Geoffray Bizouard
Stage de fin d'étude - Master 2
Du 1er avril au 28 septembre 2012.
Soutenu le 13 septembre 2012.


(UFR Sciences et Techniques)

---

# Master MIGS (Mathématiques pour l'Informatique Graphique et les Statistiques)

---

# Méta-modélisation : état de l'art et application

Maître de stage : Robert Faivre
Encadrants : Hélène Raynal, Ronan Trépos, Stéphane Couture


Unité de Biométrie et Intelligence Artificielle (Bia)
Castanet-Tolosan

3 septembre 2012

# Remerciements

Je remercie tout d'abord Hélène Raynal, Robert Faivre, Ronan Trépos, Stéphane Couture mes encadrants de stage, pour m'avoir permis de réaliser ce sujet de stage, et de m'avoir fait bénéficier de leur temps tout le long du stage.

Je souhaite également remercier Hervé Cardot le responsable du master MIGS ainsi que l'ensemble des enseignants du master pour les cours réalisés ces deux dernières années.

Merci à l'ensemble de l'unité BIA pour m'avoir accueilli chaleureusement, en particulier aux autres stagiaires et aux thésards.

# Table des matières

# Table des figures

# Liste des tableaux

# 1 Introduction

## 1.1 L'INRA : Unité Biométrie et Intelligence Artificielle

L'Institut National de la Recherche Agronomique (INRA) est un organisme de recherche scientifique publique menant des recherches afin d'améliorer la gestion des ressources de notre environnement, d'assurer la protection de notre environnement, diriger notre agriculture vers une alimentation saine et de qualité tout en étant compétitive et durable. Afin de réaliser ces recherches, l'INRA est constitué de 14 départements scientifiques répartis sur 19 centres de recherche régionaux. Mon stage s'est déroulé dans l'unité de Biométrie et Intelligence Artificielle (BIA) à Toulouse qui a pour mission de mettre à la disposition de l'INRA des méthodes et des compétences en mathématiques et en informatique appliquées. L'unité BIA est composée de deux équipes de recherche : SaAB (Statistique et Algorithmique pour la Biologie), MAD (Modélisation des Agro-écosystèmes de Décision) et de deux plate-formes : GENOTOUL (plate-forme bio-informatique) et RECORD (plate-forme de modélisation et de simulation des agro-écosystèmes). C'est dans le cadre de la plate-forme RECORD que j'ai effectué mon stage.

## 1.2 Contexte et objectif du stage

La plate-forme RECORD (RÉnovation et COorDination de la modélisation de cultures pour la gestion des agros-écosystèmes [1]) de modélisation et de simulation a pour objectif d'aider à la conception et à l'évaluation des systèmes de production. Elle propose aussi un ensemble de modèles biophysiques de simulation de nature relativement complexe. Il est souvent, pour certaines disciplines, nécessaire de disposer de modèles analytiques afin de réaliser l'optimisation de variables de décisions. La connaissance détaillée de modèles biophysiques rend ce travail parfois difficile. Il peut alors être utile d'avoir recours à un méta-modèle tiré du modèle originel afin de faciliter l'optimisation.

Notre but est donc d'effectuer une synthèse de différentes approches de méta-modélisation par expérimentation numérique. Les différentes méthodes de méta-modélisation abordées sont principalement issues de l'article de Nathalie Villa-Vialaneix (2011) [10]. L'objectif est de faire émerger des techniques de nature générique qui pourront ensuite être implémentées sur la plate-forme et de tester les différentes méthodes de méta-modélisation sur un modèle de simulation

1. Adresse de la plate-forme : http ://www4.inra.fr/record

existant sur la plate-forme afin de pouvoir les comparer et les analyser.

Dans une première partie, nous introduirons la méta-modélisation et des outils nécessaires à la création et à l'analyse des méta-modèles. Ensuite, nous présenterons les différentes approches de méta-modélisation abordées et leur utilisation par le logiciel **R**. Finalement, nous analyserons les résultats obtenus sur un modèle de simulation disponible sur la plate-forme RECORD et nous comparerons les différentes approches de méta-modélisation.

# 2 Méta-modélisation

Nous allons voir ici ce qu'est un méta-modèle et la méta-modélisation, puis comment construire ces méta-modèles et finalement comment évaluer les méta-modèles et les comparer. Nous verrons dans les parties suivantes plusieurs méthodes de méta-modélisation. Parmi celles-ci, nous présenterons des approches statistiques comme le modèle linéaire, les modèles additifs généralisés et la méthode du krigeage, et des méthodes axées sur l'apprentissage ou la classification comme les réseaux de neurones, les forêts aléatoires et les supports vecteurs machines. Il existe de nombreuses autres méthodes de méta-modélisation comme les méthodes non linéaires, les méthodes non paramétriques que nous ne présenterons pas dans le cadre de ce travail. Les méthodes abordées ici sont des méthodes "classiques", a priori "robustes" et automatisables.

## 2.1 Définition

La manière la plus courante de définir un méta-modèle est de le désigner comme un modèle du modèle. On peut trouver les méta-modèles derrière d'autres appellations comme émulateur, approximateur, modèle simplifié ou surface de réponse. L'objectif est de pouvoir explorer un modèle complexe plus facilement et de pouvoir l'évaluer, le simplifier beaucoup plus rapidement. Les méta-modèles nous mènent à la notion de méta-modélisation qui consiste à créer un méta-modèle afin de simplifier et ainsi de pouvoir exploiter plus facilement le modèle initial. De plus, on peut utiliser ces méta-modèles pour calculer des indices de sensibilité lorsque le code est trop lourd ou lorsqu'il y a trop de variables.

Afin de pouvoir construire un méta-modèle, nous avons besoin d'un jeu de données de $n$ observations contenant $p$ variables explicatives et des variables de sortie du modèle à expliquer (ici restreintes à une). Les variables explicatives se présenteront sous la forme d'une matrice $X \in \mathcal{M}_{n,p}(\Re)$ et la variable à expliquer sous la forme d'un vecteur $Y \in \Re^n$.

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

On a notre modèle initial :

$$Y = Modèle(variables)$$

et on veut avoir le méta-modèle suivant :

$$Y \approx \textit{Méta-Modèle}(variables) + \epsilon$$

$\epsilon$ est l'erreur de modélisation (différence entre valeurs observées et valeurs données par le méta-modèle). Il n'y a pas d'erreur de mesure car le modèle est créé par expérimentation numérique. La création des méta-modèles passera par la minimisation de l'erreur de modélisation.

## 2.2 Plan d'expérience

Pour créer un méta-modèle, on a besoin d'une matrice $X$ contenant un ensemble de points d'expérience et du vecteur des sorties $Y$ correspondant. Ce dernier sera établi en passant $X$ par le modèle. La matrice $X$ constitue le plan d'expérience. Il faut donc que les $n$ observations soient suffisamment bien réparties sur l'ensemble de définition des variables explicatives pour pouvoir créer un bon méta-modèle. Il existe plusieurs méthodes pour créer des plans d'expérience.

### 2.2.1 Hypercube latin

Pour créer un hypercube latin (LHS : Latin Hypercube Sampling), on commence par découper l'axe $[0, 1]$ en $n$ segments de même longueur. Ensuite, on va faire le produit cartésien de $d$ fois cet intevalle. Ainsi, on obtient un maillage de dimension $p$ composé de $n^p$ cellules de même taille. Par exemple, en dimension 2, on va placer une observation par ligne/colonne. On étend ce principe aux dimensions supérieures. Pour finir, on place aléatoirement l'observation dans la case sélectionnée.

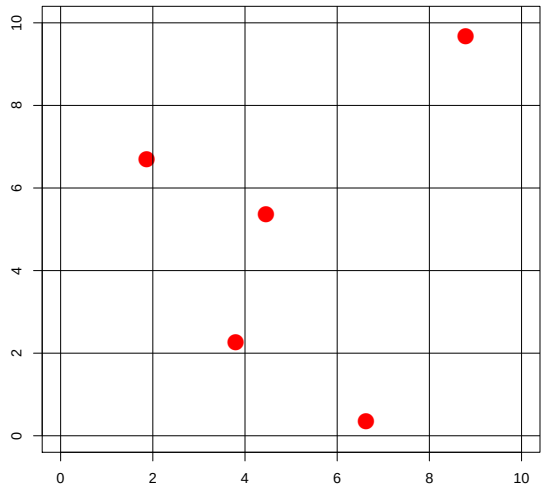


FIGURE 2.1 – Exemple de LHS en dimension 2

Sous **R**, on peut créer des plans par LHS grâce à la fonction randomLHS(n,k) ou n est le nombre d'observations voulues pour le plan d'expérience et k le nombre de dimensions. Il existe des méthodes améliorant la qualité des plans LHS (optimLHS,...) mais le temps de création du plan est fortement augmenté. On se contentera donc de plans LHS simples ou de répétition de LHS.

### 2.2.2 Suite de Sobol

Une seconde méthode pour créer des plans d'expérience est l'utilisation de suites de Sobol. L'idée est de répartir les points dans l'espace des variables en minimisant les "trous" entre les observations. Ceci est fait par les suites à faible discrépance. La discrépance est la déviation de la répartition des observations par rapport à une répartition uniforme. Elle mesure donc l'irrégularité de la distribution. Les suites de Sobol font partie de ces suites à faible discrépance qui mènent à une meilleure couverture de l'espace.

Pour générer une suite de Sobol en dimension 1, on choisit un polynôme primitif sur le corps $\mathbb{Z}_2 = \{0, 1\}$ :

$$P(x) = x^l + a_1 x^{l-1} + a_2 x^{l-2} + ... + a_{l-1} x + 1$$

Avec $a_1, ..., a_{l-1}$ prenant pour valeurs 0 ou 1. On définit la suite $(m_k)$ de la manière suivante :

$$m_k = 2a_1 m_{k-1} \oplus 2^2 a_2 m_{k-2} \oplus ... \oplus 2^{l-1} a_{l-1} m_{k-l+1} \oplus 2^l m_{k-l} \oplus m_{k-l}$$

Où $l$ est le degré du polynôme choisi, $a_i$ les coefficients du polynôme et $\oplus$ l'opérateur logique "ou exclusif bit à bit". Les valeurs initiales $m_1, ..., m_l$ doivent être des entiers impairs tels que $1 \leq m_k \leq 2^k$ pour $k = 1, ..., l$. Une suite de Sobol en dimension 1 est définie comme suit :

$$x^i = \frac{1}{2^k}(\oplus_{j=1}^{k} p_j m_j)$$

où $(p_1 ... p_m)$ est la représentation binaire de i.

Pour créer des suites de Sobol dans un espace de dimension supérieure à 1, il suffit de créer les autres composantes de $x^i$ en choisissant un polynôme primitif distinct de celui des autres composantes.

## 2.3 Comparaison des modèles

Il est intéressant d'avoir des critères de comparaison pour les différents méta-modèles afin de pouvoir estimer leur qualité. Pour commencer, nous avons besoin de deux jeux de données, un pour apprendre le modèle et un pour tester la qualité du modèle. Lors de la phase d'apprentissage du modèle, il faudra faire attention au problème de surapprentissage (overfitting) qui consiste à trop bien suivre les données du jeu d'apprentissage au détriment de la qualité d'estimation/prédiction sur un nouveau jeu de données. Ce phénomène peut être évité de plusieurs manières comme par validation croisée à la création du modèle ou par le principe même

du modèle comme les forêts aléatoires qui évitent le problème en construisant le modèle par des répétitions.

On notera $\bar{y}$ la moyenne du vecteur $y$ et $\hat{y}_i$ la valeure prédite par le modèle. L'estimation de la qualité du modèle peut se faire à deux niveaux. Le premier consiste à évaluer le modèle pour la prédiction sur le jeu d'apprentissage. Pour cela, on peut utiliser les critères du $R^2$ (coefficient de détermination) et $R^2$-ajusté correspondant au pourcentage de la variance totale expliquée par le modèle (le $R^2$-ajusté prend en compte le nombre de variables).

$$R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}, \qquad R^2\text{-}ajusté = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

Le second niveau se situe à l'estimation de la qualité du modèle sur un nouveau jeu de données. On utilisera ici le MSEP (mean square error prediction) :

$$MSEP = \frac{\sum_i (\hat{y}_i - y_i)^2}{n}$$

Ce dernier critère n'est pas influencé par le surapprentissage ; par conséquent, il peut paraître plus fiable.

