SMURFER V1.0 :

SENSITIVITY, METAMODEL, UNCERTAINTY AND RELIABILITY FEATURED BY R

Bertrand IOOSS

Résumé

Ce document présente l'outil SMURFER V1.0 (Sensitivity, Metamodel, Uncertainty and Reliability FEatured by R) développé à partir du logiciel de statistique *R* (logiciel « freeware »). Les principales fonctionnalités de SMURFER V1.0 sont :

- l'analyse statistique de données (statistiques élémentaires, corrélations, tests d'adéquation à des lois de probabilité, visualisation de convergences, …) ;
- les calculs de fiabilité (formule de Wilks) ;
- l'échantillonnage (ou la simulation) de jeux de paramètres (avec ou sans contrainte) par les méthodes aléatoire simple, LHS (hypercubes latins) et LHS optimaux ;
- l'analyse de sensibilité entre une variable réponse et des paramètres d'entrée basée sur : les outils graphiques (scatterplots et cobweb plots), les indices issus de la régression linéaire, de la régression des rangs et des tests statistiques, la méthode de Morris, les indices de Sobol (par Monte-Carlo ou FAST) ;
- la construction de métamodèles de différents types : modèle linéaire généralisé, Partial Least Squares, Support Vector Machines, boosting d'arbres de régression, forêts aléatoires, réseaux de neurones, modèle additif généralisé, krigeage ;
- le reéchantillonnage de données pour estimer des critères de qualité sur les métamodèles (techniques de leave-one-out, validation croisée et bootstrap) ;
- les calculs des effets élémentaires et des indices de sensibilité de Sobol (par Monte-Carlo) sur un métamodèle préalablement ajusté ou sur une fonction analytique.

Ce logiciel n'est pas dédié au calcul intensif. Son intérêt réside dans son adaptabilité (utilisable sous Windows, Unix, Linux, Mac OS), sa facilité d'installation et d'utilisation (seul *R* est requis et très peu de commandes sont à connaître), sa modularité (ajouter des programmes est facile) et sa capacité d'évolutivité (intégration de méthodes statistiques modernes de *R*). Il permet également l'automatisation des tâches et le lancement de calculs en « batch » (tâche de fond). Il est donc parfaitement adapté aux phases de tests, aux études de taille moyenne et à la R&D.

Ce document décrit chacune des fonctionnalités de SMURFER, accompagnées de rappels théoriques de façon à ce qu'il soit autoportant.

REMERCIEMENTS

Ce logiciel est le fruit d'un travail collectif. Il convient donc de remercier particulièrement les personnes suivantes :

- Pierre-Mathieu Pair qui a conçu la version initiale,
- Ariane Ducellier qui a développé les modèles polynomiaux,
- Gilles Pujol pour le package « sensitivity »,
- Mathieu Petelet pour l'échantillonnage LHS sous contraintes,
- Didier Barthel pour les l'analyse de sensibilité par tests statistiques,
- M. Kadiri pour la formule de Wilks,
- Paul Lemaître pour les cobweb plots,
- Michel Marquès, Nadia Perot, Nicolas Devictor, Agnès de Crecy et Pascal Bazin pour les nombreux tests et les compléments qu'ils ont réalisés.

Je tiens également à remercier ici tous les auteurs des packages utilisés par SMURFER.

1. INTRODUCTION

De nombreux phénomènes physiques sont aujourd'hui modélisés par des équations mathématiques, transposées et résolues dans des codes de calcul. Un code de calcul est un programme informatique qui, à partir de données d'entrée (variables physiques, paramètres du modèle), calcule des sorties ou réponses. Il est utilisé à la fois dans un but de simulation lorsque les expérimentations sont trop chères ou même impossibles mais aussi dans un but de prédiction. L'utilisation de ce code de calcul s'accompagne d'une incertitude. Celle-ci résulte des approximations faites par la modélisation du phénomène physique, par sa transposition informatique et par la paramétrisation du modèle. Elle résulte également des incertitudes sur les données d'entrée, dues à une connaissance imparfaite ou à une variabilité stochastique intrinsèque des paramètres physiques.

Les techniques de traitement des incertitudes appliquées aux codes de calcul s'intéressent à l'impact des incertitudes des données d'entrée et des paramètres du modèle sur les réponses. Elles ont pour but d'estimer l'intervalle de confiance et/ou la distribution de probabilité du résultat. Elles permettent également d'extraire les données les plus influentes sur l'incertitude du résultat (phase de l'analyse de sensibilité). Cette dernière étape a souvent pour objectif d'orienter la R&D afin d'améliorer le processus, d'approfondir la connaissance des données d'entrée influentes, ou de mettre en œuvre des parades pour mitiger les effets des incertitudes. Ses résultats vont également servir à valider l'analyse d'incertitude : si l'incertitude sur des paramètres influents est grande, les résultats de l'analyse d'incertitude vont être fortement liés aux choix faits sur la distribution des paramètres d'entrée. Les aspects méthodologiques des techniques de propagation d'incertitudes et d'analyse de sensibilité sont détaillés dans plusieurs articles et ouvrages récents. Parmi ceux-ci, on peut noter Helton & Davis [25], Saltelli et al. [53] [55] [54], Santner et al. [57], Fang et al. [16], Dean & Lewis [11], Kleijnen [38], De Rocquigny et al. [12], Volkova et al. [69], Cacuci & Ionescu-Bujor [6], Jacques [35].

Depuis la fin des années 90, le logiciel de statistique *R* (langage interprété « freeware » possédant de nombreuses fonctionnalités statistiques) a connu un intérêt grandissant dans la communauté internationale de chercheurs et d'ingénieurs en statistique. Les programmes développés en *R* se révèlent particulièrement adaptés aux phases de tests, aux études de taille moyenne et au prototypage de méthodes. De plus, la mise en commun de package R permet une diffusion rapide de méthodes spécialisées et un débogage rapide des programmes.

De nombreux travaux ont ainsi conduit à implémenter les algorithmes d'analyses d'incertitudes et de sensibilité en R (Pair & Iooss [48], Ducellier [14], Pujol & Cannamela [51], Kadiri [37], Heyraud et al. [27], Petelet et al. [50], Barthel [1], Lemaître[41]). Une harmonisation de ces programmes a conduit au logiciel nommé SSURFER (Sensibilités et SURFaces de réponse dans l'Environnement *R*). Une première documentation de ce logiciel est sortie en 2006 (SSURFER V1.2, Iooss [31]). En 2010, le nom de ce logiciel a été modifié en SMURFER (Sensitivity, Metamodel, Uncertainty, Reliability FEatured by R) afin de rendre plus visible l'aspect complet du logiciel, qui inclue aussi des outils d'analyses d'incertitude et de fiabilité.

Les principales fonctionnalités de la version courante de SMURFER (SMURFER V1.0) sont les suivantes :

- analyse statistique de données (statistiques élémentaires, corrélations, tests d'adéquation à des lois de probabilité, inégalités de Bienaymé-Tchebytchev et de Vysocanskii-Petunin, visualisation de la convergence des moments en fonction de la taille de l'échantillon, …) ;
- les calculs de fiabilité (formule de Wilks) ;
- échantillonnage (ou simulation) de jeux de paramètres (avec ou sans contrainte) par la méthode aléatoire simple, celle des hypercubes latins (prise en compte d'éventuelles corrélations) et celle des hypercubes latins optimaux ;
- l'analyse de sensibilité entre une variable réponse et des paramètres d'entrée basée sur : les outils graphiques (scatterplots et cobweb plots), les indices issus de la régression linéaire, de la régression des rangs et des tests statistiques, la méthode de Morris, les indices de Sobol (calculés par par Monte-Carlo ou FAST) ;
- construction de surfaces de réponse (métamodèle à temps de calcul négligeable pouvant remplacer le code de calcul pour les analyses d'incertitude) de différents types : modèle linéaire généralisé, Partial Least Squares, Support Vector Machines, Boosting d'arbres de régression, forêts aléatoires, réseaux de neurones, modèle additif généralisé, processus gaussien (krigeage) ;
- reéchantillonnage de données pour estimer les qualités des métamodèles (techniques de leave-one-out, validation croisée et bootstrap) ;

- analyse de sensibilité par la méthode de Morris ;
- calculs des effets élémentaires et des indices de sensibilité de Sobol par divers algorithmes de Monte-Carlo (Sobol, Saltelli) ou par la méthode FAST, sur un métamodèle préalablement ajusté ou sur une fonction analytique.

Pour disposer de toutes ces fonctionnalités, la version requise de *R* est la version 2.6 ou une version ultérieure.

Un package R spécifique d'analyse de sensibilité, nommé « sensitivity » a été développé lors du travail de Pujol & Cannamela [51]. Ce package est disponible librement sur le site internet de R (http://www.r-project.org). Une harmonisation des fonctions de ce package avec celles de SMURFER n'a pas été possible pour l'instant car il y a trop de différences au niveau de la programmation. Cependant, les fonctions du package peuvent être utilisées en complémentarité à celles de SMURFER. Par exemple un métamodèle créé par SMURFER peut être utilisée dans les fonctions d'analyse de sensibilité du package « sensitivity ».

Le logiciel SMURFER n'est pas dédié au calcul intensif. Son intérêt réside dans son adaptabilité (utilisable sous Windows, Unix, Linux, Mac OS), sa facilité d'installation (seul *R* est requis), sa simplicité d'utilisation (langage interprété et très peu de commandes sont à connaître), sa modularité (ajouter des programmes est facile) et sa capacité d'évolutivité (intégration de méthodes statistiques modernes de *R*). Il permet également l'automatisation des tâches, le lancement en « batch » (tâche de fond), le stockage des résultats sous forme d'objets et le couplage avec d'autres langages (par exemple des shells python). Il est donc parfaitement adapté aux phases de tests, aux études de taille moyenne et à la R&D. Ses inconvénients sont ceux de *R* (langage interprété et logiciel non commercial) : calculs plus lents que des algorithmes programmés dans des langages non interprétés, problème d'espace mémoire limitée pour les données trop volumineuses (problème dépendant de la machine et de la mémoire alloué à l'utilisateur), fonctionnalités graphiques limitées et peu ergonomiques, modules difficilement intégrables dans une plate-forme logicielle extérieure (mais ceci reste possible).

Après une présentation générale du logiciel SMURFER dans la section suivante, le document décrit chacune de ses fonctionnalités dans des parties distinctes. La troisième partie porte sur les différents outils d'analyse statistique de données, utilisables pour la modélisation probabiliste mais aussi pour le post-traitement des résultats de simulation. La quatrième partie s'intéresse aux différents programmes d'échantillonnage aléatoire de données. La cinquième partie décrit les outils d'analyse de sensibilité entre une variable de sortie et les paramètres d'entrée d'un modèle. Les différents métamodèles sont expliqués dans la dernière partie. Des rappels théoriques sont faits dans chaque section de façon à ce que ce document soit autoportant.

2. PRESENTATION DE SMURFER

SMURFER V1.0 est un programme fonctionnant sous le logiciel *R*. *R* est un système d'analyse statistique et graphique créé par Ihaka & Gentleman [29]. *R* est à la fois un logiciel et un langage qualifié de dialecte du langage *S* créé par AT&T Bell Laboratories. *S* est devenu un standard pour les chercheurs de la communauté statistique internationale. Il intègre un nombre très important de méthodes statistiques, même parmi les plus récentes (Venables & Ripley [66]). *S-PLUS* est la version commerciale de *S* alors que *R* est distribué librement sous les termes de la *GNU General Public Licence*. Le développement et la distribution de *R* sont assurés par plusieurs statisticiens rassemblés dans le *R Development Core Team*. Il est disponible sur le site Internet du *Comprehensive R Archive Network* (CRAN) : http://www.r-project.org. De bonnes introductions à la programmation en *R* sont données par Paradis [49], Verzany [68] et Venables et al. [65].

2.1 PHILOSOPHIE DE SMURFER

La première version de SMURFER a été développée par Pair & Iooss [48]. L'idée fondatrice avait pour objectif d'obtenir le programme le plus simple et le plus intuitif à utiliser. Lors de la conception, plusieurs axes fondamentaux ont été retenus :

- Simplicité : SMURFER doit pouvoir être utilisé comme une « boîte noire », dans laquelle il suffit d'intégrer les données pour travailler.
- Souplesse : SMURFER doit pouvoir être paramétré dans les moindres détails. Cette fonctionnalité est prévue pour permettre une utilisation spécifique ou plus précise à l'utilisateur désireux d'aller plus loin.
- Modularité : le programme doit être le plus générique possible, pour permettre des évolutions futures, par exemple le rajout d'autres méthodes de régression ou d'autres critères de qualité, sans avoir besoin de réécrire tout le code.

Pour satisfaire à la contrainte de modularité, il a été décidé de séparer au maximum les fonctionnalités du programme : un sous-programme pour l'adéquation aux critères de qualité, un autre sous-programme faisant tourner les différentes méthodes, ... Le plus difficile a été d'harmoniser les entrées et sorties de tous les composants, compte tenu des contraintes informatiques posées (type, dimensions,…).

Pour satisfaire à la fois aux contraintes de souplesse et de simplicité, le programme offre un grand nombre d'options permettant de paramétrer tous les détails de l'exécution, mais ces options sont toutes facultatives. On peut le paramétrer selon ses besoins, en omettant une ou plusieurs méthodes, en changeant les paramètres des méthodes, en changeant les critères de qualité utilisés, … Cependant, le programme peut très bien fonctionner avec comme seule entrée les données à traiter.

On décrit dans cette partie l'organisation générale du logiciel. `Dans la suite, on utilisera ce style de police pour distinguer le texte issu des entêtes des fichiers et des programmes de SMURFER du reste de ce manuel.`

2.2 GESTION DU LOGICIEL

Un répertoire, non fourni aux utilisateurs, contient une sauvegarde de toutes les versions des fonctions *R* de SMURFER sous forme de répertoires "v0.1", "v0.2", …, "v1.0.0", ... Il contient aussi les fichiers de documentation et de validation, ainsi que le fichier « livraisons.txt » (dates et noms des personnes auxquelles SMURFER a été fourni). Ce répertoire est géré et conservé par B. Iooss.

2.3 DISPOSITION DES FICHIERS

Le logiciel est placé dans un répertoire principal nommé SMURFER. Ci-dessous, on donne les éléments contenus dans le fichier README.

```
Le repertoire SMURFER contient le fichier install_pkg.R, programme
d'installation automatique des packages requis par SMURFER

Les sources sont conservees dans SMURFER/current :
   * Programmes R :
       affichebootstrap.R affichecontributions.R afficheloo.R
```

```
        affichemodeleGLM.R affichestepwise.R affichetest.R
        affichevc.R biais.R bootstats.R bootstrap.R bornemaxIC.R
        cobweb.R contributions.R correlationS.R creationformule.R
        criteres.R cross.R effects.R exploreSVM.R formuleWilks.R
        leaveoneout.R mae.R maformule.R moyenne.R mse.R nmse.R
        predictSR.R quantileconverg.R r2.R R2ajuste.R residus.R
        samplingLHS.R samplingLHScontraint.R samplingOptLHS.R
        samplingSimple.R samplingSimplecontraint.R select.R
        sensibilite.R smse.R sobolS.R sobolS_joint.R smurfer.R
         statS.R statsconverg.R truncated.R
   * Repertoire ex-fonctions contenant le fichier "fonctions.R" :
       exemples de fonctions mathematiques analytiques qui
       peuvent etre appelees directement par des programmes R
   * Repertoire ex-tableaux contenant des exemples de fichiers
       "tableauN.R" (ou N est le nombre de parametres d'entree)
       definissant les termes que l'on inclut dans la regression
       polynomiale
   * Repertoire "test" contenant le fichier start.R qui donne des
       exemples d'utilisation des programmes

D'autres programmes d'analyse de sensibilite sont integres dans le
package "sensitivity" :
          src pcc morris sb fast99 sobol sobol2002 tell testmodels

Le repertoire SMURFER/VALIDATION contient start_VALIDATION.R qui a
permis de valider SMURFER sur des fichiers de donnees qui sont dans
"VALIDATION/VALIDATION_v0.5", "VALIDATION/VALIDATION_v1.1" et
"VALIDATION/VALIDATION_v1.3"
Il montre aussi des exemples d'utilisation du package "sensitivity"

Le repertoire SMURFER/doc contient les fichiers suivant
   * Doc-SMURFER_v*.pdf : documentation de la version courante
   * sensitivity.pdf : documentation du package "sensitivity"
   * README : documentation generale sur la version courante
   * RELEASE : evolutions entre les versions
```

2.4 ORGANIGRAMME

La Figure 1 illustre synthétiquement la structure des programmes de SMURFER. La première colonne de cet organigramme correspond aux fonctions que l'on peut appeler en programme principal : smurfer.R, moyenne.R, statS.R, formuleWilks.R, correlationS.R, statsconverg.R, quantileconverg.R, bornemaxIC.R, cobweb.R, sensibilite.R, samplingSimple.R, samplingLHS.R, samplingOptLHS.R, samplingSimplecontraint.R, samplingLHScontraint.R, sobolS.R, sobolS_joint.R, effectS.R, predictSR.R. Le premier encadré décrit plus en détail les différentes phases du programme SMURFER.R, avec les différents modèles qui peuvent être appelés. Au niveau des méthodes de métamodèles, on cite le nom de la méthode et le nom de la fonction appelée dans *R* (*e.g.* « gbm » pour la méthode « boosting »). Les liaisons avec les flèches correspondent aux appels à des fonctions. Par exemple, à l'intérieur de SMURFER.R, on appelle sobolS.R, qui appelle samplingSimple.R, qui appelle truncated.R.

