INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

The University Of Sheffield.

# Rapport de stage de 4ème année

Département Mathématiques Appliquées

Année scolaire 2021 - 2022

Etudiant : Hugo Lelièvre,
lelievre@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

Pour mon stage de 4ème année, j'ai opté pour l'Université de Sheffield. J'ai décidé de le réaliser là bas sous la tutelle d'Arthur Leroy pour différentes raisons. Tout d'abord, je devais passer 16 semaines à l'étranger pour valider mon diplôme INSA. Voyant cette contrainte comme une occasion de progresser en anglais, mon choix s'est rapidement porté sur l'Angleterre. Ensuite, j'ai pour objectif de travailler dans les mathématiques appliquées au sport une fois mon diplôme INSA obtenu. Ainsi, le stage proposé par M. Leroy s'est présenté comme une parfaite opportunité pour découvrir une nouvelle facette de ce milieu et engranger de l'expérience.

En particulier, nous avons travaillé avec Alexia Grenouillat sur une base de données fournie par la Fédération Française de Natation (FFN), que M. Leroy avait lui-même exploitée durant sa thèse, et qui l'a conduit à inventer les algorithmes Magma et MagmaClust, avant de développer son package R MagmaClustR. L'objectif de la FFN était le suivant : développer un outil d'aide à la décision pour la fédération, basé sur des méthodes statistiques et / ou des algorithmes de machine learning, et qui permet d'identifier les jeunes nageurs prometteurs. Tout au long de monn stage j'ai pu utiliser cette base de données comme exemple, et j'ai pu ainsi découvrir l'étude des données fonctionnelles (FDA), et des multiples séries temporelles supposées partager de l'information commune, généralement observées à pas de temps irréguliers.

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

- approfondir mes connaissances sur les GPs. Sheffield étant la ville où se regroupe le plus de chercheurs dans ce domaine, 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 efficacement une base de données (BDD) sur R ;
- 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 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 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'expliciter 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 Découverte des données FFN et du package MagmaClustR

## 2.1 Base de données des nageurs

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.

Afin d'optimiser au mieux mon utilisation de R, et rendre mon code propre et réutilisable, j'ai appris à utiliser les packages du tidyverse. Ces packages contiennent de nombreuses fonctions très utiles pour manipuler des données, avec une syntaxe simple d'utilisation et de compréhension.

Après avoir appris à manipuler notre BDD et l'avoir explorée, j'ai commencé à écrire mes premiers scripts R pour l'organiser et la rendre utilisable pour Magma et MagmaClust. J'ai notamment fait :

- Une fonction permettant de filtrer les données, en gardant uniquement les lignes correspondant à l'épreuve, le sexe et la taille de bassin choisis. J'ai conservé uniquement les performances réalisées entre 10 et 20 ans, dans un objectif de prédiction des jeunes talents. De plus, cette fonction sélectionne uniquement les nageurs ayant au moins 5 performances dans l'épreuve choisie, afin d'avoir des courbes d'évolution plus pertinentes. Pour rendre les données utilisables par Magma et MagmaClust, cette fonction conserve uniquement les colonnes Identifiant, Age et Temps, qu'elle renomme ID, Input et Output. Enfin, cette fonction classe la BDD en sortie par ID, et par ordre croissant des âges.
- Une fonction pour supprimer les doublons. En effet, beaucoup de nageurs avaient plusieurs temps au même âge, cela est dû au format des compétitions : ils réalisent une course en série le matin, et s'ils sont qualifiés ils réalisent une deuxième course en finale l'après-midi. Cependant, un GP ne peut pas passer en même temps à 2 points différents, cette fonction a donc permis de filtrer les données pour garder uniquement la meilleure performance de la journée.
- Une fonction pour sélectionner aléatoirement un nombre N1 d'individus pour l'entrainement, et un nombre N2 pour le test. Cette fonction prend également en argument alpha, qui correspond au pourcentage de données à cacher. On obtient en sortie 3 BDD : data_train qui contient les lignes correspondant aux individus d'entrainement, data_pred qui contient les lignes des individus à prédire, et data_test qui contient les données cachées des individus à prédire, qu'on comparera avec les prédictions.

Grâce à ces fonctions, nous avons pu organiser nos données pour appliquer Magma et MagmaClust sur toutes les courses qui nous intéressaient. Nous avons alors pu prendre en main les différentes fonctions du package MagmaClustR. Que ça soit avec Magma ou MagmaClust, le processus général se décline en 3 grandes étapes :

- L'entrainement du modèle ;
- La prédiction ;
- L'affichage des résultats.

## 2.2 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 :

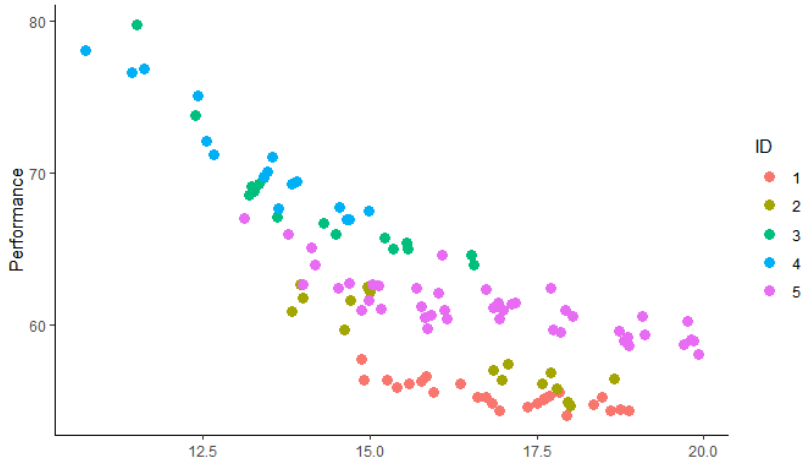
- La base de données **FFN**. Ici, les Outputs correspondent aux performances des nageurs sur différentes courses ; les Inputs, à l'âge du nageur au moment de ladite performance. Comme tous les nageurs ne concourent pas aux mêmes âges (l'âge, en années, peut prendre jusqu'à 2 décimales), la grille de 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)).

