- continuer à manipuler des algorithmes de Machine Learning ;
- apprendre à manipuler correctement une base de données (BDD) ;
- améliorer mon niveau de programmation en langage R en découvrant de nouveaux packages ;
- développer mon esprit critique en analysant les sorties des différents algorithmes ;
- prendre des initiatives tout en respectant les tâches assignées par mon maître de stage.

1.1 Etat de l'art : outils de prédiction utilisés pour les données fonctionnelles

Dans le domaine des sciences du sport (et, plus précisément, au niveau élite), l'un des principaux enjeux repose sur le repérage de "jeunes talents prometteurs". Avec le développement des outils et méthodes d'entraînement ainsi que la professionnalisation du sport de haut niveau, les écarts de performance entre les athlètes en compétition deviennent de plus en plus ténus. Les fédérations sportives ont tout intérêt à ce que la détection soit ciblée et fructueuse, afin de proposer à ces jeunes sportifs d'intégrer des structures favorisant la performance de haut niveau.

Même si de nombreux éléments influencent la performance sportive (comme les qualités physiques naturelles, le nombre d'heures d'entraînement, la structure dans laquelle évolue le sportif, son niveau de fatigue ...), plusieurs travaux (*ie* Ericsson et al., 2018) soulignent que l'évolution d'un athlète au cours du temps est plus adaptée que les valeurs brutes à des âges donnés pour prédire les performances futures d'un athlète. Tous les sportifs ne progressent pas de la même manière ; il semble donc important de tenir compte des

différents types de progression si l'on souhaite améliorer la qualité des outils de détection de talents.

Dans l'optique de modéliser la progression des performances des sportifs de haut niveau, et potentiellement prédire leurs performances futures, il semble donc naturel de modéliser ce phénomène continu comme une fonction aléatoire du temps. Formellement, la courbe d'évolution des performances est vue comme un processus stochastique, et les données de la base FFN sont simplement des observations d'un objet infini dimensionnel. Tout au long du stage, j'ai eu à manipuler des données dites fonctionnelles, que l'on peut définir ainsi :

Définition - **Variable aléatoire fonctionnelle :** Variable aléatoire qui prend ses valeurs dans un espace vectoriel de dimension infinie : $X : \Omega \to \mathcal{F}$, où $\mathcal{F}$ est un espace de fonctions (un espace de Banach séparable).

Définition - **Donnée fonctionnelle :** Réalisation d'une variable aléatoire fonctionnelle X.

Dans le contexte ainsi défini, les Processus Gaussiens (GPs) apparaissent comme étant des outils mathématiques particulièrement adaptés. D'une part, ils proposent une procédure analytique avec une prédiction a posteriori exprimée sous la forme d'une distribution gaussienne. Ils fournissent également une base pratique pour élaborer des modèles plus complexes. D'autre part, la définition de la fonction de covariance permet à elle seule d'exprimer une large palette d'hypothèses (Duvenaud, 2014) concernant les interactions entre les données. De plus, ces hypothèses de modélisation sont encodées dans des noyaux de covariance comportant, dans la plupart des cas, un nombre limité de paramètres à estimer (pour le Squared Exponential (SE), on n'en compte que deux). Ce nombre limité de paramètres permet de réduire la complexité de l'optimisation et d'éviter le piège du surajustement. Toutefois, si ce cadre élégant des GPs présente de nombreux avantages du point de vue de la modélisation, sa mise en oeuvre pratique a aussi un coût élevé.

Malgré leurs nombreuses qualités, les GPs souffrent également de limitations qui empêchent une applicabilité pratique plus large. Comme mentionné précédemment, le large artéfact de propriétés pouvant être modélisées par les GPs est très corrélé au choix du noyau de covariance. Si les GPs permettent d'apporter une grande flexibilité grâce au choix des noyaux, ils introduisent également une difficulté supplémentaire, qui est de savoir quelle forme de noyau est la plus adaptée à chaque structure particulière de données. Par ailleurs, l'étape d'apprentissage des GPs nécessite l'inversion d'une matrice pouvant conduire à des problèmes numériques (matrice proche de la singularité par exemple). Si la matrice de covariance à inverser est de taille $N * N$, une complexité en $\mathcal{O}(N^3)$ est nécessaire pour l'apprentissage. Il faut également rajouter à cela un produit matrice-vecteur en $\mathcal{O}(N^2)$ dans l'étape de prédiction. Ainsi, la complexité des méthodes GPs augmente très rapidement en fonction du nombre de données. La littérature considère que les GPs peuvent être appliqués "directement" sur des bases de données de taille modérée (*ie* environ 10^4 observations). La base de données FFN comptant plusieurs millions de lignes, cet aspect "complexité" constitue donc un véritable frein à l'utilisation des GPs classiques. Des méthodes d'approximation sparse ont été développées dans la littérature pour pallier à ce problème ; elles consistent principalement à introduire des "pseudo-inputs" $u = \{u_1, ..., u_n\}$ comme des évaluations latentes des GPs aux temps t_u. Ces pseudo-inputs sont supposés être bien moins nombreux que les inputs initiaux des GPs ($n << N$). Cela permet de réduire l'inversion de la matrice $n * n$ de covariance à une complexité de $\mathcal{O}(Nn^2)$, et le produit matrice-vecteur à $\mathcal{O}(n^2)$.

1.2 Utilisation de modèles multi-tâches avec partage d'information : Magma et MagmaClust

Comme notre dataset contient un nombre important d'individus (et chaque individu étant observé de l'ordre de 10^1 fois), la définition d'un modèle multi-tâches offre la possibilité de partager de l'information

entre les individus et permet de résoudre le problème de coût induit par les GPs. C'est l'approche qui a été choisie par M. Leroy à travers Magma (Multi tAsk Gaussian process with coMmon meAn), un modèle de GPs multi-tâches. Grâce à ce cadre probabiliste non-paramétrique, on peut définir une loi a priori sur les fonctions qu'on suppose avoir généré les données de plusieurs individus. La nouveauté de son approche consiste à introduire un GP moyen, commun à tous les individus, dont la distribution hyperposterior est calculable. Ce GP moyen fournit également à chaque individu une moyenne a priori contenant de l'information sur toute la base de données, permettant ainsi d'améliorer la capacité de prédiction du modèle. Ce partage d'informations communes entre les individus à travers le GP moyen offre une modélisation plus complète que celle d'un GP classique et une prise en compte de l'incertitude. La complexité du modèle est également réduite à $\mathcal{O}(MN^3)$, où M correspond au nombre d'individus que Magma utilise pour partager de l'information.

Afin de pouvoir tenir compte des différents types de progression des nageurs, et ainsi améliorer la qualité de la prédiction, M. Leroy a également proposé un prolongement de Magma via la définition d'un mélange de GPs multi-tâches : MagmaClust. Pour pallier à l'hypothèse d'un unique GP moyen sous-jacent, qui peut paraître trop restrictive, il étend le modèle Magma pour prendre en compte la possible présence de structures de groupes dans les données. Il introduit ainsi une composante de clustering dans la procédure, qui repose sur des mixtures de gaussiennes. Ainsi, la combinaison d'un modèle de mélange de GPs avec les avantages de l'aspect muti-tâches offre des capacités de prédiction améliorées grâce au partage d'information entre les individus via plusieurs processus moyens spécifiques au cluster.

Mon stage a donc principalement gravité autour de la compréhension de ces deux méthodes, Magma et MagmaClust, ainsi que de leur implémentation sous forme de package R. Au cours de la première semaine, j'ai essentiellement lu et essayé de comprendre la thèse de M. Leroy ainsi que les deux articles qu'il a publiés à ce sujet (*MAGMA: inference and prediction using multi-task Gaussian processes with common mean* et *Cluster-Specific Predictions with Multi-Task Gaussian Processes*). Le premier jour, M. Leroy a fait une séance générale d'explications au tableau afin d'expliquer les grandes lignes du modèle et donner des justifications mathématiques aux étapes intermédiaires. J'ai ainsi pu lui poser toutes les questions qui me sont venues lors des explications plus théoriques, ce qui m'a permis d'appréhender la lecture des articles avec tous les outils nécessaires à leur compréhension. Je propose ci-après de faire un résumé rapide des grandes étapes de Magma afin d'explicter plus clairement le fonctionnement de l'algorithme.

Grandes lignes de l'algorithme Magma.
Comme tout algorithme destiné à réaliser de la prédiction, l'algorithme Magma se décompose en 3 grandes étapes :

- L'entraînemment du modèle grâce à la fonction **train_magma()**. On précise :
  - La base de données sur laquelle on réalise l'entraînement du modèle ;
  - Le noyau de covariance pour les GPs relatifs aux individus ;
  - Le noyau de covariance pour le GP moyen ;
  - Les hyper-parameters initiaux des noyaux de covariance (dans mon cas, je les initialise toujours aléatoirement) ;
  - Si tous les individus partagent les mêmes hyper-paramètres ou non.

  train_magma fait appel à un algorithme d'Expectation-Maximisation (EM) afin de calculer les hyper-posteriors (voir **Section 3 - Paragraphe 4** pour le détail de l'algorithme en question).
- Une fois l'entraînement du modèle terminé, on peut prédire les performances futures de l'individu de notre choix grâce à la fonction **pred_magma()**. Il nous faut préciser plusieurs paramètres :
  - Les données du nageur pour lequel on va réaliser la prédiction ;
  - La grille de temps pour laquelle on souhaite avoir notre prédiction ; pour les nageurs, entre 10 et 20 ans.
  - Le modèle précédemment entraîné par **train_magma()**.
- L'affichage des résultats grâce à la fonction **plot_magma()**. Plusieurs options d'affichages sont disponibles : pour un nageur à prédire, on peut choisir d'afficher son processus individuel avec l'intervalle de crédibilité à 95% associé, ou bien une heatmap de probabilités pour avoir une idée plus précise de la confiance que l'on accorde à la prédiction.

2 Prise en main des données FFN et manipulation du package MagmaClustR

2.1 Contenu de la base de données

Une fois achevée l'étape préliminaire de lecture des travaux, j'ai commencé à explorer la BDD FFN afin de me familiariser avec les données (et, plus particulièrement, leur format). Cette base est constituée de plusieurs millions de lignes ; chacune correspond à une performance réalisée sur le territoire français par un(e) nageur(se) entre 2002 et 2016 sur une course donnée. Chaque performance est décrite par quantité d'informations :

- **L'identifiant du nageur** constitué de 3 variables : le nom, le prénom et la date de naissance, afin d'éviter les homonymes ;
- **L'âge** du nageur au moment de la performance (en années) ;
- **L'épreuve** sur laquelle le nageur s'est aligné (*Ex :* 100m nage libre, 50m brasse, 400m 4 Nages ...). On comptabilise 35 épreuves différentes, dont 18 épreuves individuelles officielles ;
- **Le sexe** du nageur ;

- **La performance** du nageur, *ie* le temps total réalisé par le nageur sur une course donnée ;
- **Les temps intermédiaires** de la performance, *ie* les temps cumulés sur chaque 50m de la course. Ils ne sont pas toujours disponibles car tous les bassins ne disposent pas de plaques chronométriques à chaque mur ;
- **La taille du bassin :** petit (25m) ou grand (50m) ;
- **Le club / pôle** dans lequel le nageur est licencié ;
- **La compétition** au cours de laquelle la performance a été réalisée.

Notre objectif principal étant de prédire les performances des nageurs, il m'a fallu apprendre à filtrer correctement la base de données pour obtenir les informations dont j'avais besoin. En effet, l'atout principal de Magma est le partage d'informations entre les individus ; si l'on s'amuse à mélanger des performances réalisées sur des courses différentes, le partage d'informations n'est dès lors plus pertinent. De même, si l'on ne dissocie pas les femmes et les hommes, on risque d'avoir des problèmes en terme de prédiction. J'ai donc appris à utiliser les principales fonctions du **tidyverse** pour pouvoir traiter la base de données. Le principal avantage du **tidyverse** est qu'il permet de manipuler facilement les données ; sa syntaxe est simple d'utilisation et assez naturelle pour moi qui débutais sur le traitement de bases de données volumineuses.

2.2 Influence des hyper-paramètres du modèle

Afin de mieux comprendre l'influence du choix du noyau et du partage des hyper-paramètres dans Magma, j'ai réduit la base de données en ne gardant que les filles ayant réalisé au moins 10 performances sur le 100m nage libre entre 2002 et 2016. J'ai ensuite utilisé une fonction mise au point par Hugo Lelièvre pour supprimer les doublons. En effet, pour pouvoir utiliser Magma, il faut que chaque nageur ait des **Inputs** (ici, des âges) uniques. Or, dans la base de données FFN, il est fréquent que plusieurs performances soient associées au même nageur et aux mêmes âges. Ces doublons sont dus au fait que certains individus nagent la même course 2 fois dans la journée (une fois en séries, une fois en finale). La fonction créée par Hugo résout ce problème en ne conservant que la meilleure performance à un âge donné pour tous les nageurs de la base de données.