# 3 Méta-Modèles

Nous allons étudier ici plusieurs approches de méta-modélisation : modèle linéaire, modèle additif généralisé, forêt aléatoire, réseau de neurones, krigeage, support vecteur machine. On verra également comment utiliser ces méthodes sur le logiciel **R**.

## 3.1 Régression linéaire

La méthode la plus connue de méta-modélisation est la régression linéaire. Celle-ci consiste à exprimer la relation entre les variables explicatives et la variable à expliquer de manière linéaire. On pourrait également voir les modèles linéaires généralisés qui eux s'appliquent sur des variables discrètes.

### 3.1.1 Modèle linéaire

On veut ici décrire le modèle et son fonctionnement. On veut donc trouver une fonction $f$ linéaire en $X$ telle que $Y = f(X)$ où $Y$ est notre variable à expliquer et $X$ l'ensemble des p variables explicatives.

On cherche un modèle linéaire simple, c'est-à-dire sans terme de degré supérieur à 1. On pose le modèle linéaire simple :

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \epsilon_i$$

On pose $\theta = [\beta_0, \beta_1, ..., \beta_n]^T$. Sous forme matricielle :

$$Y = X\theta + \epsilon$$

Afin d'utiliser ce modèle, nous devons déterminer le vecteur $\theta$. On va l'estimer par méthode des moindres carrés. On dispose pour cela d'un échantillon de $n$ observations $(Y_i, X_i)$ et on veut $\theta$ qui minimise l'erreur d'estimation au carré :

$$\min_{\theta} \sum_{i=1}^{n} (Y_i - X_i\theta)^2$$

Sous forme matricielle :

$$\min_{\theta}(Y - X\theta)'(Y - X\theta)$$

$X'$ est la transposée de $X$. On obtient l'estimateur des moindres carrés :

$$\hat{\theta} = (X'X)^{-1}X'Y$$

**Démonstration :**

$(Y - X\theta)'(Y - X\theta) = Y'Y - 2\theta'X'Y + \theta'X'X\theta$

Pour que ce soit minimal, il faut que :

$\frac{\partial(Y'Y - 2\theta'X'Y + \theta'X'X\theta)}{\partial\theta} = 0$

Donc :

$-2X'Y + 2X'X\theta = 0$

$\Rightarrow \theta = (X'X)^{-1}X'Y$ ■

On a ainsi déterminé le paramètre $\hat{\theta}$ nous permettant de prédire la sortie Y sur un nouveau jeu de données selon notre modèle.

### 3.1.2 Cas polynomial

Dans la plupart des cas, le modèle précédent est trop simple pour pouvoir bien suivre les données. C'est pourquoi nous rajoutons des termes de degré supérieur au modèle permettant de prendre en compte les interactions entre variables. Le modèle s'écrit maintenant (pour degré = 2) :

$$Y = \beta_0 + \sum_{j=1}^{p}\beta_j X_j + \sum_{j=1}^{p}\beta_{jj}X_j^2 + \sum_{j=1}^{p}\sum_{l=j+1}^{p}\beta_{jl}X_jX_l + \epsilon$$

La matrice $X$ est alors composée de :

$$X = \begin{bmatrix} x_{i1} & \cdots & x_{ip} & x_{i1}^2 & x_{i1}x_{i2} & \cdots \end{bmatrix},\ i = 1, ..., n$$

On peut étendre ceci aux polynômes d'ordre 3, 4... . Comme précédemment, on pose $\theta = [\beta_0, \beta_1, ..., \beta_n, \beta_{11}, ..., \beta_{pp}]^T$. L'estimateur des moindres carrés $\theta$ s'obtient de la même manière que pour le modèle linéaire simple :

$$\hat{\theta} = (X'X)^{-1}X'Y$$

### 3.1.3 R : fonction lm()

La création de modèle de régression linéaire sous R se fait par le biais de la fonction lm(). Son utilisation est simple :

```
lm(formula,data)
```

L'argument data de la fonction doit contenir les données du modèle sous forme de dataframe. formula sert à donner la forme voulue pour le modèle, par exemple dans un cas simple y ~$x_1$ + $x_2$ + ....
On peut également ajouter des termes de degré supérieur afin d'obtenir des modèles polynomiaux. Pour ce faire, il est possible d'utiliser la fonction polym(..., d) qui évalue les coefficients du polynôme de degré d, de manière à ce que le polynôme soit orthogonal, ce qui facilite ainsi les calculs (la matrice $X'X$ est orthogonale). L'argument ... doit contenir l'ensemble des variables du modèle sous forme de liste (V1, V2, V3, ...) et d est le degré voulu pour le modèle. L'élément ... de la fonction polym n'est pas utilisable de manière générique c'est pourquoi nous avons écrit la fonction polym_bis (le code est présent en annexe) permettant d'entrer la matrice $X$ des variables à la place de ... . Ainsi pour un modèle polynomial de degré 2 on écrira :

lm(y ~polym_bis(X,2))

De plus, les fonctions polym et polym_bis ont un inconvénient. Il devient rapidement impossible de les exécuter pour des données ayant trop de variables (avec 15 variables cela ne fonctionne pas). C'est pourquoi on a créé la fonction polym_ter présente en annexe réglant ce problème.

En faisant un summary() de notre sortie, il peut être intéressant de regarder l'élément *Coefficients* présenté sous forme de tableau, nous donnant le vecteur $\theta$ avec *Estimate* et une estimation de l'information apportée par la variable. On peut également regarder le $R^2$-*ajusté* nous donnant la qualité du modèle estimé sur les données d'apprentissage.
On a vu précédemment qu'il était possible de simplifier le modèle en sélectionnant uniquement les variables les plus informatives. Sous R, il est possible d'effectuer cette sélection grâce à la fonction step(object) où object est le modèle créé par la fonction lm().

```
Call:
lm(formula = r ~ ., data = HL)

Residuals:
    Min      1Q  Median      3Q     Max
-67.315  -4.683   0.127   4.316  62.044

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.7726     0.5058   7.458  1.9e-13 ***
V1           -0.6001     0.1752  -3.425 0.000641 ***
V2            0.0508     0.1754   0.290 0.772098
V3           -0.1638     0.1753  -0.934 0.350388
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16 on 996 degrees of freedom
Multiple R-squared: 0.0126,Adjusted R-squared: 0.009625
F-statistic: 4.236 on 3 and 996 DF,  p-value: 0.005496
```

TABLE 3.1 – Exemple de sortie lm()

La prédiction de nouvelles données sur le modèle linéaire créé se fait par la fonction predict(lm,newdata) avec lm le modèle créé et newdata le jeu de données sur lequel effectuer la prédiction.

## 3.2 Modèle additif généralisé

Le modèle additif généralisé (GAM) est une extension du modèle linéaire généralisé. La principale différence sera pour le prédicteur qui ne sera pas forcément linéaire pour GAM. On suppose ici que l'on peut approcher notre variable Y par une combinaison linéaire de fonctions non paramétriques. On pourra se référer à Wood S. (2006) [11]

### 3.2.1 Modèle

Pour les modèles GAM, il est nécessaire que la fonction de distribution de Y appartienne à la famille exponentielle. Le modèle se présentera ainsi :

$$g(\mu_i) = \alpha + \sum_{j=1}^{p} f_j(x_{ij})$$

où $\mu_i = E[y_i]$. g est une fonction connue, appelée fonction de lien, monotone et deux fois dérivables. Cette fonction peut avoir plusieurs formes comme logistique ou identité (menant aux modèles additifs). Les fonctions $f_j$ sont des fonctions d'une ou plusieurs variables décomposables sur une base de fonction (polynomiale, trigonométrique, splines, ...). Chaque $f_j$ aura donc pour forme :

$$f(x) = \sum_{i=1}^{q} \beta_i b_i(x)$$

avec $b_i$ les éléments de la base de fonction et $\beta_i$ les paramètres à estimer. Ces fonctions seront ici estimées par décomposition sur une base de fonctions splines par régression sur splines pénalisées.

L'ajustement du modèle peut se faire par maximum de vraisemblance ou par moindres carrés nous donnant des résultats légèrement différents excepté quand la loi de probabilité de $Y$ est normale. Nous allons utiliser la méthode du maximum de vraisemblance qui consiste à trouver les paramètres maximisant la vraisemblance :

$$\max_{\beta} \sum_{i=1}^{n} \mathcal{L}(y_i, x_i, \beta) - \lambda \int_a^b \left[ \frac{d^2 f(x)}{dx^2} \right]^2 dx$$

On pénalise la vraisemblance afin de lisser le modèle et ainsi éviter le surapprentissage. Le paramètre $\lambda$, obtenu par validation croisée nous permettra ainsi de contrôler l'effet de la pénalisation.

### 3.2.2 R : mgcv

Les modèles additifs généralisés peuvent être construits dans la librairie mgcv. Celle-ci propose la fonction gam() :

```
gam(formula,data)
```

L'argument data doit contenir l'ensemble des données du modèle. Pour formula, on met la formule du modèle : y~x1+x2+.... La différence avec un modèle linéaire pour la formule est que l'on peut y inclure des termes de lissage s() et te() (correspondant aux fonctions splines). On pourra ainsi mettre un terme de lissage pour chaque variable de la formule : y~s(x1)+s(x2)+.... Le terme te() sera utilisé dans le cas multidimensionnel pour prendre en compte des effets d'interaction entre les variables : te(x1,x2).Les termes s() et te() peuvent tous deux être modifiés en changeant le type de splines utilisées. Par défaut, on a bs="tp" pour s() correspondant aux splines à plaques minces, et bs="cs" pour te() correspondant aux splines cubiques de régression. On peut également préciser le degré maximal de la base de décomposition splines avec l'argument k. Ainsi, on aura des formules de type :

y~s(x1,k=3)+s(x2,k=3)
y~te(x1,x2,bs="tp",k=3)

```
Family: gaussian
Link function: identity

Formula:
r ~ s(V1) + s(V2) + s(V3)

Estimated degrees of freedom:
7.1757 8.2524 1.0000  total = 17.42813

GCV score: 155.7657
```

TABLE 3.2 – Exemple de sortie gam()

La fonction :

predict(object, newdata)

sera utilisée pour la prédiction de nouvelles observations avec : object le modèle renvoyé par la fonction gam() et newdata notre nouveau jeu d'observations.

## 3.3 Random Forest

Les forêts aléatoires ou random forest est une méthode généralement efficace et facile à comprendre introduite par L. Breiman (2001) [1]. Elle se base sur la construction d'un grand nombre d'arbres de régression.

### 3.3.1 Arbre de régression

Afin de comprendre la construction des random forest, il est nécessaire de connaître les arbres de régression.
L'idée est d'effectuer un découpage binaire et itératif de l'espace $R^p$ des variables, nous conduisant à un ensemble de pavés. Les nœuds de l'arbre (point de séparation des branches) sont créés en sélectionnant la variable séparant au mieux l'espace par minimisation du critère d'hétérogénéité suivant :

$$\sum_{i \in N^i} (y_i - \bar{y}^{N^i})^2$$

Avec $N^i$, i=1,2 les deux fils du nœud $N$ et $\bar{y}^{N^i}$ la moyenne des $y_i$ appartenant au nœud $N^i$.
On arrêtera de séparer les données lorsqu'elles seront suffisamment bien classées (à un seuil de notre critère donné) évitant ainsi des arbres trop complexes et le surapprentissage. La valeur de chaque feuille sera la moyenne des valeurs contenues dans la feuille.