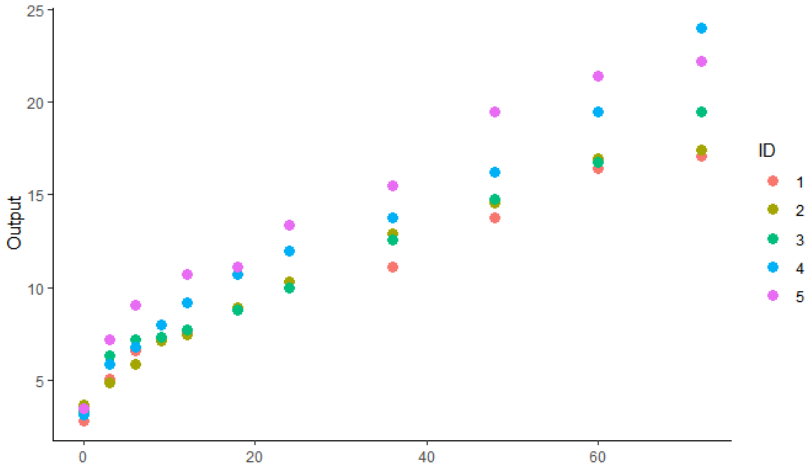
Notre objectif est réussir à produire, grâce à MagmaClust, des courbes d'évolution des performances des nageurs / du poids des enfants, et de prédire leurs futurs Outputs. On cherche également à tenir compte de la présence de structures de groupes dans les données à travers l'aspect "clustering" de MagmaClust.

## 2.3 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 :




(a)




(b)

Figure 1:

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

- $t_i = \{t_i^1, ..., t_i^{N_i}\}$ 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, pour MagmaClust, 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 $\sum_{k=1}^{K} \pi_k = 1$.

## 2.4 Définition du modèle Magma

Supposons que les données fonctionnelles proviennent de la somme d'un processus moyen, commun à tous les individus, et d'un processus centré spécifique à l'individu. Soit T l'espace des inputs, le modèle est alors :

$$y_i(t) = \mu_0(t) + f_i(t) + \epsilon_i(t), \forall t \in \mathcal{T}$$

où :

- $\mu_0(.) \sim GP(m_0(.), k_{\theta_0}(.,.))$ est le GP moyen ;
- $f_i(.) \sim GP(0, \sum_{\theta_i}(.,.))$ est le GP spécifique au i-ème individu ;
- $\epsilon_i(.) \sim GP(0, \sigma^2)$ correspond à un terme d'erreur.

Ce modèle général dépend des paramètres qui correspondent à des choix de modélisation, et des hyper-paramètres que l'on doit estimer :

- $m_0(.)$ est la moyenne *a priori* ;
- $k_{\theta_0}(.,.)$ est le noyau de covariance d'hyper-paramètres $\theta_0$ ;
- $\forall i \in \mathcal{I}$, $c_{\theta_i}(.,.)$ est le noyau de covariance d'hyper-paramètres $\theta_i$ ;
- $\forall i \in \mathcal{I}$, $\sigma_i^2$ est le bruit de la variance associée au i-ème individu ;
- $\theta = \{\theta_0 \{\theta_i\}_i, \{\sigma_i^2\}_i\}$ est l'ensemble des hyper-paramètres du modèle qui sont à apprendre.

On admet que :

- $f_{ii}$ sont indépendants ;
- $\epsilon_{ii}$ sont indépendants ;
- $\forall i \in \mathcal{I}, \mu_0, f_i, \epsilon_i$ sont indépendants.

La Figure 2 récapitule les différentes dépendances entre les variables du modèle.

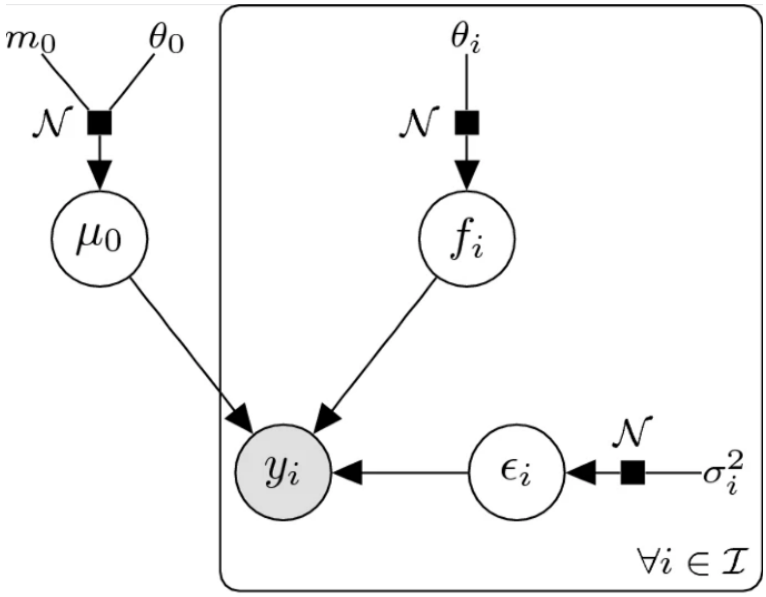



Figure 2: Graphique modélisant les dépendances entre variables au sein de Magma.

## 2.5 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 d'un processus individuel centré :

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

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 $\gamma_k$ ;
- $\forall i \in \mathcal{I}$, $\sum_{\theta_i}(.,.)$ est le noyau de covariance d'hyper-paramètre $\theta_i$ ;
- $\forall i \in \mathcal{I}$, $\sigma^2$ 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 :

- $\mu_{kk}$ sont indépendants ;
- $f_{ii}$ sont indépendants ;
- $Z_{ii}$ sont indépendants ;
- $\epsilon_{ii}$ 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 3.

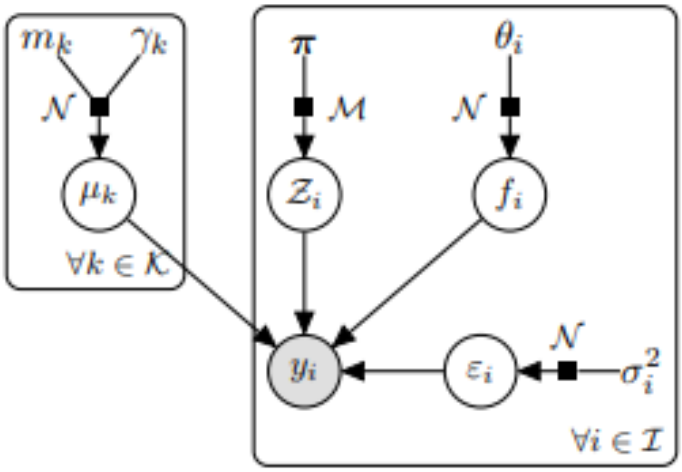



Figure 3: Graphique modélisant les dépendances entre variables au sein de MagmaClust.