Une fois les doublons supprimés, j'ai fait tourner Magma sur une base d'environ 30 individus. Au début du stage, la version de l'algorithme n'était pas encore très optimisée en terme de temps de calcul ; si je voulais pouvoir le faire tourner fréquemment pour étudier l'influence de différents paramètres, il me fallait réduire considérablement le nombre de nageurs. De plus, même si les performances de Magma en terme de prédiction augmentent en fonction du nombre d'individus utilisés pour l'entraînement, 30 suffisent pour avoir une idée assez précise des différents comportements de l'algorithme.

2.2.1 Noyaux de covariance du GP moyen et des GPs individuels

Le choix des noyaux de covariance du modèle Magma influence significativement l'allure des courbes des différents GPs. 4 noyaux sont actuellement implémentés dans le package MagmaClustR (parmi les noyaux les plus fréquemment utilisés dans la littérature) :

- Le **Squared Exponential (SE)**, *ie* le noyau "par défaut" des GPs. Il dispose de bonnes propriétés, comme le caractère lisse des courbes qu'il engendre et le fait que son prior ait des dérivées infinies ;

- Le **Linear (LIN)**, utile si l'on observe des tendances linéaires chez nos individus. Contrairement à la majorité des autres noyaux, il est non-stationnaire (*ie* il dépend de l'emplacement absolu de 2 inputs, et pas uniquement de leur position relative) ;
- Le **Periodic (PERIO)**, qui est adapté aux données présentant un motif répétitif au cours du temps ;
- Le **Rational Quadatric (RQ)**, qui équivaut à additionner ensemble plusieurs noyaux SE avec des lengthscales différentes.

M. Leroy a également rendu possible la combinaison de ces 4 noyaux par le biais des opérateurs "+" (équivalent du "OU" logique) et "*" (équivalent du "ET" logique). J'ai donc dû réfléchir au noyau le plus adapté à notre base de données, en tenant compte des spécificités d'une saison de natation. En particulier :

- Les performances des nageurs décroissent avec l'âge, *ie* les nageurs s'améliorent en grandissant, ce qui semble logique puisqu'ils augmentent le nombre d'heures d'entraînement, gagnent en taille et en puissance musculaire ... La décroissance est très rapide dans les jeunes années (de 10 à 13 ans environ), puis les performances commencent rapidement à stagner. J'ai donc tenté de spécifier des noyaux "SE*LIN" et "SE+LIN" pour le GP moyen et les GPs individuels. Je pensais que la partie linéaire permettrait de rendre compte de la décroissance assez nette de la "première phase", puis que la partie SE prendrait la relève pour la suite. Malheureusement, cette combinaison de noyaux ne semble pas bien adaptée à nos nageurs ; l'aspect linéaire s'impose trop par rapport à la partie SE, ce qui force nos GPs à avoir des variations assez raides. Si l'on se contente simplement du noyau linéaire, c'est encore pire ; les GPs sont beaucoup trop rigides, et les prédictions atterrissent loin des performances réellement observées.
- Au cours d'une saison sportive (d'environ septembre à fin juillet), les performances des nageurs évoluent généralement périodiquement en fonction de leurs pics de forme. J'ai étudié l'allure des GPs obtenus en combinant les noyaux SE et PERIO. La version la plus performante est "SE*PERIO" ; si l'on affiche la heatmap du GP de l'individu à prédire, on remarque que l'incertitude est moins étendue qu'avec un simple noyau SE. Toutefois, cette combinaison tient trop compte de nos connaissances sportives, et il arrive (dans un petit nombre de cas) que la prédiction soit assez éloignée des performances réellement observées, ce qui n'est pas le cas avec un noyau SE traditionnel.
- Concernant le noyau RQ, j'ai vite abandonné l'idée de l'utiliser. En effet, il est difficile à manipuler, rallonge la compilation de Magma par rapport aux autres noyaux et n'aboutit pas à des résultats très satisfaisants.

Finalement, le noyau SE semble être le plus adapté à la base de données FFN. J'ai donc décidé de le garder dans toute la suite du stage.

2.2.2 Partage des hyper-paramètres entre les individus

Un autre hyper-paramètre qui influence grandement les GPs obtenus en sortie concerne le partage des hyper-paramètres entre les individus de la base de données.

- Si l'on décide que l'ensemble des hyper-paramètres est commun à tous les nageurs, on tire un meilleur parti de l'information provenant des individus d'entraînement par le biais du GP moyen. Ainsi, on modélise un contexte dans lequel les individus représentent différentes trajectoires du même processus.
- Si l'on décide que l'ensemble des hyper-paramètres est propre à chaque individu, on indique implicitement que les structures de covariance entre les individus ne sont pas toujours les mêmes. On définit ainsi un modèle plus flexible, mais qui exploite moins du partage d'information entre individus.

Il est important de noter que la spécification d'hyper-paramètres propres à chaque individu peut conduire à des problèmes numériques dans certains cas pathologiques. En effet, les inversions de matrice constituent un problème classique dans la littérature des GPs ; à cause du nombre potentiellement élevé d'hyper-paramètres différents à entraîner, la probabilité pour qu'au moins l'un d'entre eux conduise à une matrice presque singulière augmente. Dans ce cas précis, un individu pourrait submerger les autres dans le calcul de la moyenne hyper-postérieure et donc conduire à une sous-estimation de la variance postérieure. Ce problème ne se produit pas avec des hyper-paramètres communs grâce au partage d'information.

Ainsi, je me suis cantonnée pour la suite du stage à garder des hyper-paramètres communs pour tous les individus. En ayant fait tourner Magma avec des hyper-paramètres communs et non-communs, on remarque que dans la plupart des cas, les prédictions obtenues sont meilleures dans le cas des hyper-paramètres communs.

3 Définition d'un nouveau modèle : partage des hyper-paramètres communs au sein d'un même cluster

3.1 Généralités

Dans la suite, on appelle :

- "variable d'entrée" (*ie* Input) une grille de temps, que l'on peut assimiler à une série temporelle ;
- "variable de sortie" (*ie* Output) la variable que l'on souhaite observer ;
- "variable d'identification" (*ie* ID) la variable qui nous permet d'identifier anonymement les individus de la base de données.

Pour utiliser MagmaClust, il faut obligatoirement que notre base de données possède ces 3 variables (nommées "ID", "Input", "Output"). Des variables additionnelles peuvent être ajoutées à la base ; elles seront traitées comme des covariables. On précise également que les observations de notre série temporelle peuvent être régulières ou non. Ici, nous avons appliqué MagmaClust à 2 bases de données différentes :

- La base de données **FFN**. Ici, les Outputs correspondent aux performances des nageurs sur différentes courses ; les Inputs, à l'âge du nageur au moment de ladite performance. Comme tous les nageurs ne concourent pas aux mêmes âges (l'âge, en années, peut prendre jusqu'à 2 décimales), la grille de temps est très irrégulière ;
- Une base de données **poids**, qui regroupe les poids d'environ 350 enfants âgés de 0 à 72 mois. Ici, les Outputs correspondent aux poids des enfants et sont observés aux mêmes âges pour tous les enfants (certainement à cause des visites obligatoires chez le pédiatre). La grille de temps est donc ici régulière.

Afin de donner un aperçu de nos deux bases de données, on affiche sur les figures ci-dessous l'allure de 5 individus (nageurs Figure 1 (a), enfants Figure 1 (b)).

Notre objectif est réussir à produire, grâce à MagmaClust, des courbes d'évolution des performances des nageurs / du poids des enfants, et de prédire leurs futurs Outputs. On cherche également à tenir compte de la présence de structures de groupes dans les données à travers l'aspect "clustering" de MagmaClust.

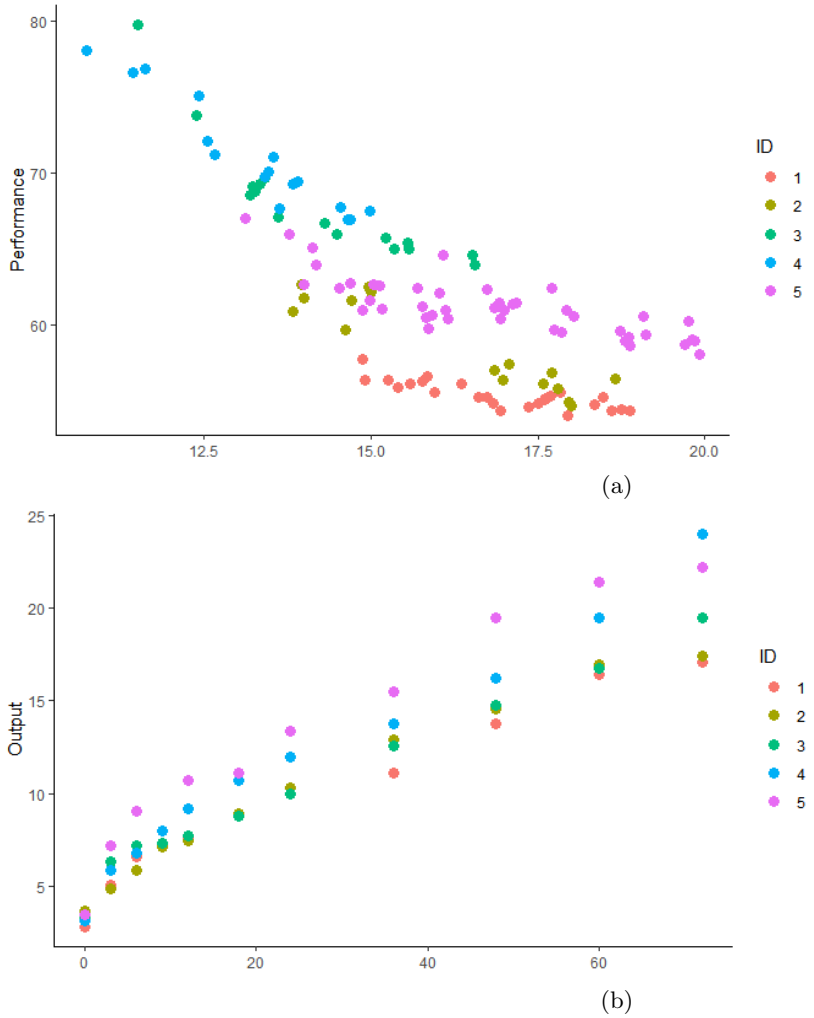


Figure 1:

(a) 5 nageurs sélectionnés aléatoirement dans la BDD FFN.
(b) 5 enfants sélectionnés aléatoirement dans la BDD poids.

3.2 Notations

Comme nous supposons que l'ensemble de données est composé d'observations ponctuelles de plusieurs fonctions, on note $I \subset \mathbf{N}$ l'ensemble de tous les indices (qui contient notamment $\{1, ..., M\}$, les indices des individus observés, *ie* d'entrainement). Les valeurs d'entrée étant définies sur un continuum, on nomme $\mathcal{T}$ cet espace d'entrée (dans notre cas, $\mathcal{T} \subset \mathbf{R}$). De plus, MagmaClust étant défini à des fins de clustering, on désigne par $\mathcal{K} = \{1, ..., K\}$ l'ensemble des clusters d'individus. On définit également les notations suivantes :

- $t_i = \{t_i^1, ..., t_i^{N_i}\}$ l'ensemble des pas de temps disponibles pour le i-ème individu (où N_i est le nombre d'observations pour cet individu) ;
- $\mathbf{y_i} = y_i(t_i)$ le vecteur d'Ouputs pour le i-ème individu ;
- $t = \bigcup_{i=1}^{M} t_i$ l'ensemble de tous les temps observés dans la base de données ;

- $N = card(t)$ le nombre total de temps observés.

Enfin, puisque l'on veut définir un modèle de mélanges gaussiens, on associe à chaque individu un vecteur aléatoire binaire $Z_i = (Z_i^1, ..., Z_i^K)^T$ pour indiquer à quel cluster l'individu appartient. Par exemple, si le i-ème individu est issu du cluster k, alors $Z_i^k = 1$ et 0 ailleurs. On considère que ces vecteurs aléatoires sont issus de la même distribution multinomiale : $Z_i \sim \mathcal{M}(1, \pi), \forall i \in \mathcal{I}$, avec un vecteur de proportions de mélange $\pi = (\pi_1, ..., \pi_K)^T$ et $\sum_{k=1}^K \pi_k = 1$.