L'encadré en bas à droite liste les noms des fonctions intégrées dans le package « sensitivity ».

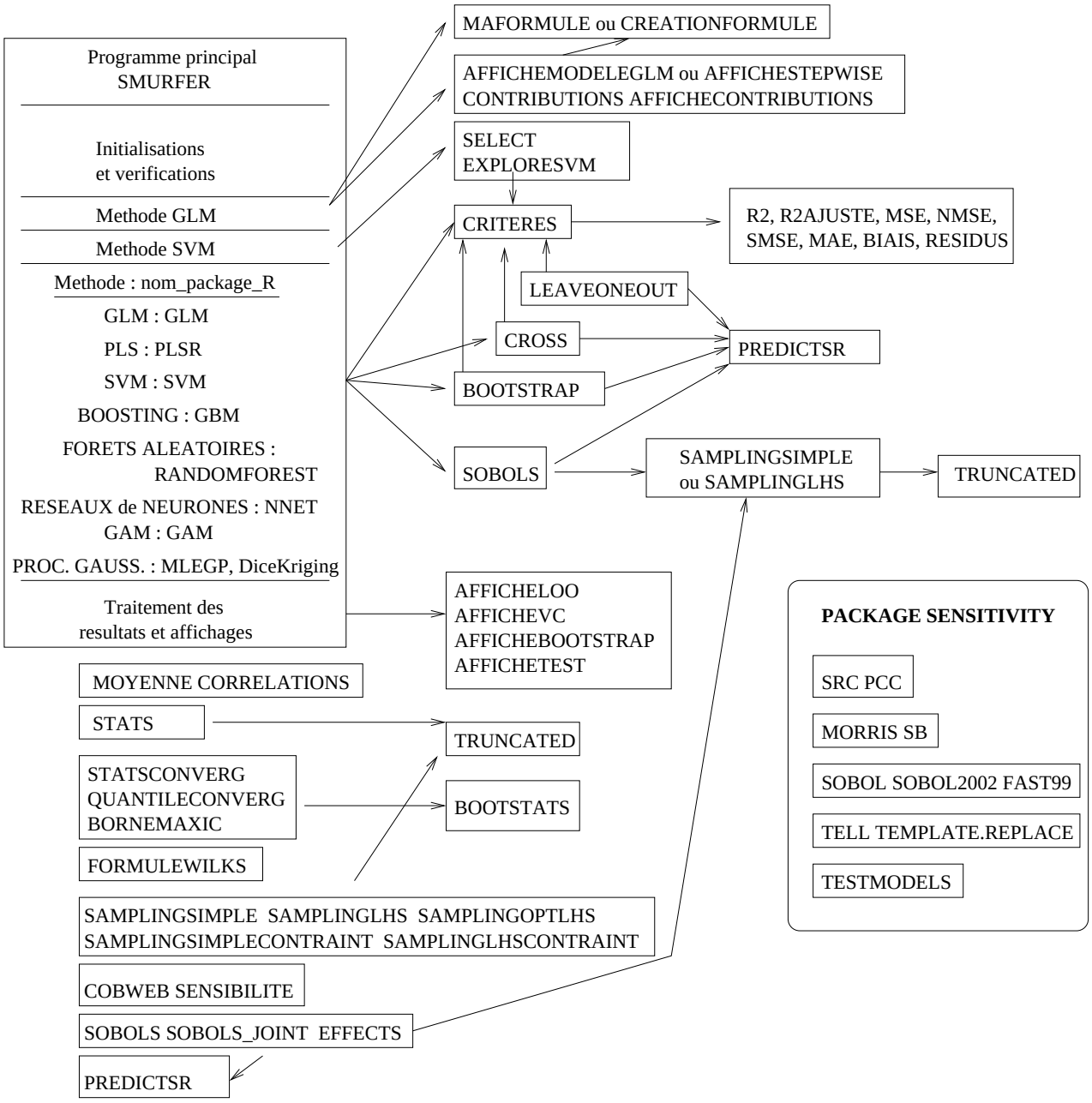

Figure 1 : Organigramme de SMURFER V1.0.

2.5 INSTALLATION

Ci-dessous, on donne les éléments contenus dans le fichier README pour l'installation du logiciel.

```
Logiciel requis : R version 2.10 (ou ulterieure)

INSTALLATION des PACKAGES R :
    Installation automatique : executer le programme install_pkg.R
      situe a la racine de SMURFER

    Installation manuelle sous Windows :
      Packages : car DiceKriging e1071 evd gbm JointModeling lhs
            mlegp pls randomForest sensitivity triangle
        Cliquer sur Packages puis sur Installer le(s) package(s)

    Installation sous Unix/Linux :
      Packages : car DiceKriging e1071 evd gbm JointModeling lhs
```

```
            mlegp pls randomForest sensitivity triangle
          Installer ces packages dans un repertoire $R_LIBRARY_PATH
          Ajouter $R_LIBRARY_PATH a la variable environnement R_LIBS
          dans le fichier d'environnement (.bashrc, .kshrc ou .cshrc)

AUTRES PACKAGES UTILISES (fournis avec l'installation de R) :
      boot mgcv nnet

INSTALLATION de SMURFER sous Unix/Linux :
      Placer SMURFER.tar.gz dans $VOTRE_REP (par ex: $HOME/src)
      gunzip SMURFER.tar.gz
      tar xvf SMURFER.tar

INSTALLATION de SMURFER sous Windows :
      Placer SMURFER.tar.gz ou SMURFER.zip dans VOTRE_REP
          (par ex: ~\Mes Documents\Programmes)
      Decompresser et extraire le contenu dans un repertoire
        nomme SMURFER
```

2.6 UTILISATION

Ci-dessous, on donne les éléments contenus dans le fichier README concernant l'utilisation du logiciel.

```
Lancement du programme sous Unix ou sous Windows :

   * Se mettre dans son repertoire de travail.
   * Creer le fichier "start.R" (modeles dans SMURFER/current/test
     et dans SMURFER/VALIDATION).
     Ce programme est destine a charger les librairies,
     les programmes de SMURFER et les fichiers de donnees.
     A faire imperativement :
       * definir la variable .PATH_SMURFER (chemin de SMURFER)
       * affecter les entrees/sorties dans des variables x et y
   * Lancer "R"
   * Sous Windows : se mettre dans son repertoire de travail sous R
   * Taper "source("start.R")" pour executer start.R (chargement
     des librairies et des donnees)

Liste des programmes que l'on peut executer
(on peut inserer ces executions dans "start.R") :

   * smurfer.R : le programme principal de metamodeles
     execution : "smurfer(x,y)"

   * predictSR.R : fonction de prediction a partir d'un modele R
       sur une matrice
     execution : "predictSR(modele,x)"

   * moyenne.R : resume statistique des parametres d'entree ou
       de sortie
     execution : "moyenne(x)"

   * statS.R : toutes les stats elementaires des parametres
       d'entree ou de sortie + tests d'adequation aux principales
       lois de proba + graphiques + appication des inegalites de
       Bienaymé-Tchebytchev et Vysocanskii-Petunin
     execution : "statS(x)"
       Exemple d'utilisation dans VALIDATION/start_VALIDATION.R

   * correlationS.R : matrice de correlation des entrees
```

```
  execution : "correlationS(x)"

* statsconverg.R : visualisation de la convergence des moyennes,
    ecart-types, skewness et kurtosis de x (avec intervalles de
  confiance bootstrap) + convergence des accroissements

* quantileconverg.R : visualisation de la convergence des
    quantiles a 75%, 90%, 95% et 99% de x (avec intervalles de
    confiance bootstrap)

* bornemaxIC.R : calcul des bornes min et max de l'intervalle de
    confiance bootstrap sur les moyenne et ecart-type quand on
    rajoute des nouveaux points

* formuleWilks.R : calcul la taille N d'un echantillon pour
    appliquer la formule de Wilks

* cobweb.R : trace les cobweb plots pour analyse de sensibilite

* sensibilite.R : tous les indices de sensibilite lineaires et
    monotones entre la sortie et les entrees + les R2 et R2*
    pour valider/invalider les hypotheses lineaire et monotone
    + les indices bases sur le test CL
    + les graphiques scatterplots entre la sortie et les entrees
  execution : "sensibilite(x,y)"

* samplingSimple.R : fonction d'echantillonnage, simulation
    aleatoire simple de N valeurs de p parametres suivant des
    lois donnees
  execution : "samplingSimple(dim_x)"

* samplingLHS.R : fonction d'echantillonnage, simulation LHS
    de N valeurs de p parametres suivant des lois donnees
  execution : "samplingLHS(dim_x)"

* samplingOptLHS.R : fonction d'echantillonnage, simulation d'un
    plan LHS optimal (maximin ou S-optimal)
    de N valeurs de p parametres suivant des lois donnees
  execution : "samplingOptLHS(dim_x)"

* samplingSimplecontraint.R : fonction d'echantillonnage,
    simulation aleatoire simple avec des contraintes de croissance
    entre parametre d'entree
  execution : "samplingSimplecontraint(dim_x)"

* samplingLHScontraint.R : fonction d'echantillonnage,
    simulation LHS avec des contraintes de croissance
    entre parametre d'entree
  execution : "samplingLHScontraint(dim_x)"

* sobolS.R : calcul des indices de Sobol a partir d'un modele ou
    d'une fonction R
  execution : "sobolS(modele,dim_x)"

* sobolS_jointR : calcul des indices de Sobol a partir d'un modele
    joint GLM ou GAM
  execution : "sobolS(model_mean,model_disp,dim_x)"

* effectS.R : visualisation des effets elementaires a partir d'un
    modele ou d'une fonction R
  execution : "effectS(model,dim_x)"
```

```
   * src (package "sensitivity") : calcul des indices src et srrc
     a partir d'un echantillon d'entrees et d'un modele

   * pcc (package "sensitivity") : calcul des indices pcc et prcc
     a partir d'un echantillon d'entrees et d'un modele

   * morris (package "sensitivity") : creation des echantillons
     et/ou calcul des indices de Sobol par la methode FAST sur un
     modele ou une fonction R

   * fast99 (package "sensitivity") : creation des echantillons et/ou
     calcul des indices par la methode de Morris sur un modele
     ou une fonction R

   * sobol (package "sensitivity") : creation des echantillons
     et/ou calcul des indices de Sobol par l'algorithme de
     Sobol (1993) sur un modele ou une fonction R

   * sobol2002 (package "sensitivity") : creation des echantillons
     et/ou calcul des indices de Sobol par l'algorithme de
     Saltelli (2002) sur un modele ou une fonction R

Le synopsis de smurfer.R est reproduit a la fin de ce README.
Pour les autres fonctions, il faut consulter l'entete des sources R
ou la documentation de SMURFER.
L'utilisation des fonctions du package "sensitivity" est décrite
en tapant "help(nom_de_la_fonction)"
```

Dans le répertoire current/test, on a disposé un fichier « start.R » contenant des exemples d'exécution de chaque programme avec quelques variantes au niveau des options.

2.7 SORTIES

Certaines fonctions (smurfer.R, predictSR.R, sampling*.R, effectS.R) retournent des variables, vecteurs, matrices ou objets à la fin de leur exécution. Les fonctions SMURFER.R, moyenne.R, statS.R, formuleWilks.R, correlationS.R, sensibilite.R et sobolS.R créent des fichiers de sortie au format texte avec l'extension « .out », contenant les résultats des différents calculs. Ces sorties sont également imprimées sur la console de l'utilisateur.

2.8 VISUALISATIONS GRAPHIQUES

Ci-dessous, on donne les éléments contenus dans le fichier README concernant les fonctionnalités de visualisation graphique contenues dans le logiciel.

```
Le programme smurfer.R cree des fichiers postscript qui sont
ouverts automatiquement sous Unix et qu'il faut ouvrir manuellement
sous Windows.

Les programmes statS.R, cobweb.R, sensibilite.R, effectS.R,
statsconverg.R, quantileconverg.R et bornemaxIC.R ouvrent des
fenetres graphiques  sous R. Celles-ci peuvent etre sauvees de la
maniere suivante :
   * sous Windows : clic droit sur le graphique puis sauver comme
     bitmap ou comme postscript
   * sous Unix : taper sous R
                 dev2bitmap("file.pdf",type="pdfwrite")
     Cela sauve la 1ere fenetre graphique, puis taper
                 dev.off()
     et on peut recommencer pour les figures suivantes
     On peut remplacer le format pdf par un autre type de formats
     (jpg, bmp, ps, ...)
```

3. OUTILS D'ANALYSE STATISTIQUE

3.1 STATISTIQUES ELEMENTAIRES (MOYENNE.R)

La fonction utilisée est la suivante :

```
moyenne ( x , nom =  c ( NA ) )
```

```
#-------------------------------------------------------------------------
#Fonction calculant la moyenne, l'ecart-type, le minimum et le maximum |
#des colonnes de x                                                      |
#-----------------------------------------------------------------------|
#Argument                                                               |
#obligatoire : x : matrice des variables d'entree                       |
#                                                                       |
#Argument                                                               |
#facultatif : nom : vecteur des noms des variables (6 caracteres)       |
#-----------------------------------------------------------------------|
#Sortie : la moyenne, l'ecart-type, le minimum et le maximum des        |
#         colonnes de la matrice sont affiches dans un fichier nomme    |
#         res-MOYENNE.out                                               |
#-----------------------------------------------------------------------|
#      Auteur : B. Iooss et A. Ducellier                                |
#-----------------------------------------------------------------------|
```

Pour que le programme fonctionne, x doit être une matrice et non un vecteur. Si x est un vecteur, on le transforme en une matrice à une colonne en tapant la ligne suivante :

```
x <- matrix ( x , nr = N , nc = 1 , dimnames = list ( 1 : N , c ( "V1" ) ) )
```

où N est le nombre d'observations et V1 est le nom de la variable (que l'utilisateur peut choisir). L'option « dimnames » peut également être omise. Ensuite, on exécute le programme en tapant :

```
moyenne ( x )
```

3.2 PARAMETRES, GRAPHIQUES ET TESTS STATISTIQUES (STATS.R)

La première phase d'une analyse d'incertitudes consiste en la modélisation probabiliste (par une fonction de densité de probabilité) des paramètres d'entrée auxquels on s'intéresse. Cette évaluation peut être faite par jugement d'expert ou à partir d'une analyse statistique de données (Saporta [56], De Rocquigny et al. [12]). De nombreux logiciels possèdent la plupart des outils de l'analyse statistique de données. On a développé notre propre programme en *R* afin de disposer d'un outil semi-automatique de tests d'adéquation de lois de probabilité sur des données (Heyraud et al. [27]). Quelques compléments y ont été également ajoutés.