## 2.6 Influence des hyper-paramètres sur le comportement des GP 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 ma fonction pour supprimer les doublons.

J'ai ensuite 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.

### 2.6.1 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.

### 2.6.2 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 Vignettes du package MagmaClustR

## 3.1 Objectifs 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 succinctes : 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 Alexia, nous avons décidé de nous occuper de 2 vignettes chacun ; j'ai donc rédigé et mis en forme *Gaussian Process regression* et *Introduction to MagmaClust*.

## 3.2 *Gaussian Process regression*

Il n'existe pas de package dans R permettant de tracer simplement des processus gaussiens. Ainsi, en plus de Magma et MagmaClust, Mr Leroy a décidé d'implémenter dans son package des processus gaussiens classiques. Cette vignette avait donc pour but d'illustrer le fonctionnement des processus gaussiens, et de montrer leur utilité dans un cas concret. Mr Leroy m'a demandé de faire une vignette courte et synthétique

pour que les utilisateurs puissent comprendre au premier coup d'œil comment utiliser les GP, et reproduire l'exemple de la vignette dans leur propre cas.

Bien que beaucoup plus succinte que les vignettes de Magma et MagmaClust, l'organisation de la vignette GP reste globalement la même :

- Présentation du jeu de données ;
- Mise en forme du dataset ;
- Entraînement ;
- Prédiction ;
- Affichage du résultat.

Avec Alexia, nous avons décidé d'utiliser la même nageuse pour la vignette "Gaussian Process regression" et celle "Introduction to Magma". Cette idée est venue naturellement afin de pouvoir comparer les performances des 2 algorithmes. On peut voir ci-dessous les données de notre nageuse.

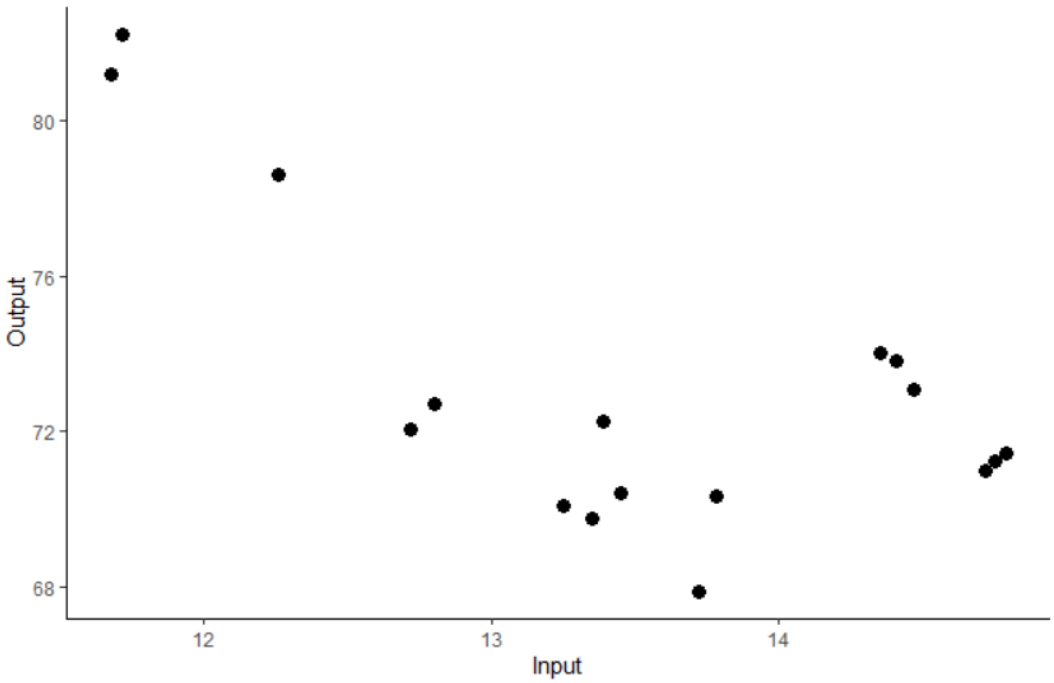



Figure 4: Données de la nageuse sélectionnée.

J'ai ensuite illustré les 3 étapes de l'algorithme. J'ai expliqué l'intérêt des paramètres comme le choix du noyau ou de la moyenne à priori pour l'entraînement, et le choix de la grille d'inputs sur laquelle effectuer la prédiction.

Une fois les étapes d'entraînement et de prédiction éffectuées, il ne restait plus qu'à afficher le résultat. On peut voir la courbe du GP en violet assortie d'un intervalle de crédibilité à 95% en rose, et des points de notre nageuse en noir.