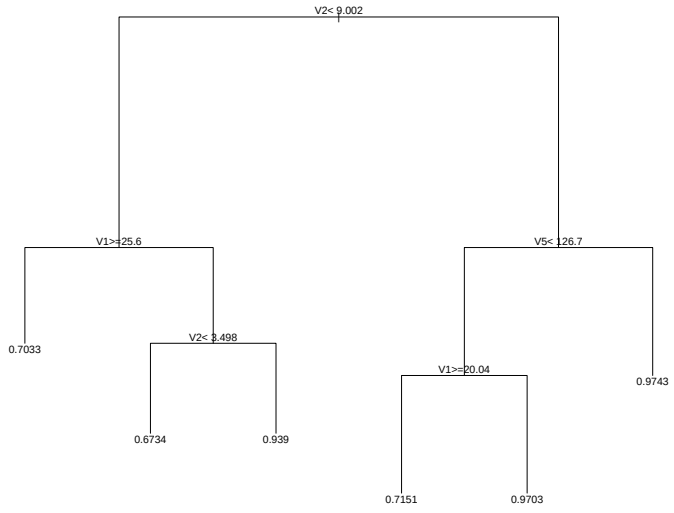


FIGURE 3.1 – Exemple d'arbre de régression

### 3.3.2 Modèle

On a vu la construction des modèles d'arbre de régression. Un modèle random forest s'effectue par la construction d'un nombre important (k choisi) d'arbres aléatoires de régression. Dans la forêt, chacun des arbres créé sera construit de la manière suivante :

- On choisit aléatoirement m observations parmi les n observations initiales.
- À chaque nœud, on choisit aléatoirement j (choisi) variables parmi les p variables. Et on choisit parmi ces j variables celles qui séparent au mieux les données.

Pour les random forest, il y a plusieurs paramètres à établir avant la construction du modèle. Le premier est le nombre d'arbres dans la forêt, le second est le nombre de variables choisies aléatoirement à chaque nœud et le troisième est le nombre d'observations maximales à chaque feuille de l'arbre (souvent différent de 1 pour éviter des arbres trop complexes et le surapprentissage).
Pour la prédiction de nouvelles valeurs, on effectuera la prédiction pour chaque arbre de la forêt puis on donnera la moyenne de ces résultats.

### 3.3.3 R : randomForest

Sous R, la construction de forêts aléatoires peut se faire avec le package randomForest. On a vu précédemment que deux paramètres étaient à choisir avant de créer la forêt. Par la suite, le nombre d'arbres de la forêt a été laissé au nombre par défaut (k=500). Le nombre optimal j de variables sélectionnées aléatoirement parmi les p variables initiales peut être choisi grâce à la fonction :

```
tuneRF(x,y)
```

x doit être la matrice des variables explicatives et y la variable à prédire. D'autres arguments peuvent être ajoutés à la fonction. On peut noter parmi eux ntreeTry permettant d'augmenter le nombre d'arbres utilisés lors de l'optimisation et ainsi améliorer la qualité, stepFactor la valeur par laquelle le nombre de variables est augmenté à chaque itération.

On peut maintenant créer la forêt aléatoire. Pour cela, il faut utiliser la fonction randomForest() en changeant le paramètre calculé précédemment. Ce qui nous donne :

```
randomForest(formula,data,ntree,mtry)
```

L'argument data contient l'ensemble des données du modèle. formula demande la variable à prédire ainsi que les variables explicatives : y~x1+x2+.... On peut modifier le nombre d'arbres contenu dans la forêt en modifiant ntree qui par défaut est à 500. mtry, le nombre de variables sélectionnées pour chaque nœud devra être fixé à la valeur déterminée par la fonction tuneRF(). On peut également noter que le nombre d'observations maximal par feuille est par défaut à 5 pour la régression.

```
Call:
 randomForest(formula = r ~ ., data = HL, mtry = best.m, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 14.57093
                    % Var explained: 94.35
```

TABLE 3.3 – Exemple de sortie randomForest()

La fonction nous donnera en sortie le pourcentage de variance expliquée sur l'ensemble d'apprentissage. Il est maintenant possible sur un nouveau jeu de données de prédire suivant le modèle de forêt aléatoire :

predict(rf,newdata)

Où rf est le modèle créé par la fonction randomForest et newdata le nouveau jeu de données à prédire.

## 3.4 Réseau de neurones

Le modèle de réseau de neurones est basé sur le principe des neurones du cerveau connectés entre eux par les axones. Il existe plusieurs méthodes permettant d'établir un modèle de réseau de neurones. Nous verrons ici uniquement le modèle du perceptron multicouches (MLP) qui est capable d'approximer n'importe quelle fonction.

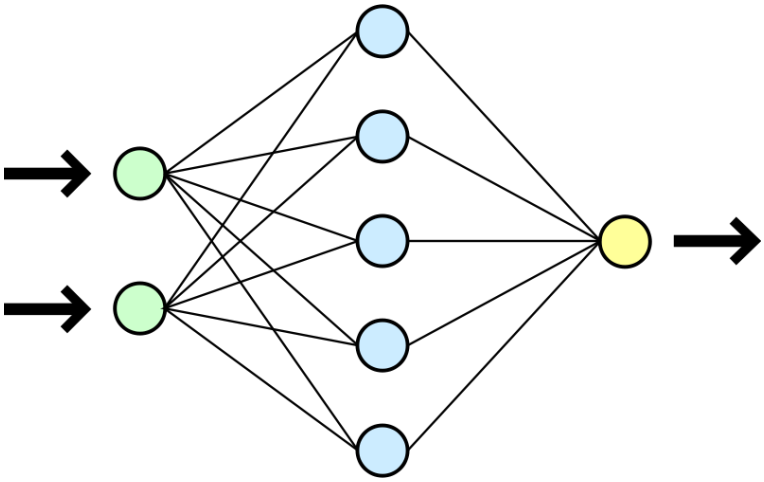

FIGURE 3.2 – Exemple de réseau de neurones

### 3.4.1 MLP

Afin de pouvoir établir le modèle, il est intéressant de comprendre le fonctionnement d'un neurone artificiel. Celui-ci fonctionne de la manière suivante : on lui donne en entrée différentes informations ($x_i$) chacune pondérée par un poids($w_i$), on fait la somme pondérée de celles-ci, puis on leur applique une fonction d'activation $\phi$ afin de produire une sortie. Un neurone s'écrira donc de la manière suivante :

$$\phi(\sum_{i=1}^{p} w_i x_i + w_0)$$

La fonction d'activation appliquée à un neurone peut avoir diverses formes. Les plus couramment utilisées dans le cadre de régression sont les fonctions :

- sigmoïde (ou logistique) : $\frac{1}{1+e^{-x}}$
- identité

Il est maintenant possible d'établir le modèle de réseau de neurones. Pour ce faire, on va créer une série de neurones (couche) à partir des données puis créer une seconde couche de neurones à partir de la première. On peut répéter ceci un certain nombre de fois afin de créer plusieurs couches. La couche finale nous renverra la ou les sorties du réseau. Dans notre cas, le réseau ne

comportera qu'une seule couche (couche cachée) nous donnant ainsi :

$$f_w(x) = g_1\left(\sum_{i=1}^{Q} w_i^{(2)} g_2(x^T w_i^{(1)} + w_i^{(0)}) + w_0^{(2)}\right)$$

Avec $Q$ le nombre de neurones sur la couche cachée, $g_1, g_2$ fonctions d'activation données et $w$ l'ensemble des poids. Dans le cadre de la régression, la fonction $g_1$ est le plus souvent l'identité alors que la fonction $g_2$ est généralement logistique.
Il nous reste ainsi à déterminer les poids $w$. Pour ce faire, on cherche $\hat{w}$ qui minimise la somme des erreurs quadratiques d'estimation pénalisées par les poids :

$$\hat{w} = argmin \sum_{i=1}^{n} \|y_i - f_w(x_i)\|^2 + C\|w\|^2$$

Le paramètre $C$ sert à contrôler la pénalisation de l'erreur quadratique par les poids $w$. Cette pénalisation sert à lisser l'estimation du modèle pour éviter les problèmes de surapprentissage. De plus, ce paramètre est choisi tout comme $Q$ avant de déterminer les poids $w$. Ces deux paramètres sont le plus souvent déterminés par validation croisée.
Un problème peut dès lors être soulevé. Cette fonction n'est pas quadratique en $w$ ce qui implique qu'il n'existe pas de méthode exacte pour résoudre ce problème d'optimisation. Cette minimisation pourra être effectuée par méthode de gradient descendant, nous donnant ainsi une approximation de la solution. Cette méthode va être répétée un certain nombre de fois en changeant le point de départ (les poids de départ) de l'algorithme afin d'éviter les minima locaux.

### 3.4.2 R : nnet

Il existe plusieurs package R permettant de construire des réseaux de neurones. Nous utiliserons et détaillerons ici le package nnet. La fonction principale de ce package est nnet() permettant de construire le modèle. Elle se présente et s'utilise de la manière suivante :

nnet(formula, data, size, decay = 0, maxit = 100, linout = FALSE, trace=TRUE)

Le principal élément de cette fonction est formula qui est de la forme y ~$x_1$ + $x_2$ + ..., data contient les données du modèle. Les arguments size et decay sont le nombre de neurones $Q$ sur la couche cachée et le paramètre de pénalisation des poids $C$. On peut encore noter trois paramètres pouvant améliorer la qualité du modèle. Le premier, maxit permet de changer le nombre d'itérations maximal effectué par la méthode pour éviter les minima locaux. Le nombre d'itérations avant convergence étant très dépendant du modèle et du point de départ, il peut être utile d'augmenter ce nombre pour avoir un bon modèle. Le second, linout permet de changer la seconde fonction d'activation à l'identité (de base logistique). Le troisième, trace sert à effectuer ou non l'affichage des étapes de la méthode. Enlever l'affichage permet un gain de temps lors de l'estimation du réseau de neurones.

Les valeurs optimales des paramètres size et decay étant très dépendantes du modèle sur lequel on cherche à établir le réseau de neurones, on peut les déterminer par le biais de la fonction tune() du package e1071. Cette fonction peut être utilisée pour différentes méthodes. Dans le cas des réseaux de neurones elle s'emploie de la manière suivante :

```
tune(nnet, formula, data, ranges = list(decay, size), tunecontrol)
```

La fonction comporte de nouveaux arguments : ranges doit contenir la liste des paramètres à tester ; decay et size doivent contenir l'ensemble des valeurs à tester pour $Q$ et $C$ ; tunecontrol sert à choisir le type de sélection pour les paramètres (validation croisée, échantillon test/apprentissage). Le choix des valeurs pour ces deux paramètres peut être délicat car la valeur optimale est très dépendante du modèle de départ et donc mettre de nombreuses valeurs à tester augmentera fortement le temps d'estimation du modèle, mais mettre peu de valeurs nous verra potentiellement éloigné de la valeur optimale. La fonction renvoie ainsi un vecteur des paramètres optimaux.

```
a 3-30-1 network with 151 weights
inputs: V1 V2 V3
output(s): r
options were - linear output units  decay=0.1
```

TABLE 3.4 – Exemple de sortie nnet()

Une fois le modèle établi par la fonction nnet(), on peut prédire le modèle sur un nouveau jeu de données $X$ par la fonction predict() de la manière suivante :

```
predict(n1,X)
```

Où n1 est le modèle de réseau de neurones créé précédemment. On a ainsi les valeurs $Y$ prédites sur le nouvel échantillon $X$.

## 3.5 Krigeage

Le krigeage est une technique de modélisation souvent utilisée dans le cadre de statistique spatiale. Le krigeage s'apparente à un modèle linéaire avec résidus non indépendants tout en minimisant la variance de l'erreur d'estimation, ceci menant à des interpolations.