La fonction stats.R calcule de nombreux paramètres statistiques élémentaires de chaque colonne du fichier d'entrée :

- Coefficients de position : moyenne, minimum, maximum, médiane, 1^{er} quartile, $3^{ème}$ quartile.
- Coefficients de dispersion : écart-type, coefficient de variation, étendue, intervalle interquartile, écart médian.
- Coefficients de forme : coefficient d'asymétrie (*skewness*), coefficient d'aplatissement (kurtosis).
- La « Boxplot » (boîte à moustaches) fournit la valeur médiane, les valeurs délimitées par les quartiles supérieur et inférieur, les valeurs minimum et maximum associées à ces quartiles ainsi que les valeurs aberrantes.
- L'histogramme donne une idée de l'allure de la distribution de probabilité. Celui-ci est ajusté par une densité issue d'un lissage par noyau gaussien (fonction *density* de R).

La fonction fournit par ailleurs des tests d'adéquation aux principales lois (uniforme, normale, normale tronquée, log normale, lognormale tronquée, triangulaire, exponentielle, Gamma et GEV (*Generalized Extreme Values*) :

- Les « Q-Q Plot » (graphiques quantiles empiriques vs. quantiles théoriques) visualisent l'adéquation de la distribution des données aux principales lois théoriques.
- La fonction met en œuvre le test statistique de Kolmogorov-Smirnov (pour toutes les lois) et celui de Shapiro-Wilks pour les lois normale et log-normale. Ces tests fournissent des p-values, que l'on traduit en niveaux de confiance avec lesquels on peut rejeter l'hypothèse d'adéquation à une loi donnée. Par exemple, si la p-value de l'adéquation à une loi normale est supérieure à 0.1, on en déduit que l'on ne peut pas rejeter l'hypothèse que la variable suit une loi normale à des niveaux supérieurs à 90%. On pourra donc modéliser la variable par une loi normale.

Tous les détails sur ces propriétés et méthodes statistiques peuvent être trouvées dans les ouvrages de référence en statistique (par exemple Saporta [56]). Dans certains cas, on observe un problème aux niveaux des tests d'adéquation aux lois (qui sont continues) quand les données d'entrée sont des séries discrètes avec des valeurs fréquemment identiques. Dans ces conditions, il est difficile d'ajuster une loi continue sur une série discrète : les graphiques « Q-Q plot » et les tests statistiques sont de mauvaise qualité. Pour rendre ces variables continues, on introduit aux grandeurs scalaires un bruit aléatoire uniforme ou « tremblement » ou « dithering » (Vivier [67]) qui permet en quelque sorte de rendre à la variable son caractère continu. Le choix des bornes du bruit dépend de la précision des données. Par exemple, si la précision est de 0.001, on ajoute un bruit aléatoire uniforme dans l'intervalle [-0.0005,0.0005]. On a donc « détronqué » ou « désarrondisé » la variable.

Enfin, la fonction met en œuvre les inégalités de Bienaymé-Tchebytchev et de Vysocanskii-Petuinin pour plusieurs niveaux de confiance (Iooss & Perot [32]). Ces inégalités sont utilisées pour déterminer la précision de l'estimateur de la moyenne d'un échantillon ou la taille nécessaire d'un échantillon pour que sa moyenne atteigne une certaine précision.

La fonction utilisée est la suivante :

```
statS<-function(x,nom=c(NA)){

 #-------------------------------------------------------------------------
 #Fonction calculant de nombreux parametres statistiques elementaires   |
 #des colonnes de x, avec des tests d'adequation aux principales lois,  |
 #et des graphiques (histogramme, densite, boxplot et Q-Q Plot)         |
 # Mise en oeuvre des inegalites de Tchebytchev et Vysocanskii-Petunin  |
 #-------------------------------------------------------------------------|
 #Arguments :                                                            |
 #                                                                       |
 #obligatoire : x : matrice des variables d'entree (format colonnes)    |
 #                  x peut posseder des elements vides (contenant "NA") |
 #                                                                       |
 #facultatif : nom : vecteur des noms des variables (7 caracteres max)  |
 #                   (par defaut : header du tableau x)                  |
 #-------------------------------------------------------------------------|
 #Sortie : tous les resultats sont affiches dans le fichier res-STAT.out|
 #-------------------------------------------------------------------------|
 #Utilisation :                                                          |
 #  - Au prealable, il faut allouer x sous R                             |
 #      Exemple a partir d'un fichier avec header :                      |
 #            mat<-read.table(file="MOX_DEI.txt",header=TRUE,dec=",")   |
 #            x<-mat[,2:length(mat)]                                     |
 #            x<-x+runif(dim(x)[[1]]*dim(x)[[2]],min=-0.0005,max=0.0005)|
 #                      # on rajoute un bruit pour avoir 1 var. continue|
 #            statS(x)                                                   |
 #      Exemple a partir d'un fichier sans header :                      |
 #            x<-read.table(file="nom_fichier")                          |
 #                                                                       |
 #  - Il faut le meme nombre de donnees dans chaque colonne              |
 #    ou alors on complete avec des "NA"                                 |
 #                                                                       |
 # WARNING : PROBLEMES si les intervalles de variation des donnees sont |
 # a valeurs trop faibles (<1e-6) ou trop fortes (>1e6)                  |
 # En ce cas ("system is computationally singular", "Erreur dans optim")|
```

```
# il suffit de travailler sur des variables standardisees              |
#                                                                       |
#-----------------------------------------------------------------------|
#      Auteur : B. Iooss                                                |
#-----------------------------------------------------------------------|

# ***********************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# ***********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
library(e1071)
library(evd)
library(triangle)
```

3.3 CALCUL DE LA MATRICE DE CORRELATION (CORRELATIONS.R)

La fonction utilisée est la suivante :

```
correlationS<-function(x,nom=c(NA)){

#-----------------------------------------------------------------
#Cette fonction permet de calculer les correlations entre     |
#les variables x et les correlations entre les rangs des x    |
#-------------------------------------------------------------|
#Argument                                                     |
#obligatoire : x : la matrice des variables d'entree          |
#                                                            |
#Argument                                                     |
#facultatif : nom : vecteur des noms des variables d'entree   |
#             (6 caracteres)                                 |
#-------------------------------------------------------------|
#Sortie : la matrice de correlations                          |
#         l'indice de conditionnement                         |
#         la matrice de correlations des rangs                |
#                                                            |
#Les resultats (correlations simples + indice de condit.)     |
#s'affichent dans un fichier nomme "res-CORRELATIONS.out"     |
#qui se trouve dans le repertoire de travail                  |
#                                                            |
#2 fichiers "matcorrel.dat" et "matcorrelrank.dat" contenant  |
#les matrices de correlation simple et de correlation sur les |
#rangs sont aussi crees                                       |
#-------------------------------------------------------------|
#      Auteur : B. Iooss et A. Ducellier                      |
#-------------------------------------------------------------|
```

3.4 CONVERGENCE DES MOMENTS (STATSCONVERG.R)

Ce type d'analyse est utilisé pour connaître la représentativité d'un échantillon, c'est-à-dire pour savoir si la taille de l'échantillon est suffisante quand on veut estimer les moments statistiques de la variable (moyenne, écart-type, …). La méthode est décrite en détail dans Iooss & Perot [32]. La fonction utilisée est la suivante :

```
statsconverg<-function(x,nom=c(NA),cvplot=T,nrepet=1,nboot=100,fboot=2,
                  kboot=1,conf=0.95,moments=FALSE,ampl=2){

#-----------------------------------------------------------------------
#Etude de la representativite des donnees en analysant la convergence  |
#des moyennes, ecart-types, skewness et kurtosis des colonnes de x,    |
```

```
#avec des intervalles de confiance donnes par bootstrap                 |
#Etude de la convergence de la somme des accroissements des moments et |
#de la somme des accroissements des IC des moments                      |
#-----------------------------------------------------------------------|
#Arguments :                                                            |
#                                                                       |
#obligatoire : x : matrice des variables d'entree (format colonnes)     |
#                                                                       |
#facultatif : nom    : vecteur des noms de variable (7 caracteres max) |
#                     (par defaut : header du tableau x)                |
#             cvplot : TRUE -> tous les graphes de convergence          |
#                      FALSE -> seulement la cvence des accroissements  |
#             nrepet : nombre de repetitions de graphes de convergence  |
#             nboot  : nombre de repliques bootstrap                    |
#             fboot  : 1ere taille d'echantillon pour la convergence    |
#             kboot  : increment de la taille d'echantillon             |
#             conf   : niveau de confiance pour l'intervalle bootstrap  |
#             moments: TRUE pour avoir les skewness et kurtosis         |
#                      en plus des moyennes et ecart-type               |
#                      (ces calculs ne fonctionnent que pour 1 colonne)|
#             ampl   : facteur d'amplification de l'axe des ordonnees   |
#                      pour graphes de cvence des sommes des accroiss   |
#-----------------------------------------------------------------------|
#Sortie : Graphes de convergence                                        |
#-----------------------------------------------------------------------|
#Utilisation :                                                          |
#  -Au prealable, il faut allouer x sous R                              |
#    Exemple : mat<-read.table(file="MOX_DEI.txt",header=TRUE,dec=",")  |
#            x<-mat[,2:length(mat)]                                     |
#            x<-x+runif(dim(x)[[1]]*dim(x)[[2]],min=-0.0005,max=0.0005)|
#            x<-read.table(file="nom_fichier")                          |
#                                                                       |
#  - Il faut le meme nombre de donnees dans chaque colonne              |
#-----------------------------------------------------------------------|
#      Auteur : B. Iooss                                                |
#-----------------------------------------------------------------------|

# ********************************************************************
# FONCTION UTILISEE :
#      bootstats.R (intervalle de confiance bootstrap)
# ********************************************************************
# LIBRAIRIES REQUISES
  library(boot)
  library(e1071) # A INSTALLER
```

3.5 CONVERGENCE DES QUANTILES (QUANTILECONVERG.R)

La fonction utilisée est la suivante :

```
    quantileconverg<-function(x,nom=c(NA),nrepet=1,nboot=100,fboot=2,kboot=1,
                      conf=0.95,moments=FALSE){

#-----------------------------------------------------------------------
#Etude de la representativite des donnees en analysant la convergence  |
#des quantiles a 75%, 90%, 95% et 99% d'une variable x,                 |
#avec des intervalles de confiance donnes par bootstrap                 |
#-----------------------------------------------------------------------|
#Arguments :                                                            |
#                                                                       |
#obligatoire : x : variable d'entree                                    |
#                                                                       |
#facultatif : nom    : vecteur des noms de variable (7 caracteres max) |
```

```
#                     (par defaut : header de x)                        |
#             nrepet : nombre de repetitions de graphes de convergence  |
#             nboot  : nombre de repliques bootstrap                    |
#             fboot  : 1ere taille d'echantillon pour la convergence    |
#             kboot  : increment de la taille d'echantillon             |
#             conf   : niveau de confiance pour l'intervalle bootstrap  |
#             moments: TRUE pour avoir les quantiles a 95% et 99%       |
#                      en plus des quantiles a 50% et 75%               |
#--------------------------------------------------------------------------|
#Sortie : Graphes de convergence                                           |
#--------------------------------------------------------------------------|
#Utilisation :                                                             |
#  -Au prealable, il faut allouer x sous R                                 |
#    Exemple : mat<-read.table(file="MOX_DEI.txt",header=TRUE,dec=",")     |
#            x<-mat[,2:length(mat)]                                        |
#            x<-x+runif(dim(x)[[1]]*dim(x)[[2]],min=-0.0005,max=0.0005)|
#            x<-read.table(file="nom_fichier")                             |
#--------------------------------------------------------------------------|
#      Auteur : B. Iooss                                                   |
#--------------------------------------------------------------------------|

# ***********************************************************************
# FONCTION UTILISEE :
#      bootstats.R (intervalle de confiance bootstrap)
# ***********************************************************************
# LIBRAIRIES REQUISES
  library(boot)
  library(e1071) # A INSTALLER
```

3.6 SIMULATION D'AJOUT DE NOUVEAUX POINTS (BORNEMAXIC.R)

La fonction utilisée est la suivante :

```
     bornemaxIC<-function(x,nom=c(NA),kmax=1,nech=100,nboot=100,conf=0.95){

#--------------------------------------------------------------------------
#Calcul des bornes min et max de l'intervalle de confiance bootstrap       |
#sur les moyenne et ecart-type si on rajoute k nouveaux points             |
#(k varie de 1 a kmax)                                                     |
#--------------------------------------------------------------------------|
#Arguments :                                                               |
#                                                                          |
#obligatoire : x : matrice des variables d'entree (format colonnes)        |
#                                                                          |
#facultatif : nom    : vecteur des noms de variable (7 caracteres max)     |
#                     (par defaut : header du tableau x)                   |
#             kmax   : nombre maximal de points que l'on ajoute            |
#             nech   : nombre de nouveaux echantillons par echantillon     |
#             nboot  : nombre de repliques bootstrap                       |
#             conf   : niveau de confiance pour l'intervalle bootstrap     |
#--------------------------------------------------------------------------|
#Sortie : Graphes d'evolution des bornes de l'intervalle de confiance      |
#--------------------------------------------------------------------------|
#Utilisation :                                                             |
#  -Au prealable, il faut allouer x sous R                                 |
#    Exemple : mat<-read.table(file="MOX_DEI.txt",header=TRUE,dec=",")     |
#            x<-mat[,2:length(mat)]                                        |
#            x<-x+runif(dim(x)[[1]]*dim(x)[[2]],min=-0.0005,max=0.0005)|
#            x<-read.table(file="nom_fichier")                             |
#                                                                          |
#  - Il faut le meme nombre de donnees dans chaque colonne                 |
#--------------------------------------------------------------------------|
```

```
#      Auteur : B. Iooss                                                    |
#-----------------------------------------------------------------------|

# ***********************************************************************
# FONCTION UTILISEE :
#      bootstats.R (intervalle de confiance bootstrap)
#      Auteur : G. Pujol (DEC/SESC)
# ***********************************************************************
# LIBRAIRIE REQUISE
  library(boot)
```

4. OUTILS DE FIABILITE : LA FORMULE DE WILKS

L'intervalle de tolérance d'une variable aléatoire *Y* est un intervalle *[m,M],* α-fractile bilatéral au niveau de confiance β, tel que :

$$\mathrm{P}\{\mathrm{P}(m \le Y \le M) \ge \alpha\} \ge \beta.$$

Une telle relation signifie que l'on peut affirmer, avec au plus (1-β) pour cent de risque de se tromper, que α pour cent au moins des valeurs possibles de la réponse *Y* sont comprises entre les valeurs *m* et *M*.

Pour calculer les bornes *m* et *M*, les techniques couramment utilisées sont :

- les méthodes de Monte Carlo et ses variantes ;
- les méthodes basées sur les statistiques d'ordre (David & Nagaraja [10]).

Parmi le second type de méthodes, la formule de Wilks (David & Nagaraja [10], Kadiri [37], de Rocquigny et al. [12], Lemaitre [41]) permet de déterminer la taille minimale *N* d'un échantillon i.i.d. ($Y_1, Y_2, \ldots, Y_N$) à générer aléatoirement en fonction des valeurs de α et β.

Remarque : Pour un code de calcul prenant en entrée un vecteur **X** et donnant en sortie une variable *Y*, un échantillon i.i.d. de *N* points $\{\mathbf{X}^{(j)}, j=1,\ldots,N\}$ est généré aléatoirement, puis un échantillon de *N* valeurs de *Y* est obtenue par application du code de calcul.