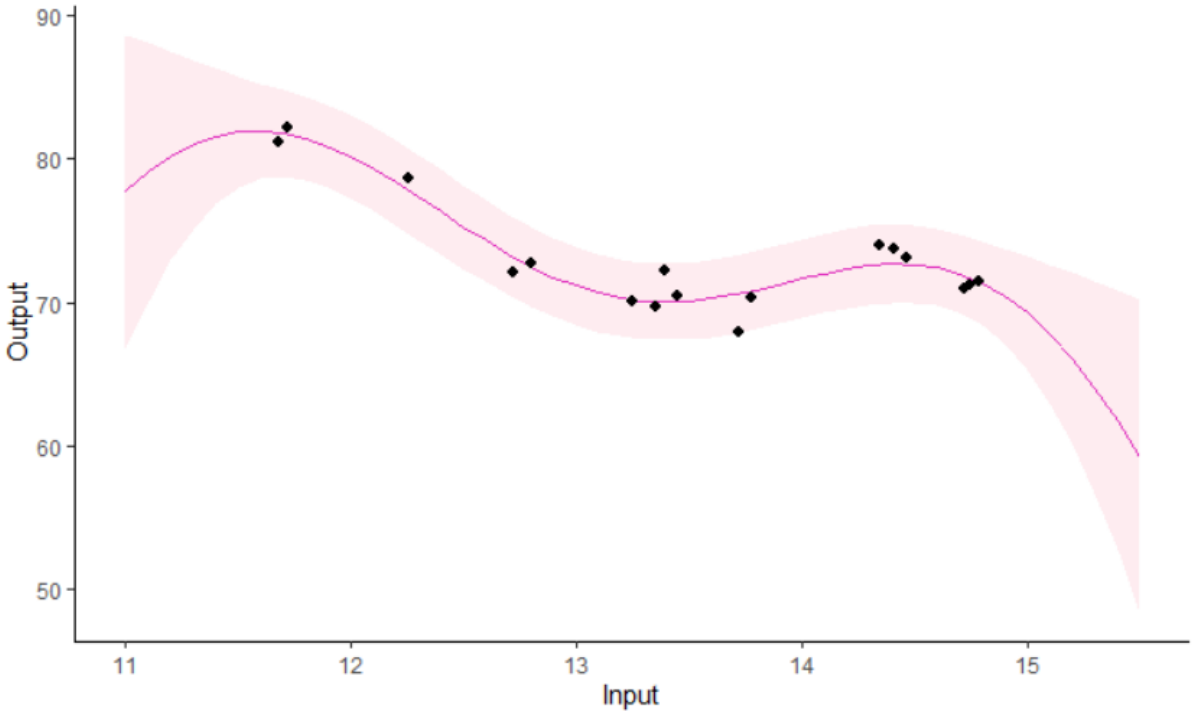



Figure 5: GP de la nageuse sélectionnée.

### 3.3 *Introduction to MagmaClust*

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 poids des enfants. 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 poids des enfants âgés de 0 à 6 ans). Contrairement à la vignette *Introduction to Magma* rédigée par Alexia, les données sont régulières : les poids des enfants sont observés à intervalles de temps réguliers (ce qui est probablement dû à des visites obligatoires chez le pédiatre).

Pour avoir une idée de l'allure des données, j'ai commencé par afficher les poids de 5 enfants de 0 à 6 ans sélectionnés aléatoirement (voir Figure 6). On remarque bien que les trajectoires sont assez similaires : elles croissent assez rapidement lors des premiers mois de la vie des enfants, puis l'augmentation devient plus linéaire à partir de 18 mois. Toutefois, on distingue clairement des différences de profil des enfants ; certains sont corpulents à la naissance mais leur prise de poids ralentit rapidement, tandis que d'autres naissent plus fins mais grandissent plus vite. La structure sous-jacente de groupes au sein de cette base de données souligne donc l'intérêt d'utiliser MagmaClust plutôt que Magma.

La principale difficulté que j'ai rencontrée lors de la rédaction de cette vignette était de réussir à montrer toutes les nouveautés de MagmaClust par rapport à Magma, et donc de bien choisir l'individu sur lequel réaliser la prédiction pour que le lecteur se rende pleinement compte de l'utilité de MagmaClust.

Afin d'être cohérent avec la vignette *Introduction to Magma* d'Alexia, j'ai organisé les principales parties de la même façon :

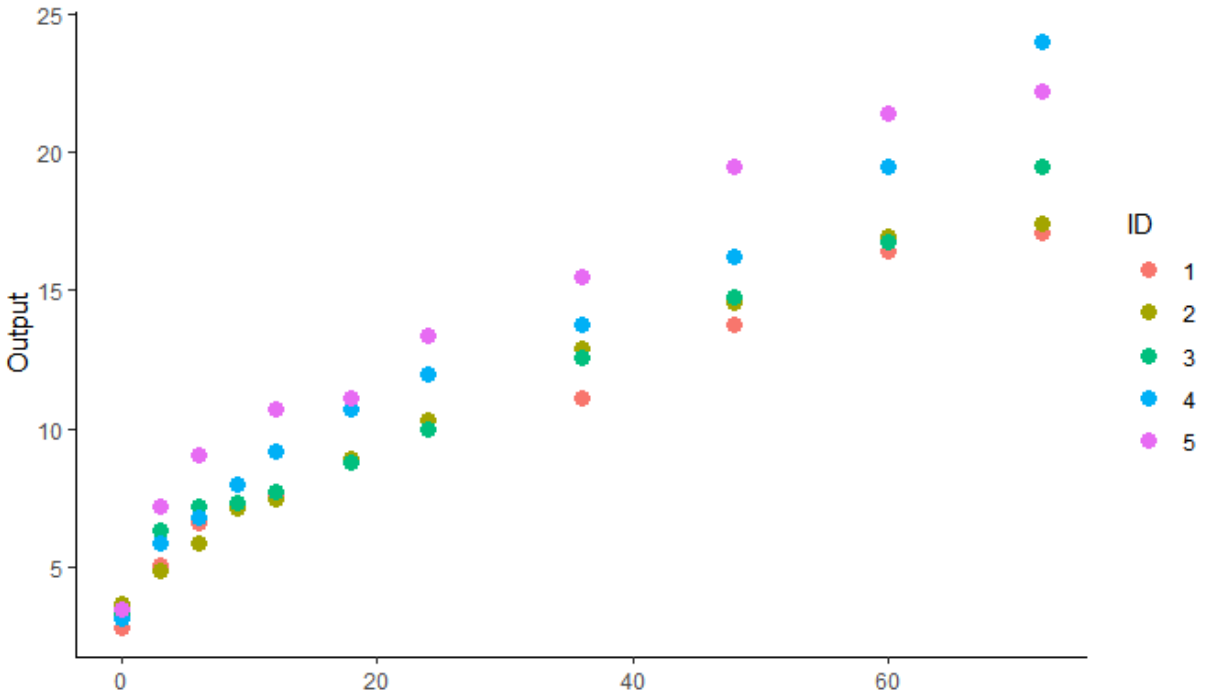



Figure 6: 5 enfants sélectionnés aléatoirement dans la BDD poids.