### 3.5.1 Modèle

On veut exprimer un échantillon observé $Y_i$, i=1,...,n grâce à un ensemble de données $x_i = [x_1...x_p]$ avec $x_i \in \Re^n$. Le modèle du krigeage peut être vu comme la réalisation d'un modèle de régression et d'un processus aléatoire $Z$, nous donnant ainsi :

$$\hat{y}(x) = \beta_1 f_1(x) + ... + \beta_q f_q(x) + Z(x)$$

où les $q$ fonctions $f_j : \Re^n \mapsto \Re$ sont choisies (constante, linéaire, quadratique,...), les coefficients $\beta_j$ étant les paramètres de la régression. On peut également définir le vecteur $f(x) = [f_1(x)...f_q(x)]^T$. Le processus aléatoire $Z$ est d'espérance nulle (krigeage simple) et de covariance entre $Z(v)$ et $Z(w)$ :

$$Cov(Z(v), Z(w)) = \sigma^2 R(\theta, v, w)$$

où $\sigma^2$ est la variance du processus $Z$ et $R(\theta, v, w)$ la fonction de corrélation de paramètre $\theta$. Les fonctions de corrélation peuvent être définies comme :

$$R(\theta, v, w) = \prod_{j=1}^{d} R_j(\theta_j, v_j, w_j)$$

Elles peuvent avoir différentes formes, par exemple, $R_j(\theta_j, v_j, w_j) = exp(-\theta_j |w_j - v_j|)$.

Nous allons voir comment établir $\hat{y}(x)$, la prédiction du krigeage à un nouveau point x. On peut définir $\mathbf{R}$ comme la matrice des processus de corrélation sur les données d'apprentissage $x_i$ :

$$\mathbf{R}_{ij} = R(\theta, x_i, x_j), i, j = 1, ..., m$$

Pour un nouveau point x, le vecteur des corrélations entre $Z$ sur les données d'apprentissage et x est :

$$r(x) = [R(\theta, x_1, x)...R(\theta, x_m, x)]^T$$

Il est également nécessaire de définir la matrice :

$$F = [f(x_1)...f(x_m)]^T$$

Notre but étant de déterminer $\hat{y}$, considérons le prédicteur linéaire $\hat{y}(x) = c^T Y$. On va déterminer c pour résoudre notre problème. On cherche à minimiser la moyenne des erreurs au carré (MSE) :

$$\begin{aligned} MSE &= E[(\hat{y}(x) - y(x))^2] \\ &= \sigma^2(1 + c^T \mathbf{R} c - 2c^T r) \end{aligned}$$

Avec $Y = F\beta + Z$, $y(x) = F(x)^T\beta + Z$ et pour que le prédicteur soit non biaisé $F^T c = f(x)$. Le problème de minimisation du MSE nous mène au système d'équations matricielles suivant :

$$\begin{bmatrix} \mathbf{R} & F \\ F^T & 0 \end{bmatrix} \begin{bmatrix} c \\ -\frac{\lambda}{2\sigma^2} \end{bmatrix} = \begin{bmatrix} r \\ f \end{bmatrix}$$

dont la solution est :

$$c = \mathbf{R}^{-1}(r - F(-\frac{\lambda}{2\sigma^2}))$$

On peut montrer par la méthode des moindres carrés généralisés que le problème de régression $F\beta \simeq Y$ a pour solution $\beta^\star = (F^T\mathbf{R}^{-1}F)^{-1}F^T\mathbf{R}^{-1}Y$. En reprenant $\hat{y}(x) = c^T Y$ on obtient le prédicteur :

$$\hat{y}(x) = f(x)^T\beta^\star + r(x)^T\gamma^\star$$

avec $\gamma^\star = \mathbf{R}^{-1}(Y - F\beta^\star)$. Pour prédire un nouvel élément x, il nous suffira de calculer $f(x)$ et $r(x)$. Plus de détails sur les calculs peuvent être trouvés dans la documentation de la toolbox matlab DACE (Lophaven et al.) [3].

Lors de l'estimation du modèle de krigeage, on a utilisé des fonctions de corrélation. Celles-ci nécessitent la sélection d'une valeur pour le paramètre $\theta$. On estimera la valeur de ce paramètre par maximum de vraisemblance. On peut rappeler que le krigeage est une méthode d'interpolation sur l'échantillon d'apprentissage. Il est possible de modifier le modèle de krigeage en ajoutant un effet pépite au modèle. L'effet pépite est une variation naturelle du paramètre mesuré : il correspond à un petit terme d'erreur, menant à un lissage et non plus à une interpolation. Cet effet pépite peut ainsi améliorer la qualité du modèle.

### 3.5.2 R : DiceKriging

Le package DiceKriging utilisé ici pour établir des modèles de krigeage sous R est tiré de la toolbox Matlab DACE. Pour créer un modèle de krigeage, il faut utiliser la fonction km() se présentant de la manière suivante :

km(design, response)

On doit fournir à la fonction un ensemble de données d'apprentissage $s_i = [s_1...s_m]$ par le paramètre design et la variable observée $Y$ avec response. Cette fonction peut prendre d'autres arguments, dont certains peuvent permettre d'améliorer grandement le modèle. On a vu qu'il était possible d'ajouter un effet pépite au modèle de krigeage. Cet effet pépite peut être pris en compte lors de l'estimation du modèle en ajoutant le paramètre nugget.estim=TRUE. Estimer l'effet pépite permet généralement d'améliorer le modèle final. La fonction km() nous permet également un effet de dérive avec le paramètre formula. Cette dérive correspond au vecteur $f(x)$ de la partie précédente. De base, le paramètre est à formula=~1 correspondant à aucun effet de dérive. On peut demander un effet linéaire des variables par formula=$\sim x_1 + ... + x_n$. Le dernier argument qu'il peut être intéressant de modifier est covtype correspondant à la structure de covariance utilisée lors de l'estimation du krigeage. Nous utiliserons la structure de covariance par

défaut : matern5_2.
On pourra observer en sortie de la fonction les valeurs du vecteur $\theta$, la variance du modèle sur l'échantillon d'apprentissage et la valeur de l'effet pépite s'il est estimé.

```
Call:
km(formula = ~1, design = HL[, -Y], response = HL[, Y], nugget.estim = TRUE)

Trend  coeff.:
               Estimate
 (Intercept)   -62.6777

Covar. type  : matern5_2
Covar. coeff.:
               Estimate
   theta(V1)     4.8460
   theta(V2)    14.4747
   theta(V3)     9.0085

Variance estimate: 61941.02

Nugget effect estimate: 0.0006194102
```

TABLE 3.5 – Exemple de sortie km() avec effet pépite

Le modèle établi, on va chercher à prédire de nouvelles données. Pour cela, on utilise la fonction predict qui se présente de la manière suivante :

predict(object,newdata,type,se.compute=TRUE)

object correspond au modèle de krigeage construit grâce à la fonction km et newdata aux données à prédire. Nous mettrons type="SK" afin de préciser que l'on effectue un krigeage simple. Si on laisse se.compute=TRUE, la fonction nous renverra plusieurs éléments dont la moyenne du krigeage et les intervalles de confiance à 95%. En passant le paramètre à FALSE, seulement la moyenne du krigeage sera renvoyée.

## 3.6 Support Vecteur Machine

Les supports vecteurs machines (SVM) sont à l'origine utilisés afin de résoudre des problèmes de classification. Ils ont été par la suite adaptés aux problèmes de régression par V. Vapnik (1995) [9]. Les SVM sont basés sur l'idée d'effectuer une régression linéaire dans un espace de dimension supérieure à celle de l'espace des données. En pratique, il ne sera pas nécessaire d'effectuer des calculs en dimension supérieure.

### 3.6.1 Modèle

On cherche à exprimer la variable $y \in \Re^n$ grâce à un ensemble de données $x \in M_{n,q}(\Re)$. Pour les SVM, la fonction $\hat{f}$ d'estimation de $y$ sera de la forme :

$$\hat{f}(x) = \langle w, \phi(x) \rangle + b$$

où $\phi$ est une transformation non linéaire passant $x$ à un espace de dimension supérieure, $w$ et $b$ sont des paramètres à apprendre.

Par la suite, la transformation $\phi$ n'interviendra que dans des produits scalaires. Trouver la fonction et calculer ce produit scalaire est difficile, c'est pourquoi on introduit les fonctions de Kernel (ou à noyau) qui nous permettent d'effectuer des calculs simples dans $\Re^q$. En effet, il existe une fonction $K : \Re^q \times \Re^q \to \Re$ tel que :

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

Il faut ensuite déterminer quelle fonction de Kernel $K$ peut être associée à une fonction $\phi$ donnée. En pratique, on teste les fonctions de Kernel sans se préoccuper de $\phi$. Il existe de nombreuses fonctions de Kernel dont la plus fréquemment utilisée, la fonction Kernel gaussienne $K(x, x') = e^{-\gamma \|x - x'\|^2}$, $\gamma > 0$. Cette fonction nous apporte un paramètre $\gamma$ qu'il faudra choisir correctement afin d'avoir un bon modèle.

Il existe plusieurs méthodes pour déterminer $w$ et $b$. Nous utiliserons la méthode consistant à prendre comme critère de qualité pour le modèle la fonction $\epsilon - insensitive$ :

$$L_\epsilon(x, Y) = \sum_{i=1}^{n} max(|f(x_i) - y_i| - \epsilon, 0)$$

Cette fonction permet d'ignorer les erreurs suffisamment faibles ($< \epsilon$) créant ainsi une zone d'acceptation autour de la vraie valeur. Prendre uniquement cette fonction comme critère pour trouver le modèle régression ne nous donnerait presque aucune erreur sur les données d'apprentissage mais il y aurait du surapprentissage, le modèle pourrait être trop complexe et ne pas donner de bonne estimation pour de nouvelles observations. On introduit donc un terme de régulation $\|w\|^2$. On a donc $w$ et $b$ tels que :

$$\arg\min_{w,b} \sum_{i=1}^{n} max(|f(x_i) - y_i| - \epsilon, 0) + \frac{1}{C}\|w\|^2$$

$C$ correspondant au paramètre de régulation. Plus $C$ est petit, plus on autorise des erreurs grandes et une complexité réduite, plus sa valeur est élevée, moins on autorisera d'erreur mais la complexité sera plus grande pouvant mener à du surapprentissage. Il faudra donc choisir soigneusement $C$ pour avoir un bon modèle.

On va d'abord chercher la valeur de $w$. On passe par le Lagrangien puis par le problème dual suivant :

$$\max_{\alpha_i,\alpha_i^\star} \; -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i-\alpha_i^\star)(\alpha_j-\alpha_j^\star)K(x_i,x_j) - \epsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^\star) + \sum_{i=1}^{n} y_i(\alpha_i-\alpha_i^\star)$$

avec $\sum_{i=1}^{n}(\alpha_i-\alpha_i^\star)=0$ et $\alpha_i,\alpha_i^\star \in [0,C]$. On a en dérivant le Lagrangien par rapport à $w$ :

$$w = \sum_{i=1}^{n}(\alpha_i-\alpha_i^\star)\phi(x_i)$$

où $\alpha_i,\alpha_i^\star$, $i=1,...,n$ sont des solutions du problème précédent.

Il nous reste maintenant à trouver b. Pour cela, on passe par les conditions de Karush-Kuhn-Tucher et on trouve :

$$b = y_i - \epsilon - \langle w,\phi(x_i)\rangle, \text{ pour } \alpha_i^\star \in ]0,C[$$

Ainsi, on obtient le modèle SVM suivant :

$$f(x) = \sum_{i=1}^{n}(\alpha_i-\alpha_i^\star)K(x_i,x) + b$$

Au final, pour établir le modèle, trois paramètres seront à choisir. Le premier est $\epsilon$ de la fonction de coût, le second, $\gamma$, est le paramètre de la fonction de kernel $K$ et le dernier $C$ est le paramètre de régulation. Ces trois paramètres sont généralement sélectionnés par validation croisée.