Cas bilatéral

Si on recherche un intervalle de tolérance bilatéral, la valeur de *N* est obtenue par la résolution de l'équation :

$$1-\alpha^N - N(1-\alpha)\alpha^{N-1} \ge \beta.$$

Par exemple, pour $\alpha = \beta = 95\%$, la taille *N* de l'échantillon doit être supérieure ou égale à 93. Un échantillon i.i.d. de *N* valeurs de *Y* est alors généré. Les bornes *m* et *M* de l'intervalle de tolérance sur *Y* sont obtenues en retenant les valeurs minimale et maximale de l'échantillon $\{Y_j, j=1,\ldots,N\}$:

$$m = \min_i (Y_i) \text{ et } M = \max_i (Y_i).$$

Cas unilatéral

Un α-fractile unilatéral supérieur (respectivement inférieur), au niveau de confiance β, est une borne *M* (respectivement *m*) telle que

$$\mathrm{P}\{\mathrm{P}(Y \le M) \ge \alpha\} \ge \beta.$$

Dans ces conditions, la taille de l'échantillon doit vérifier l'inégalité suivante :

$$1-\alpha^N \ge \beta.$$

Cas général

Les statistiques d'ordre permettent d'obtenir une formulation plus générale de la formule de Wilks. Soit un N-échantillon i.i.d. d'une variable aléatoire Y, rangé par ordre croissant, $(Y_{(1)}, Y_{(2)}, \ldots, Y_{(N)})$, de fonction de répartition F_Y continue et strictement croissante, on a le résultat suivant (appelé parfois lemme de Capera) :

Pour tout $1 \le r \le s \le N$, $P[Y_{(r)} \le Y \le Y_{(s)}]$ *est une variable aléatoire de loi Bêta de paramètres* $a=s-r$ *et* $b=N+1-s+r$. *On a alors*

$$G(\alpha) = \mathrm{P}\,[\mathrm{P}[Y_{(r)} \le Y \le Y_{(s)}] \ge \alpha] = \frac{1}{B(s-r, N+1-s+r)} \int_\alpha^1 t^{s-r-1}(1-t)^{N-s+r}\,dt \quad \ge \beta$$

En développant en série $G(\alpha)$, on obtient la formule de Wilks générale pour le cas bilatéral :

$$G(\alpha) = \sum_{i=0}^{s-r-1} C_N^i \alpha^i (1-\alpha)^{N-i} \geq \beta$$

Pour se ramener au cas unilatéral, on utilise la relation :

ordre n cas bilatéral = ordre 2*n cas unilatéral

La fonction programmée dans SMURFER est la suivante :

```
formuleWilks<-function(alpha=0.95,beta=0.95,bilateral=FALSE,ordre=1){

#-------------------------------------------------------------
#Calcule la taille minimale d'un échantillon à générer       |
#aléatoirement à l'aide de la Formule de Wilks               |
#                                                            |
#------------------------------------------------------------|
#Arguments                                                   |
#obligatoires : aplha : valeur du alpha pour l'intervalle    |
#                       de confiance (entre 0 et 1). Défaut  |
#                       mis à 0.95                           |
#               beta :  valeur du beta pour l'intervalle     |
#                       de tolérance (entre 0 et 1). Défaut  |
#                       mis à 0.95                           |
#               bilateral : TRUE si l'on veut un intervalle  |
#                       bilateral, FALSE par défaut          |
#               ordre : ordre de la forumule de Wilks        |
#                       L'ordre par défaut est 1             |
#------------------------------------------------------------|
#Sortie : la valeur du N minimale                            |
#------------------------------------------------------------|
#      Auteur : M. Kadiri et P. Lemaitre                     |
#------------------------------------------------------------|
```

5. OUTILS D'ECHANTILLONNAGE

A partir des lois de probabilité définies sur les paramètres d'entrée, on peut effectuer un grand nombre de simulations aléatoires de jeux de paramètres d'entrée. La technique la plus fréquente est la méthode de Monte-Carlo pure (SRS comme "Simple Random Sampling"), pour laquelle les différents vecteurs de paramètres simulés sont indépendants les uns des autres. De nombreuses autres techniques ont été développées pour permettre une convergence plus rapide de la méthode de Monte-Carlo. La méthode d'échantillonnage par hypercubes latins, implémentée dans SMURFER, est l'une des plus populaires.

5.1 ECHANTILLONNAGE ALEATOIRE SIMPLE (SAMPLINGSIMPLE.R)

La fonction est la suivante :

```
     samplingSimple<-function(dim_x,nom=c(NA),N=1,lois=rep(0,dim_x),
                 paramlois=array(0,dim=c(4,dim_x)),
                 tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x))){
#-------------------------------------------------------------------------|
#Tirage aleatoire simple de N jeux de parametres                          |
#-------------------------------------------------------------------------|
#Arguments obligatoires :                                                 |
#               dim_x : nombre de parametres d'entree dans le modele      |
#                                                                         |
#Arguments facultatifs :                                                  |
#             nom : vecteur des noms des parametres simules               |
#               (defini comme : c("V1","V2","V3",...))                    |
#             N : nombre de jeux de simulations                           |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                        |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull         |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale|
#               10=gumbel                                                 |
#               Par defaut, on prend la loi uniforme                      |
#             paramlois : tableau avec les parametres de chaque loi (max=4)|
#               pour chaque entree (range par colonne) :                  |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)   |
#               (moy,ecart-type,0,0) pour normale,                        |
#               (moy du log, ecart-type du log,0,0) pour lognormale,      |
#               (forme,echelle,0,0) pour Weibull,                         |
#               (lambda,0,0,0) pour exponentielle,                        |
#               (shape1, shape2,0,0) pour beta,                           |
#               (min,mode,max,0) pour triangulaire,                       |
#               (min,mode1,mode2,max) pour trapezoidale,                  |
#               (mode,echelle,0,0) pour Gumbel                            |
#             tronq : vecteur pour selectionner ou non une loi tronquee   |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)          |
#             paramtronq : tableau avec les parametres de troncature de   |
#               chaque loi : (min,max) range par colonne                  |
#-------------------------------------------------------------------------|
#Sortie : la matrice des N simulations des dim_x parametres               |
#-------------------------------------------------------------------------|
#      Auteur : B. Iooss                                                  |
#-------------------------------------------------------------------------|

# !!! WARNING !!!! : la troncature ne s'applique pas a la loi trapezoidale

# ********************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# ********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
```

```
  library(triangle)
  library(evd) # Gumbel
# ********************************************************************
```

Par défaut, si on ne renseigne pas les options facultatives, on a un vecteur de paramètres simulé (N=1). Ces paramètres sont de loi uniforme (lois=rep(0,dim_x) signifie que lois est un vecteur rempli de 0 de dimension dim_x), et les bornes des lois uniformes sont (0,0), car paramlois est un tableau rempli de 0.

5.2 ECHANTILLONNAGE PAR HYPERCUBES LATINS (SAMPLINGLHS.R)

Il existe plusieurs méthodes pour réduire la variance d'une variable estimée par la méthode de Monte-Carlo pure SRS (Liu [42], Saltelli et al. [53]). L'une d'elle est particulièrement efficace et assez facile à mettre en oeuvre. Il s'agit de la méthode des hypercubes latins connue sous l'acronyme LHS comme ``Latin Hypercube Sampling'' (McKay et al. [47], Iman & Conover [30], Stein [61], Helton & Davis [25]). Celle-ci consiste à obtenir un échantillon dont les points sont répartis uniformément sur toute l'étendue des paramètres incertains, en découpant simultanément l'intervalle de chaque paramètre en segments de probabilités égales, et en sélectionnant aléatoirement une valeur représentant chaque segment. Chaque paramètre ne peut prendre qu'une valeur dans chaque intervalle. Cette méthode évite donc qu'il y ait des zones sous-échantillonnées et d'autres sur-échantillonnées. Ainsi, la méthode LHS donne des résultats plus robustes que la méthode SRS pour les statistiques associées au comportement global (moyenne, variance, ...). Elle est donc préférable quand le temps de calcul du code à simuler est important et que l'on ne peut réaliser que quelques dizaines ou centaines de calculs.

Cette méthode permet aussi de simuler des variables aléatoires corrélées dans le cadre et en dehors du cadre des lois gaussiennes (comme la méthode SRS). Cette fonctionnalité est implémentée dans notre programme par l'option « correl=2 » où on introduit une matrice de corrélation sur les rangs.

La fonction est la suivante :

```
      samplingLHS <- function(dim_x,nom=c(NA),N=1,lois=rep(0,dim_x),
                  paramlois=array(0,dim=c(4,dim_x)),correl=0,
                  tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x))){

#-------------------------------------------------------------------------|
#Tirage aleatoire LHS de N jeux de parametres                             |
#(possibilite d'imposer une matrice de correlations sur les rangs)        |
#  References:                                                            |
#  Stein, M. 1987. Technometrics 29:143-151                               |
#  Iman and Conover. 1982. Commun. Stat. Simul. Comput. 11(3):311-334     |
#  McKay, Conover and Beckman. 1979. Technometrics 21: 239-245            |
#-------------------------------------------------------------------------|
#Arguments obligatoires :                                                 |
#               dim_x : nombre de parametres d'entree dans le modele      |
#                                                                         |
#Arguments facultatifs :                                                  |
#             nom : vecteur des noms des parametres simules               |
#               (defini comme : c("V1","V2","V3",...))                    |
#             N : nombre de jeux de simulations                           |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                        |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull         |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale|
#               10=gumbel                                                 |
#               Par defaut, on prend la loi uniforme                      |
#             paramlois : tableau avec les parametres de chaque loi (max=4)|
#               pour chaque entree (range par colonne) :                  |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)   |
#               (moy,ecart-type,0,0) pour normale,                        |
#               (moy du log, ecart-type du log,0,0) pour lognormale,      |
#               (forme,echelle,0,0) pour Weibull,                         |
#               (lambda,0,0,0) pour exponentielle,                        |
```

```
#               (shape1, shape2,0,0) pour beta,                                |
#               (min,mode,max,0) pour triangulaire,                            |
#               (min,mode1,mode2,max) pour trapezoidale,                       |
#               (mode,echelle,0,0) pour Gumbel                                 |
#             tronq : vecteur pour selectionner ou non une loi tronquee        |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)               |
#             paramtronq : tableau avec les parametres de troncature de        |
#               chaque loi : (min,max) range par colonne                       |
#             correl : 0 -> pas de correlation entre parametres                |
#                      1 -> pas de correlation, on supprime les correlations|
#               indesirables par la methode des permutations circulaires       |
#                      2 -> introduction d'une matrice de correlations sur     |
#               les rangs des parametres via le fichier "matcorrelrank.dat"    |
#-------------------------------------------------------------------------------|
#Sortie : la matrice des N simulations des dim_x parametres                     |
#-------------------------------------------------------------------------------|
#      Auteurs : B. Iooss                                                       |
#-------------------------------------------------------------------------------|

# !!! WARNING !!!! : le LHS et la la troncature ne s'appliquent pas a la
#                    loi trapezoidale

# ********************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# ********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
  library(triangle)
  library(evd) # Gumbel
# ********************************************************************
```

5.3 HYPERCUBES LATINS OPTIMAUX (SAMPLINGOPTLHS.R)

Il est possible d'améliorer encore les qualités de bonne répartition des points d'un plan LHS en utilisant des critères spécifiques mesurant ses qualités. L'idée est alors de générer un grand nombre de plans LHS et de choisir le meilleur au sens du critère choisi. Un package R, nommé « lhs », met en œuvre ce type de méthodes. Plus d'informations sur les critères utilisés peuvent être obtenues avec la documentation du package ou par l'aide en ligne de R :

```
library(lhs)
help(maximinLHS)
help(improvedLHS)
help(geneticLHS)
```

La fonction programmée dans SMURFER est la suivante :

```
samplingOptLHS<-function(dim_x,nom=c(NA),N=1,lois=rep(0,dim_x),
            paramlois=array(0,dim=c(4,dim_x)),
            tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x)),
            optimal="maximin",dup=1,pop=100,gen=4,pMut=0.1){

#-------------------------------------------------------------------------------|
#Tirage aleatoire d'un plan LHS optimal de N jeux de parametres                 |
#  Le plan peut etre maximin, distance-optimal ou S-optimal                     |
#-------------------------------------------------------------------------------|
#Arguments obligatoires :                                                       |
#               dim_x : nombre de parametres d'entree dans le modele           |
#                                                                              |
#Arguments facultatifs :                                                        |
#             nom : vecteur des noms des parametres simules                    |
#               (defini comme : c("V1","V2","V3",...))                         |
```

```
#             N : nombre de jeux de simulations                              |
#             lois : vecteur contenant les types de distribution de proba    |
#               pour chaque entree                                           |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull            |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale   |
#               10=gumbel                                                    |
#               Par defaut, on prend la loi uniforme                         |
#             paramlois : tableau avec les parametres de chaque loi (max=4)  |
#               pour chaque entree (range par colonne) :                     |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)      |
#               (moy,ecart-type,0,0) pour normale,                           |
#               (moy du log, ecart-type du log,0,0) pour lognormale,         |
#               (forme,echelle,0,0) pour Weibull,                            |
#               (lambda,0,0,0) pour exponentielle,                           |
#               (shape1, shape2,0,0) pour beta,                              |
#               (min,mode,max,0) pour triangulaire,                          |
#               (min,mode1,mode2,max) pour trapezoidale,                     |
#               (mode,echelle,0,0) pour Gumbel                               |
#             tronq : vecteur pour selectionner ou non une loi tronquee      |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)             |
#             paramtronq : tableau avec les parametres de troncature de      |
#               chaque loi : (min,max) range par colonne                     |
#             optimal : type d'optimalite pour le LHS (par defaut "maximin")|
#               "distance" pour distance-optimal, "S" pour S-optimal         |
#             dup : facteur pour le nb de points candidats dans les fcts     |
#               maximinLHS (plan maximin) et improvedLHS (plan dist-optimal)|
#             pop, gen, pMut : options de la fct geneticLHS (plan S-optimal)|
#               Taper help(geneticLHS) pour en savoir plus                   |
#-----------------------------------------------------------------------------|
#Sortie : la matrice des N simulations des dim_x parametres                   |
#-----------------------------------------------------------------------------|
#      Auteur : B. Iooss                                                      |
#-----------------------------------------------------------------------------|

# !!! WARNING !!!! : le LHS et la la troncature ne s'appliquent pas a la
#                     loi trapezoidale

# ********************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# ********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
  library(triangle)
  library(lhs)
  library(evd) # Gumbel
# ********************************************************************
```

5.4 ECHANTILLONNAGE ALEATOIRE SIMPLE AVEC CONTRAINTES (SAMPLINGSIMPLECONTRAINT.R)

Dans certaines applications, il est important que les paramètres d'entrée respectent certaines contraintes d'ordre entre elles, par exemple, une variable aléatoire devant être toujours supérieure à une autre. Ces contraintes créent des corrélations entre les paramètres d'entrée. Un tirage aléatoire simple des paramètres d'entrée, puis une méthode de rejet des jeux de paramètres ne respectant pas les contraintes peut s'avérer inefficace d'un point de vue temps de calcul. En utilisant les lois tronquées, on propose une fonction pour simuler des variables avec de telles contraintes. La situation proposée est celle d'une série de variables avec des contraintes de croissance ou de décroissance entre elles.

La fonction est la suivante :

```
    samplingSimplecontraint<-function(dim_x,nom=c(NA),N=1,lois=rep(0,dim_x),
                  paramlois=array(0,dim=c(4,dim_x)),contraint=rep(0,dim_x),
                  tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x))){

#-------------------------------------------------------------------------|
#Tirage aleatoire simple de N jeux de parametres avec prise en compte de  |
#contraintes entre eux (croissance ou decroissance)                       |
#  Petelet et al., 2009. http://fr.arxiv.org/abs/0909.0329                |
#-------------------------------------------------------------------------|
#Arguments obligatoires :                                                 |
#               dim_x : nombre de parametres d'entree dans le modele      |
#                                                                         |
#Arguments facultatifs :                                                  |
#             nom : vecteur des noms des parametres simules               |
#               (defini comme : c("V1","V2","V3",...))                    |
#             N : nombre de jeux de simulations                           |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                        |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull         |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale|
#               10=gumbel                                                 |
#               Par defaut, on prend la loi uniforme                      |
#             paramlois : tableau avec les parametres de chaque loi (max=4)|
#               pour chaque entree (range par colonne) :                  |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)   |
#               (moy,ecart-type,0,0) pour normale,                        |
#               (moy du log, ecart-type du log,0,0) pour lognormale,      |
#               (forme,echelle,0,0) pour Weibull,                         |
#               (lambda,0,0,0) pour exponentielle,                        |
#               (shape1, shape2,0,0) pour beta,                           |
#               (min,mode,max,0) pour triangulaire,                       |
#               (min,mode1,mode2,max) pour trapezoidale,                  |
#               (mode,echelle,0,0) pour Gumbel                            |
#             tronq : vecteur pour selectionner ou non une loi tronquee   |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)          |
#             paramtronq : tableau avec les parametres de troncature de   |
#               chaque loi : (min,max) range par colonne                  |
#             contraint : vecteur contenant les types de contraintes a    |
#               imposer pour chaque Vi                                    |
#               0=tirage normal ; 1=contrainte de croissance ; -1=contrainte|
#               de decroissance.                                          |
#               Par defaut, on realise un tirage normal.                  |
#-------------------------------------------------------------------------|
#Sortie : la matrice des N simulations des dim_x parametres               |
#-------------------------------------------------------------------------|

# !!! WARNING !!!! : la troncature ne s'applique pas a la loi trapezoidale

# *********************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# *********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
  library(triangle)
  library(evd) # Gumbel
# *********************************************************************
```

5.5 ECHANTILLONNAGE PAR HYPERCUBES LATINS AVEC CONTRAINTES (SAMPLINGLHSCONTRAINT.R)

Du fait des contraintes imposées entre les variables, l'échantillonnage aléatoire simple avec contraintes ne permet pas aux variables de respecter leurs lois marginales. Pour ce faire, il est possible d'utiliser un échantillonnage par hypercubes latins (LHS). Petelet et al. [50] ont proposé un algorithme d'échantillonnage LHS permettant d'imposer des contraintes d'ordre entre les variables.