3.3 Définition du modèle MagmaClust

Si l'on suppose que le i-ème individu appartient au k-ème cluster, on peut définir son expression mathématique comme la somme du processus moyen associé à son cluster et le processus centré sur lui-même :

$$y_i(t) = \mu_k(t) + f_i(t) + \epsilon_i(t), \forall t \in \mathcal{T}$$

où :

- $\mu_k(.) \sim GP(m_k(.), c_{\gamma_k}(.,.))$ est le GP moyen du k-ème cluster ;
- $f_i(.) \sim GP(0, \sum_{\theta_i}(.,.))$ est le GP spécifique au i-ème individu ;
- $\epsilon_i(.) \sim GP(0, \sigma^2)$ correspond à un terme d'erreur.

Or, ce modèle général dépend de plusieurs paramètres de moyenne et de variance (qui correspondent à des choix de modélisation) et des hyper-paramètres que l'on doit estimer :

- $\forall k \in \mathcal{K}$, $m_k(.)$ est la moyenne *a priori* du k-ème cluster ;
- $\forall k \in \mathcal{K}$, $c_{\gamma_k}(.,.)$ est le noyau de covariance d'hyper-paramètre γ_k ;
- $\forall i \in \mathcal{I}$, $\sum_{\theta_i}(.,.)$ est le noyau de covariance d'hyper-paramètre θ_i ;
- $\forall i \in \mathcal{I}$, σ_i^2 est le bruit de la variance associée au i-ème individu ;
- $\theta = \{\{\gamma_k\}_k, \{\theta_i\}_i, \{\sigma_i^2\}_i, \pi\}$ est l'ensemble des hyper-paramètres du modèle.

On admet que :

- μ_{kk} sont indépendants ;
- f_{ii} sont indépendants ;
- Z_{ii} sont indépendants ;
- ϵ_{ii} sont indépendants ;
- $\forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \mu_k, f_i, Z_i, \epsilon_i$ sont indépendants.

Pour avoir une idée plus claire des dépendances entre variables, voir Figure 2.

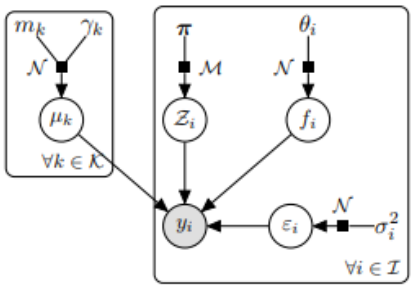


Figure 2: Graphique modélisant les dépendances entre variables au sein de MagmaClust.

3.4 Contribution : donner des hyper-paramètres communs à tous les individus d'un même cluster

MagmaClust est un modèle flexible ; on peut définir plusieurs types de sous-modèles, qui doivent leur spécificité au choix des hyper-paramètres. Il existe déjà 4 types de sous-modèles :

- $\mathcal{H}_{00}$: processus moyen commun à tous les clusters - processus individuel commun à tous les individus (2 HPs) ;
- $\mathcal{H}_{k0}$: processus moyen spécifique au cluster - processus individuel commun à tous les individus (K+1 HPs) ;
- $\mathcal{H}_{0i}$: processus moyen commun à tous les clusters - processus individuel spécifique à chaque individu (M+1 HPs) ;
- $\mathcal{H}_{ki}$: processus moyen spécifique au cluster - processus individuel spécifique à chaque individu (M+K HPs).

Dans le cas où l'on considère que tous les clusters ont le même γ_k, on admet que tous les processus moyens des clusters $\{\mu_k\}_k$ partagent la même structure de covariance. Cette propriété sous-entend que les allures / variations des courbes sont supposées être radicalement identiques d'un cluster à l'autre ; la différenciation est surtout due aux valeurs des moyennes. A l'inverse, des structures de covariance différentes offrent davantage de flexibilité tant au niveau de la dispersion, de la tendance ou de la régularité des courbes.

Pour tenir davantage compte des spécificités de chaque cluster, j'ai cherché un compromis entre $\mathcal{H}_{.0}$ et $\mathcal{H}_{.i}$. Ainsi, deux choix apparaissent :

- soit on définit un processus moyen commun à tous les clusters et un processus individuel commun à tous les individus du même cluster. On l'appelle $\mathcal{H}_{0k}$ **: processus moyen commun à tous les clusters - processus individuel commun à tous les individus de ce cluster (K+1 HPs)** ;
- soit on définit un processus moyen spécifique au cluster et un processus individuel commun à tous les individus du même cluster. On l'appelle $\mathcal{H}_{kk}$ **: processus moyen spécifique au cluster - processus individuel commun à tous les individus de ce cluster (K+K HPs)**.

Afin de mettre en oeuvre $\mathcal{H}_{0k}$ et $\mathcal{H}_{kk}$, il faut modifier l'algorithme VEM (Variational Expectation-Maximisation) implémenté dans l'entraînement du modèle. On rappelle que :

- la **E-step** de l'agorithme crée une fonction pour l'espérance de la log vraisemblance évaluée en utilisant l'estimation "courante" des hyper-paramètres ;

- la **M-step** calcule les paramètres maximisant la log-vraisemblance attendue trouvée lors de la **E-step**. Ces estimations de paramètres sont ensuite utilisées pour déterminer la distribution des variables latentes lors de la **E-step** suivante.

En répétant ces deux étapes jusqu'à convergence, on atteint un optimum local de la vraisemblance en relativement peu d'itérations (la majeure partie du temps, moins de 10 suffisent à atteindre la convergence). Dans mon cas, l'initialisation des hyper-paramètres est faite aléatoirement, bien qu'on pourrait tout à fait utiliser l'avis d'un spécialiste et spécifier des hyper-paramètres particuliers. De manière générale, comme pour n'importe quelle méthode utilisant des GPs, l'initialisation des hyper-paramètres peut avoir une influence importante sur l'optimisation finale et conduire à de mauvaises prédictions dans des cas pathologiques. L'initilisation des clusters se fait quant à elle par un k-means.

Dans notre cas, la **E-step** de l'algorithme VEM demeure inchangée ; en effet, cette étape est commune à toutes les hypothèses précédemment énoncées. Comme la forme des hyper-paramètres (communs ou non pour les clusters, communs ou non pour les individus) ne joue pas de rôle prépondérant lors de cette étape (on calcule simplement une espérance de log vraisemblance à partir d'une estimation des hyper-paramètres), nous n'avons pas besoin de modifier cette étape.

En revanche, la **M-step** nécessite quant à elle des modifications. L'optimisation des hyper-paramètres devient :

Pour $\mathcal{H}_{0k}$	Pour $\mathcal{H}_{kk}$
$\hat{\gamma}_0 = \arg\max_{\gamma_0} \sum_{k=1}^{K} \mathcal{L}_k(\hat{m}_k(t); m_k(t), C^t_{\gamma_k}),\ \forall k \in \mathcal{K}$	$\hat{\gamma}_k = \arg\max_{\gamma_k} \mathcal{L}_k(\hat{m}_k(t); m_k(t), C^t_{\gamma_k}),\ \forall k \in \mathcal{K}$
$(\hat{\theta}_k, \hat{\sigma}^2_k) = \arg\max_{(\theta_k, \sigma^2_k)} \sum_{i \in k} \mathcal{L}_i(y_i, \hat{m}_k(t_i), \psi^{t_i}_{\theta_k, \sigma^2_k}), \forall k \in \mathcal{K}$	$(\hat{\theta}_k, \hat{\sigma}^2_k) = \arg\max_{(\theta_k, \sigma^2_k)} \sum_{i \in k} \mathcal{L}_i(y_i, \hat{m}_k(t_i), \psi^{t_i}_{\theta_k, \sigma^2_k}), \forall k \in \mathcal{K}$

où $\mathcal{L}_k(x; m, S) = log\mathcal{N}(x; m, S)$ - un terme de pénalisation et $\mathcal{L}_i(x; m, S) = \sum_{k=1}^{K} \tau_{ik}(log\mathcal{N}(x; m, S)$ - un terme de pénalisation).
$\forall i \in \mathcal{I}, \forall k \in \mathcal{K}$, τ_{ik} correspond à la proprabilité pour l'individu i d'appartenir au cluster k.

3.5 Implémentation algorithmique

Pour modifier la **M-step** dans le package MagmaClust, il faut mettre à jour la fonction **vm_step()** qui y est déjà implémentée. Notre tâche consiste à récuperer la fonction de base et y ajouter un nouveau paramètre, **common_hp_i_in_k**. S'il vaut **TRUE**, alors on modifie une partie de l'algorithme ; s'il vaut **FALSE**, on utilise la version initialement implémentée. Ci-dessous se trouve la nouvelle déclaration de la fonction **vm_step** :

```
1 vm_stepNew <- function(db,
2                        old_hp_k,
3                        old_hp_i,
4                        list_mu_param,
5                        kern_k,
6                        kern_i,
7                        m_k,
```

```
                              common_hp_k,
                              common_hp_i,
                              common_hp_i_in_k,
                              pen_diag)
```

- **db** correspond à la base de données sur laquelle on souhaite appliquer la fonction. Dans notre cas, il s'agit des individus d'entraînement de notre BDD ;
- **old_hp_k, old_hp_i** correspondent aux hyper-paramètres relatifs au cluster k et à l'individu i que l'on hérite de l'itération précédente (et que l'on va mettre à jour) ;
- **list_mu_param** correspond à la liste des paramètres du processus moyen k ;
- **kern_k, kern_i** correspondent aux noyaux utilisés pour calculer la matrice de covariance du processus moyen / des individus. Dans notre cas, on utilise le noyau squared exponential, qui permet d'obtenir des courbes assez lisses et qui convient bien à tous types de données ;
- **m_k** correspond à la liste des moyennes *a priori* des K GPs ;
- **pen_diag** est un terme de pénalisation que l'on ajoute à la diagonale des matrices de covariance pour éviter qu'elles ne soient singulières. Dans notre cas, on le fixe à 10^{-10} ;
- **common_hp_k, common_hp_i** sont des booléens. Si **common_hp_k = TRUE**, alors les hyper-paramètres relatifs au cluster sont communs à tous les clusters ; sinon, ces hyper-paramètres sont tous différents. De même, si **common_hp_i = TRUE**, les hyper-paramètres relatifs aux individus sont communs à tous les individus ; sinon, on les considère tous différents ;
- **common_hp_i_in_k** est le nouveau paramètre de la fonction. Si **common_hp_i_in_k = TRUE**, alors on considère que les hyper-paramètres relatifs aux individus sont les mêmes uniquement pour les individus appartenant au même cluster (*ie* intra-cluster), mais différents pour 2 individus n'appartenant pas au même cluster (*ie* inter-clusters).

Dans la fonction **vm_step()** originale, deux cas de figure se présentent concernant les hyper-paramètres relatifs aux individus :

- Si l'on spécifie **common_hp_i = TRUE**, on rentre dans la boucle **if** suivante :

```
    if (common_hp_i) {
    ## Extract the hyper-parameters associated with the i-th individual
    par_i <- old_hp_i %>%
      dplyr::select(-.data$ID) %>%
      dplyr::slice(1)

    ## Optimise hyper-parameters of the individual processes
    new_hp_i <- optimr::opm(
      par = par_i,
      fn = elbo_clust_multi_GP_common_hp_i,
      gr = gr_clust_multi_GP_common_hp_i,
      db = db,
      hyperpost = list_mu_param,
      kern = kern_i,
      pen_diag = pen_diag,
      method = "L-BFGS-B",
      control = list(kkt = F)
    ) %>%
      dplyr::select(list_hp_i) %>%
      tibble::as_tibble() %>%
```

```
      tidyr::uncount(weights = length(list_ID_i)) %>%
      dplyr::mutate("ID" = list_ID_i, .before = 1)
  }

```

Les nouveaux hyper-paramètres des individus sont calculés à partir d'un algorithme d'optimisation antérieur au package MagmaClust. La fonction **elbo_clust_multi_GP_common_hp_i()** renvoie la valeur de l'elbo gaussien pénalisé pour la somme des M individus d'entrainement, qui ont des hyper-paramètres communs. La fonction **gr_clust_multi_GP_common_hp_i()** renvoie le gradient de l'elbo gaussien pénalisé calculé par la fonction précédente.