- Présentation du jeu de données ;
- Mise en forme du dataset pour le rendre utilisable par MagmaClust : comme pour Magma, la base de données doit obligatoirement contenir les colonnes ID, Input et Output. L'utilisateur doit donc renommer les colonnes de son jeu de données, et éventuellement en supprimer certaines (ici, la colonne Sexe).
- Séparation des données en un échantillon d'apprentissage et un échantillon d'entraînement ;
- Manipulation de la fonction **select_nb_cluster()** pour connaître le nombre de clusters sous-jacents dans notre base de données. Ici, on en détecte 3 ;
- L'entraînement du modèle avec la fonction **train_magmaclust()**. J'ai précisé quels étaient les paramètres avec lesquels jouer pour qu'ils soient adaptés à notre base de donnés ;
- La prédiction grâce à la fonction **pred_magmaclust()**. J'ai spécifié moi-même la grille d'inputs pour que notre prédiction soit comprise entre 0 et 6 ans ;
- L'affichage de la prédiction avec la fonction **plot_magmaclust()**. Par rapport à Magma, comme on a 3 clusters, on affiche 3 processus moyens en pointillés.

A la demande de M. Leroy, j'ai comparé les prédictions que l'on obtenait avec Magma et avec MagmaClust pour le même enfant de la BDD poids (voir Figure 7). On remarque bien que l'unique processus moyen de Magma ne tient pas compte des structures de groupes au sein des enfants, rendant la prédiction beaucoup moins bonne que MagmaClust. Cette dernière retrouve le cluster d'appartenance de l'individu à prédire, ce qui permet d'améliorer considérablement la prédiction.

On peut cependant remarquer des oscillations dans les GP moyens des clusters, elles proviennent de plusieurs choses :

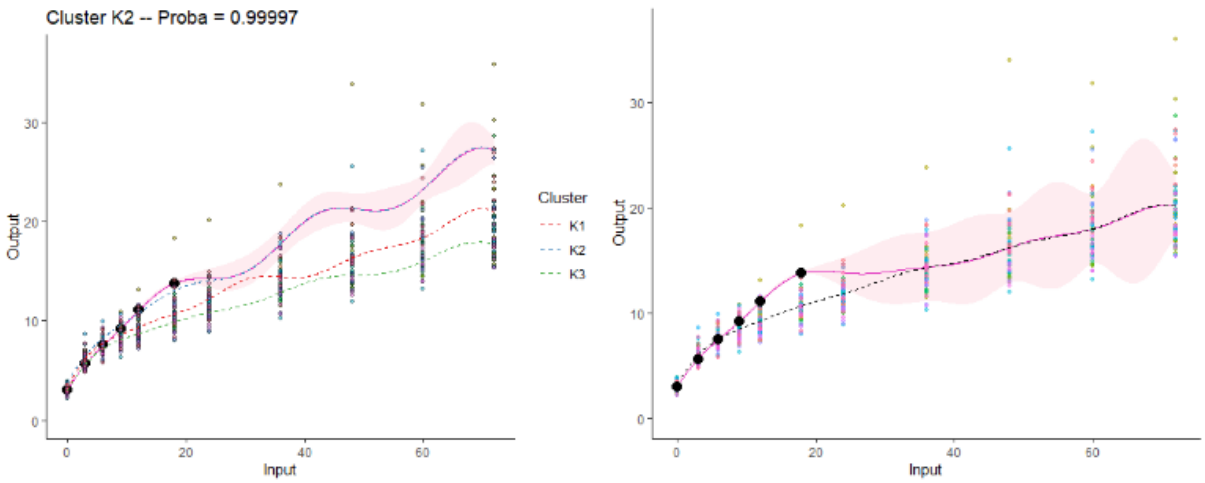



Figure 7: Comparaison de la prédiction de MagmaClust (gauche) avec celle de Magma (droite).

- Les données elles-mêmes contiennent des oscillations ;
- Nous avons pris des noyaux SE, qui favorisent les oscillations ;
- Les données ne sont observées qu'à des moments particuliers.

# 4 Nouvelle version du package plus adaptée aux Inputs multidimensionnels

Une fois les vignettes réalisées et l'objectif initial du stage atteint, M. Leroy nous a lancés sur un projet de plus grande envergure : changer l'implémentation du package pour les inputs multidimensionnels, afin d'apporter 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.

## 4.1 Problème 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.

## 4.2 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 :

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

```

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 8).

| 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 8: Nouveau format de la BDD après l'ajout de la variable Référence.

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 :

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

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.

## 4.3 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.

## 4.4 Ajout de fonctions utiles pour l'utilisateur

### 4.4.1 Créer une grille d'inputs

Lors de notre codage des algorithmes pour des inputs multidimensionnels, nous avons eu besoin de simuler des données pour tester notre code. En particulier, nous avons besoin de grilles d'inputs sur lesquelles effectuer la prédiction. Alexia a donc créé la fonction **expand_grid_inputs()**. Cette fonction permet à l'utilisateur de rentrer, pour chaque variable d'input, le vecteur des valeurs sur lequel il souhaite évaluer son processus, et elle renvoie en sortie une BDD contenant toutes les combinaisons possibles. On a ainsi une grille complète, de la précision choisie par l'utilisateur, qu'on peut directement passer en argument de **pred_gp()**, **pred_magma()** ou **pred_magmaclust()**.
En utilisant la fonction **expand_grid_inputs** comme suit, avec 3 variables d'input, on obtient la sortie affichée en Figure 9.

```
expand_grid_inputs(Input = seq(0,1,1),
                   Covariate_1 = seq(1,2,1),
                   Covariate_2 = seq(-1,1,1))
```

| | 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 9: Grille obtenue en sortie de la fonction **expand_grid_inputs()**.

### 4.4.2 Pré-traitement des données

En travaillant sur la BDD nageurs, j'ai été confronté au problème d'arrondis sur les Inputs. Plus précisément, pour définir la grille d'âge sur laquelle on effectue le training, plusieurs choix étaient envisageables :

- soit on arrondissait les âges des nageurs à 2 ou 3 décimales ;
- soit on choisissait de faire une grille régulière : on définit un pas de temps régulier de manière à avoir 1000 points d'Input possibles entre 10 et 20 ans. Ensuite, pour chaque performance réalisée, on arrondit l'âge associé à la valeur d'Input de la grille créée la plus proche.

Dans les deux cas, si, après arrondi, deux performances différentes (ou plus) d'un même nageur sont associées au même âge (*ie* dans notre cas, cela signifie que les 2 performances - ou plus - ont été réalisées à moins de 3 jours d'intervalle), on peut choisir de garder le minimum, le maximum, la moyenne, la médiane ... En une dimension, la logique adoptée par M. Leroy optait pour la seconde option ; j'ai donc décidé d'adapter le même raisonnement en 2D par souci de cohérence.

Ainsi, j'ai créé la fonction **regularize_data()**, qui permet à l'utilisateur de pré-traiter sa base de données (en particulier, de la régulariser). Le code est disponible ci-dessous :