La fonction est la suivante :

```
     samplingLHScontraint<-function(dim_x,nom=c(NA),N=1,lois=rep(0,dim_x),
                  paramlois=array(0,dim=c(4,dim_x)),contraint=rep(0,dim_x),
                  tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x))){
#-------------------------------------------------------------------------|
#Tirage aleatoire LHS de N jeux de parametres avec prise en compte de     |
#contraintes entre eux (croissance ou decroissance)                       |
#  References:                                                            |
#  Stein, M. 1987. Technometrics 29:143-151                               |
#  Iman and Conover. 1980.                                                |
#       Communications in Statistics: Theory and Methods A9: 1749-1874    |
#  McKay, Conover and Beckman. 1979. Technometrics 21: 239-245            |
#  Petelet et al., 2009. http://fr.arxiv.org/abs/0909.0329                |
#-------------------------------------------------------------------------|
#Arguments obligatoires :                                                 |
#               dim_x : nombre de parametres d'entree dans le modele      |
#                                                                         |
#Arguments facultatifs :                                                  |
#             nom : vecteur des noms des parametres simules               |
#               (defini comme : c("V1","V2","V3",...))                    |
#             N : nombre de jeux de simulations                           |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                        |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull         |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale|
#               10=gumbel                                                 |
#               Par defaut, on prend la loi uniforme                      |
#             paramlois : tableau avec les parametres de chaque loi (max=4)|
#               pour chaque entree (range par colonne) :                  |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)   |
#               (moy,ecart-type,0,0) pour normale,                        |
#               (moy du log, ecart-type du log,0,0) pour lognormale,      |
#               (forme,echelle,0,0) pour Weibull,                         |
#               (lambda,0,0,0) pour exponentielle,                        |
#               (shape1, shape2,0,0) pour beta,                           |
#               (min,mode,max,0) pour triangulaire,                       |
#               (min,mode1,mode2,max) pour trapezoidale,                  |
#               (mode,echelle,0,0) pour Gumbel                            |
#             tronq : vecteur pour selectionner ou non une loi tronquee   |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)          |
#             paramtronq : tableau avec les parametres de troncature de   |
#               chaque loi : (min,max) range par colonne                  |
#             contraint : vecteur contenant les types de contraintes a    |
#               imposer pour chaque Vi                                    |
#               0=tirage LHS ; 1=contrainte de croissance ; -1=contrainte |
#               de decroissance.                                          |
#               Par defaut, on realise un tirage LHS.                     |
#-------------------------------------------------------------------------|
#Sortie : la matrice des N simulations des dim_x parametres               |
#-------------------------------------------------------------------------|
# Auteurs : M. Petelet (DEN/SAC/DM2S/SEMT/LTA)                            |
#           B. Iooss (DEN/CAD/DER/SESI/LCFR)                              |
```

```
#-------------------------------------------------------------------------|

# !!! WARNING !!!! : le LHS et la la troncature ne s'appliquent pas a la
#                    loi trapezoidale

# ************************************************************************
# FONCTION UTILISEE :
#      truncated.R (fonctions pour lois tronquees)
# ************************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
  library(triangle)
  library(evd) # Gumbel
#************************************************************************
```

6. OUTILS D'ANALYSE DE SENSIBILITE

6.1 OUTIL GRAPHIQUE : LE COBWEB PLOT

Le ``cobweb plot'' (graphique en toile d'araignée) est un mode de représentation de jeux de données de taille importante en 2D. L'appellation a été donnée par Cooke (cf. Kurowicka & Cooke [40]) mais ce type de graphique a été initialement popularisé dans le domaine de l'analyse de données par Wegman [70] sous le nom de ``Parallel Coordinates''.

Le cobweb plot peut s'appliquer à n'importe quelle matrice. En analyse de sensibilité, il est appliqué sur une matrice de taille *n* x *m*, dans lesquelles il y a *n*-1 variables d'entrées et une variable de sortie, et ce pour *m* individus ou observations. Chaque variable est représentée par un trait vertical (que l'on espace de façon égale), et chaque individu par une ligne brisée dont les sommets sont situés sur les axes. La valeur de la position d'un individu sur un axe est donnée par la valeur prise par l'individu pour la variable correspondante. On reproduit ce tracé pour chaque individu, pour chaque axe.

La Fig. 2 montre un exemple d'un tel graphe pour la fonction d'Ishigami (Saltelli et al. [53]) qui conprend 3 variables d'entrée et une sortie.

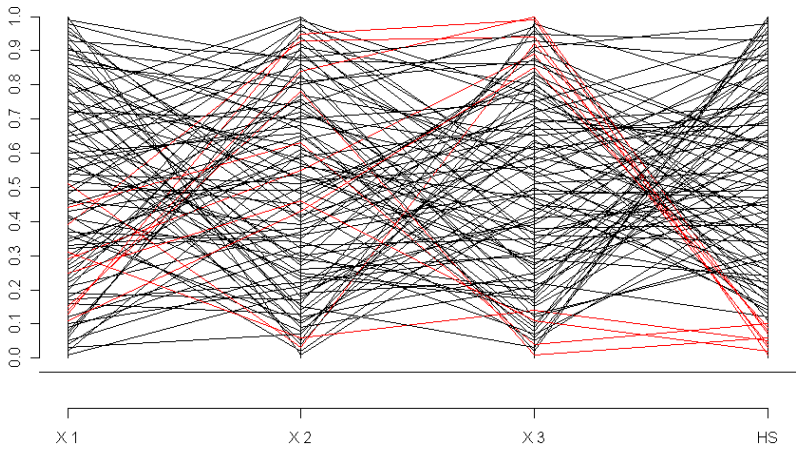

Figure 2 : Cobweb plot pour la fonction d'Ishigami sur un échantillon de taille 100. En rouge, on trace les échantillons correpondant aux 10% valeurs les plus basses de la sortie *HS*. L'échelle utilisée est « percentile ».

La méthode du cobweb plot est adaptée pour visualiser un nombre de données de taille petite à moyenne (de 1 à 1000 individus, préférence pour 50-200 individus, ainsi que 2 à 10 variables). Elle offre également des possibilités de détection d'interactions entre variables, surtout dans le cas de fonctions monotones (comme tgp5d). Elle est moins efficace dans le cadre de fonctions hautement non-monotones (comme Ishigami ou G-Sobol). Il faut néanmoins avoir `intuité'' ces interactions pour pouvoir les mettre en valeur, ce qui fait du cobweb plot une méthode complémentaire à la méthode du scatterplot, dans le cas de l'analyse d'interactions.

Dans SMURFER, nous utilisons une fonction qui dispose d'options spécifiques pour colorier les lignes et trier les axes verticaux. Nous renvoyons à Lemaitre [41] pour une explication dtéaillée de ces options. La fonction est la suivante :

```
cobweb<-function(xy,scale="percentile",MeV="basic",Afourni){
#-------------------------------------------------------------------------------|
#Fonction d'affichage du cobweb plot aussi appelé parallel coordinates     |
#-------------------------------------------------------------------------------|
#Argument                                                                       |
#obligatoire : xy : matrice de plus de deux variables d'entrée, classée en  |
#                   colonnes et une seule sortie                               |
#                                                                              |
#facultatifs : scale   : chaine de caractères, qui peut prendre les valeurs |
```

```
#                        "natural" ou "percentile"                          |
#             MeV     : chaîne de caractères qui peut prendre les valeurs   |
#                       "basic","modif", "perso"                            |
#             Afourni : empilement de listes d'instructions pour la mise    |
#                       en valeur.                                          |
#-------------------------------------------------------------------------|
#Sortie : trace le diagramme demandé                                       |
#-------------------------------------------------------------------------|
#      Auteur : P. Lemaitre                                                |
#-------------------------------------------------------------------------|
```

6.2 INDICES DE SENSIBILITE BASES SUR UN ECHANTILLON

6.2.1 Indices de sensibilités linéaires et monotones

L'analyse de sensibilité globale consiste à déterminer quelle part de variation sur la réponse d'un modèle est due à la variation sur chaque variable d'entrée. Pour évaluer cette influence numériquement on utilise différents indices de sensibilité (Saltelli et al. [53][55]). La première approche consiste à baser l'analyse de sensibilité sur la régression linéaire. Cette méthode est simple et rapide, et permet d'obtenir des résultats qualitatifs même si le modèle n'est pas linéaire ou si le nombre d'échantillons est limité. Trois différents indices de sensibilité sont considérés :

- Les coefficients de régression standard (SRC).

  Ces coefficients utilisent la régression linéaire de la réponse *Y* en fonction des *p* paramètres d'entrée X_i :

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$$

  où les β_i sont les coefficients de la régression. Les SRC (*Standardized Regression Coefficient*) sont donnés par

$$SRC(Y, X_i) = \beta_i \frac{\sigma(X_i)}{\sigma(Y)}$$

  où $\sigma(X_i)$ et $\sigma(Y)$ sont les écart-types de X_i et *Y*. Un indice SRC mis au carré mesure la part de la variance de la réponse expliquée par un facteur. Le signe du SRC indique si la réponse augmente ou diminue quand le paramètre d'entrée augmente.

  Remarque : Il faut faire attention car dans la bibliographie, le SRC est parfois défini en terme de variance et vaut donc le carré de celui qui est défini ici.

- Les coefficients de corrélation de Pearson (ρ).

  L'un des indices de sensibilité les plus souvent utilisés est le coefficient de corrélation linéaire ρ (ou coefficient de Pearson) :

$$\rho(X_i, Y) = \frac{\mathrm{cov}(X_i, Y)}{\sigma(X_i)\sigma(Y)} = \frac{\frac{1}{N}\sum_{j=1}^{N}(X_{i,j} - \overline{X}_i)(Y_j - \overline{Y})}{\sigma(X_i)\sigma(Y)}$$

  où *N* est le nombre d'éléments dans les vecteurs X_i et *Y*, et où $\overline{X}_i$ et $\overline{Y}$ sont les moyennes des variables X_i et *Y*. Cet indice élémentaire n'a de sens que dans le cas où X_i et *Y* sont liés par une relation linéaire. Il n'a pas de validité quantitative, mais il peut faire apparaître le caractère linéaire de la relation entre quelques variables dominantes et la réponse (Saporta [56]). Dans le cas parfaitement linéaire et si les paramètres X_i ne sont pas corrélés entre eux ($\rho(X_i, X_j) = 0$ pour $i \neq j$), il y a égalité entre les coefficients de corrélation et les SRC.

- Les coefficients de corrélations partielles (PCC).

  En exprimant la sensibilité de Y à une variable X_i, l'indice SRC ne tient pas compte du fait que la corrélation entre Y et X_i peut être due à une tierce variable. Les indices SRC perdent donc une grande partie de leur pertinence dans le cas où les entrées sont corrélées. Pour évaluer la sensibilité de Y à X_i en éliminant l'effet des autres variables, on considère l'indice PCC (*Partial Correlation Coefficient*). Le coefficient de corrélation partielle entre Y et X_i est défini comme le coefficient de corrélation entre $Y - \hat{Y}$ et $X_i - \hat{X}_i$, où

$$\hat{Y} = b_0 + \sum_{i=1}^{p} b_i X_i, \qquad \hat{X}_i = c_0 + \sum_{h \neq i} c_h X_h + c_i Y$$

  Le PCC mesure la corrélation linéaire entre une réponse et un facteur après avoir effectué une correction visant à neutraliser les corrélations linéaires entre les facteurs. Il est compris entre +1 et -1 et s'interprète comme un coefficient de corrélation. Ce n'est pas à proprement parler une mesure d'influence, mais plutôt une mesure de la linéarité. L'interprétation est délicate et il est préférable de ne considérer cette mesure que qualitativement.

  Remarque : Dans la bibliographie, la définition du PCC peut être légèrement différente de la notre, à savoir, avec $\hat{Y}$ issue d'une régression sur les X_j en ôtant le paramètre X_i, et $\hat{X}_i$ issue d'une régression uniquement sur les X_j (sans le Y). Cette différence n'a pas d'impact sur les valeurs du PCC.

Ces trois types de coefficients sont basés sur l'hypothèse que le modèle est linéaire. Pour valider cette hypothèse, il est important de calculer le coefficient de détermination de la régression linéaire :

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{N} (\overline{Y} - Y_i)^2}$$

Le R^2 représente le pourcentage de la variance de la réponse expliquée par le modèle de régression. Il est préférable de calculer ce coefficient sur une base de test, constituée de points n'ayant pas été utilisés dans la régression. Il est alors nommé coefficient de prédictivité et noté Q_2. Plus R^2 est proche de 1, plus la réponse Y est linéaire par rapport aux paramètres d'entrée. Si tel est le cas, les coefficients SRC vont alors donner des indices de sensibilité pour chaque paramètre d'entrée, qui permettront de classer définitivement les contributions des paramètres à la variabilité du résultat. Cependant, chaque indice aura une valeur véritablement quantitative si la relation entre Y et X_i est linéaire, c'est-à-dire si le coefficient PCC de X_i est proche de 1.

Lorsque la relation entre Y et les X_i n'est pas linéaire mais monotone, l'analyse précédente peut être réalisée sur les rangs (Saporta [56], Saltelli et al. [53]). Au lieu de considérer les valeurs des réalisations de Y et X_i, nous calculons leurs rangs : 1 pour la valeur la plus petite, 2 pour la suivante, …, N pour la plus grande. En effet une relation monotone par rapport à chaque facteur se traduit par une relation linéaire sur les rangs. Les mêmes indices sont alors calculés, seuls les noms changent :

- les coefficients de corrélation de Pearson deviennent ceux de Spearman ;
- les SRC deviennent les SRRC (*Standardized Rank Regression Coefficient* : coefficient de régression standard des rangs) ;
- les PCC deviennent les PRCC (*Partial Rank Correlation Coefficient* : coefficient de corrélation partielle des rangs) ;
- le R^2 devient le R^{2*} (coefficient de détermination sur les rangs).

L'interprétation est plus délicate car le modèle n'est plus étudié directement. L'information est donc à considérer qualitativement. Les coefficients basés sur les rangs sont souvent plus élevés que les coefficients linéaires car l'hypothèse de monotonie est moins restrictive que l'hypothèse linéaire. La différence entre R^2 et R^{2*} peut être un indicateur utile de la non linéarité du modèle.

En conclusion, ces coefficients permettent de classer les variables d'entrée par ordre croissant d'influence avant d'effectuer la régression. Cependant, ce classement ne sera valable que si la variable de sortie varie de manière linéaire ou monotone en fonction des variables d'entrée.

6.2.2 Indices basés sur des tests statistiques

L'article de Helton et al. [26] fournit une liste détaillée d'indices basés sur des tests statistiques (paramétriques ou non paramétriques). On se référera à cet article pour la description détaillée de ces tests. Une analyse détaillée a été aussi réalisée dans Barthel [1] L'idée est de partitionner en plusieurs morceaux le domaine de variation d'une entrée (et éventuellement de la sortie) et de tester ensuite si la distribution est différente entre les partitions. Il faut donc utiliser les tests qui permettent de comparer plusieurs (notamment plus de 2) échantillons : F-ANOVA, χ^2, Kruskal-Wallis, … Les résultats sont donc donnés en terme de p-value, et n'ont donc pas un sens quantitatif comme celui des SRC où on parle en terme de contribution à la variance de la sortie. Par contre l'hypothèse de linéarité ou de monotonie n'est plus nécessaire pour la validation des indices obtenus. La seule hypothèse émise concerne l'indépendance des observations. Pour les plans LHS, où les observations ne sont pas indépendantes, il semble que cette hypothèse ne soit pas contraignante et que les tests soient robustes et pleinement applicables.