- Sinon, on rentre dans le **else** suivant :

```
  else {
  loop2 <- function(i) {
    ## Extract the hyper-parameters associated with the i-th individual
    par_i <- old_hp_i %>%
      dplyr::filter(.data$ID == i) %>%
      dplyr::select(-.data$ID)
    ## Extract the data associated with the i-th individual
    db_i <- db %>% dplyr::filter(.data$ID == i)

    ## Optimise hyper-parameters of the individual processes
    optimr::opm(
      par = par_i,
      fn = elbo_clust_multi_GP,
      gr = gr_clust_multi_GP,
      db = db_i,
      pen_diag = pen_diag,
      hyperpost = list_mu_param,
      kern = kern_i,
      method = "L-BFGS-B",
      control = list(kkt = F)
    ) %>%
      dplyr::select(list_hp_i) %>%
      tibble::as_tibble() %>%
      return()
  }
  new_hp_i <- sapply(list_ID_i, loop2, simplify = FALSE, USE.NAMES = TRUE) %>%
    tibble::enframe(name = "ID") %>%
    tidyr::unnest(cols = .data$value)
}

```

Cette fois, au lieu d'optimiser les nouveaux hyper-paramètres sur la somme des M individus d'entraînement, on optimise des hyper-paramètres différents pour chaque individu. Ainsi, on crée une boucle **for**, que l'on applique à chaque individu d'entraînement, et on utilise les fonctions **elbo_clust_multi_GP()** et **gr_clust_multi_GP()**, qui ne supposent plus que les hyper-paramètres sont communs à tous les individus.

Dans **vm_stepNew()**, on rajoute un autre cas de figure :

```
else if (common_hp_i_in_k){

   # On recupere le cluster d appartenance de chaque individu
   max_clust <- proba_max_cluster(list_mu_param$mixture)

```

```
    # On ne garde que les clusters non nuls pour une itération donnee
    names_k <- max_clust %>% pull(Cluster) %>% unique()

    # On cree un dataframe vide avec les noms de colonnes des hyper-parametres du noyau
    table_hp <- data.frame(matrix(vector(), 0, length(old_hp_i %>% names()),
                          dimnames=list(c(),old_hp_i %>% names())))

    # On convertit la colonne "ID" en character (au lieu de logical)
    table_hp$ID <- as.character(table_hp$ID)

    # On boucle sur les clusters
    loop_all_clust <- function(clust)
    {
      ## Create a list which contains the ID that are in clust
      list_ID_in_clust <- max_clust %>% filter(Cluster %in% clust) %>% pull(ID)

      loop_clust <- function(list_ID_in_clust){
        ## Extract the hyper-parameters associated with the i-th individual of
        ## the cluster
        par_i <- old_hp_i %>%
          dplyr::filter(.data$ID %in% list_ID_in_clust) %>%
          dplyr::select(-.data$ID) %>%
          dplyr::slice(1)
        ## Extract the data associated with the i-th individual
        db_i <- db %>% dplyr::filter(.data$ID %in% list_ID_in_clust)

        ## Optimise hyper-parameters of the individual processes
        optimr::opm(
          par = par_i,
          fn = elbo_clust_multi_GP_common_hp_i,
          gr = gr_clust_multi_GP_common_hp_i,
          db = db_i,
          pen_diag = pen_diag,
          hyperpost = list_mu_param,
          kern = kern_i,
          method = "L-BFGS-B",
          control = list(kkt = F)
        ) %>%
          dplyr::select(list_hp_i) %>%
          tibble::as_tibble() %>%
          return()
      }

      # On applique la fonction loop_clust aux individus presents dans le cluster
      HP_i_temp <- loop_clust(list_ID_in_clust)

      # On convertit le resultat en tibble
      new_hp_i_temp <- tibble::tibble(ID=list_ID_in_clust,HP_i_temp)
      return(new_hp_i_temp)
    }

    # On applique la boucle globale a chacun des clusters non vides
    temp <- lapply(names_k, loop_all_clust)

    # On ajoute les nouveaux HP au tableau des HP
    new_hp_i <- bind_rows(table_hp,temp)
    #print.data.frame(new_hp_i)

  }
```

Ici, il faut boucler à la fois sur les individus et sur les clusters. Lorsque l'on rentre dans la **loop_clust**, on réalise la même opération que dans le cas **common_hp_i = TRUE** ; la seule différence est que cette fois, la valeur de l'elbo gaussien pénalisé est obtenue pour la somme sur les M_k individus contenus dans le cluster

k, et non plus pour la somme complète sur les M individus. Pour cela, il nous faut donc préalablement récupérer la liste des individus que comporte chaque cluster. Une fois que l'on a obtenu les nouveaux hyper-paramètres relatifs au cluster k, on réitère l'opération pour le cluster $k+1$. On procède ainsipour tous les clusters non vides.

Remarque : il est primordial d'optimiser les hyper-paramètres des individus uniquement pour les clusters non vides, grâce à la commande

```
names_k <- max_clust %>% pull(Cluster) %>% unique()
```

En effet, même si l'on a spécifié **nb_cluster = 3** au début de l'entraînement, il est assez probable que l'un des 3 clusters de départ finisse par se vider au fil des itérations. L'algorithme renvoie alors une erreur, et il est impossible de terminer l'entraînement des données.

3.6 Application à nos deux bases de données

Une fois cette nouvelle version implémentée et débugguée, je l'ai appliquée aux 2 bases de données évoquées dans la partie introduction. Par souci de concision, je ne montrerai que les résultats relatifs à la base de données "poids des enfants" ; il est toutefois important de préciser que les résultats et comportements observés sont sensiblement les mêmes dans les deux cas. J'ai fait tourner le programme avec plusieurs ensembles d'entraînement différents, et ai essayé de prédire les futurs Outputs pour des individus différents. Pour l'entraînement, 25 enfants sont suffisants pour mettre en lumière les principaux résultats et comportements de l'algorithme. Pour réaliser une étude plus fine, il faudrait entraîner le modèle sur une centaine d'individus, mais cela devient rapidement assez coûteux en temps.

Pour les graphiques affichés ci-dessous, j'ai volontairement caché certaines données de l'enfant à prédire (représentées en jaune) afin de pouvoir comparer les prédictions des 2 versions de MagmaClust avec les données réellement observées. J'ai également affiché une heatmap de probabilités pour avoir une meilleure estimation de l'incertitude associée à nos prédictions.

J'ai pu remarquer deux types de comportement :

- **1er cas (figures 3 (a) et (b)) :** on spécifie **nb_cluster = 3** au début de l'entraînement. Au fur et à mesure des itérations, le cluster 3 se vide, ce qui explique que son processus moyen reste à zéro. Si l'on affiche le contenu des clusters, on remarque que ces derniers ne sont pas très désésquilibrés (on tombe souvent sur du 17 - 8, donc quasiment 2/3 - 1/3). Dans ces cas-là, l'optimisation des hyper-paramètres avec la nouvelle version est réussie puisque l'on a suffisamment d'individus dans chaque cluster pour éviter de tomber dans le piège du surajustement.

  On remarque dans ce cas que la prédiction est meilleure avec la nouvelle version de MagmaClust qu'avec l'ancienne. Les prédictions sont toutes situées dans une aire de "forte probabilité", ce qui n'était pas le cas avec l'ancienne version.

  Ainsi, si les clusters sont suffisamment "consistants" (*ie* s'il y a un nombre suffisant d'individus dans chaque cluster) et assez bien dissociés, la nouvelle version de MagmaClust est plus performante - en terme de prédiction - que l'ancienne.

- **2eme cas (figures 4 (a) et (b)) :** on spécifie à nouveau **nb_cluster = 3** au début de l'entraînement. Ici aussi, au fur et à mesure des itérations, le cluster K2 se vide. Cette fois, si l'on affiche le contenu

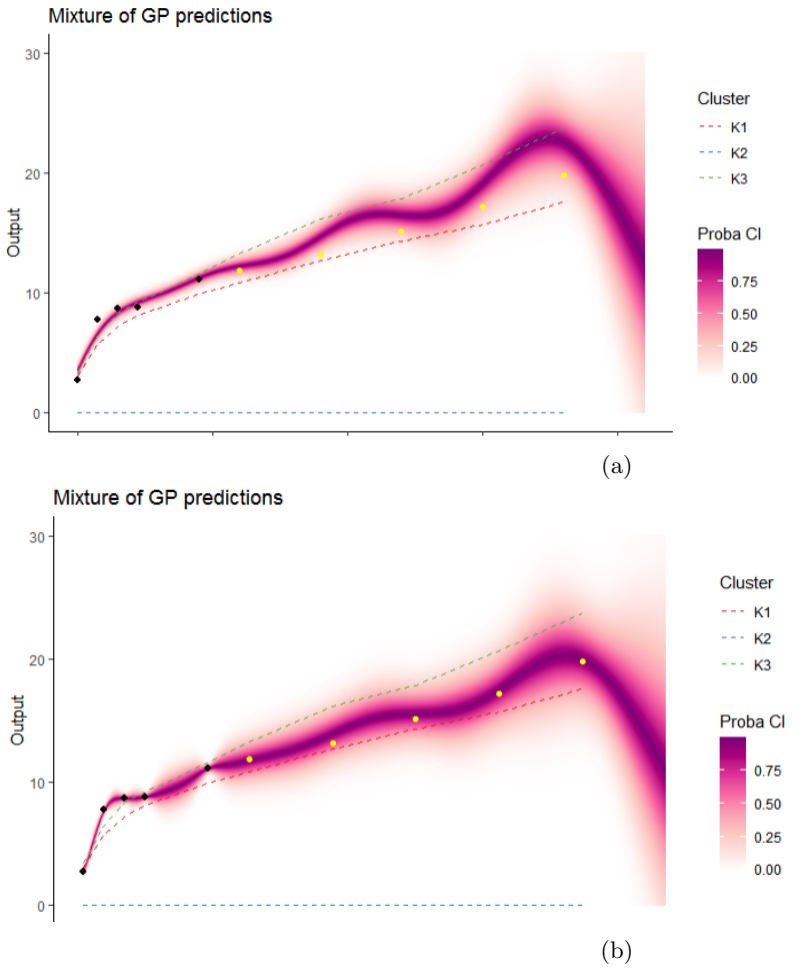
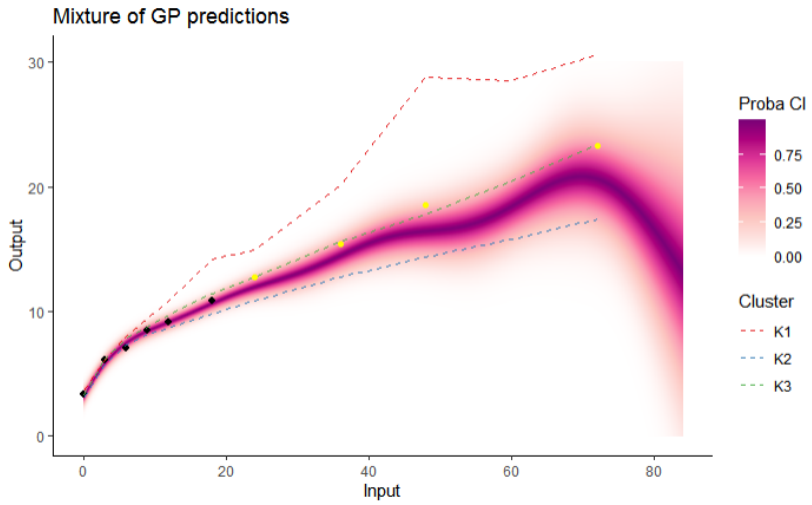


Figure 3:

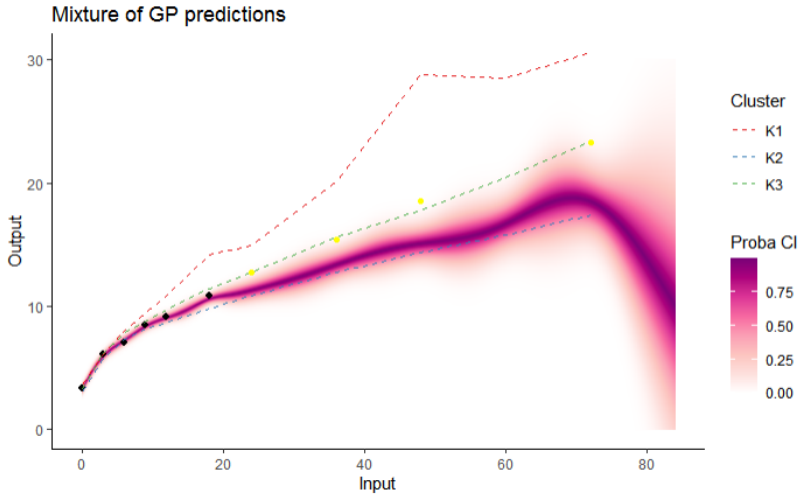
(a) **1er cas :** Prédiction obtenue avec l'ancienne version de MagmaClust, avec **common_hp_i = TRUE**.
(b) **1er cas :** Prédiction obtenue avec la nouvelle version de MagmaClust, avec **common_hp_i_in_k = TRUE.**

des clusters en fin d'entraînement, on remarque qu'ils sont très déséquilibrés (souvent 21 - 4 ou 21 - 3). Si l'individu à prédire se situe dans le cluster à faible effectif, on tombe dans du surajustement ; la courbe passe quasiment exactement au milieu des points noirs, mais dès qu'on manque d'information, la prédiction est détériorée significativement. Cela s'explique par le fait que les individus partagent de l'information par le biais du processus spécifique à chaque cluster. Les prédictions plongent progressivement au processus spécifique ; si le cluster contient très peu d'individus, la moindre variation dans les données peut entraîner d'importants changements dans l'allure du processus spécifique. Au contraire, si l'individu à prédire se situe dans le cluster contenant quasiment tous les individus, la nouvelle version de MagmaClust ne diffère que très peu de l'ancienne. Son intérêt est donc nul dans ce cas-là.