```
regularize_data <- function(data,
                            size_grid = 30,
                            scale = FALSE,
                            grid_inputs = NULL,
```

```
                                summarise_fct = base::mean) {
  if (data %>% is.data.frame()) {
    if (!all(c("ID", "Output") %in% names(data))) {
      stop(
        "The 'data' argument should be a tibble or a data frame containing ",
        "at least the mandatory column names: 'ID', 'Output'"
      )
    }
  } else {
    stop(
      "The 'data' argument should be a tibble or a data frame containing ",
      "at least the mandatory column names: 'ID', 'Output'"
    )
  }
  ## Function to scale columns
  scale_function <- function(data) {
    (data - mean(data)) / sd(data) %>% return()
  }

  ## Get the Input columns names
  names_col <- data %>%
    dplyr::select(-.data$ID, -.data$Output) %>%
    names()

  ## Scale the Input columns if required
  if (scale) {
    data <- data %>%
      dplyr::mutate_at(names_col, scale_function)
  }

  if (is.null(grid_inputs)) {
    ## Put the data on a grid node
    fct_round <- function(data, size_grid) {
      round_step <- (max(data) - min(data)) / size_grid
      data <- data %>%
        plyr::round_any(round_step)
    }

    data %>%
      dplyr::mutate_at(names_col, fct_round, size_grid) %>%
      dplyr::group_by_at(c("ID", names_col)) %>%
      dplyr::summarise_all(summarise_fct) %>%
      dplyr::ungroup() %>%
      return()
  } else {
    if (!(setequal(names(grid_inputs), names_col))) {
      stop("Input column names in grid_inputs must be the same as in data.")
    } else {
      round_col <- function(col_name) {
        vector_input <- data %>% dplyr::pull(col_name)
        vector_grid_input <- grid_inputs %>%
          dplyr::pull(col_name) %>%
          unique() %>%
          sort()

        vector_grid_input[match_closest(vector_input, vector_grid_input)] %>%
          return()
      }

      inputs <- sapply(names_col, round_col) %>%
        as_tibble()
      tibble::tibble(ID = data$ID, inputs, Output = data$Output) %>%
        dplyr::group_by_at(c("ID", names_col)) %>%
        dplyr::summarise_all(summarise_fct) %>%
        dplyr::ungroup() %>%
```