Certains tests sont plus robustes que d'autres, et notamment ne supposent pas que l'échantillon suive une loi normale. Dans SMURFER, on utilise les tests suivants :

- test des localisations communes (« Common Locations » noté CL) basé sur le test de Kruskal-Wallis (Conover [7]),
- test des moyennes communes (« Common Means » noté CMN) basé sur le test de Fisher,
- test des médianes communes (« Common Medians » noté CMD) basé sur le test du χ^2,
- test des variances communes (« Common Variances » noté CV) basé sur le test de Levene,
- test d'indépendance statistique (« Statistical Independence » noté SI) basé sur le test χ^2.

6.2.3 Le programme sensibilite.R

La fonction utilisée est la suivante :

```
     sensibilite<-function(x,y,nseg=1,nsegy=0,xnom="V",ynom="y",
                     nom=c(NA),affiche=TRUE){

#-------------------------------------------------------------
#Fonction qui calcule des indices de sensibilite bases sur des  |
#echantillons ("sampling-based sensitivity indices")            |
#                                                               |
# 1) On calcule les coefficients de correlation et de           |
#regression entre des parametres x et des variables y :         |
#coef de correlation (Pearson), de regression standard (SRC) et |
#de correlation partielle (PCC).                                |
#Le coefficient de determination R2 (entre les x et y) est donne|
#pour (in)valider l'hypothese de linearite.                     |
#                                                               |
# 2) On calcule les SRC et PCC des variables centrees-reduites  |
#(a utiliser si les ordres de grandeur des parametres x sont    |
#tres differents)                                               |
#                                                               |
# 3) La fonction calcule les coefficients bases sur les rangs:  |
#coef de correlation des rangs (Spearman), de regression        |
#standard des rangs (SRRC), de correlation partielle standard   |
#des rangs (PRCC).                                              |
#Le coefficient de determination R2* (entre les rangs des x,y)  |
#est donne pour (in)valider l'hypothese de monotonie.           |
#                                                               |
# 4) On calcule les indices bases sur des tests statistiques    |
#et le decoupage du domaine des entrees :                       |
#test CL (Common Locations), test CMN (Common Means),           |
#test CMD (Common Medians), test CV (Common Variances) et       |
# test SI (Statistical Independance)                            |
#                                                               |
#Tous ces indices sont tries par ordre decroissant              |
#                                                               |
#Cette fonction affiche aussi les graphiques scatterplots entre |
```

```
#les y et chaque parametre x                                          |
#-----------------------------------------------------------------|
#Arguments                                                          |
#obligatoires : x : matrice des variables d'entree                  |
#               y : matrice des variables de sortie                 |
#                                                                   |
#Arguments                                                          |
#facultatifs : nseg : nombre de partitions sur les entrees          |
#               pour les indices bases sur les tests stats          |
#               Par defaut : nseg=1 (pas de tests)                  |
#             nsegy : nb de partition sur la sortie (test SI)       |
#               Par defaut : nsegy=nseg                             |
#             xnom : "nom" pour les variables d'entree              |
#               (on utilise xnom suivi d'un numero par entree)      |
#             ynom : "nom" pour les variables de sortie             |
#               (on utilise ynom suivi d'un numero par sortie)      |
#               ynom est utilise si la variable n'a pas de nom      |
#             nom : vecteur des noms des x et y (6 caracteres)      |
#               (xnom et ynom ne sont alors pas pris en compte)     |
#             affiche : type d'affichage                            |
#               TRUE -> scatterplot + resultats detailles           |
#               FALSE->resultats resumes (OK si bcp de sorties)     |
#-----------------------------------------------------------------|
#Sortie : Affiche tous les indices de sensibilite entre les y et    |
#           chaque x : indices lineaires (Pearson, SRC, PCC),       |
#           indices SRC et PCC des x centrees-reduites,             |
#           indices monotones (Spearman, SRRC, PRCC),               |
#           indices CL, CMN, CMD, CV, SI                            |
#         Affiche les R2, R2 ajuste, R2*, R2* ajuste pour           |
#           (in)valider les hypotheses lineaire et monotone         |
#         Ces resultats sont dans le fichier res-SENSIBILITE.out    |
#         Si affiche=TRUE, trace les graphiques scatterplots        |
#           entre les y et chaque parametre x                       |
#         Si affiche==FALSE, on conserve les vecteurs etmatrices    |
#           R2_n R2aju_n pear_n src_n pcc_n R2star_n R2staraju_n    |
#           spear_n srrc_n prcc_n CL_n CMN_n CMD_n CV_n SI_n        |
#           qui contiennent les coefficients de sensibilite         |
#-----------------------------------------------------------------
# Auteurs : B. Iooss, A. Ducellier, A. De Crecy, D. Barthel         |
#-----------------------------------------------------------------

# ********************************************************************
# LIBRAIRIES REQUISES (A INSTALLER)
  library(car)
# ********************************************************************
```

6.3 CALCUL DES INDICES DE SOBOL PAR MONTE-CARLO

Quand le modèle n'est ni linéaire, ni monotone, des indices de sensibilité peuvent être donnés par la méthode de Sobol. Dans SMURFER, ces indices sont calculés par la méthode de Monte-Carlo à partir d'un métamodèle.

6.3.1 Aspects théoriques

On cherche à exprimer la variance de la variable de sortie Y en fonction des variables d'entrée X_1, X_2, …, X_p. Cette variance $Var(Y)$ peut se décomposer de la manière suivante (Sobol [60], Homma & Saltelli [28]) :

$$Var(Y) = \sum_i D_i + \sum_{i<j} D_{ij} + \sum_{i<j<k} D_{ijk} + \ldots + D_{12\ldots p}$$

où

$D_i = Var(E(Y/X_i))$, $D_{ij} = Var(E(Y/X_iX_j)) - Var(E(Y/X_i)) - Var(E(Y/X_j))$,
$D_{ijk} = Var(E(Y/X_iX_jX_k)) - Var(E(Y/X_iX_j)) - Var(E(Y/X_iX_k)) - Var(E(Y/X_jX_k)) + Var(E(Y/X_i))$
$+ Var(E(Y/X_j)) + Var(E(Y/X_k))$, …

Les indices de sensibilité sont alors définis de la manière suivante :

$$S_i = \frac{D_i}{Var(Y)}$$

$$S_{ij} = \frac{D_{ij}}{Var(Y)}, \quad S_{ijk} = \frac{D_{ijk}}{Var(Y)}, \ldots$$

Les indices de sensibilité d'ordre 1, S_i , expriment la sensibilité de la variance de *Y* à la variable X_i, les indices de sensibilité d'ordre 2, S_{ij} , expriment la sensibilité de la variance de *Y* à l'interaction des variables X_i et X_j, et ainsi de suite.

Lorsque le nombre de variables d'entrée est élevé, on a intérêt à utiliser les indices de sensibilité totaux S_{T_i} . S_{T_i} est défini comme la somme de tous les indices de sensibilité ayant i comme indice. Ainsi, si on a trois variables d'entrée, on aura :

$$S_{T_1} = S_1 + S_{12} + S_{13} + S_{123}$$

On peut aussi écrire que où $D_{\sim i}$ est la part de la variance totale qui n'est pas issue de la variable X_i.

La méthode de Sobol (Sobol [60], Homma & Saltelli [28], Jacques [35]) permet alors de calculer une estimation par Monte-Carlo de la valeur des indices de sensibilité en simulant les variables d'entrée puis en estimant les variances par une somme de Riemann sur ces simulations. Soit f le modèle permettant de déterminer la variable de sortie en fonction des variables d'entrée. Le principe de la méthode de Sobol est le suivant :

- On considère deux matrices indépendantes (à S lignes et *p* colonnes) $x^{(1)}$ et $x^{(2)}$ de S réalisations du vecteur aléatoire $(X_1, X_2, \ldots, X_p)$, représentant deux jeux des variables d'entrée. Dans la pratique, S doit être très élevé et largement supérieur à *p*.

$$x_1 = \begin{pmatrix} X_{1,1}^{(1)} & X_{1,2}^{(1)} & \ldots & X_{1,p}^{(1)} \\ X_{2,1}^{(1)} & X_{2,2}^{(1)} & \ldots & X_{2,p}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ X_{S,1}^{(1)} & X_{S,2}^{(1)} & \ldots & X_{S,p}^{(1)} \end{pmatrix} \text{ et } x_2 = \begin{pmatrix} X_{1,1}^{(2)} & X_{1,2}^{(2)} & \ldots & X_{1,p}^{(2)} \\ X_{2,1}^{(2)} & X_{2,2}^{(2)} & \ldots & X_{2,p}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ X_{S,1}^{(2)} & X_{S,2}^{(2)} & \ldots & X_{S,p}^{(2)} \end{pmatrix}$$

On fait varier aléatoirement les variables X_i dans tout leur intervalle de variation.

- On estime la moyenne de *Y* par :

$$\hat{D}_0 = \frac{1}{S}\sum_{k=1}^{S} f(X_{k,1}^{(1)}, \ldots, X_{k,p}^{(1)})$$

- La variance de *Y* est alors estimée par :

$$\hat{D} = \frac{1}{S}\sum_{k=1}^{S} f(X_{k,1}^{(1)}, \ldots, X_{k,p}^{(1)})^2 - \hat{D}_0{}^2$$

- Les D_i sont estimés par :

$$\hat{D}_i = \frac{1}{S}\sum_{k=1}^{S} f(X_{k,1}^{(1)},\ldots,X_{k,p}^{(1)})\, f(X_{k,1}^{(2)},\ldots,X_{k,i-1}^{(2)},X_{k,i}^{(1)},X_{k,i+1}^{(2)},\ldots,X_{k,p}^{(2)}) - \hat{D}_0^{\,2}$$

c'est-à-dire que l'on considère les deux jeux de variables mais en gardant le même $X_{k,i}$.

- Les D_{ij} sont estimés par :

$$\hat{D}_{ij} = \frac{1}{S}\sum_{k=1}^{S} f(X_{k,1}^{(1)},\ldots,X_{k,p}^{(1)})\, f(X_{k,1}^{(2)},\ldots,X_{k,i-1}^{(2)},X_{k,i}^{(1)},X_{k,i+1}^{(2)},\ldots,X_{k,j-1}^{(2)},X_{k,j}^{(1)},X_{k,j+1}^{(2)},\ldots,X_{k,p}^{(2)}) - \hat{D}_i - \hat{D}_j - \hat{D}_0^{\,2}$$

c'est-à-dire que l'on considère les deux jeux de variables mais en gardant les mêmes $X_{k,i}$ et $X_{k,j}$.

- Les $D_{\sim i}$ sont estimés par :

$$\hat{D}_{\sim i} = \frac{1}{S}\sum_{k=1}^{S} f(X_{k,1}^{(1)},\ldots,X_{k,p}^{(1)})\, f(X_{k,1}^{(1)},\ldots,X_{k,i-1}^{(1)},X_{k,i}^{(2)},X_{k,i+1}^{(1)},\ldots,X_{k,p}^{(1)}) - \hat{D}_0^{\,2}$$

c'est-à-dire que l'on considère un seul jeu de variables et qu'on ne change que $X_{k,i}$.

On peut alors en déduire une estimation des indices de sensibilité d'ordre 1 S_i, des indices de sensibilité d'ordre 2 S_{ij}, et des indices de sensibilité totaux S_{T_i}.

En conclusion, le calcul des indices de Sobol est une méthode d'analyse de la variance globale de la variable de sortie d'un modèle (mais ce n'est pas la seule). Les indices de Sobol permettent de classer les variables d'entrée par ordre d'influence sans hypothèse de linéarité ou de monotonie. Cependant, leur calcul demande malheureusement de très longs temps de calculs. Par exemple, l'estimation avec un bon niveau de confiance d'un indice nécessite environ 10000 calculs. Pour avoir une estimation plus précise et un intervalle de confiance, il est utile de répéter au moins 10 fois cette estimation, ce qui multiplie le nombre de calculs par 10. Dans un cas où on a 10 paramètres d'entrée, l'estimation des indices S_i et S_{T_i} nécessite donc environ un million d'évaluations du modèle.

6.3.2 Le programme sobolS.R

La fonction utilisée est la suivante :

```
sobolS<-function(f=a,modele=b,methode=0,dim_x,x=NULL,
                logx=rep(FALSE,dim_x),logy=FALSE,S=10000,R=1,Sij=TRUE,
                LHS=FALSE,lois=rep(0,dim_x),
                paramlois=array(0,dim=c(4,dim_x)),
                tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x)),...){

#appel : sobolS(f=hs,dim_x=3,S=10000,R=1,LHS=TRUE,
#           paramlois=array(c(-pi,pi,0,0,-pi,pi,0,0,-pi,pi,0,0),dim=c(4,3)))

#-------------------------------------------------------------------------|
#Fonction pour calculer les indices de sensibilite de Sobol               |
#-------------------------------------------------------------------------|
#Arguments obligatoires dans le cas d'un modele sous forme d'objet R :    |
#       modele : objet contenant le modele                                |
#       methode : numero de la methode, 0 si fonction utilisateur         |
#         1=GLM, 2=PLS, 3=SVM, 4=BOOSTING, 5=RF, 6=RN, 7=GAM, 8=PG, 9=PGdice|
#       dim_x : nombre de parametres d'entree dans le modele              |
#                                                                         |
```

```
#Arguments obligatoires dans le cas d'une fonction :                          |
#       f : objet representant la fonction (prealablement declaree)          |
#       dim_x : nombre de parametres d'entree dans le modele                 |
#                                                                            |
#Arguments facultatifs :                                                     |
#             x : matrice de variables d'entree (pour calculer les bornes    |
#               de la loi uniforme si necessaire)                            |
#             logx : vecteur logx=c(TRUE,TRUE,...,TRUE), correspondant aux    |
#               parametres d'entree transformees en log lors de              |
#               l'ajustement du modele                                       |
#             logy : mettre logy=TRUE si la variable de sortie a ete         |
#               transformee en log lors de l'ajustement du modele            |
#             S : nombre de simulations                                      |
#             R : nombre de repetitions du programme                         |
#             Sij : TRUE pour calculer les indices d'ordre 2                 |
#             LHS : TRUE pour utiliser un echantillonnage LHS lors du        |
#               calcul des indices de Sobol                                  |
#               Par defaut : FALSE pour un echantillonnage simple            |
#             lois : vecteur contenant les types de distribution de proba    |
#               pour chaque entree                                           |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull            |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale   |
#               10=gumbel                                                    |
#               Par defaut, on prend la loi uniforme                         |
#             paramlois : tableau avec les parametres de chaque loi (max=4)  |
#               pour chaque entree (range par colonne) :                     |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)      |
#               (moy,ecart-type,0,0) pour normale,                           |
#               (moy du log, ecart-type du log,0,0) pour lognormale,         |
#               (forme,echelle,0,0) pour Weibull,                            |
#               (lambda,0,0,0) pour exponentielle,                           |
#               (shape1, shape2,0,0) pour beta,                              |
#               (min,mode,max,0) pour triangulaire,                          |
#               (min,mode1,mode2,max) pour trapezoidale,                     |
#               (mode,echelle,0,0) pour Gumbel                               |
#                Par defaut, on a la loi uniforme avec (min,max) calcules a  |
#                partir des parametres d'entree x                            |
#             tronq : vecteur pour selectionner ou non une loi tronquee      |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)             |
#             paramtronq : tableau avec les parametres de troncature de      |
#               chaque loi : (min,max) range par colonne                     |
#-----------------------------------------------------------------------------|
#Sortie : les indices Si, Sij et STi a la fin du fichier "res-SOBOL.out"     |
#         (fichier non écrasé)                                               |
#         Pour chaque indice, on donne son estimation moyenne, minimale,     |
#         maximale, ses ecart-types empirique et asymptotique (issu du TCL)  |
#-----------------------------------------------------------------------------|
#      Auteur : B. Iooss et A. Ducellier                                     |
#-----------------------------------------------------------------------------|
```