Sur la Figure 4 - (b), on voit très bien le surajustement de la nouvelle version de MagmaClust. L'individu que l'on prédit se situe dans un cluster quasiment vide, ce qui explique la mauvaise qualité



(a)



(b)

Figure 4:

(a) **2eme cas :** Prédiction obtenue avec l'ancienne version de MagmaClust, avec **common_hp_i = TRUE**.
(b) **2eme cas :** Prédiction obtenue avec la nouvelle version de MagmaClust, avec **common_hp_i_in_k = TRUE.**

de nos prédictions par rapport à l'ancienne version.

Ainsi, l'ajout d'une nouvelle caractéristique à la fonction **vm_step()** pour tenir davantage compte des spécificités des individus n'est intéressante que dans un cadre particulier : un nombre suffisant d'individus dans chaque cluster pour éviter le surapprentissage, et des clusters suffisamment dissociés les uns des autres.

3.7 Limites du modèle

Si cette nouvelle version semblait plutôt bien marcher en pratique, avec des résultats satisfaisants dans un cadre particulier, elle soulève toutefois des problèmes d'un point de vue théorique. En effet, si l'on assigne à un individu les hyper-paramètres du cluster dans lequel il est le plus probable, on le "force" à appartenir à un unique cluster. Ainsi, on ne tient plus compte de la composante "mixture gaussienne", qui est pourtant au coeur de l'algorithme MagmaClust. J'ai repris la démonstration qui prouve que le modèle est valable pour les 4 hypothèses originellement proposées par M. Leroy et ai essayé de l'adapter pour le nouveau modèle. Malheureusement, je n'ai trouvé aucun moyen de l'écrire prorement. En y réfléchissant avec M. Leroy, nous sommes parvenus à la conclusion qu'il n'était sans doute pas possible de le développer théoriquement, justement à cause de cette contradiction mixture gaussienne / forcer un individu à appartenir à un unique cluster. Ayant le sentiment d'être arrivée au bout de ce projet-là, j'ai décidé (avec l'accord de M. Leroy) de m'atteler à une autre tâche.

4 Rédaction de vignettes d'aide à l'utilisation du package MagmaClustR

4.1 Enjeux des vignettes explicatives

Dans l'optique de rendre son package suffisamment clair pour être utilisé par le plus grand nombre, M. Leroy nous a demandé de réaliser des vignettes HTML. Ces vignettes ont pour but d'expliquer au lecteur ce qu'il est possible de faire grâce à MagmaClustR et comment utiliser les différentes fonctions du package. Les points à respecter étaient les suivants :

- Le format des vignettes doit s'inspirer de celui du **tidyverse**, avec la même organisation des différentes parties.
- Les vignettes doivent être construites autour d'un exemple précis et ne pas rester trop générales (la documentation est là pour ça). L'exemple sert de fil rouge pour la rédaction et la lecture de la vignette : le lecteur doit pouvoir rapidement comprendre ce que l'on cherche à faire et transposer à son propre projet.
- Les explications doivent rester claires et succintes : on ne cherche pas à submerger le lecteur d'informations, au risque de le perdre. Le temps de lecture doit être compris entre 5mins (pour la vignette la plus courte) et 10mins (pour la plus complexe).
- Les possibilités plus poussées qu'offre le package (en particulier au niveau de l'affichage) doivent être traitées comme des "bonus" et placées en fin de vignette. Elles sont destinées aux utilisateurs plus expérimentés qui souhaiteraient aller plus loin dans la customization des graphiques.
- Les graphiques doivent être bien choisis. Chaque graphique doit illustrer une information bien précise ; si on essaie de mettre en valeur trop d'informations différentes, on perd en clarté.
- La vignette constitue la vitrine du package. Elle doit donc mettre en valeur les nouveautés apportées par le package par rapport à ce qui existait déjà avant.

4 vignettes devaient ainsi être réalisées :

- la première (*Gaussian Process regression*) pour montrer comment effectuer une régression gaussienne grâce au package ;
- la seconde (*Introduction to Magma*), pour expliquer le fonctionnement de Magma en se basant sur la BDD nageurs ;

- la troisième (*Introduction to MagmaClust*) pour développer les améliorations apportées par MagmaClust en utilisant la BDD poids des enfants;
- la dernière (*Introduction to Magma 2D*) pour illustrer l'utilisation de Magma sur un exemple en 2 dimensions (*ie* on a désormais 2 Inputs : 'Input' et 'Covariable')

Avec Hugo, nous avons décidé de nous occuper de 2 vignettes chacun ; j'ai donc rédigé et mis en forme *Introduction to Magma* et *Introduction to Magma 2D*.

4.2 *Introduction to Magma*

La principale difficulté pour la rédaction de cette vignette consistait à trouver un compromis entre une explication complète et précise, et un message clair et concis. Il fallait garder à l'esprit que l'utilisateur lambda du package est censé posséder des connaissances suffisantes en GP pour pouvoir appliquer MagmaClustR sur son jeu de données. Toutefois, il n'est pas supposé connaître la démarche complète de l'algorithme ; je n'avais donc pas à m'attarder sur les subtilités de l'algorithme EM ou le partage des informations entre les individus, au risque de noyer d'informations l'utilisateur.

Afin de rendre la vignette plus agréable à lire et pour que l'utilisateur puisse comprendre rapidement ce que l'on peut réaliser grâce au package, j'ai utilisé la BDD nageurs FFN. Cette dernière sert de fil conducteur : je l'utilise pour que l'exemple paraisse concret au lecteur et qu'il comprenne tout l'intérêt de MagmaClustR par rapport aux autres algorithmes de l'état de l'art. J'ai commencé par évoquer rapidement le contenu de la base de données ainsi que l'objectif principal que l'on cherche à atteindre (*ie* prédire les performances futures des nageurs dans le cadre de la détection de jeunes talents mise en place par la FFN). Afin d'avoir une idée de l'allure des données, j'ai commencé par afficher les performances de 5 nageurs de 100m nage libre sélectionnés aléatoirement (voir Figure 1 (a)). On remarque bien que les trajectoires sont assez similaires : elles décroissent assez lentement en fonction de l'âge, puis finissent par stagner à un certain niveau en oscillant légèrement.

J'ai ensuite créé une section "**Data format**" pour expliquer au lecteur comment mettre en forme sa base de données. En effet, pour pouvoir utiliser Magma, on doit obligatoirement avoir les colonnes ID, Input et Output. Il faut donc que l'utilisateur renomme les colonnes de sa BDD, et éventuellement qu'il en supprime certaines (dans notre cas, la colonne Genre). J'explique ensuite les grandes fonctions de l'algorithme :

- la séparation des données en un échantillon d'apprentissage et un échantillon d'entraînement ;
- l'étape de training grâce à **train_magma()**. Je me suis attachée à préciser quels étaient les paramètres d'importance de la fonction et comment modifier leurs valeurs pour qu'elles soient adaptées au jeu de données qu'on utilise ;
- l'étape de prédiction grâce à **pred_magma()**. Ici, seule la grille d'inputs peut être modifiée par l'utilisateur ; il est libre de choisir la fenêtre d'inputs sur laquelle Magma va réaliser une prédiction ;
- l'affichage de la prédiction grâce à **plot_gp()**. Magma affiche par défaut le GP de l'individu à prédire en traits pleins, le GP moyen en pointillés et un intervalle de crédibilité à 95% (ou une heatmap de probabilités).

A la demande de M. Leroy, j'ai comparé les prédictions que l'on obtenait avec Magma sur la BDD femmes avec la BDD hommes. En effet, si les courbes de progression sont sensiblement identiques entre les deux genres jusqu'à environ 14 ans, les hommes continuent à progresser de manière plus importante que les filles à partir de cet âge là. J'ai aussi comparé la prédiction de Magma avec celle d'un GP classique pour montrer l'intérêt du package par rapport aux outils déjà existants. L'amélioration majeure de Magma réside dans la phase où l'on n'a plus de données observées pour l'individu à prédire, mais qu'on dispose des données

des individus d'entraînement. Ainsi, on voit graphiquement l'intérêt du partage d'informations entre les individus, qui est au coeur du fonctionnement de Magma.

J'ai ensuite ajouté une partie "bonus" pour les utilisateurs souhaitant obtenir des graphiques plus élaborés, avec une heatmap de probabilités à la place d'un intervalle de confiance ou un affichage de la prédiction sous forme de GIF. Pour terminer, j'ai introduit un paragraphe d'accroche pour rediriger le lecteur vers une version plus évoluée de Magma (*ie* MagmaClust) en lui montrant les limites de l'algorithme actuel (et, *de facto*, la raison pour laquelle il serait intéressant de tenir compte des structures de groupe au sein des données).

4.3 *Introduction to Magma 2D*

Une fois la vignette *Introduction to Magma* finalisée, relue et validée par M. Leroy, je me suis attelée à rédiger celle d'*Introduction to Magma 2D*. Si la structure de cette seconde vignette s'inspire largement de la première, quelques ajustements ont tout de même dû être opérés :

- la simulation des données : dans *Introduction to Magma 2D*, on ne se sert plus d'une base de données déjà existante ; on en simule une grâce à la fonction **simu_db()** du package MagmaClustR. J'ai donc expliqué comment customiser les paramètres de cette fonction pour que l'utilisateur obtienne en sortie une base de données comportant le nombre d'individus, de points par individus et de clusters sous-jacents qu'il souhaite. En particulier, je me suis servie de **simu_db()** pour créer un jeu de données en 2D grâce à l'argument **covariate = TRUE**.

- la prédiction : dans Magma, l'utilisateur peut choisir de spécifier lui-même la grille d'inputs sur laquelle il désire réaliser la prédiction, ou bien laisser **pred_magma()** en générer une automatiquement pour lui. Or, dans le premier cas, l'utilisateur doit être vigilant : comme on travaille désormais en 2D, la grille d'inputs n'est plus un simple segment découpé à intervalles réguliers mais une véritable grille en 2D. Afin de faciliter la création de cette grille et que l'utilisateur n'ait plus à le faire à la main, j'ai créé la fonction **expand_grid_inputs()** : elle permet de faire toutes les combinaisons d'Input et de Covariate possibles et de construire une grille 2D à partir de ces combinaisons (voir Section 5.4 pour plus de détails). A présent, un point est défini par le triplet (Input, Covariate, Output) et se place facilement dans un plan 2D.

- l'affichage du résultat. Il s'agit sans doute de la plus grande modification par rapport à la version 1D ; comme on augmente la dimension, certaines fonctionnalités de **plot_gp()** ne sont désormais plus affichables, en particulier :

  - les données ayant servi au training, auparavant disponibles en arrière-plan. C'est un choix de M. Leroy ; en les affichant sur la heatmap, on aurait eu beaucoup trop de labels et le graphe aurait perdu en clarté.
  - le processus moyen (représenté en pointillés noirs dans la version 1D).

  La Figure 5 illustre le type de graphique que l'on obtient grâce à MagmaClustR en 2D. On affiche les valeurs de la moyenne a posteriori prédites par Magma pour un individu de la base de données. La prédiction est représentée comme une heatmap de probabilités : l'axe des abscisses correspond à la colonne Input ; celui des ordonnées, à la Covariate. Chaque couple d'inputs (*ie* (Input, Covariate)) est associé à un gradient de couleur. Plus la couleur tire sur le violet, plus la valeur de cette moyenne est haute. Comme pour la version 1D, on obtient une quantification de l'incertitude grâce au niveau de transparence de la heatmap. Plus l'intervalle de crédibilité (à 95%) est étroit, plus la couleur associée est opaque. Sur la Figure 5, on remarque bien que lorsque l'on se situe autour des points observés, la couleur est assez opaque ; quand on s'en éloigne, elle devient plus claire, signe que l'intervalle de crédibilité s'élargit.