### 3.6.2 R : e1071

Il existe plusieurs packages permettant de construire des modèles SVM. Nous utiliserons le package e1071 qui propose de nombreux outils parmi lesquels la fonction svm(). La fonction s'utilise de la manière suivante :

svm(formula, data, gamma, cost, epsilon)

Comme pour les autres packages, l'argument data doit contenir le jeu de données et formula la variable à prédire et les variables de prédiction (y~.). Ensuite, Nous avons gamma pour $\gamma$ le paramètre de la fonction de Kernel, cost pour $C$ la régulation de la pénalisation du modèle et epsilon pour $\epsilon$ le paramètre de la fonction $\epsilon$-*insensitive*. Ces trois paramètres ont tous une valeur

par défaut mais celle-ci n'est pas optimisée et cela peut conduire à un modèle ayant une mauvaise capacité de prédiction. On va donc les déterminer par validation croisée en utilisant la fonction tune() présente dans le package et permettant de paramétrer plusieurs autres méthodes. Dans le cas des SVM, on aura :

tune(svm, formula, data, ranges = list(gamma, cost, epsilon), tunecontrol)

les arguments formula et data sont les mêmes que dans svm(). L'argument ranges doit contenir l'ensemble des valeurs à tester pour les paramètres gamma, cost et epsilon des SVM. Le dernier argument à régler est tunecontrol. Celui-ci permet de régler la méthode de sélection des paramètres. Nous voulons effectuer une validation croisée donc on mettra tunecontrol = tune.control(sampling = "cross").

```
obj = tune(svm, r~., data = HL,ranges = list(gamma = c(0.01,0.1,1), cost =
      c(50,100,150,200),epsilon=c(0.01,0.1,1)),tunecontrol = tune.control(
      sampling = "cross"))


Call:
svm(formula = r ~ ., data = HL, gamma = as.numeric(obj$best.parameters[1]),
    cost = as.numeric(obj$best.parameters[2]), epsilon = as.numeric(
    obj$best.parameters[1]))

Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  200
      gamma:  1
    epsilon:  1

Number of Support Vectors:  55
```

TABLE 3.6 – Exemple de recherche de paramètres et sortie de svm()

Pour la prédiction de nouvelles observations, on utilisera la fonction :

predict(object, newdata)

object sera le modèle renvoyé par la fonction svm() et newdata notre nouveau jeu d'observations.

# 4 Application et analyse des résultats

Le but de cette partie est de comparer (en termes de temps, de qualité d'estimation et de prédiction) les méthodes étudiées sur un modèle de la plate-forme afin de voir si ces méthodes peuvent être appliquées de manière générique et implémentées sur la plate-forme.

## 4.1 Modèle Azodyn

Les tests réalisés par la suite ont été effectués sur des modèles issus du modèle Azodyn-colza de la plate-forme RECORD. Le modèle Azodyn-colza est un modèle dynamique effectuant des simulations sur des parcelles de colza pour une durée de 381 jours (du début de la simulation jusqu'à la récolte). Les méta-modèles abordés sont testés sur deux ensembles de variables différents issus du modèle Azodyn-colza. Le premier est composé de 4 variables et le second de 15 variables. Ces variables sont de plusieurs types :

- 4 variables de décision nous donnant une valeur d'apport en azote (variables du premier modèle).
- 3 variables sur l'azote contenu dans le sol.
- 3 variables sur les conditions d'eau du sol.
- 2 variables sur l'enracinement de la plante.
- 3 variables sur la plante.

Ces variables seront appliquées au modèle Azodyn-colza et nous récupèrerons la variable MSTg au jour 379 correspondant à la matière sèche générée au jour de la récolte. On a donc deux modèles composés de deux nombres de variables différents, nous permettant ainsi d'avoir une estimation de l'effet de l'augmentation du nombre de variables sur les méta-modèles abordés.

On a ensuite appliqué les méta-modèles à nos deux modèles. Pour ce faire, on a pris deux types de plans d'expérience (LHS et Sobol) pour nos simulations et pour chacun de ces plans, on a créé trois plans de tailles différentes (100, 300 et 1000 observations). Finalement, on a répété 5 fois nos méta-modèles sur l'ensemble des plans créés précédemment nous donnant un large panel de données permettant de comparer efficacement les méta-modèles et les plans d'expérience.

## 4.2 Méthodes de comparaison

### 4.2.1 Les méta-modèles

On a réalisé plusieurs expériences (réplicas). Chacun de ces réplicas a été effectué pour un plan d'expérience LHS ou Sobol donné. On distinguera ces réplicas par un numéro (ex : LHS-1, LHS-2 ...). Pour deux réplicas différents, les plans seront différents. Pour chaque réplica, on effectue 5 répétitions de nos méta-modèles sur le plan d'expérience afin d'estimer la stabilité de la méthode sur un plan donné (donne-t-elle toujours les mêmes résultats ? ). La stabilité provient du caractère aléatoire ou non intervenant dans les méthodes. Par exemple, les svm n'ont pas de facteurs aléatoires donc pour un même jeu de paramètres, on obtiendra les mêmes résultats.

Parmi les méta-modèles, on peut rappeler que pour l'apprentissage des réseaux de neurones et des supports vecteurs machines il est nécessaire de sélectionner des paramètres optimaux. Pour ce faire, une validation croisée est effectuée pour l'ensemble des paramètres. Pour cause de temps trop important, on effectuera une sélection des paramètres des réseaux de neurones par validation échantillon apprentissage/test.

Il est nécessaire de préciser pour la suite les notations des méta-modèles. Le modèle linéaire sera noté l1, les modèles polynomiaux de degré 2 et 3, l2 et l3, les modèles GAM g1, de krigeage sans effet pépite k2, de krigeage avec effet pépite k1, de forêt aléatoire rf, de réseau de neurones nnet, de support vecteur machines svm. De plus, pour le modèle avec 15 variables, le g1 pour 100 observations n'a pas pu être effectué car limités par la taille de données insuffisante. De plus, nnet a été réalisé pour 3 valeurs de decay et 3 valeurs de size et svm pour 3 valeurs de gamma, 4 de cost et 3 de epsilon.

### 4.2.2 Critères d'évaluation

Nos méta-modèles ainsi créés vont être comparés. Cette comparaison est faite selon trois critères : le temps, le $R^2$ et le msep. Le temps nous donnera deux informations : est-il coûteux en temps d'estimer les modèles (quels sont les modèles les plus et les moins rapides à estimer) et le temps de prédiction de la méthode peut-il être une contrainte de la méthode ? Le $R^2$ :

$$R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

quant à lui exprime la part de variance expliquée par le méta-modèle. Il nous donnera donc une évaluation de la qualité du modèle pour de futures prédictions basées sur les données d'apprentissages. Le msep (mean square error prediction) :

$$MSEP = \frac{\sum_i (\hat{y}_i - y_i)^2}{n}$$

calcule la moyenne des erreurs de prédiction au carré pour des données prédites sur un nouveau plan d'expérience servant d'échantillon test (ici un plan LHS de 300 observations). Le msep nous servira ainsi à valider ou non la qualité du méta-modèle.

## 4.3 Méta-modèle : Temps

Une première manière de comparer les méta-modèles entre eux est le temps de calcul. Celui-ci est présent à deux niveaux, le temps d'estimation et le temps de prédiction du méta-modèle.

### 4.3.1 Temps d'estimation

Le temps d'estimation du méta-modèle est le temps nécessaire à la création du modèle. Il est intéressant de connaître la stabilité du temps d'estimation des méthodes suivant plusieurs variations comme le nombre d'observations pour créer le méta-modèle et le nombre de variables du modèle.

| | l1 | l2 | l3 | rf | nnet | svm | g1 | k1 | k2 |
|---|---|---|---|---|---|---|---|---|---|
| n=100 | **0.002** | 0.02 | 0.024 | 0.389 | **171.9** | 9.136 | 0.120 | 0.767 | 0.366 |
| n=300 | **0.0048** | 0.032 | 0.042 | 2.644 | **192.3** | 88.28 | 0.114 | 6.626 | 3.259 |
| n=1000 | **0.004** | 0.036 | 0.052 | 11.493 | **704.1** | 626.4 | 0.588 | 60.58 | 50.32 |

TABLE 4.1 – Temps moyen de prédiction/observations (en seconde) sur un plan LHS, modèle à 4 variables

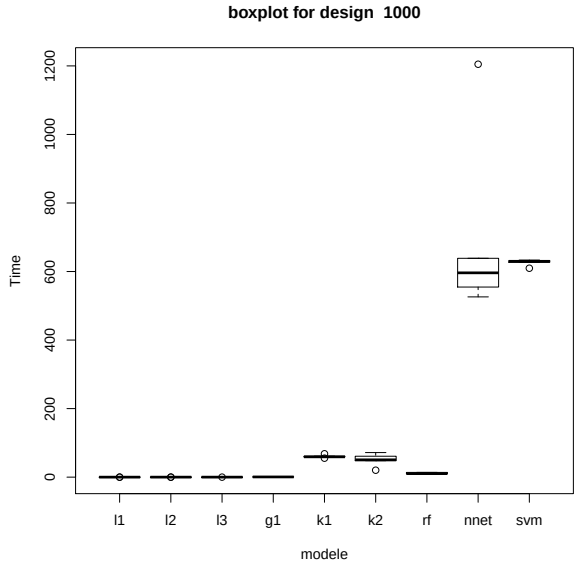


FIGURE 4.1 – Temps d'estimation (en seconde) pour 1000 observations sur un plan LHS, 4 variables

On peut remarquer dans la table 4.1 que les modèles linéaires, polynomiaux et de type GAM sont tous très rapides à estimer (<1s). De plus, on voit que l'évolution du temps en fonction du

nombre d'observations n'est pas linéaire avec les méthodes de krigeage, de réseau de neurones et de SVM pour qui le temps d'estimation augmente très rapidement avec la taille des données, nous donnant des méthodes difficiles à utiliser dans le cas de nombreuses observations.

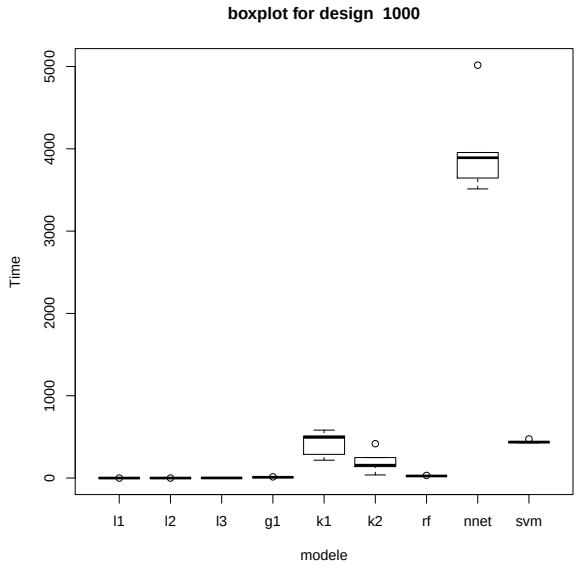


FIGURE 4.2 – Temps d'estimation (en seconde) pour 1000 observations sur un plan LHS, 15 variables

Une seconde analyse sur les temps d'estimation des méta-modèles nous permet grâce aux figures 4.1 et 4.2 de voir l'importance de l'augmentation de la dimension. En effet, celle-ci n'a quasiment aucune influence sur les méta-modèles l1, g1, rf et svm. En revanche, on voit une très forte augmentation du temps pour les méthodes de krigeage montrant le principal problème de ces méthodes : le temps d'estimation augmente très fortement avec l'augmentation de la taille des données (nombre d'observations et nombre de dimensions) du fait de la manière de construction de la méthode. On peut relever un nouveau problème : le temps d'estimation des réseaux de neurones augmente fortement avec la dimension. De plus, on remarque des temps très variables pour cette méthode.

On pourra dire que pour peu de variables, excepté nnet et svm, les méthodes sont rapides. L'augmentation de dimension a un effet majeur sur le krigeage et les réseaux de neurones. On a vu que les réseaux de neurones sont ici la méthode la plus lente à estimer d'autant plus que l'estimation a été effectuée avec un test échantillon apprentissage/test au lieu d'une validation croisée, car le temps d'estimation était trop élevé dans le cas de validation croisée.