```
70            return()
71        }
72    }
73  }
```

On commence par vérifier que l'argument fourni en entrée est bien une base de données ; en particulier, elle doit être au même format que lorque l'on veut utiliser Magma.

J'ai ensuite laissé à l'utilisateur la possibilité de centrer et réduire ses données ; cette fonctionnalité peut être appréciable pour donner la même importance à chaque variable si elles ne sont pas du même ordre de grandeur.

Il est également possible pour l'utilisateur de fournir une grille d'inputs sur laquelle il souhaite que ses données soient arrondies. Cette grille peut éventuellement provenir de la fonction **expand_grid_inputs()**. Il faut toutefois être bien vigilant :

- La grille d'inputs en entrée doit avoir le même nombre d'Inputs que la BDD que l'on souhaite régulariser ;
- Les noms de colonnes de la grille d'inputs doivent être strictement les mêmes que ceux de la BDD de départ.

Une fois ces précautions prises, on utilise la fonction **match_closest()** du package **MALDIquant**. Elle prend deux vecteurs en entrée (dans notre cas, une des colonnes d'Inputs et la colonne de la grille d'inputs éponyme) : pour chaque valeur du premier vecteur, elle trouve la valeur du second vecteur qui en est la plus proche. On effectue la même opération pour toutes les colonnes d'Inputs de notre BDD. Si plusieurs points se retrouvent avec les mêmes coordonnées (*ie* toutes les valeurs d'Inputs sont égales) mais des valeurs d'Output différentes, l'utilisateur est libre d'effectuer l'opération qu'il souhaite grace à l'argument **summarise_fct** : il peut garder le minimum / le maximum / la moyenne / la médiane des Outputs. Le choix lui revient en fonction des données qu'il manipule.

Si l'utilisateur ne fournit pas de grille d'inputs en entrée, la fonction lui en crée une elle-même : pour chaque colonne d'Inputs, on prend le minimum et le maximum et on divise par la taille de la grille (qui peut être plus ou moins précise selon les besoins de l'utilisateur). Ainsi, on obtient le "pas" de la colonne (*ie* l'écart constant entre 2 valeurs de la grille). On itère ensuite ce procédé sur toutes les colonnes d'Inputs.

En sortie, on obtient donc une nouvelle base de données "régularisée", qui rend l'utilisation de Magma plus optimale. Cette fonction permet surtout à l'utilisateur de gagner du temps et d'éviter de pré-traiter ses données à la main.

L'intérêt majeur de cette fonction est qu'elle permet à l'utilisateur de contrôler la taille de la matrice de covariance du GP. L'inversion de cette matrice étant l'opération demandant le plus de temps de calcul dans les algrithmes **Magma** et **MagmaClust**, **regularize_data()** permet d'approximer les inputs sans trop perdre de précision, tout en gagnant beaucoup de temps.

### 4.4.3 Ajout d'une version 2D à simu_db()

Mr Leroy a codé une la fonction **simu_db()**, qui permet à l'utilisateur de simuler des données. Il peut notamment choisir le nombre d'individus, le nombre de points par individu, la présence et le nombre de clusters dans les données. La fonction simule alors des hyper-paramètres aléatoires, et évalue le GP correspondant sur certains inputs sélectionnés au hasard. On obtient alors des données avec des fluctuations gaussiennes.

Pour tester notre version multidimensionnelle des algorithmes, nous avions besoin de données multidimensionnelles également. Mr Leroy avait déjà laissé la possibilité d'ajouter une colonne Covariate aux données simulées par **simu_db()**. Cependant, les valeurs de la colonne Covariate étaient des valeurs tirées aléatoirement, avec aucun impact sur la valeur d'Output. Dans l'optique d'utiliser Magma en dimension supérieure, il fallait des données où chaque dimension d'Input avait un réel intérêt pour avoir des résultats intéressants. Etant donné qu'on ne peut pas représenter des données en plus de 2 dimensions (car il y a également la variable d'Output), j'ai donc décidé d'ajouter la possibilité de simuler des données 2D dans **simu_db()**. Pour cela, les données ne correspondent plus à une courbe avec des oscillations gaussiennes, mais à une nappe avec des bosses et des creux.

## 4.5 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.

L'ancienne version de la fonction **simu_db()** n'étant pas adaptée à la 2D, on a commencé par simuler 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()** 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 10, en haut à gauche. On remarque que la gaussienne n'est pas parfaite et qu'il y a un léger bruit. Cela s'explique par l'approximation des inputs sur une grille relativement large. Cependant, on reconnait facilement l'allure des données, et en pratique il y a toujours des imprécisions de mesure, donc on se rapproche encore plus d'un cas concret.

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.

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, tous les graphes étant à la même échelle, on se rend bien compte que 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 10 (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 10 (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).

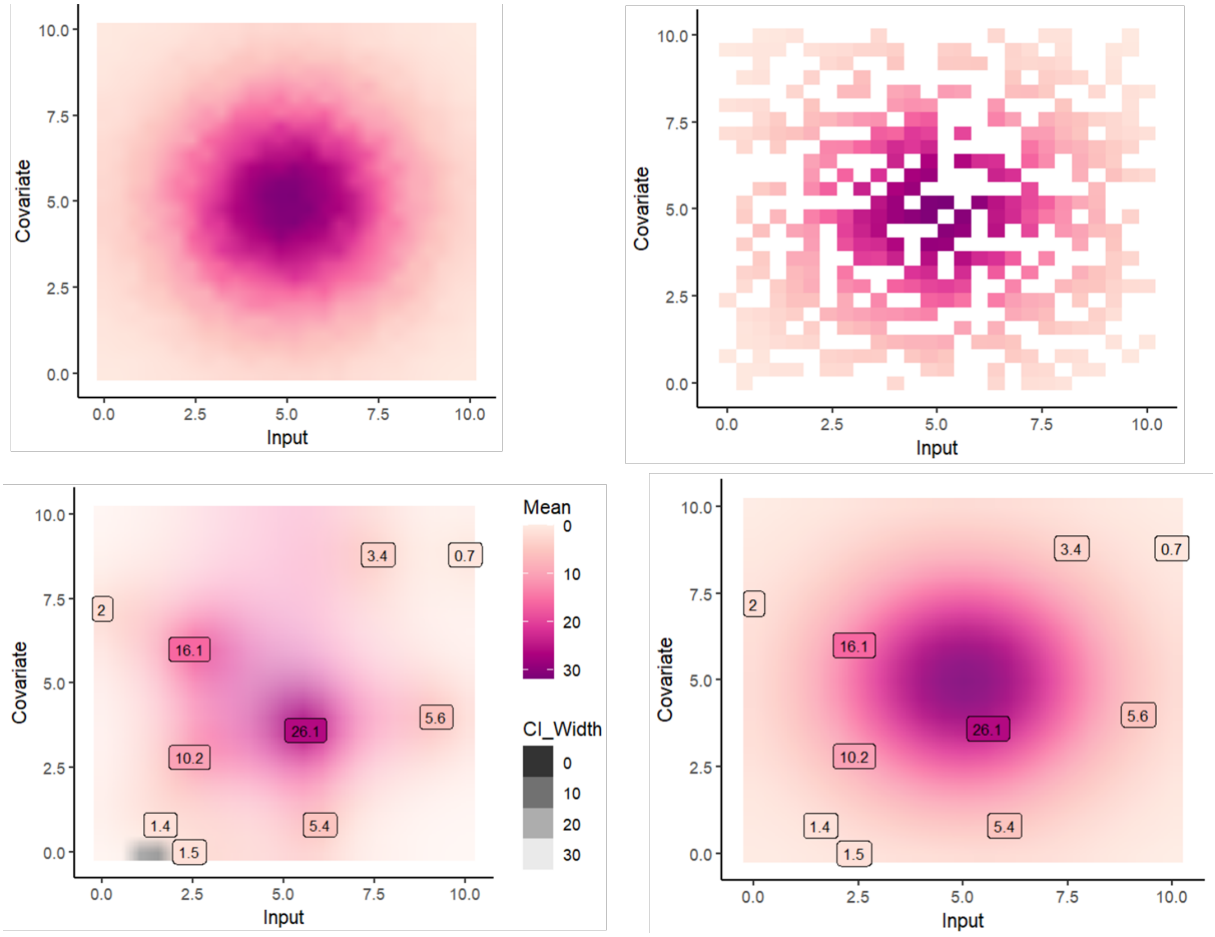
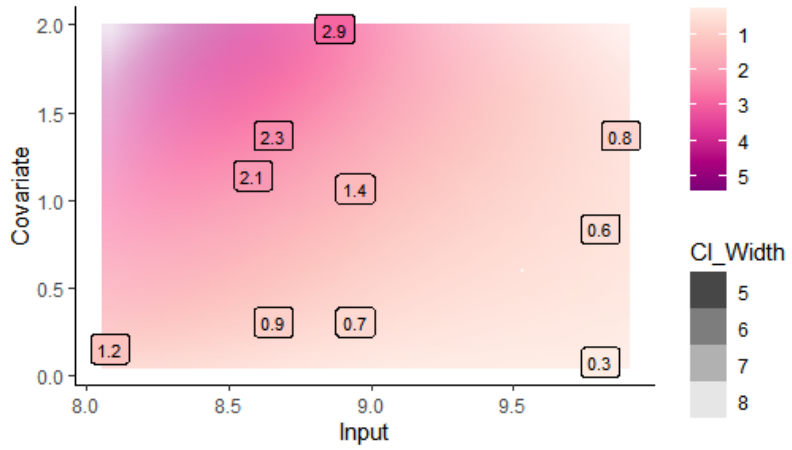



Figure 10:

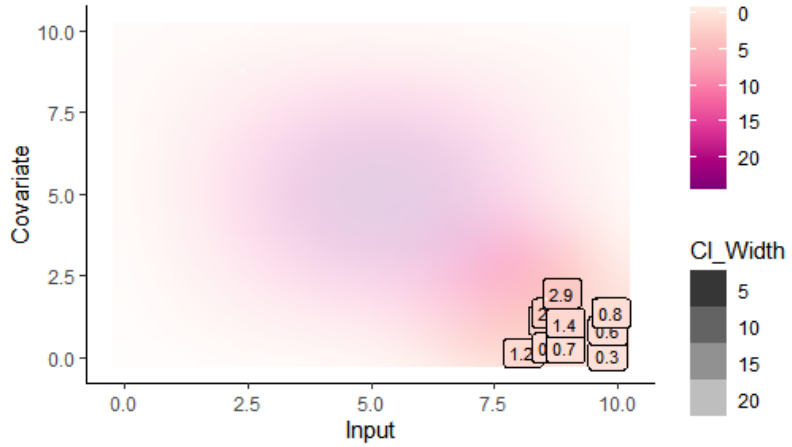
En haut à gauche : densité de la gaussienne 2D centrée en (5,5).
En haut à droite : points ayant servi à l'entraînement du modèle.
En bas à gauche : prédiction obtenue avec l'ancienne version 2D de Magma.
En bas à droite : prédiction obtenue avec la nouvelle version 2D de Magma.