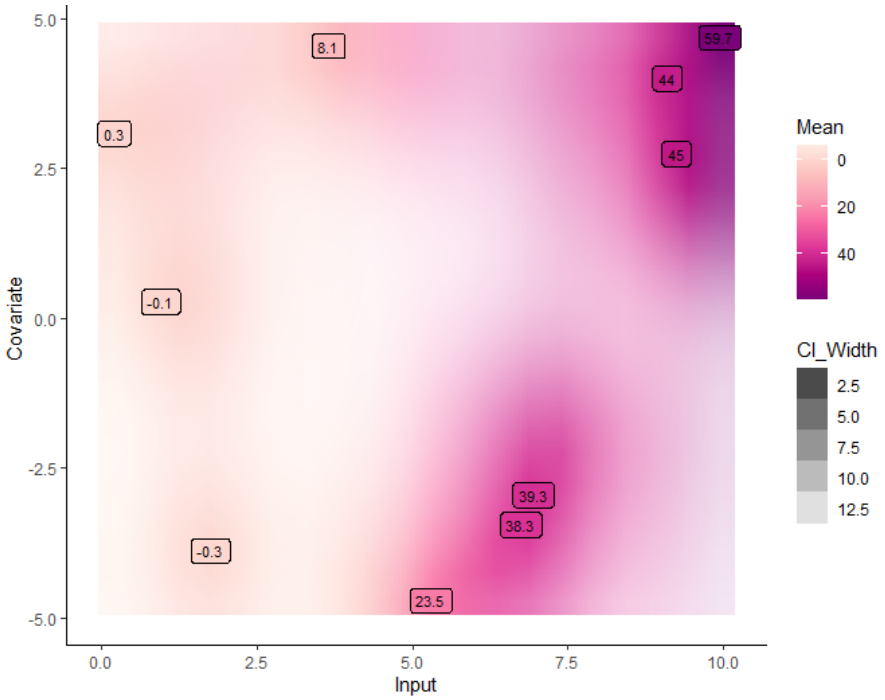


Figure 5: Prédiction obtenue grâce à MagmaClustR en 2 dimensions.

5 Elaboration d'une nouvelle version du package plus adaptée aux Inputs multidimensionnels

Une fois les vignettes réalisées, M. Leroy nous a lancés sur un projet de plus grande envergure : réaliser une nouvelle version 2D de Magma et MagmaClust, qui apporterait une vraie plue value aux utilisateurs souhaitant les appliquer à des bases de données plus complexes.

Jusqu'à présent, il était possible d'utiliser Magma avec une base de données contenant plusieurs variables : un Input (dans la plupart des cas, le temps exprimé sous forme de série temporelle) et une ou plusieurs Covariates (traitées comme de véritables covariables ; pour le poids des enfants, on pourrait considérer la taille). L'entraînement et la prédiction ne différaient pas fondamentalement de la version 1D ; seul l'affichage de la prédiction était significativement modifié (voir la section ***Introduction to Magma 2D***). Toutefois, cette version souffrait d'une lacune : dans l'algorithme, le calcul de la matrice de covariance n'était réalisé que sur la variable Input, et pas sur les éventuelles Covariates du modèle. Ces Covariates n'étaient donc pas réellement prises en compte dans l'entraînement et la prédiction, devenant superficielles par rapport à la variable d'importance Input. On pouvait facilement s'en rendre compte lorsque l'on affichait le résultat de la prédiction en 2D : dans certains cas, les dégradés de couleur se faisaient selon Input et restaient quasiment constants selon Covariate.

L'objectif pour nous était donc de faire en sorte que toutes les variables du modèle aient la même importance dans le calcul de la matrice de covariance. Ainsi, cette nouvelle version aurait vocation à être adaptée à n'importe quelle dimension, et pas uniquement à la 2D.

5.1 Malédiction de la grande dimension

La première précaution à garder en tête si l'on souhaite appliquer Magma sur des BDD contenant plusieurs Inputs concerne la malédiction de la dimension. On rappelle ici que l'atout majeur de l'algorithme réside dans le partage des informations entre les individus. Lorsque l'on travaillait en 1D (*ie* avec seulement une variable d'Input et aucune Covariate), chaque point d'observation était relativement proche de plusieurs autres, rendant l'utilisation de Magma appropriée. En revanche, lorsque l'on augmente le nombre de variables (et donc, par conséquent, la dimension), on "éloigne" nos points observés ; plus la dimension est grande, plus les points se retrouvent isolés. Le partage d'informations devient donc très compliqué : en 1D, un point d'observation donné possède plus de caractéristiques communes avec les points qui sont proches de lui (au sens de la distance euclidienne), et moins avec ceux qui en sont plus éloignés. En grande dimension, tous les points sont éloignés les uns des autres, rendant impossible le partage d'informations.

Pour que le modèle Magma puisse être suffisamment performant, il faudrait donc multiplier drastiquement le nombre d'observations de la base de données. Or, ce nombre d'observations "raisonnable" augmente exponentiellement en fonction de la dimension ; il devient donc rapidement compliqué d'exploiter Magma lorsque le nombre de variables explicatives est grand. Toutefois, cette faiblesse demeure un problème d'apprentissage machine classique, un utilisateur possédant des connaissances suffisantes dans ce domaine pourra l'appréhender sans souci.

5.2 Définition d'un nouveau format d'Input

Dans la précédente version de Magma, on considérait l'Input comme étant la variable de référence ; en particulier, les lignes et les colonnes des différentes matrices de covariance étaient nommées selon les valeurs d'Input. Par exemple, si on a une base de données avec des Inputs compris entre 1 et 10 (avec une précision de 10^{-1}), la matrice de covariance regroupant tous les individus ressemblera à cela :

$$\begin{array}{c} \\ 1 \\ 1.1 \\ \dots \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 1.1 & \dots & 10 \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,100} \\ a_{2,1} & a_{2,2} & \dots & a_{2,100} \\ \dots & \dots & \dots & \\ a_{100,1} & a_{100,2} & \dots & a_{100,100} \end{pmatrix} \end{array}$$

Afin que tous les Inputs (*ie* l'Input de référence et les éventuelles Covariates) soient désormais pris en compte, il nous faut trouver un format de variable qui concatène tous les Inputs. Nous avons décidé d'ajouter une nouvelle variable, nommée "Référence", dont les valeurs désignent les différentes combinaisons d'Inputs de la BDD, grâce au code suivant :

```
1      data <- data %>%
2          purrr::modify_at(names_col, signif) %>%
3          tidyr::unite("Reference", names_col, sep=":", remove = FALSE) %>%
4          dplyr::arrange(.data$Reference)
5
```

On a préalablement stocké dans la variable names_col le nom de toutes les colonnes d'inputs. On commence par arrondir toutes les valeurs des différents inputs à 6 chiffres significatifs afin d'éviter des problèmes numériques par la suite. On utilise ensuite les deux points pour séparer les valeurs d'Inputs et rendre la Référence lisible. La base de données possède désormais un nouveau format (voir Figure 6).

Les matrices de covariance sont elles aussi modifiées ; si on a les mêmes valeurs d'Input que dans l'exemple précédent et des valeurs de Covariate comprises entre -1 et 1 (avec un pas de 10^{-1}), la matrice de covariance ressemble à cela :

ID	Reference	Input	Covariate	Output
Indiv_1	1:-2	1	-2	4
Indiv_2	3:-1	3	-1	7
Indiv_3	5:1	5	1	9
Indiv_4	7:3	7	3	2

Figure 6: Nouveau format de la BDD après l'ajout de la variable Référence.

$$
\begin{array}{c}
 \\
1:-1 \\
1:-0.9 \\
\dots \\
1:1 \\
1.1:-1 \\
\dots \\
1.1:1 \\
\dots \\
10:1
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
1:-1 & 1:-0.9 & \dots & 1:1 & 1.1:-1 & \dots & 1.1:1 & \dots & 10:1
\end{array} \\
\begin{pmatrix}
. & . & \dots & . & . & \dots & . & \dots & . \\
. & . & \dots & . & . & \dots & . & \dots & . \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
. & . & \dots & . & . & \dots & . & \dots & . \\
. & . & \dots & . & . & \dots & . & \dots & . \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
. & . & \dots & . & . & \dots & . & \dots & . \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
. & . & \dots & . & . & \dots & . & \dots & .
\end{pmatrix}
\end{array}
$$

Autre point important : avec la précédente version de Magma, pour chaque individu, les doublons d'Input n'étaient pas autorisés (une erreur s'affichait lorsque l'on essayait de faire tourner l'algorithme). A présent, pour un même individu, ce ne sont pas les valeurs de la variable Input mais les Références qui doivent être uniques. Par exemple, les couples (Input = 1, Covariate = 2) et (Input = 1, Covariate = 3) n'auraient pas pu être traités dans l'ancienne version de Magma 2D ; désormais, comme les deux couples forment deux références différentes, le problème de doublons ne se pose plus.

5.3 Calcul des matrices de covariance

La principale nouveauté apportée par notre version 2D réside dans la forme des matrices de covariance. On en distingue 2 types :

- Les matrices de covariance **individuelles**. Chaque individu $\mathcal{I}$ possède une matrice de covariance qui lui est propre ; si $n_{\mathcal{I}}$ valeurs de Référence lui sont associées, on lui attribue une matrice de covariance carrée de taille $n_{\mathcal{I}}$ x $n_{\mathcal{I}}$. Chaque ligne / colonne aura pour nom l'une des valeurs de Référence de l'individu.

- La matrice de covariance **collective**. Si on considère une grille pour laquelle chaque intersection correspond à un couple (Input, Covariate) de la BDD, alors les lignes et colonnes de la matrice collective correspondent à chaque noeud de cette grille. Dans la précédente version de Magma, seuls les valeurs d'Input de tous les individus de la BDD constituaient les lignes / colonnes de cette matrice.

Cette nouvelle forme de la BDD (après l'ajout de la colonne Référence) et des matrices de covariance a entraîné la modification de nombreuses fonctions du package. Nous avons fait en sorte de toujours travailler avec des BDD contenant la colonne Référence ; ainsi, dans l'entraînement et la prédiction, nous avons rajouté cette colonne aux BDD et aux grille d'Inputs pour garder une certaine cohérence. Il a donc fallu repenser la majorité des fonctions internes pour que l'algorithme repose à présent sur Référence (et non plus sur Input).

En particulier, nous avons dû recoder l'ancienne fonction **kern_to_cov()** pour tenir compte de cette nouvelle forme. Le calcul des coefficients des matrices de covariance reste quant à lui inchangé. Cependant, le nom et le nombre de lignes / colonnes ne dépend plus de la variable Input mais des noeuds qui figurent dans Référence.

Nous avons adapté cette démarche aux fonctions permettant de coder des GPs classiques, Magma et MagmaClust.
Remarque : nous avons dû être viligants sur le fait que nos modifications ne devaient pas perturber le fonctionnement habituel de Magma en 1D. En particulier, les nouvelles fonctions utilisées sur une BDD contenant un seul Input sont censées donner le même résultat qu'avec la précédente version du package.

5.4 expand_grid_inputs() : aide à l'utilisateur dans la construction de sa grille d'inputs

Lorsque j'ai réalisé la vignette ***Introduction to Magma 2D***, j'ai dû adapter le format de la grille d'inputs que je passais en argument de la foncton **pred_magma()**. Dans la version 1D, il suffisait de spéficier un vecteur de taille finie (par exemple,

```
seq(from = 1, to = 10, by = 0.1)
```

pour obtenir un vecteur de taille 100). Avec la version 2D, le format vectoriel n'est plus approprié ; il faut désormais définir une grille d'inputs en 2D, composée d'une colonne Input et une colonne Covariate.

Afin d'éviter à l'utilisateur de créer cette grille à la main, tâche qui peut être assez pénible s'il ne connait pas bien le tidyverse, j'ai créé la fonction **expand_grid_inputs()** dont le code est disponible ci-dessous :

```
expand_grid_inputs <- function(Input, ...){
  arguments <- list(Input,...)

  if(!(arguments %>% every(is.numeric))){
    stop("The arguments must all be numerical sequences.")
  }

  dim_all <- sapply(arguments,length) %>% prod()

  if(dim_all > 10000) {
    warning("The number of grid points is too high. Magma has a cubic complexity,
            so the execution will be extremely long. We advise to reduce the length
            of your grid of inputs.")
  }

  expand_grid(Input, ...) %>% return()
}
```

Comme notre objectif est d'adapter Magma aux dimensions supérieures ou égales à 2, cette fonction doit pouvoir créer une grille d'inputs contenant autant de variables que l'on souhaite. Pour cela, on définit l'argument Input de base (qui doit être présent dans n'importe quelle BDD sur laquelle on voudrait appliquer Magma), puis on rajoute des points de suspension qui permettent à l'utilisateur de définir autant de Covariates qu'il le souhaite.