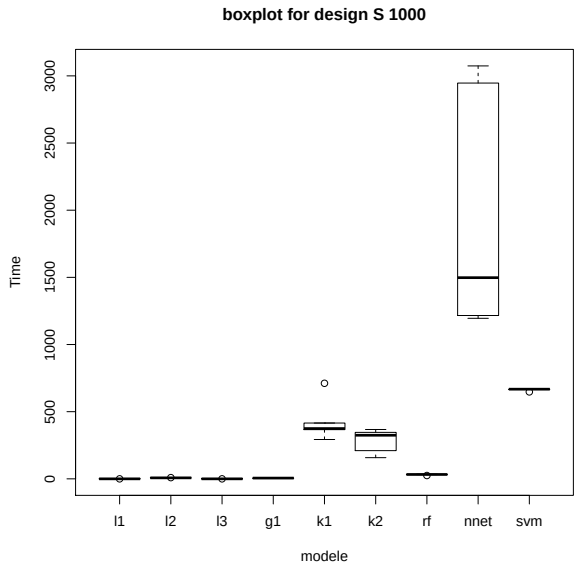


FIGURE 4.3 – Temps d'estimation (en seconde) pour 1000 observations sur un plan de Sobol, 4 variables

On voit également (figure 4.3) que le choix d'un plan d'expérience différent semble ne pas avoir une grande influence sur le temps d'estimation des méta-modèles (les ordres de grandeurs restent les même).

### 4.3.2 Temps de prédiction

Le temps de prédiction est le temps nécessaire pour prédire une sortie sur un nouveau jeu de données.

On remarque (figure 4.4) que le temps de prédiction des méta-modèles abordés n'est pas un problème car il est très court (<1s). C'est également le cas pour le modèle à 15 variables. Il pourrait être intéressant de voir le temps de prédiction des méta-modèles pour de très grosses tailles de données. Certaines méthodes comme le krigeage pourraient prendre beaucoup de temps à prédire.

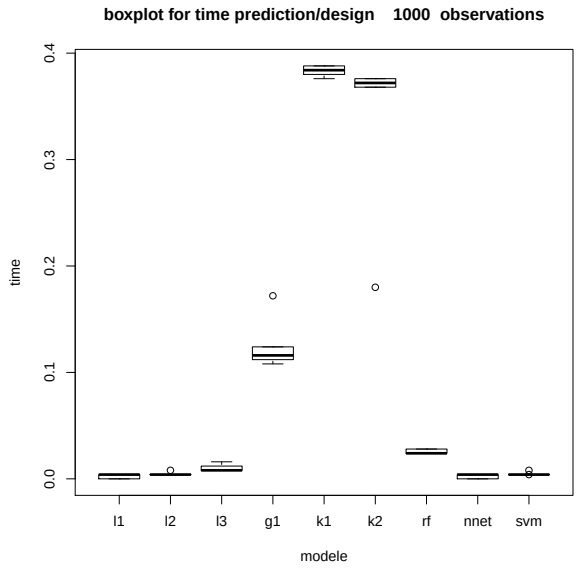


FIGURE 4.4 – Temps de prédiction (en seconde) sur un jeu de 300 observations, 4 variables

## 4.4 Qualité des modèles

Nous utilisons ici deux critères pour estimer la qualité des méta-modèles : la qualité d'estimation avec le $R^2$-ajusté et la qualité de prédiction avec le msep.

### 4.4.1 Taille d'échantillon

Nous allons ici comparer l'influence du nombre d'observations dans l'échantillon d'apprentissage sur la qualité des méta-modèles en terme de msep. Pour cela, nous avons comparé les méthodes suivant trois nombres d'observations (100, 300, 1000).

La figure 4.5, nous permet de voir une amélioration de msep pour tous les méta-modèles en augmentant le nombre d'observations du plan d'expérience, ceci est normal car on augmente ainsi la précision de la méthode. Cependant, on peut dégager deux types d'amélioration : la première, une quasi-stagnation, voire une légère amélioration entre 100 et 300 observations puis une stagnation entre 300 et 1000 observations pour l1, l2, l3, g1 et la seconde avec une très nette amélioration entre 100-300 et 300-1000 observations pour les modèles k1, k2, rf, nnet, svm. Il peut être difficile de voir ces améliorations de msep sur les figures précédentes du fait que les modèles n'ont pas les mêmes ordres de grandeur de msep.

Par la suite, afin d'obtenir les meilleurs résultats pour l'ensemble des méthodes, nous les comparerons uniquement lorsqu'elles seront construites à partir d'un plan d'expérience composé de 1000 observations.

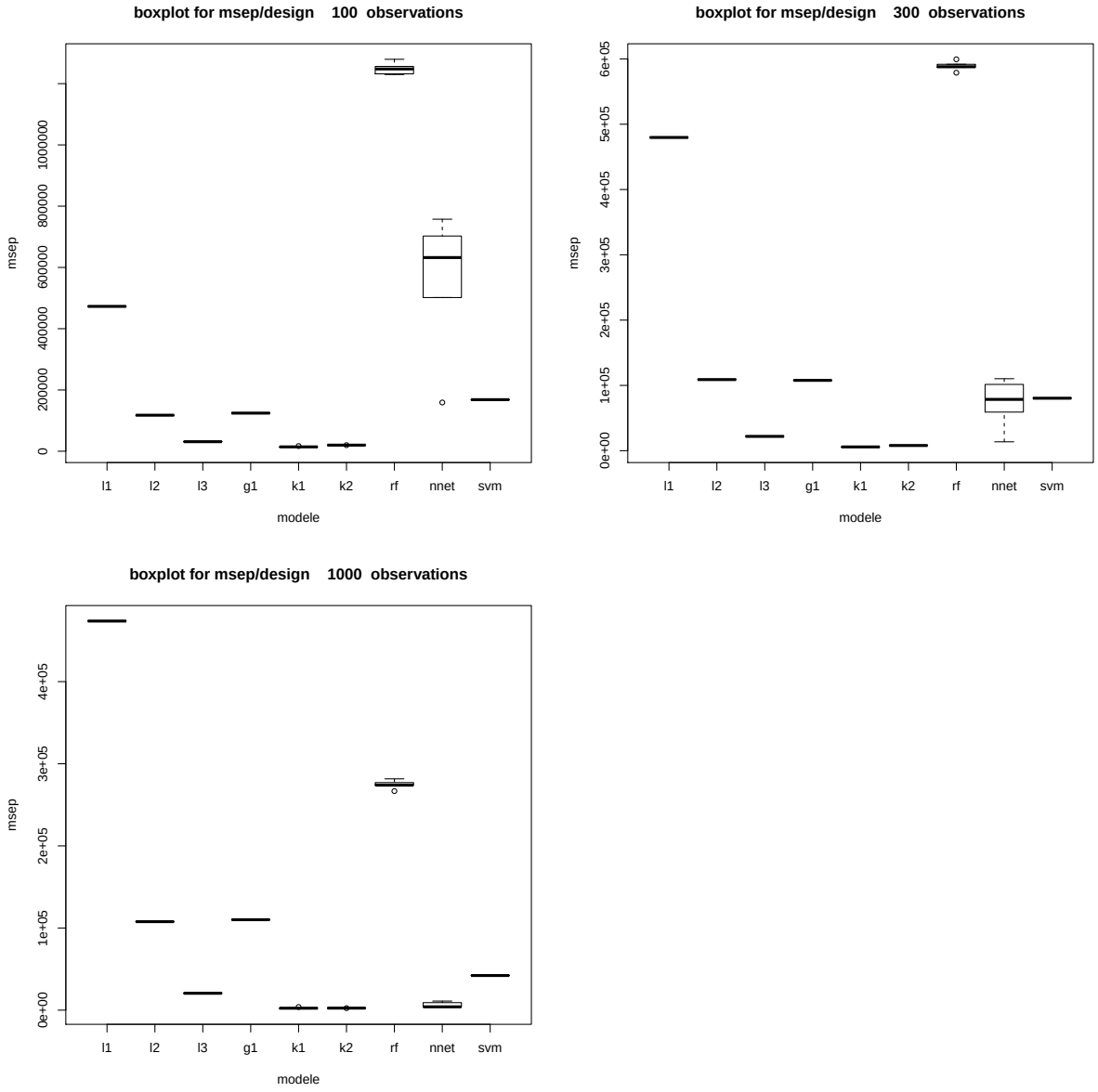


FIGURE 4.5 – prédiction (msep) pour des plans d'expérience LHS de 100, 300, 1000 observations, 4 variables

### 4.4.2 Estimation

On cherche ici à comparer les méta-modèles au niveau de l'estimation et ainsi avoir une première vision de leur qualité au moment de l'estimation. Pour ce faire, on va utiliser le $R^2$. On peut voir sur les figures 4.6 et 4.7 les $R^2$ associés aux différents méta-modèles créés.

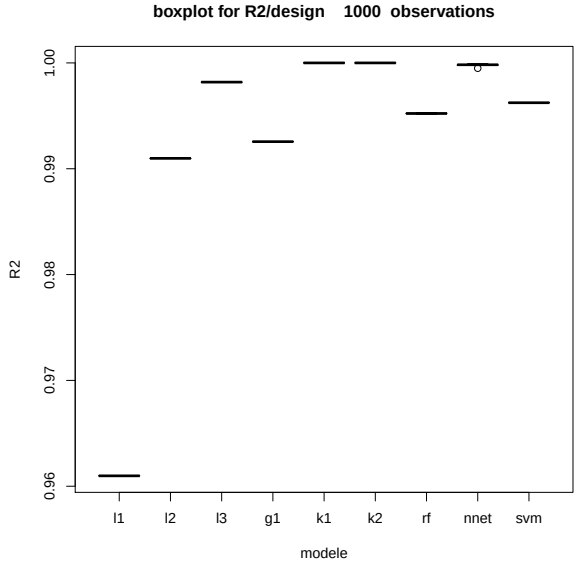


FIGURE 4.6 – $R^2$ pour un plan d'expérience LHS de 1000 observations, 4 variables

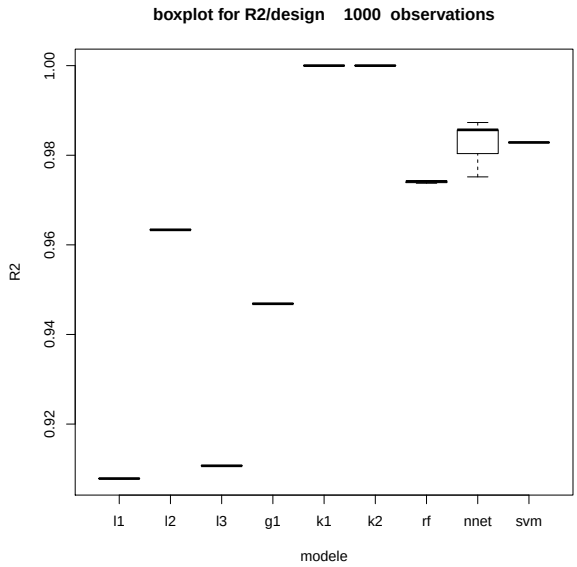


FIGURE 4.7 – $R^2$ pour un plan d'expérience LHS-1 de 1000 observations, 15 variables

Ces deux figures nous permettent de faire plusieurs remarques. La méthode du krigeage donnera toujours un $R^2$ égal à un du fait qu'elle correspond à une interpolation.

On peut également remarquer que l'ordre de qualité des modèles suivant le $R^2$ est similaire à celui suivant le msep (4.5, 4.8, excepté pour le krigeage : $R^2$ toujours égal à 1), nous confirmant la possibilité d'avoir un premier a priori sur la qualité des méthodes.

### 4.4.3 Prédiction

On va ici effectuer une comparaison des méta-modèles par le biais du msep (mean square error prediction) afin d'avoir un aperçu de l'efficacité des méthodes pour la prédiction. Pour ce faire, on prend un échantillon de données permettant de tester les méta-modèles (ici créés par des plan LHS de 300 observations). On observera plus particulièrement les figures 4.5 et 4.8.