6.3.3 Le programme sobolS_joint

Dans le cas d'un métamodèle joint de type GLM ou GAM [33]) et construit avec la librairie « JointModeling » (procédure d'ajustement pas encore dans SMURFER), on peut calculer les indices de Sobol à l'aide de la fonction suivante :

```
sobolS_joint<-function(model_mean,model_disp,dim_x,methode=1,x=NULL,
            logx=rep(FALSE,dim_x),logy=FALSE,S=10000,R=1,Sij=TRUE,
            LHS=FALSE,n.trees=25000,lois=rep(0,dim_x),
            paramlois=array(0,dim=c(4,dim_x)),
            tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x))){
```

```
#-------------------------------------------------------------------------|
#Fonction pour calculer les indices de sensibilite de Sobol               |
# pour un modele joint de type GLM (methode=1) ou GAM (methode=7)         |
#-------------------------------------------------------------------------|
#Arguments obligatoires :                                                 |
#               model_mean : objet contenant la composante moyenne       |
#               model_disp : objet contenant la composante dispersion    |
#               dim_x : nombre de parametres d'entree dans le modele     |
#                                                                        |
#Arguments facultatifs :                                                  |
#             methode : numero de la methode (1 pour GLM, 7 pour GAM)     |
#             x : matrice de variables d'entree (pour calculer les bornes |
#               de la loi uniforme si necessaire)                        |
#             logx : vecteur logx=c(TRUE,TRUE,...,TRUE), correspondant aux |
#               parametres d'entree transformees en log lors de          |
#               l'ajustement du modele                                   |
#             logy : mettre logy=TRUE si la variable de sortie a ete     |
#               transformee en log lors de l'ajustement du modele        |
#             S : nombre de simulations                                  |
#             R : nombre de repetitions du programme                     |
#             Sij : TRUE pour calculer les indices d'ordre 2             |
#             LHS : TRUE pour utiliser un echantillonnage LHS lors du    |
#               calcul des indices de Sobol                              |
#               Par defaut : FALSE pour un echantillonnage simple        |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                       |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull        |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale |
#               10=gumbel                                                |
#               Par defaut, on prend la loi uniforme                     |
#             paramlois : tableau avec les parametres de chaque loi (max=4) |
#               pour chaque entree (range par colonne) :                 |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)  |
#               (moy,ecart-type,0,0) pour normale,                       |
#               (moy du log, ecart-type du log,0,0) pour lognormale,     |
#               (forme,echelle,0,0) pour Weibull,                        |
#               (lambda,0,0,0) pour exponentielle,                       |
#               (shape1, shape2,0,0) pour beta,                          |
#               (min,mode,max,0) pour triangulaire,                      |
#               (min,mode1,mode2,max) pour trapezoidale,                 |
#               (mode,echelle,0,0) pour Gumbel                           |
#                Par defaut, on a la loi uniforme avec (min,max) calcules a |
#                partir des parametres d'entree x                        |
#             tronq : vecteur pour selectionner ou non une loi tronquee   |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)         |
#             paramtronq : tableau avec les parametres de troncature de  |
#               chaque loi : (min,max) range par colonne                 |
#-------------------------------------------------------------------------|
#Sortie : les indices Si, Sij et STi a la fin du fichier "res-SOBOL.out"  |
#         (fichier non écrasé)                                            |
#         Pour chaque indice, on donne son estimation moyenne, minimale,  |
#         maximale, ses ecart-types empirique et asymptotique (issu du TCL) |
#-------------------------------------------------------------------------|
#      Auteur : B. Iooss                                                  |
#-------------------------------------------------------------------------|
# LIBRAIRIES REQUISES (A INSTALLER)
  library(JointModeling)
#-------------------------------------------------------------------------|
```

6.4 CALCUL DES EFFETS ELEMENTAIRES

A l'aide de l'analyse de variance fonctionnelle, il est possible de visualiser l'effet d'un paramètre d'entrée (resp. de l'interaction entre paramètres d'entrée) dans tout son (resp. leur) domaine de variation (Santner et al. [57], Fang et al. [16], Dean & Lewis [11]). Ses effets sont appelés effets élémentaires et peuvent être estimés à l'aide d'un nombre d'évaluations du modèle souvent prohibitif. On propose donc de les calculer sur des fonctions analytiques ou sur des métamodèles (cf. section suivante). De part leur nature additive, les modèles de surface de réponse GLM et GAM permettent d'extraire naturellement et sans difficulté ces fonctions élémentaires. La formule analytique du prédicteur du modèle processus gaussien (PG) permet également leur calcul rapide (Marrel et al. [44]).

Nous proposons ici, une approche par intégration numérique. Cette approche n'a pas été peaufinée et a été très peu testée. Elle est donc relativement faillible et nécessite quelques améliorations.

La fonction utilisée est la suivante :

```
     effectS <- function(f=a,modele=b,methode=0,dim_x,x=NULL,
                logx=rep(FALSE,dim_x),logy=FALSE,S=1000,R=100,
                LHS=FALSE,lois=rep(0,dim_x),
                paramlois=array(0,dim=c(4,dim_x)),
                tronq=rep(F,dim_x),paramtronq=array(0,dim=c(2,dim_x)),...){

#-------------------------------------------------------------------------|
#Fonction pour calculer les effets elementaires                           |
#-------------------------------------------------------------------------|
#Arguments obligatoires dans le cas d'un modele sous forme d'objet R :    |
#       modele : objet contenant le modele                                |
#       methode : numero de la methode, 0 si fonction utilisateur         |
#       dim_x : nombre de parametres d'entree dans le modele              |
#                                                                         |
#Arguments obligatoires dans le cas d'une fonction :                      |
#       f : objet representant la fonction (prealablement declaree)       |
#       dim_x : nombre de parametres d'entree dans le modele              |
#                                                                         |
#Arguments facultatifs :                                                  |
#             x : matrice de variables d'entree (pour calculer les bornes |
#               de la loi uniforme si necessaire)                         |
#             logx : vecteur logx=c(TRUE,TRUE,...,TRUE), correspondant aux |
#               parametres d'entree transformees en log lors de           |
#               l'ajustement du modele                                    |
#             logy : mettre logy=TRUE si la variable de sortie a ete      |
#               transformee en log lors de l'ajustement du modele         |
#             S : nombre de simulations pour l'integration MC             |
#             R : nombre de pas de discretisations dans le calcul des effets|
#             LHS : TRUE pour utiliser un echantillonnage LHS lors du     |
#               calcul des indices de Sobol                               |
#               Par defaut : FALSE pour un echantillonnage simple         |
#             lois : vecteur contenant les types de distribution de proba |
#               pour chaque entree                                        |
#               0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull         |
#               4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale|
#               10=gumbel                                                 |
#               Par defaut, on prend la loi uniforme                      |
#             paramlois : tableau avec les parametres de chaque loi (max=4)|
#               pour chaque entree (range par colonne) :                  |
#               (min,max,0,0) pour uniforme (par défaut : min=0, max=1)   |
#               (moy,ecart-type,0,0) pour normale,                        |
#               (moy du log, ecart-type du log,0,0) pour lognormale,      |
#               (forme,echelle,0,0) pour Weibull,                         |
#               (lambda,0,0,0) pour exponentielle,                        |
#               (shape1, shape2,0,0) pour beta,                           |
```

```
#               (min,mode,max,0) pour triangulaire,                          |
#               (min,mode1,mode2,max) pour trapezoidale,                     |
#               (mode,echelle,0,0) pour Gumbel                               |
#                Par defaut, on a la loi uniforme avec (min,max) calcules a  |
#                partir des parametres d'entree x                            |
#             tronq : vecteur pour selectionner ou non une loi tronquee      |
#               TRUE pour loi tronquee, FALSE sinon (par défaut)             |
#             paramtronq : tableau avec les parametres de troncature de      |
#               chaque loi : (min,max) range par colonne                     |
#-------------------------------------------------------------------------------|
#Sortie : 1 graphe avec tous les effets elementaires superposes                |
#         La matrice des effets elementaires EE                               |
#-------------------------------------------------------------------------------|
#      Auteur : B. Iooss                                                      |
#-------------------------------------------------------------------------------|
```

7. METAMODELES

La méthode des surfaces de réponse est un outil connu depuis longtemps qui a pour objectif de construire une fonction qui simule le comportement d'un phénomène physique ou chimique dans le domaine de variation des paramètres influents, à partir d'un certain nombre d'expériences (Box & Draper [4]). Des généralisations ultérieures ont amené cette méthode à être utilisée pour construire des modèles simplifiés se substituant à l'exécution de codes de calculs nécessitant trop de temps ou de ressources (Sacks et al. [52], Kleijnen & Sargent [39], Iooss et al. [34]). On les nomme à présent « métamodèles ».

Construire un métamodèle a donc pour objectif d'obtenir un modèle mathématique représentatif du code étudié (bonne approximation), ayant de bonnes capacités de prédiction, et dont le temps de calcul pour évaluer une réponse est négligeable. Une tel métamodèle sera donc efficace pour les analyses d'incertitude et de sensibilité, nécessitant plusieurs milliers de simulations.

Les données nécessaires pour construire un métamodèle sont :
- la connaissance du code *H* qui modélise le phénomène étudié ;
- un échantillon *D* de *N* points $(\boldsymbol{X_i}, Y_i)$, où $\boldsymbol{X_i}=(X_{i,1}, \ldots, X_{i,p})$ est un vecteur des paramètres d'entrée (de taille *p* le nombre de paramètres), $Y_i = H(\boldsymbol{X_i})$ est la réponse du code *H*, et *i* varie de 1 à *N* ;
- une famille *F* de fonctions $f(\boldsymbol{X},c)$ où *c* est un vecteur de paramètres (régression paramétrique) ou d'indices (régression non paramétrique) qui permet d'identifier les différents éléments de *F*.

En général, on utilise la technique des moindres carrés pour obtenir le meilleur représentant f_0 de *F*. On minimise la fonctionnelle :

$$R(f) = \frac{1}{N}\sum_{i=1}^{N}\left\{Y_i - f\ [\ \boldsymbol{X_i},\boldsymbol{c}\ ]\ \right\}^2$$

par rapport aux paramètres *c*, pour obtenir c_0 et le métamodèle $f_0(X)=f(X,c_0)$. Hastie et al. [24] décrivent en détail les principes de la méthode de l'apprentissage statistique et le problème du compromis entre biais et variance. En effet, un modèle avec peu de paramètres peut être très faux (biais grand) mais est peu sensible aux variations des données (variance faible), alors qu'un modèle comportant beaucoup de paramètres a souvent un biais faible mais peut avoir une variance importante.

Il y a de multiples choix possibles de familles de métamodèle : les polynômes, les fonctions à base d'interpolants radiaux, les splines, les réseaux de neurones, les modèles linéaires généralisés (GLM, dont les polynômes forment une sous-famille), les processus gaussiens, ... Les méthodes de régression mises en œuvre dans SMURFER sont les suivantes : GLM (modèles linéaires généralisés), SVM (machines à vecteurs supports), PLS (Partial Least Squares), boosting d'arbres de régression, forêts aléatoires, réseaux de neurones, GAM (modèles additifs généralisés), processus gaussiens (*i.e.* krigeage, réalisés par deux packages différents). La section suivante décrit brièvement ces différents modèles, ainsi que leur implémentation dans la fonction smurfer.R. Les critères de validation seront explicités dans la deuxième section. Enfin, le paragraphe 0 présentera de manière générale le programme smurfer.R.

7.1 LES DIFFERENTS METAMODELES

7.1.1 Modèles linéaires généralisés (GLM)

Les GLM forment une classe de modèles englobant les modèles linéaires classiques. Ils les généralisent notamment en relaxant l'hypothèse de normalité sur les observations au profit d'une distribution appartenant à la famille exponentielle. Le lecteur pourra se référer à la bibliographie (Besse [3], McCullagh & Nelder [46]) pour plus de détails sur ces modèles.

7.1.1.1 Aspects théoriques des GLM

Les modèles linéaires généralisés sont caractérisés par trois composantes :

- ✓ La *composante aléatoire* identifie la distribution de probabilité de la variable à expliquer. On suppose que l'échantillon statistique est constitué de *n* variables aléatoires Y_i $(i = 1, \ldots, n)$ indépendantes admettant des distributions issues d'une *structure exponentielle*. Cela signifie que les lois de ces

variables sont dominées par une même mesure dite de référence et que la famille de leurs densités par rapport à cette mesure se met sous la forme :

$$f(y_i;\theta_i,\phi)=\exp\left\{\frac{y_i\theta_i-v(\theta_i)}{u(\phi)}+w(y_i,\phi)\right\}$$

Cette formulation inclut la plupart des lois usuelles comportant un ou deux paramètres : gaussienne, gaussienne inverse, gamma, Poisson, binomiale. . . . Le paramètre θ_i est appelé *paramètre naturel* de la famille exponentielle.

- ✓ Les observations planifiées des variables explicatives sont organisées dans la matrice *X* de planification d'expérience (*design matrix*). Le prédicteur linéaire, *composante déterministe* du modèle, est le vecteur à *n* composantes $\eta = X\beta$, où β est un vecteur de *p* paramètres.

✓ La troisième composante exprime une *relation fonctionnelle* entre la composante aléatoire et le prédicteur linéaire. Soit $\{\mu_i = E(Y_i); i = 1...n\}$, on pose

$$\eta_i = g(\mu_i), i = 1...n$$

où *g*, appelée fonction lien, est supposée monotone et différentiable. Ceci revient à écrire un modèle dans lequel une *fonction de la moyenne* appartient au sous-espace engendré par les variables explicatives :

$$g(\mu_i) = x'_i\,\beta, i = 1...n$$

La fonction lien qui associe la moyenne μ_i au paramètre naturel est appelée *fonction lien canonique*. Dans ce cas,

$$g(\mu_i) = \theta_i = x'_i\,\beta$$

L'estimation des paramètres β_j est calculée en maximisant la log-vraisemblance du modèle linéaire généralisé. Celle-ci s'exprime pour toute famille de distributions mise sous la forme d'une structure exponentielle.

Pour calculer les coefficients de la régression, on utilise la méthode des moindres carrés itérativement repondérés (Faraway [17]). Cette méthode consiste à effectuer plusieurs fois une régression linéaire à l'aide de la méthode des moindres carrés pondérés en modifiant à chaque fois la valeur des poids. La méthode des moindres carrés pondérés consiste à utiliser la méthode des moindres carrés simple en affectant un poids w_i à chaque donnée.

7.1.1.2 Quelques aspects de la régression polynomiale utilisés dans SMURFER

Dans le cas gaussien, le modèle linéaire généralisé se ramène au modèle linéaire. On se place dans le cadre d'un modèle polynomial et on rappelle quelques algorithmes et résultats utiles associés à ce modèle. Ceux-ci sont mis en œuvre dans le modèle GLM utilisé par SMURFER.

Sélection de termes

L'algorithme nommé "stepwise" permet d'enlever du modèle de régression ses termes les moins influents. La méthode utilisée est la suivante : on part du modèle complet et pour chaque terme du modèle, on calcule ce que deviendraient la déviance et le critère AIC (*Akaike's Information Criterion*) si on remplaçait le modèle complet par le modèle auquel on a enlevé uniquement cet effet. La déviance est la somme des carrés des résidus et l'AIC est égal à *–2log-vraisemblance + 2k* où *k* est le nombre de paramètres inconnus du modèle. On calcule donc la déviance et l'AIC pour chacun des nouveaux modèles (les modèles auxquels on a enlevé un effet) et pour le modèle initial. On choisit alors d'enlever du modèle l'effet pour lequel l'AIC est le plus faible. Si cet effet est un effet principal (X_i) et s'il apparaît dans un effet de degré supérieur (X_i^2 ou X_iX_j par exemple), on décide de ne pas l'enlever du modèle. On continue ainsi à minimiser l'AIC jusqu'à ce qu'il ne soit plus possible de le faire varier significativement.

L'option *penaltyAIC* permet de régler cette procedure de selection : c'est le multiple du nombre de degrés de liberté utilise pour la pénalité. Augmenter cette valeur offre donc une sélection plus dure (on garde moins de termes).