La grille d'inputs doit réaliser toutes les combinaisons d'Inputs possibles ; pour cette raison, il faut trouver un compromis entre la taille de la grille et le temps d'exécution de Magma. Pour éviter tout problème, on calcule le produit des dimensions : s'il est supérieur à 10000, on affiche un warning à destination de l'utilisateur

pour lui rappeler que la complexité de Magma est cubique (et donc que l'exécution pourra être longue si les dimensions de la grille d'inputs sont trop grandes). On applique ensuite la fonction **expand_grid()** sur l'ensemble des vecteurs ayant été passés en arguments par l'utilisateur. En sortie, on obtient bien la grille d'inputs désirée.

Pour plus de clarté, considérons l'exemple suivant : un utilisateur souhaite créer une grille d'inputs prenant en compte 3 variables :

- *Input*, un vecteur allant de 0 à 1 avec un pas de 0.5 ;
- *Covariate*_1, un vecteur allant de 1 à 2 avec un pas de 0.5 ;
- *Covariate*_2, un vecteur allant de -1 à 1 avec un pas de 1.

En utilisant **expand_grid_inputs(Input = seq(0,1,1), Covariate_1 = seq(1,2,1), Covariate_2 = seq(-1,1,1))**, on obtient la sortie affichée en Figure 7:

	Input	Covariate_1	Covariate_2
1	0	1	-1
2	0	1	0
3	0	1	1
4	0	2	-1
5	0	2	0
6	0	2	1
7	1	1	-1
8	1	1	0
9	1	1	1
10	1	2	-1
11	1	2	0
12	1	2	1

Figure 7: Grille obtenue en sortie de la fonction **expand_grid_inputs()**.

Cette fonction est surtout utile en vue d'utiliser **train_magma()** et **pred_magma()** sur des bases de données à plus d'une dimension, mais peut également se révéler utile dans d'autres contextes (traitement de données notamment).

5.5 Comparaison entre l'ancienne version de Magma 2D et la nouvelle

Une fois les codes de Magma et MagmaClust adaptés, nous avons fait tourner les anciennes et les nouvelles versions sur les mêmes jeux de données afin de pouvoir les comparer.

La fonction **simu_db()** n'étant pas réellement adaptée à la 2D (la Covariate est simulée aléatoirement, donc la nouvelle version 2D de Magma n'a pas d'intérêt particulier), on simule nous-mêmes une BDD en 2D. Plus précisément :

- on simule Input et Covariate indépendamment selon 2 lois uniformes entre 0 et 10 ;
- pour chaque paire (Input, Covariate), l'Output correspond à la valeur de la densité d'une gausienne calculée en ce point. Sans perte de généralité, on tire une gaussienne d'espérance (5,5) et de matrice de covariance **matrix (c(5,0,0,5), nrow = 2, ncol = 2)**.

Chaque individu de la BDD possède 10 couples (Input, Covariate) observés. On utilise notamment la fonction **regularize_data()** créée par Hugo pour placer nos données sur une grille de dimensions 20 x 20. Le graphique correspondant à la BDD ainsi régularisée est affiché sur la Figure 9, en haut à gauche.

On tire ensuite aléatoirement 50 individus issus de cette BDD 2D ; ils constituent notre échantillon d'apprentissage. On sélectionne aussi un autre individu au hasard ; c'est sur ce dernier que la prédiction va être réalisée. Voir Figure 8 pour obtenir la comparaison des 2 versions 2D de Magma.

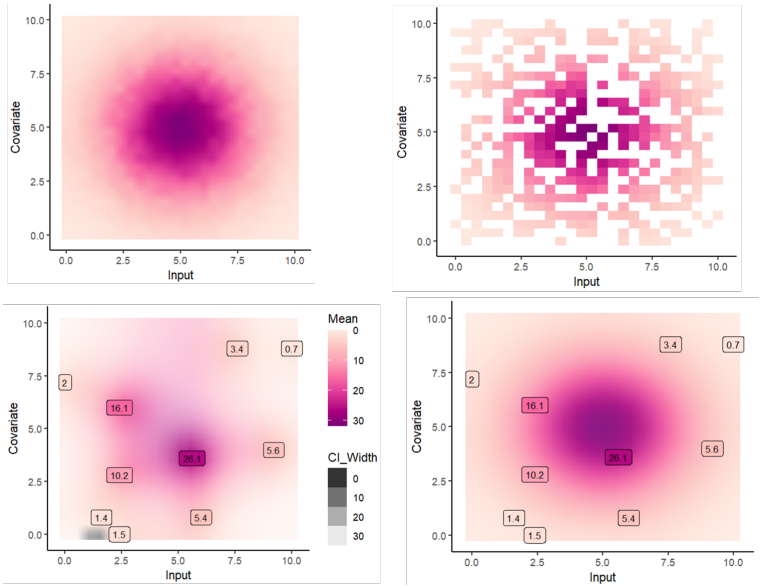


Figure 8:

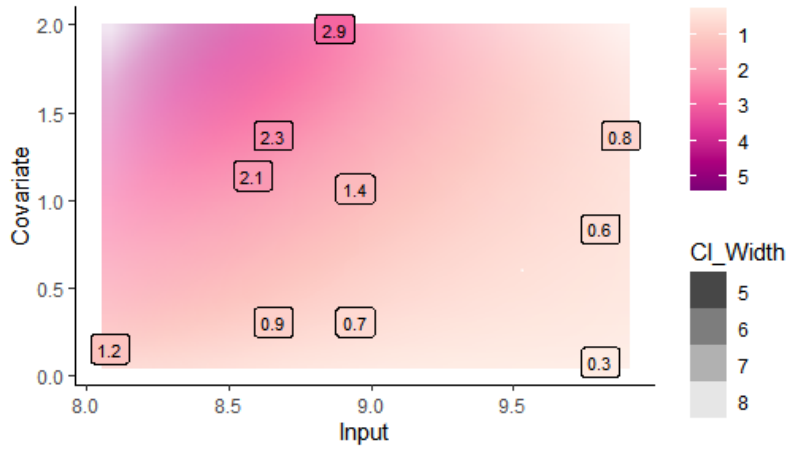
En haut à gauche : densité de la gaussienne 2D centrée en (5,5).
En haut à droite : points ayant servi à l'entraînement du modèle.
En bas à gauche : prédiction obtenue avec l'ancienne version 2D de Magma.
En bas à droite : prédiction obtenue avec la nouvelle version 2D de Magma.

On remarque tout d'abord qu'avec très peu de points (seulement 50 individus !), la nouvelle version de Magma arrive à parfaitement reconstituer la densité de la gausienne 2D. De plus, la largeur de l'intervalle de crédibilité est beaucoup plus faible que pour la prédiction obtenue avec l'ancienne version de Magma : on a donc davantage confiance en nos valeurs prédites, ce qui constitue une vraie plus-value pour l'utilisateur.

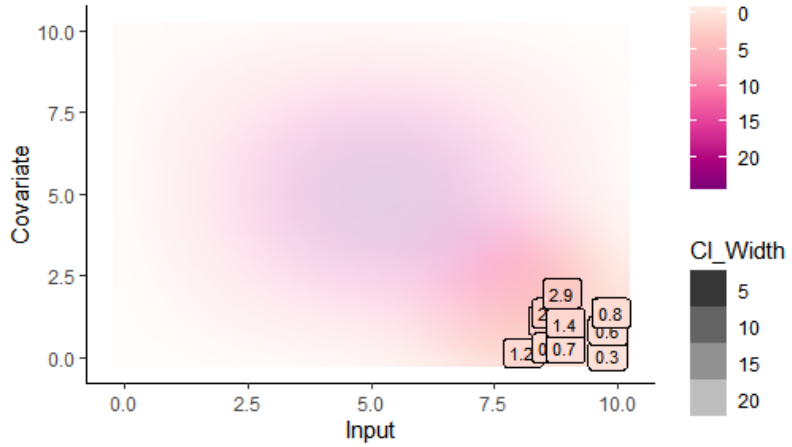
Afin de bien se rendre compte des performances de la nouvelle version 2D, nous avons changé les individus d'entraînement et celui de prédiction :

- les individus d'entraînement ont tous des données situées dans la partie supérieure-gauche de la Figure 8 (en haut à gauche) (*ie* leurs valeurs d'Input sont toutes comprises entre 0 et 2 ; celles de Covariate, entre 8 et 10. On les tire selon des lois uniformes) ;
- l'individu à prédire dispose d'observations situées dans la partie inférieure-droite de la Figure 8 (en haut à gauche) (*ie* ses valeurs d'Input sont toutes comprises entre 8 et 10 ; celles de Covariate, entre 0 et 2. On les tire selon des lois uniformes).

Avec 50 individus d'entraînement, on obtient les graphiques de la Figure 9.



(a)



(b)

Figure 9:

(a) Graphique de la prédiction obtenue avec la nouvelle version 2D (grille centrée sur les données de l'individu à prédire).
(b) Graphique de la prédiction obtenue avec la nouvelle version 2D (grille large).
Le dégradé de rose représente la moyenne ; la légende a été tronquée par R.

Certes, dès que l'on s'éloigne de l'aire contenant les données de prédiction, la largeur de l'intervalle de crédibilité augmente significativement. Toutefois, la prédiction demeure excellente (et fiable) lorsque l'on reste dans l'aire évoquée précédemment, et ce en dépit du fait que les données d'entraînement soient situées complètement à l'opposé du puits de la gaussienne. Si l'on s'intéresse au graphique sur grille large, on se rend compte que, même si l'incertitude augmente, la forme "puits de gaussienne" reste bien définie : plus on se rapproche du centre de la grille, plus les valeurs de la moyenne augmentent, et inversement lorsqu'on s'en éloigne. Cet exemple met donc bien en exergue la puissance de Magma, adapté au cas multidimensionnel.

5.6 Adaptation de la vignette *Introduction to Magma 2D* avec la nouvelle version

Une fois cette nouvelle version 2D mise au point, il a fallu modifier la vignette 2D précédemment créée. En effet, ***Introduction to Magma 2D*** s'appuie intégralement sur l'ancienne version 2D de Magma ; en particulier, la fonction **simu_db()** que l'on utilise pour simuler aléatoirement une BDD n'est pas vraiment adaptée à la 2D. Plus précisément, l'Output généré par **simu_db()** n'est pas corrélé avec la variable Covariate. Celle-ci n'a donc aucun intérêt si l'utilisateur désire réaliser une prédiction avec Magma.

Hugo s'est donc occupé de développer une nouvelle version de **simu_db()** afin que l'Output soit corrélé avec la variable Covariate. Une fois **simu_db()** adaptée, j'ai pu mettre à jour la vignette ***Introduction to Magma 2D***. Les changements effectués sont les suivants :

- Dans la section **Data simulation**, une fois que la BDD a été simulée, on applique la fonction **regularize_data()** pour la régulariser. Une nouvelle fois, j'ai expliqué à l'utilisateur quelle était l'utilité de chaque paramètre ;
- Dans la section **Even prettier graphics**, je propose une comparaison des prédictions obtenues avec Magma 2D et les GPs classiques en 2D sous la forme de GIFs. En effet, on remarque qu'avec un GP classique, la prédiction s'améliore nettement au fur et à mesure que les données se rajoutent sur la heatmap. Si l'on ne dispose que de 2 ou 3 points de prédiction, le GP a du mal à retrouver la forme originale du processus sous jacent (voir Figures 10 et 11). C'est exactement le comportement que l'on observait déjà en 1D : tant que l'on dispose de données pour l'individu à prédire, le GP individuel a un bon comportement (*ie* il passe à proximité des données, de manière assez lisse). En revanche, dès que ces données viennent à manquer, le GP individuel plonge à la moyenne a priori sans tirer parti des informations apportées par les données d'entraînement.
  Au contraire, avec Magma, la prédiction est excellente même si l'on dispose de très peu de points pour l'individu à prédire, illustrant une nouvelle fois la force du partage d'information entre les individus (voir Figures 12 et 13).

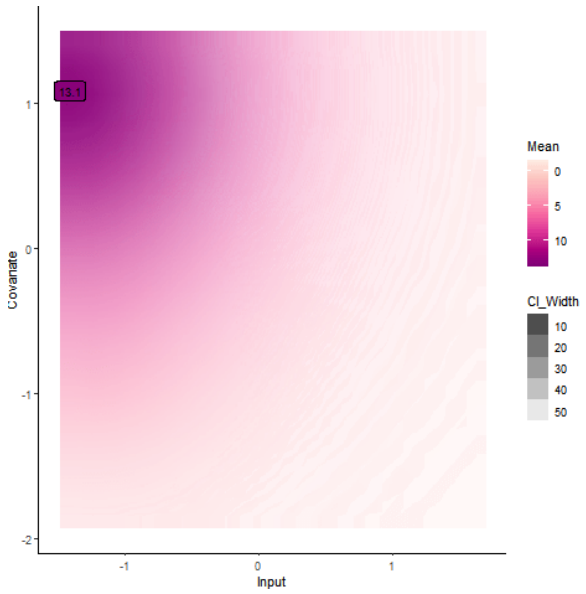


Figure 10:
Graphique de la prédiction obtenue avec la version 2D des GPs et une seule donnée pour l'individu à prédire.