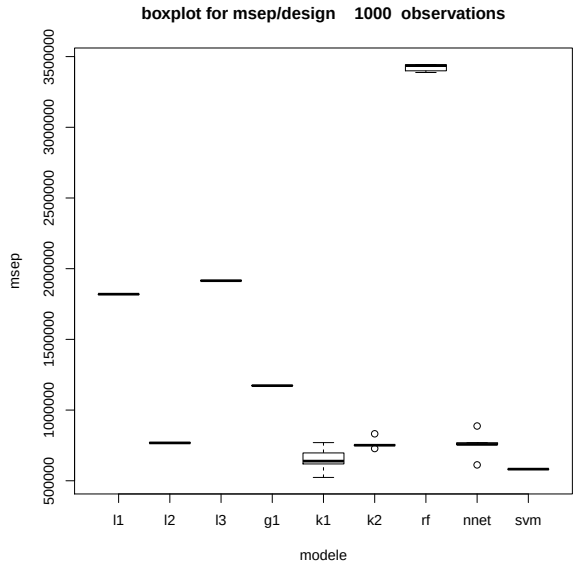


FIGURE 4.8 – prédiction (msep) pour un plan d'expérience LHS-1 de 1000 observations, 15 variables

On voit que, dans notre cas, les méthodes du krigeage et de réseau de neurones sont les plus efficaces. Bien qu'elles soient efficaces, ces méthodes ont certains défauts. Les méthodes de krigeage ont un temps d'estimation qui augmente fortement avec la dimension, donc tant que nous restons dans notre cas (faible taille de données), ces méthodes sont intéressantes à employer (très efficaces, temps d'estimation raisonnable). On peut constater un autre souci, figure 4.9, il est possible dans de rares cas que la méthode du krigeage fasse une très mauvaise estimation du modèle (probablement due à la forme des données de départ). La méthode des réseaux de neurones quant à elle pose de plus nombreux problèmes. On peut reprendre le temps d'estimation qui est très important dans notre situation qui mène à choisir une validation échantillon test/apprentissage pour la sélection des paramètres. De plus, bien qu'elle permette d'obtenir de bon résultats en termes de prédiction dans la majorité des cas, il arrive régulièrement que le

modèle estimé soit très mauvais (figure 4.9). Cette erreur se situe lors de la création du modèle : il arrive que l'optimisation du paramètre $w$ de la méthode se termine alors que les poids choisis mènent à un modèle de mauvaise qualité.

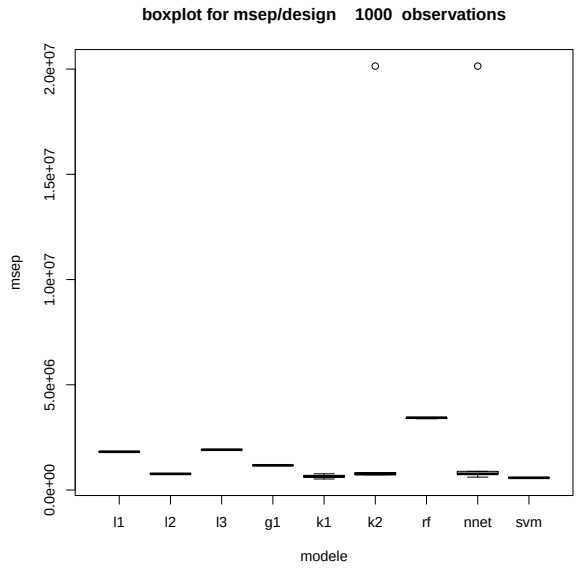


FIGURE 4.9 – prédiction (msep) pour un plan d'expérience LHS-2 de 1000 observations, 15 variables

La méthode des SVM nous donne de bons résultats pour les deux nombres de variables. Le seul souci apparaissant ici est, comme pour les réseaux de neurones, le temps d'estimation dû à la recherche des paramètres. Cette méthode peut être intéressante si l'on a un a priori de la valeur des paramètres (moins de paramètres en entrée) à utiliser, réduisant ainsi considérablement le temps d'estimation de la méthode sans perturber la qualité du modèle.

En augmentant le degré des méthodes linéaires, il est possible d'obtenir des modèles de bonne qualité (surtout pour le degré 3) mais la fonction polym ne nous permet pas d'effectuer des modèles polynomiaux alors que le modèle originel contient 15 variables. Ces modèles peuvent être très efficaces suivant les données étudiées. Le modèle GAM nous donne des résultats similaires au modèle polynomial d'ordre 2 pour 4 variables. Ce modèle est rapide à estimer mais sa construction par la fonction s() le rend compliqué à généraliser pour des modèles ayant n'importe quel nombre de variables. La méthode des forêts aléatoires nous donne ici des résultats peu concluants. Elle donne même pour 15 variables des résultats moins intéressants que ceux ontenus par le modèle linéaire. On peut en revanche préciser que cette méthode est très rapide à estimer et qu'elle est peu sensible aux paramètres et donc facile à utiliser.

La dernière chose à préciser est que les résultats obtenus nous ont fait déduire certains aspects des méta-modèles mais il est difficile de les comparer entre eux car leur efficacité peut être grandement altérée par le modèle sur lequel il sont construits. Ainsi, par exemple, les forêts aléatoires qui généralement, fournissent de très bons résultats, ont, pour notre modèle, un pouvoir prédictif très faible.

## 4.5 Plans d'expérience

Nous avons effectué des simulations sur deux types de plans d'expériences afin de voir s'il est possible d'améliorer le méta-modèle par le biais du plan. On compare donc ici les résultats donnés pour des plans LHS et des plans créés grâce à des suites de Sobol qui remplissent mieux l'espace des variables.

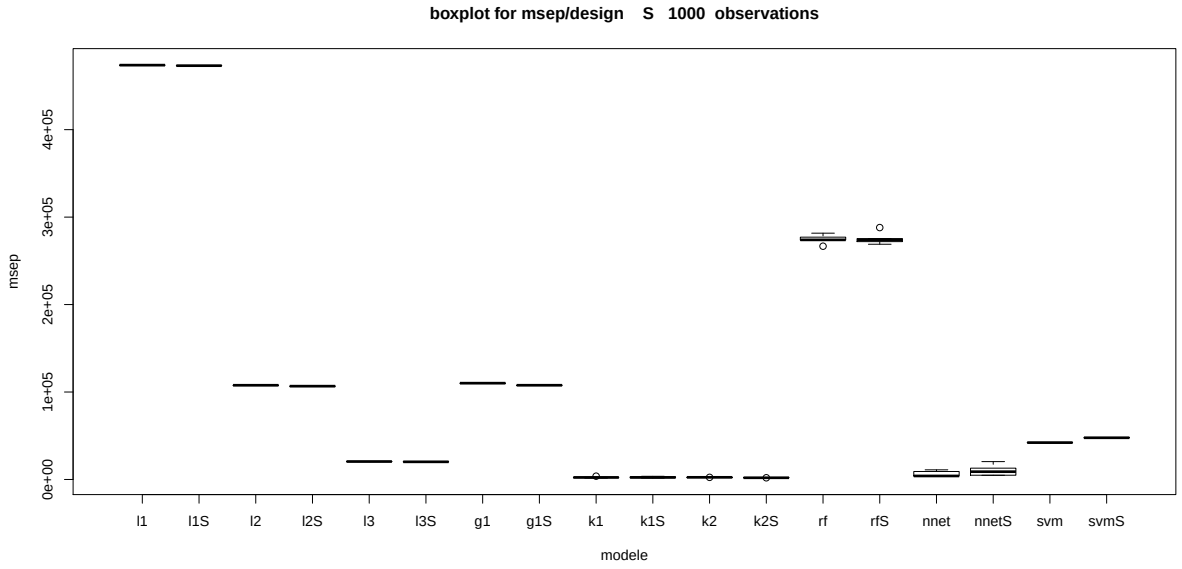


FIGURE 4.10 – msep pour un plan d'expérience LHS et Sobol de 1000 observations, 4 variables

Les figures 4.10 et 4.11 ne nous permettent pas de conclure précisément si les plans LHS ou les plans issus de suites de Sobol permettent d'obtenir de meilleurs résultats. En revanche, en observant l'ensemble des résultats obtenus (l'ensemble des réplicas effectués), il semblerait que les résultats obtenus par des plans issus de suites de Sobol sont plus stables que ceux obtenus par des plans LHS, c'est-à-dire qu'ils ont une moins grande variabilité. On peut illustrer cette conclusion avec le cas des forêts aléatoires (4.12) où l'on note que le msep pour un plan d'expérience LHS est plus affecté lors des fifférents réplicas que celui issu du plan d'expérience de Sobol. Ceci vient probablement de la construction des deux types de plans. En effet, les plans LHS ne sont pas optimisés et donc peuvent dans certain cas remplir très "bien" l'espace des variables et dans

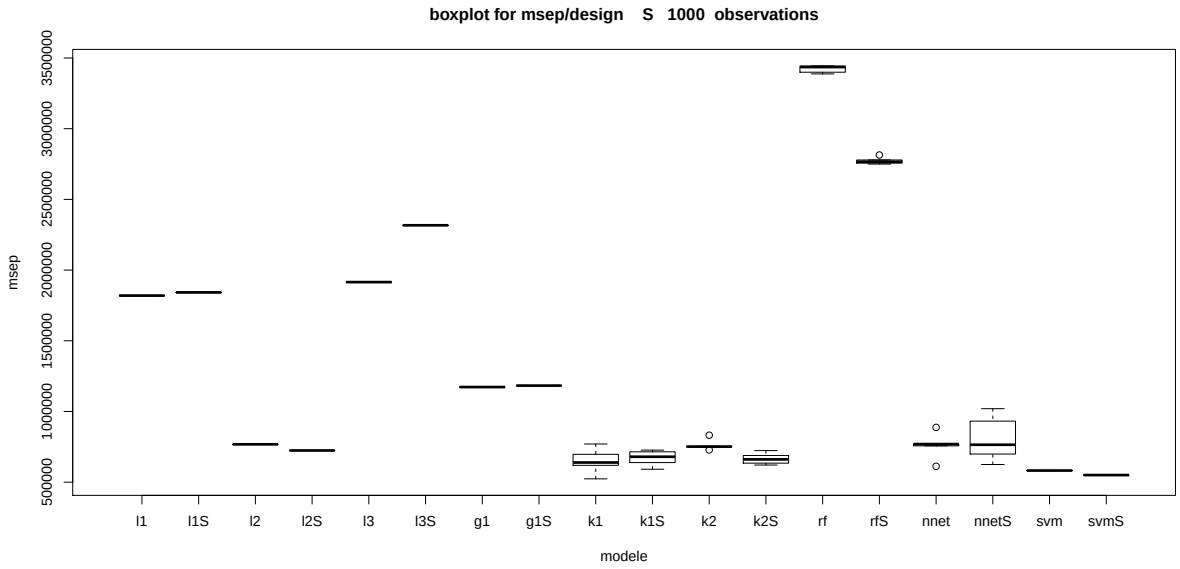


FIGURE 4.11 – msep pour un plan d'expérience LHS-1 et Sobol de 1000 observations, 15 variables

d'autres très "mal" le remplir. Ce problème n'intervient pas dans le cas des plans issus de suites de Sobol du fait de leur caractéristique (suite à faible discrépance) menant à minimiser les "trous". Ainsi il est possible d'obtenir de très bon plans par LHS (meilleurs que par suite de Sobol) ainsi que des mauvais plans. On pourra préférer les plans créés par suites de Sobol car ils entrainent une meilleure stabilité des résultats entre les réplicas.