Avec 50 individus d'entraînement, on obtient les graphiques de la Figure 11.

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.




(a)




(b)

Figure 11:

(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.

# 5 Conclusion et bilan sur l'ensemble du stage

Pour clôturer ce rapport, j'aimerais effectuer un petit bilan de l'ensemble du stage, et de tout ce qu'il m'a apporté. Tout d'abord au niveau de l'anglais, j'ai lu beaucoup d'articles de différents chercheurs, j'ai donc appris beaucoup de vocabulaire technique dans le domaine des mathématiques, et en particulier des GP. Effectuer un stage en Angleterre, entouré de personnes parlant anglais était une très bonne occasion pour améliorer ma compréhension et ma pratique de la langue, même si je dois l'avouer, l'accent du Yorkshire est assez dur à comprendre au début... Grâce à Mr Leroy, nous avons également pu assister à un séminaire sur le machine learning. Cette journée m'a permis de découvrir ce qu'était un séminaire dans le monde de la recherche, les travaux sur lesquels les différents chercheurs travaillent, et d'écouter leurs présentations. C'était aussi l'occasion de discuter avec tous les gens présents et de d'échanger en anglais. Ensuite, au

niveau théorique, où j'ai appris beaucoup de choses en particulier sur les GP. Bien qu'ils ne soient pas totalement nouveaux pour moi puisqu'on les avait étudiés en cours durant le second semestre de 4eme année, il ne s'agissait que d'une partie d'un cours optionnel, et donc mes connaissances sur le sujet restaient assez succinctes. J'ai ainsi pu découvrir :

- la différence entre un intervalle de confiance et un intervalle de crédibilité ;
- différents noyaux de covariance, leurs spécificités et leurs impacts sur l'allure du processus gaussien correspondant ;
- la façon dont on utilise le noyau de covariance et la moyenne pour tracer un processus ;
- le concept d'hyper-paramètres et leur influence sur l'allure de la courbe ;

Grâce à Magma et MagmaClust, j'ai pu comprendre les limites des GP classiques au niveau de la prédiction en l'absence de données, et en particulier l'importance de la moyenne dans ce cas. J'ai découvert le fonctionnement des algorithmes EM (Expectation Maximization), et VEM (Variationnal EM), la différence entre moyenne a priori et a posteriori. L'idée derrière Magma et MagmaClust, qui est d'utiliser les informations apportées par d'autres individus afin de prédire l'évolution d'un nouvel individu, est relativement intuitive. Cependant, les équations qui permettent de mettre en pratique cette idée sont relativement complexes, et il m'aura fallu prendre beaucoup de recul pour avoir les idées claires sur le fonctionnement des algorithmes. Ainsi, j'ai donc travaillé dans un cadre très théorique, et mis un premier pied dans le monde de la recherche.

Enfin, ce stage m'a permis d'énormément progresser en R. Ayant un niveau assez basique, je termine ce stage en me sentant très à l'aise en R, et capable de coder de nombreuses choses. J'aimerais remercier Mr Leroy qui a joué un grand rôle là-dedans en nous donnant des conseils et en nous expliquant les bonnes habitudes à prendre pour coder efficacement, proprement, et pour avoir un code facile à comprendre pour les autres utilisateurs. Il nous a notamment fait découvrir les différents packages du tidyverse, parmi lesquels on retrouve dplyr, un package qui permet de manipuler des données grâce à des fonctions simples et intuitives, ou ggplot2 qui permet de créer des graphiques de tous types simplement. Grâce à tout cela j'ai pu commencer à coder des fonctions et utiliser le debugger de R.

Après avoir bien progressé en R et terminé les vignettes, Mr Leroy nous a proposé de coder une nouvelle version du package en changeant l'aspect multidimensionnel. Nous avons accepté avec plaisir et nous nous sommes lancés dans ce challenge ambitieux. J'ai ainsi pu voir l'organisation d'un package, la répartition des fonctions dans les différents fichiers, la documentation du code, et me rendre compte de la difficulté pour avoir une cohérence dans l'ensemble du code de tous les fichiers. J'ai dans le même temps pu découvrir tous les fichiers autres que ceux du code qu'il faut ajouter dans un package R, comme le README, la description du package, les bibliothèques à importer pour faire tourner le code etc... C'était aussi l'occasion d'apprendre à mieux manipuler GitHub, la façon dont on télécharge un package en développement dessus ou comment push sur une branche. Je suis très fier d'avoir pu apporter ma contribution au package, et heureux de savoir qu'elle sera utile à Mr Leroy et aux utilisateurs de MagmaClustR. Je suis très content d'avoir pu découvrir les algorithmes de Mr Leroy qui permettent d'améliorer grandement la prédiction future des GP, de les avoir compris et maîtrisés. Ils sont des outils très précieux que je suis heureux de savoir utiliser, et dont je me servirai à nouveau dans le futur avec certitude.

Pour conclure j'aimerais remercier Mr Leroy pour sa gentillesse, sa patience et sa bienveillance tout au long du stage, et même avant où il a dû se battre avec l'administration pour que nous soyons acceptés en stage à Sheffield. Il a toujours pris le temps de nous expliquer avec pédagogie quand nous en avions besoin sans jamais nous dénigrer. Je suis très heureux d'avoir pu effectuer mon stage avec lui et je n'aurais pas pu rêver meilleur encadrant. Donc pour les éventuels étudiants à la recherche de stage qui liraient ce rapport : foncez !

# 6 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/).