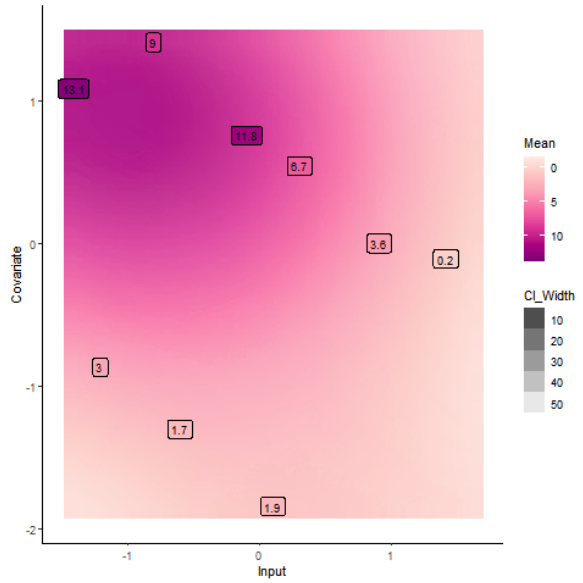


Figure 11:
Graphique de la prédiction obtenue avec la version 2D des GPs et 10 données pour l'individu à prédire.

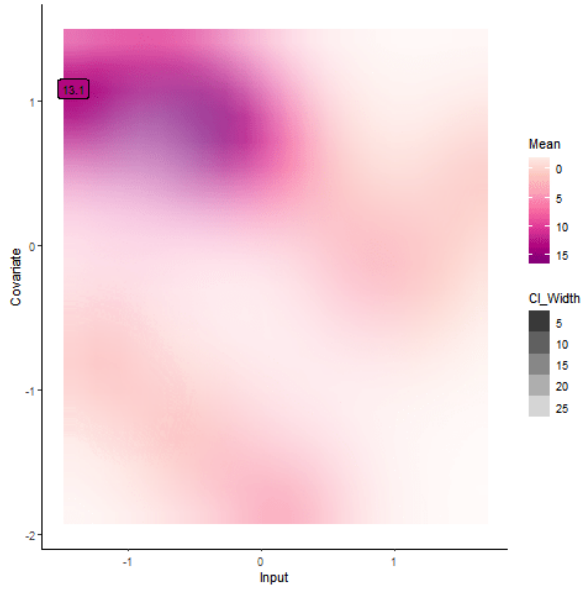


Figure 12:
Graphique de la prédiction obtenue avec la nouvelle version 2D de Magma et une seule donnée pour l'individu à prédire.

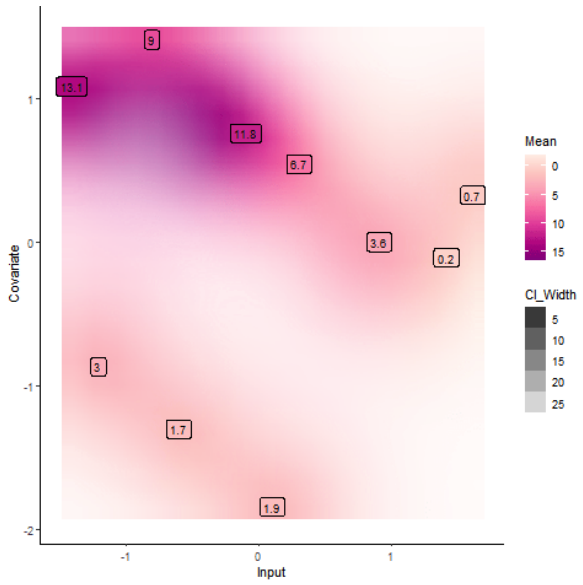


Figure 13:
Graphique de la prédiction obtenue avec la nouvelle version 2D de Magma et 10 données pour l'individu à prédire.

6 Conclusion : regard rétrospectif sur l'ensemble du stage

Maintenant que l'essentiel du contenu de ce stage a été retranscrit sur ce rapport, il me paraît important de faire un retour sur les différents apports personnels que j'en ai tiré.

En commençant ce stage, je ne connaissais pas du tout le package R **tidyverse**, qui est extrêmement pratique si l'on souhaite manipuler une BDD complexe. Les premières semaines m'ont permis de me familiariser avec ce package, qui s'est révélé par la suite être un précieux outil pour le codage de la nouvelle version de Magma. Puisque je souhaite travailler en tant que data scientist, cette étape de manipulation et traitement des données me paraît être essentielle pour débuter.

D'un point de vue plus mathématique, je pense avoir assimilé beaucoup de connaissances sur les GPs. A l'INSA, nous avons eu un cours de GPs au cours du 2ème semestre de 4ème année ; toutefois, il s'agissait plus d'une introduction aux GPs qu'une étude poussée de ces derniers. En commençant le stage, je ne savais pas simuler un GP sur R ... Que de chemin parcouru depuis ! J'ai apprécié aborder cet outil :

- sous l'angle théorique d'abord, en lisant les articles de M. Leroy pour comprendre le fonctionnement de Magma. A plusieurs reprises au cours du stage, il a fallu travailler des démonstrations assez complexes, qui manipulaient les GPs de façons encore jamais abordées en cours ;
- sous l'angle pratique ensuite, en manipulant les (hyper-)paramètres des GPs et la valeur de la moyenne a priori pour étudier leur influence sur les processus.

Grâce à ce stage, j'ai pu mieux saisir le fonctionnement des GPs et me faire une idée de ce qu'il était possible de modéliser grâce à eux.

Dans un autre registre, j'ai apprécié que M. Leroy me laisse suivre mon idée de début de stage (*ie* définir un nouveau modèle pour MagmaClust avec un partage d'hyper-paramètres communs au sein d'un même cluster). Il m'a aiguillée lorsque j'avais des questions à lui poser et m'a permis d'aller au bout de mon intuition de départ. Malgré l'échec de cette nouvelle feature, l'expérience a tout de même été enrichissante : j'ai pu davantage comprendre le fonctionnement de Magma et de MagmaClust (qui restait assez obscur par endroits, même après avoir lu les articles) et être plus à l'aise en langage R. J'ai aussi pu développer mon esprit critique avec l'analyse des différents résultats en sortie, en essayant de détecter les cas pour lesquels cette feature pouvait être intéressante. Même si, au final, mon idée n'était pas réellement applicable (à cause du mélange de gaussiennes), cela aura permis à M. Leroy de gagner du temps en n'explorant pas davantage cette piste, qu'il avait envisagée lors du développement du package.

Le travail sur les vignettes a quant à lui été très enrichissant ; en particulier, il m'a permis de développer des compétences dans plusieurs domaines différents, du traitement de données à la rédaction, en passant par la manipulation du package. Pour chacun des graphiques présentés, j'ai dû explorer les bases de données pour choisir l'exemple qui illustrait le mieux ce que je cherchais à montrer. De plus, les grilles sur lesquelles j'effectuais la prédiction ont dû être modifiées à de nombreuses reprises pour que l'on obtienne exactement l'effet recherché (pas trop larges pour ne pas tasser les données et minimiser l'intérêt de Magma en affichant des segments / aires d'Input(s) pour lesquels on ne disposait plus de données, et pas trop étroites non plus pour que l'on distingue suffisamment les 3 grandes phases de Magma).

Il m'a également fallu étudier plus en détails les différentes fonctions du package pour m'y familiariser et être plus à l'aise avec leurs paramètres. L'utilisation de ces fonctions sur ces bases de données "réelles" a notamment permis de faire remonter plusieurs petits bugs présents dans les fonctions du package ; M. Leroy a pu corriger certains points problématiques qui n'apparaissaient pas sur des bases de données artificielles. Enfin, n'ayant pas pour habitude d'être très concise dans mes rapports à l'INSA, j'ai dû apprendre à me

concentrer sur l'essentiel et ne pas assommer le lecteur de trop de détails. Les vignettes m'ont également permis d'améliorer mon niveau linguistique en apprenant à réfléchir directement en anglais (plutôt que de penser en français et traduire ensuite).

Enfin, la dernière mission de ce stage (qui n'était pas prévue au départ) a été la réalisation de la nouvelle version 2D de Magma. Avec Hugo, nous l'avons abordée comme un nouveau défi à relever. L'ampleur du travail à réaliser était assez importante : il a fallu repenser le format des données en entier, parcourir toutes les fonctions du package pour modifier tous les morceaux qui devaient l'être et bien comprendre l'utilité de chacune d'entre elles. Nous avons notamment appris à utiliser le debugger **browser()** de R, que nous ne connaissions pas du tout et qui nous a été d'une aide précieuse pour modifier les fonctions du package. Nous avons également dû corriger tous les petits bugs présents dans le code et testé nos fonctions en modifiant les valeurs des paramètres pour être sûrs que tous les cas de figure soient correctement traités par l'algorithme. Enfin, nous avons nettoyé notre code pour qu'il soit facile à lire et à comprendre pour l'utilisateur : en particulier, nous avons veillé à respecter au maximum l'alignement des pipes du **tidyverse** et à ne jamais dépasser la limite des 80 caractères sur une même ligne de code (dans le but d'éviter les problèmes d'affichage sur GitHub).

Cette expérience a sans doute été la plus enrichissante ; je pense pouvoir dire que mon niveau en programmation R a bien évolué depuis le début du stage, tout comme ma façon d'aborder une tâche complexe. J'ai vraiment appris à découper un grand problème en plusieurs sous-problèmes plus simples à résoudre, et avancer progressivement en testant chaque nouveauté pour être assurée de son bon fonctionnement. J'ai aussi eu le sentiment de faire partie de la grande aventure qu'est Magma, puisque notre contribution a permis de faire significativement évoluer le package proposé au départ. J'ai pu explorer tous les aspects d'un package complexe, de son implémentation à son utilisation, en passant par la correction des bugs et l'explication de son fonctionnement.

Pour conclure, je tenais à remercier M. Leroy pour toutes les choses qu'il a faites pour nous. Nous avons eu beaucoup de difficultés d'un point de vue administratif en amont de ce stage ; il s'est toujours rendu disponible, a été très réactif et à l'écoute. Sans lui, je pense que nous ne serions jamais partis à Sheffield ...

Il a également été un super maître de stage, toujours prêt à répondre à nos questions, à l'écoute de nos suggestions. C'est aussi un très bon pédagogue, qui nous a poussé à donner le meilleur de nous-mêmes et appris à être rigoureux et propres dans notre manière de coder en R. Il a toujours été encourageant et de bon conseil, ce qui est vraiment très appréciable pour un stagiaire. La diversité des tâches qu'il nous a confiées constitue aussi, à mon sens, un énorme point positif : nous ne nous sommes pas ennuyés en répétant les mêmes tâches tout au long du stage. Enfin, je tenais à le remercier d'avoir accepté de nous prendre en binôme. Je pense que nous n'aurions clairement pas pu accomplir autant de tâches différentes et être aussi efficaces en étant seuls. Cela nous aura également permis d'apprendre à se répartir le travail équitablement, à nous entraider et à tirer parti des qualités de notre binôme.

Enfin, ce stage m'aura permis de mettre un second pied dans le monde de la data appliquée au sport. Après une première expérience avec le CREPS de Toulouse au cours du projet de 4ème année, j'ai pu découvrir une autre facette de ce milieu, davantage orientée mathématiques et plus rigoureuse. M. Leroy étant cofondateur du groupe français de "Sport et Statistique", il nous a fait part de son expérience personnelle, ce qui m'a permis de me faire une idée plus précise de cet univers, encore méconnu et assez restreint aujourd'hui. Grâce à ce stage je pense désormais avoir les armes nécessaires pour aborder ma future vie professionnelle (dans le sport, je l'espère) avec plus de confiance et de sérénité.

7 Bibliographie

- *MAGMA: Inference and Prediction using Multi-Task Gaussian Processes with Common Mean*, A. Leroy, P. Latouche, B. Guedj, S. Gey - Machine Learning - 2022 (https://link.springer.com/article/10.1007/s10994-022-06172-1);
- *Multi-task learning models for functional data and application to sports performances prediction*, A. Leroy - PhD Thesis - 2020 (https://arthur-leroy.netlify.app/files/Thesis-Arthur_LEROY.pdf);
- *Cluster-Specific Predictions with Multi-Task Gaussian Processes*, A. Leroy, P. Latouche, B. Guedj, S. Gey - PREPRINT - 2020 (https://arxiv.org/pdf/2011.07866.pdf);
- *The Kernel Cookbook: Advice on Covariance functions*, David Duvenaud (https://www.cs.toronto.edu/ duvenaud/cookbook/).