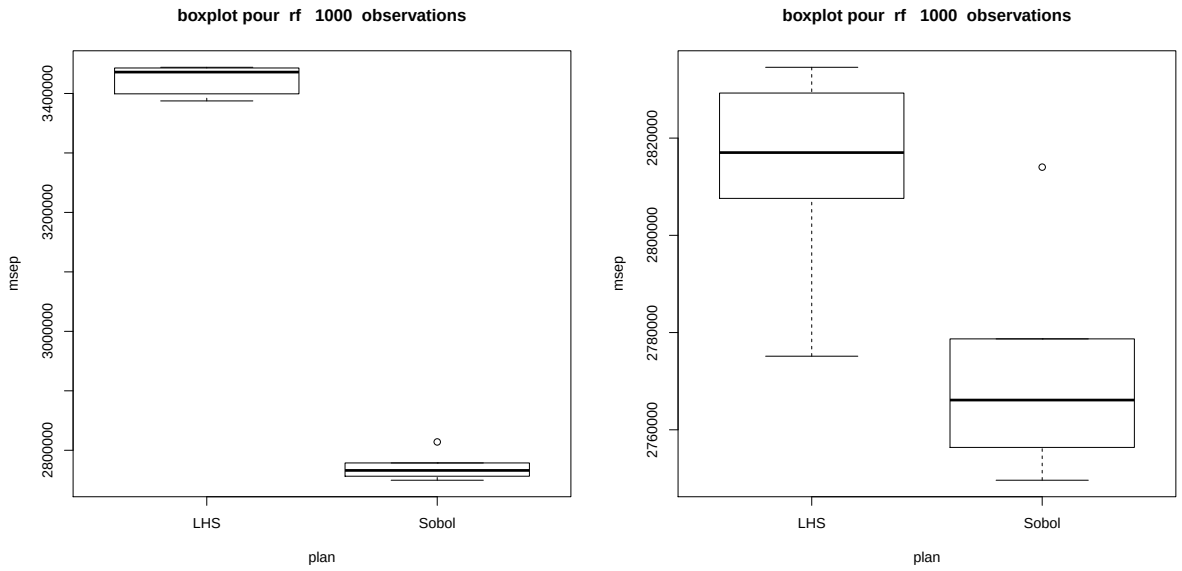


FIGURE 4.12 – msep pour un plan d'expérience LHS-1 et Sobol-1 de 1000 observations et pour un plan LHS-2 et Sobol-2, 15 variables

# 5 Conclusion

## 5.1 Bilan

L'étude réalisée sur les méta-modèles a permis d'effectuer une synthèse de plusieurs méthodes de méta-modélisation permettant ainsi d'en avoir une approche simple. Ces méthodes ont ensuite été comparées sur un modèle de la plate-forme RECORD suivant plusieurs critères (temps d'estimation, temps de prédiction,$R^2$, msep). Ces comparaisons ont mis en avant des aspects intéressants des méthodes comme l'augmentation très importante du temps d'estimation de certaines méthodes avec l'augmentation de la dimension (krigeage, réseau de neurones), mais également un premier a priori de la qualité des méthodes en termes de msep et $R^2$ en fonction du nombre d'observations. Une comparaison directe des méthodes entre elles est à prendre avec précaution parce que la plupart des méthodes ont une grande influence selon le modèle de départ. Le meilleur exemple est le modèle linéaire qui apportera des résultats excellents et meilleurs que les autres méthodes si le modèle d'origine a un comportement linéaire mais qui sera très mauvais dans le cas contraire. On peut tout de même rappeler que sur le modèle étudié ici, la méthode du krigeage donne d'excellents résultats. Il en est de même pour les réseaux de neurones bien qu'ils aient de grandes instabilités. Au contraire, le modèle de forêt aléatoire et le modèle linéaire donnent de mauvais résultats.

Le stage avait également pour objectif de voir si les méta-modèles pouvaient être intégrés et utilisés de manière générique sur la plate-forme RECORD. Une fonction prenant un plan d'expérience, un nom de méthode et d'éventuels arguments (suivant la méthode) a ainsi été créée. Ainsi, en entrant un plan d'expérience (avec la variable à estimée associée), le nom de la méthode voulue et les évantuels arguments, la fonction créer le modèle et le sauvegarde afin qu'il puisse être réexploité. La fonction permet également de récupérer le temps d'estimation et le $R^2$ du modèle.

D'un point de vue personnel, ce stage m'a permis de mettre en application différentes compétences statistiques acquises pendant mon master comme les méthodes de forêt aléatoire, de krigeage, de gam. Il m'a également permis d'améliorer mes connaissances statistiques avec l'apprentissage de nouvelles méthodes statistiques. En effet, les réseaux de neurones sont pour moi une méthode nouvelle et les supports vecteurs machines ont été abordés dans un cadre statistique pour la première fois (cette méthode ayant été étudiée en cours d'algorithmique pendant le master). De plus, ce stage m'a fait développer un intérêt certain pour le domaine de la recherche.

## 5.2 Perspectives

Le stage visait à la réalisation d'une synthèse de différentes méthodes de méta-modélisation et à leur application de manière générique sur des modèles de la plate-forme RECORD. Une fonction permettant d'utiliser ces méta-modèles sur la plate-forme a été créée mais les méta-modèles ne prennent pas en compte certains paramètres stochastiques importants comme la météo qui entraine alors une difficulté liée à l'aléa des données climatiques. De plus, la création des plans d'expérience n'a pas été faite de manière générique. Il est donc possible d'améliorer les travaux effectués en ajoutant ces éléments. Il serait aussi intéressant d'augmenter le nombre de méta-modèles utilisables sur la plate-forme.

Il pourrait être intéressant de chercher d'autres méthodes pour créer des plans d'expérience remplissant "mieux" l'espace des variables, pouvant ainsi améliorer la qualité des méta-modèles. Ces améliorations effectuées, on aurait atteint l'objectif principal, qui était d'avoir un programme simple d'utilisation et robuste permettant d'obtenir un méta-modèle efficace pour faciliter l'optimisation des variables de décisions du modèle originel.

# Bibliographie


[1] L. Breiman. Random forests. *Machine Learning*, 45 :5–32, 2001.

[2] G. Dellino, C. Meloni, and J. Kleijen. Robust simulation-optimisation using metamodels. *Proceedings of the 2009 Winter Simulation Conferance*, 2009.

[3] S. Lophaven, H. Nielsen, and J. Søndergaard. Dace - a matlab kriging toolbox, version 2.0. technical report imm-tr-2002-12. informatics and mathematical modeling, technical university of denmark. 2002.

[4] B. Reich, C. Storlie, and H. Bondell. Variable selection in bayesian smoothing spline anova model : Application to deterministic computer codes. *Tecnometrics*, 51 :110–120, 2009.

[5] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. Sensitivity analysis in practice : A guide to assessing scientific models. 2004.

[6] A. Smola and B. Scholkopf. A tutorial on support vector regression. 2003.

[7] C. Storlie, H. Bondell, B. Reich, and H. Zhang. Surface estimation, variable selection, and the nonparametric oracle property. *Statistica Sinica*, 21 :679–705, 2011.

[8] C. Storlie and J. Helton. Multiple predictor smoothing methods for sensitivity analysis : Description of techniques. *Reliability Engineering and System Safety*, 91 :28–54, 2007.

[9] V. Vapnik. The nature of statistical learning theory. springer verlag, new york, usa. 1995.

[10] N. Villa-Vialaneix. A comparison of eight metamodeling techniques for the simulation of n2o fluxes and n leaching from corn crops. *Environmental Modeling & Software*, 2011.

[11] S. Wood. *Generalized Additive Models An Introduction with R*. 2006.

[12] S. Wood and N. Augustin. Gams with integrated model selection using penalized regression splines and applications to environmental modelling. *Ecological Modelling*, 2002.


# Logiciel

– **R** version 2.15.1

# Annexe

# Fonction : polym_bis

```
# Function for use polym in case we want to enter a matrix as argument instead of V1,V2,...
# see polym function for more details
#
#
polym_bis = function(data, degree = 1, raw = TRUE)
{
if (sum(is.na(data)) != 0)
stop("data not full")

nd <- dim(as.matrix(data))[2]

if (nd == 1)
return(poly(data, degree, raw = raw))

z <- do.call("expand.grid", rep.int(list(0:degree), nd))
s <- rowSums(z)
ind <- (s > 0) & (s <= degree)
z <- z[ind, ]
s <- s[ind]
res <- cbind(1, poly(data[,1], degree, raw = raw))[, 1 + z[, 1]]

for (i in 2:nd) res <- res * cbind(1, poly(data[,i], degree, raw = raw))[, 1 + z[, i]]
     colnames(res) <- apply(z, 1, function(x) paste(x, collapse = "."))
     attr(res, "degree") <- as.vector(s)
     res
}
```

# Fonction : polym_ter

```
# Function for use polym in case we want to enter a matrix as argument
# instead of V1,V2,...
# and when there are too many variables
# see polym function for more details
#
# funcZ allow us to use the function polym when there are too many variables
# it construct a matrix containing all degrees possibles on the variables
# (for a degree max)
# this function work for degree 1, 2 and 3
funcZ = function(nd, degree)
{
z2 = NULL
z=rep(0,nd)
for(j in 1:nd)
{
z[j]=1
z2=cbind(z2,z)
z[j]=0

if(degree >= 3)
{
z[j]=3
z2=cbind(z2,z)
z[j]=0
}
if(degree >=2)
{
z[j]=2
z2=cbind(z2,z)
z[j]=0

for(i in 1:nd)
{
```

```
if(i>j)
{
z[j]=1
z[i]=1
z2=cbind(z2,z)
z[j]=0
z[i]=0
}

if(degree>=3)
{
if(i!=j)
{

z[j]=2
z[i]=1
z2=cbind(z2,z)
z[j]=0
z[i]=0
}
for(p in 1:nd)
{
if(i>j)
{
if(p>i)
{
z[j]=1
z[i]=1
z[p]=1
z2=cbind(z2,z)
z[j]=0
z[i]=0
z[p]=0
}
}
}
}
}
}
}

return (t(z2))
}
```

```
polym_ter = function(data, degree = 1, raw = TRUE)
{
if (sum(is.na(data)) != 0)
stop("data not full")

nd <- dim(as.matrix(data))[2]

if (nd == 1)
return(poly(data, degree, raw = raw))

z <- funcZ(nd,degree)
s <- rowSums(z)

res <- cbind(1, poly(data[,1], degree, raw = raw))[, 1 + z[, 1]]

for (i in 2:nd) res <- res * cbind(1, poly(data[,i], degree, raw = raw))[, 1 + z[, i]]
     colnames(res) <- apply(z, 1, function(x) paste(x, collapse = "."))
     attr(res, "degree") <- as.vector(s)
     res
}
```

# Résumé :

L'INRA possède une plate-forme de modélisation et de simulation d'agroécosystème : RECORD. Celle-ci propose un ensemble de modèles biophysiques de simulation relativement complexes. Il peut être intéressant de simplifier ces modèles afin de faciliter l'optimisation de variables de décisions. Cette simplification se fait par le biais de méta-modèles : modèle tiré du modèle originel. Une synthèse de différentes méthodes de méta-modélisation a donc été effectuée. Ces méta-modèles ont ensuite été appliqués à un modèle de la plate-forme puis comparés suivant divers aspects (temps, $R^2$, msep) afin de voir s'il est possible de les appliquer génériquement aux modèles de la plate-forme.



**Mots-clés :** méta-modèle, émulateur, surface de réponse, plan d'expérience, LHS, Sobol, SVM, forêt aléatoire, réseau de neurones, GAM, krigeage.



# Abstract :

INRA has a platform of modelisation and simulation : RECORD. It offers a set of relatively complex biophysics simulation models. It can be interesting to simplify these models in order to make the optimization of decision variables easier. This simplification is done using meta-models, *ie* models derived from the original model. A synthesis of many meta-modeling methods will be done. These methods will be applied on a model of the platform, then we will compare their results according to various aspects (time, $R^2$, msep) to assess the possibility of their generic application to the models of the platform.



**Keywords :** meta-model, surrogate, design of experiment , LHS, Sobol, SVM, random forest, neural network, GAM, krigeage.