La t-value

Pour chacun des termes de la fonction polynomiale choisie, on calcule une estimation du coefficient β_j associé à ce terme, l'écart type de ce coefficient, sa t-value et *P(>|t|),* c'est-à-dire la probabilité que |*t*| soit supérieur à la t-value si *t* suit une loi de Student à *n-k-1* degrés de liberté (*n* est le nombre d'observations et *k* le nombre de termes dans la régression). Ces deux dernières valeurs servent à évaluer le degré de confiance avec lequel sont déterminés les coefficients du modèle. Calculer la t-value et *P(>|t|)* revient à tester l'hypothèse H_0 : $\beta_j = 0$. En effet, si on ne peut pas rejeter cette hypothèse, il est inutile de calculer l'estimation b_j de β_j. Pour tester cette hypothèse, on calcule la valeur de la t-value $t(\alpha/2, n-k-1)$:

$$t(\alpha/2, n-k-1) = \frac{b_j}{\sigma_{b_j}}$$

On en déduit la valeur de α = P(>|t|) grâce à la relation entre α, n-k-1 et $t(\alpha/2, n-k-1)$ donnée dans les tables de la loi de Student à n-k-1 degrés de liberté. α proche de 0 signifie que l'hypothèse $\beta_j = 0$ est rejetée avec un niveau de confiance proche de 1 et qu'il faut donc garder l'effet lié à β_j.

Les contributions des effets

Pour calculer les contributions des effets (des termes du polynôme), on reprend la méthode proposée dans le logiciel *Lumière* (SIER [59]). On suppose qu'on a n observations et k +1 coefficients à déterminer dans le modèle. On note le modèle de la manière suivante :

$$Y = Z\beta + \varepsilon$$

où Z est le vecteur des termes du polynôme. Pour chaque observation i, on écrit

$$Y_i = \beta_0 + \beta_1 Z_{i,1} + \ldots \beta_j Z_{i,j} + \ldots + \beta_k Z_{i,k} + \varepsilon_i$$

La matrice des effets est notée :

$$Z = \begin{pmatrix} 1 & Z_{1,1} & Z_{1,2} & \cdots & Z_{1,k} \\ 1 & Z_{2,1} & Z_{2,2} & \cdots & Z_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & Z_{p,1} & Z_{p,2} & \cdots & Z_{p,k} \end{pmatrix}$$

L'estimation b de β est donnée par l'algorithme des moindres carrés suivant la formule :

$$b = [{}^tZZ]^{-1}{}^tZY = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{pmatrix}$$

La contribution du terme Z_j à la variation totale est alors :

$$C_j = b_j Cov(Z_j, Y)$$

Le fait d'avoir une covariance signifie qu'on fait une projection, comme dans la méthode PLS [63]. Cette définition de la contribution donne une idée de la contribution de l'effet Z_j à la variance de Y, *Var(Y).* Elle permet donc d'éliminer du modèle les effets peu influents. Cette information est complémentaire de celle fournie par l'analyse de sensibilité qui porte, elle, uniquement sur les variables d'entrée X_i. Il faut noter que la somme des contributions des effets est égale à la variation de *Y* expliquée par la régression (Ducellier [14]) :

$$Var(Y) = \sum_{j=0}^{k} b_j Cov(Z_j, Y) + Var(\varepsilon)$$

Il arrive que l'on obtienne par cette méthode des contributions négatives. Cela montre qu'il existe une forte corrélation entre le terme dont la contribution est négative et un des autres termes. On peut alors se demander s'il n'est pas préférable de supprimer du modèle les termes générant des contributions négatives. Cependant, il ne convient pas d'enlever systématiquement du modèle tous les termes dont la contribution est négative. En effet, d'une part, il se peut que le signe d'un des coefficients du modèle soit

incertain, et, par conséquent, le signe de la contribution correspondante sera également incertain. D'autre part, deux termes, même fortement corrélés, peuvent être nécessaires à expliquer la variable de sortie, et en enlever un peut conduire à dégrader fortement les critères de qualité du modèle.

Lorsque les effets sont fortement corrélés entre eux, il est difficile d'effectuer la régression. Par exemple, c'est parfois le cas entre les effets principaux X_i et les effets quadratiques X_i^2 ou les effets croisés X_iX_j. Pour vérifier si on ne risque pas de rencontrer de problèmes dus à cette trop grande corrélation entre les effets, on peut calculer la matrice de corrélation des effets. On appelle C la matrice de corrélation des effets Z_i *(i=1, …, k)* où l'élément $C_{i,j}$ de la i ème ligne et de la j ème colonne est défini par :

$$C_{i,j} = \frac{Cov(Z_i, Z_j)}{\sigma_{Z_i}\sigma_{Z_j}}$$

où σ_{Z_i} et σ_{Z_j} sont les écarts-types de Z_i et Z_j.

Il peut être risqué de réaliser la régression lorsqu'il est difficile d'inverser la matrice de corrélation des effets. Les qualités numériques d'inversion de la matrice sont quantifiées par l'indice de conditionnement. Celui-ci est défini de la manière suivante. Si λ_1, λ_2, … , λ_k sont les valeurs propres de la matrice de corrélation des effets rangées dans l'ordre croissant, alors l'indice de conditionnement est défini par le quotient de la plus grande valeur propre sur la plus petite (Besse [2]) :

$$idc = \frac{\lambda_k}{\lambda_1}$$

Si l'indice de conditionnement est inférieur à 100, on considère que la matrice de corrélation des effets est bien conditionnée. Si l'indice de conditionnement est supérieur à 1000, la matrice est très mal conditionnée : il y a des effets fortement liés.

7.1.1.3 Implémentation dans SMURFER

Pour construire le modèle GLM, on utilise la fonction « glm » contenue dans le package de base de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « help(glm) » dans *R*. On décrit ci-dessous les différentes options liées au modèle GLM et disponibles dans la fonction SMURFER.R.

Choix de la famille du GLM

L'option famille permet de changer le type de fonction de densité des observations. Par défaut, on a famille=gaussian, ce qui correspond aux modèles linéaires classiques. Les autres choix possibles sont « binomial », « Gamma », « inverse.gaussian », « poisson », …

Algorithme des moindres carrés itérativement repondérés

Si on désire fixer le nombre maximal d'itérations effectuées dans la régression, il faut préciser iteration=TRUE. Par défaut, les itérations sont réalisées jusqu'à ce qu'il y ait convergence. Quand on choisit de fixer le nombre maximal d'itérations, il est possible que *R* renvoie un message d'avertissement indiquant que l'algorithme n'a pas convergé. Quand on a choisi iteration = TRUE, l'option niter indique le nombre maximal d'itérations qu'effectuera SMURFER. Par défaut, on a niter = 1.

Création de la formule de régression

L'option forme permet de choisir la façon dont sera créée la formule de régression permettant de calculer le modèle. On peut choisir "creationformule" (choix par défaut) ou "maformule".

Si forme = « creationformule », le programme calcule automatiquement le polynôme incluant tous les paramètres d'entrée. L'option degreformule (par défaut degreformule=1) permet d'indiquer au programme le degré du polynôme à créer. Toutes les puissances et toutes les interactions à cet ordre et aux ordres inférieurs sont prises en compte. Le degré maximal que l'on peut prendre est égal à 4.

Si forme = « maformule », l'utilisateur définit lui-même la formule du polynôme. Pour ce faire, deux voies sont possibles :

- .On peut utiliser dans les arguments de SMURFER.R les options suivantes :

  - ✓ intercept : on met intercept = TRUE pour inclure une constante dans la formule.

  - ✓ tabA1 (permet d'inclure tous les termes en X_i), tabA2 (permet d'inclure tous les termes en X_i^2), tabA3 (permet d'inclure tous les termes en X_i^3), tabA4 (permet d'inclure tous les termes en X_i^4). Ces options sont des vecteurs de dimension *p,* le nombre de paramètres d'entrée, remplies par des « yes » et/ou des « no ». On met « yes » à la position *i* du vecteur pour inclure le paramètre X_i dans la formule. Par exemple, tabA2=rep(« yes »,dimx) permet d'inclure tous les X_i^2 dans la formule ; tabA3=c(« no », « yes », « no ») permet d'inclure uniquement le terme X_2^3 dans la formule.

  - ✓ tabB1 (permet d'inclure tous les termes en X_iX_j). Cette option est une matrice triangulaire (avec $i<j$) à deux dimensions (chacune égale au nombre de paramètre d'entrée) remplies de « yes » et/ou de « no ». Par exemple, tabB1=array("yes",dim=c(dimx,dimx)) permet d'inclure tous les X_iX_j dans la formule ; tabB1[1,3]= « yes » concerne uniquement le terme X_1X_3.

  - ✓ tabC1 (permet d'inclure tous les termes en $X_iX_jX_k$), tabC2 (permet d'inclure tous les termes en $X_i^2X_j$). tabC1 est une matrice à trois dimensions (avec $i<j<k$) , alors que tabC2 est une matrice à deux dimensions (avec $i \neq j$).

  - ✓ tabD1 (permet d'inclure tous les termes en $X_iX_jX_kX_l$), tabD2 (permet d'inclure tous les termes en $X_i^3X_j$), tabD3 (permet d'inclure tous les termes en $X_i^2X_jX_k$), tabD4 (permet d'inclure tous les termes en $X_i^2X_j^2$). tabD1 est une matrice à quatre dimensions (avec $i<j<k<l$), tabD2 est une matrice à deux dimensions (avec $i \neq j$), tabD3 est une matrice à trois dimensions (avec $i \neq j$, $i \neq k$, $j<k$) et tabD4 est une matrice à deux dimensions (avec $i \neq j$).

- On peut rentrer ces options dans des fichiers « tableauN.R » (où *N* est le nombre de paramètres d'entrée) qu'il faudra faire charger par SMURFER, avant l'exécution de smurfer.R. Des exemples de tels fichiers tableauN.R sont disponibles dans le répertoire SMURFER/current/TABLEAUX.

Sélection de termes dans la formule de régression

Il est possible d'utiliser la procédure de sélection automatique de termes du polynôme par l'algorithme « stepwise » (cf. paragraphe 7.1.1.2). Il faut alors mettre l'option stepwise à TRUE (par défaut stepwise = FALSE). Cette sélection est très pratique pour obtenir une formule optimale. L'option trace = 1 permet d'afficher les détails des étapes de l'algorithme « stepwise » (élimination itérative de termes). Par défaut, trace = 0.

7.1.2 Modèles PLS (« Partial Least Squares »)

La régression PLS (Tenenhaus [63]) univariée est un modèle non linéaire reliant une variable dépendante *Y* à un ensemble de variables indépendantes quantitatives ou qualitatives $X_1, \ldots, X_p$. Elle peut être obtenue par une suite de régressions simples et multiples. En exploitant les tests statistiques associés à la régression linéaire, il est possible de sélectionner les variables explicatives significatives à conserver dans la régression PLS et de choisir le nombre de composantes à retenir. La régression PLS usuelle est issue d'une utilisation itérative des moindres carrés ordinaires.

La régression PLS est particulièrement intéressante quand il y a des colinéarités entre les facteurs de la régression, c'est-à-dire si certains de ces facteurs sont approximativement des combinaisons linéaires d'autres facteurs. La régression PLS est également utile quand il n'y a pas assez d'observations par rapport au nombre de paramètres d'entrée.

7.1.2.1 Aspects théoriques

On suppose que les variables $Y, X_1, \ldots, X_p$ sont toutes centrées. Le modèle de la régression PLS à *m* composantes s'écrit :

$$Y = \sum_{h=1}^{m} c_h (\sum_{j=1}^{p} w_{hj}^{*} x_j) + résidu$$

avec la contrainte que les composantes PLS

$$t_h = \sum_{j=1}^{p} w_{hj}^{*} x_j$$

soient orthogonales. On peut considérer que les paramètres c_h et w^*_{hj} du modèle sont des paramètres à estimer (de là provient la non linéarité du modèle).

La régression PLS est un algorithme d'estimation des paramètres du modèle que nous allons décrire en reliant chaque étape de calcul à une régression linéaire simple ou multiple.

La première composante $t_1 = X\, w_1^*$ est définie par

$$t_1 = \frac{1}{\sqrt{\sum_{j=1}^{p} \mathrm{cov}(Y, X_j)^2}} \sum_{j=1}^{p} \mathrm{cov}(Y, X_j) X_j$$

La quantité $cov(Y, X_j)$ est aussi le coefficient de régression a_{1j} dans la régression simple reliant la variable Y à la variable $X_j/var(X_j)$:

$$Y = a_{1j}(X_j/\mathrm{var}(X_j)) + \text{résidu}$$

En effet :

$$a_{1j} = \frac{\mathrm{cov}(Y, \frac{1}{\mathrm{var}(X_j)} X_j)}{\mathrm{var}(\frac{1}{\mathrm{var}(X_j)} X_j)} = \mathrm{cov}(Y, X_j)$$

Un test sur le coefficient de régression a_{1j} permet donc d'évaluer l'importance de la variable X_j dans la construction de t_1. Ceci dit on pourrait tout aussi bien étudier la régression simple de Y sur X_j :

$$Y = a'_{1j} X_j + résidu$$

Les tests portant sur la nullité des a_{1j} et a'_{1j} sont naturellement équivalents. Dans la formule du calcul de t_1 on pourrait remplacer les covariances non significatives par 0.

On construit les régressions simples de Y et des X_j sur t_1 :

$$Y = c_1 t_1 + Y_1$$

et pour chaque $j = 1$ à p

$$X_j = p_{1j} t_1 + X_{1j}$$

La deuxième composante t_2 est définie par

$$t_2 = \frac{1}{\sqrt{\sum_{i=1}^{p} \mathrm{cov}(Y_1, X_{1j})^2}} \sum_{j=1}^{p} \mathrm{cov}(Y_1, X_{1j}) X_{1j}$$

La quantité $cov(Y_1, X_{1j})$ est aussi le coefficient de régression a_{2j} dans la régression multiple reliant la variable Y aux variables t_1 et $X_{1j}/var(X_{1j})$:

$$Y = c_{1j} t_1 + a_{2j}(X_{1j}/var(X_{1j})) + résidu$$

Ce résultat provient de l'orthogonalité entre le résidu X_{1j} et la composante t_1. La corrélation partielle entre Y et X_j conditionnellement à t_1 est définie comme la corrélation entre les résidus Y_1 et X_{1j}. De même la covariance partielle entre Y et X_j conditionnellement à t_1 est définie comme la covariance entre les résidus Y_1 et X_{1j} :

$$cov(Y, X_j | t_1) = cov(Y_1, X_{1j})$$

D'où une autre écriture de la deuxième composante PLS :

$$t_2 = \frac{1}{\sqrt{\sum_{j=1}^{p} \mathrm{cov}(Y, X_j | t_1)^2}} \sum_{j=1}^{p} \mathrm{cov}(Y, X_j | t_1) X_{1j}$$

Un test sur le coefficient de régression a_{2j} permet d'évaluer l'importance de la variable X_{1j} dans la construction de t_2. On pourrait aussi tester l'apport de la variable X_j à la construction de la deuxième composante PLS en étudiant directement la régression de Y sur $t1$ et X_j :

$$Y = c'_{ij} + a'_{2j} X_j + résidu$$

Les tests portant sur la nullité de a_{2j} et a'_{2j} sont équivalents car les vecteurs (t_1, X_{1j}) et (t_1, X_j) engendrent le même espace. Dans la formule de t_2 on pourrait remplacer les covariances non significatives par 0. La composante t_2 s'exprime aussi en fonction des variables d'origine X_j puisque les résidus $X_{1j} = X_j - p_{1j} t_1$ sont fonctions des X_j. La composante t_2 exprimée en fonction des X_j s'écrit $t_2 = Xw^*_2$.

On procède de la même manière pour le calcul des autres composantes $t_h = Xw^*_h$. On arrête en utilisant une procédure de validation croisée ou bien lorsque toutes les covariances partielles sont statistiquement non significatives.

Dans la première formule, on estime les coefficients c_h par régression multiple de Y sur les composantes PLS t_h. L'équation de régression estimée peut ensuite s'exprimer en fonction des variables d'origine X_j :

7.1.2.2 Implémentation dans SMURFER

Pour construire le modèle PLS, on utilise la fonction « plsr » contenue dans le package « pls » de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « library(pls) » puis « help(plsr) » dans *R*.

Les options liées au modèle PLS et disponibles dans la fonction SMURFER.R sont les mêmes que celles de la méthode GLM au niveau de la création de la formule de régression : la méthode PLS utilise la même formule polynomiale que dans la méthode GLM. Si l'algorithme « stepwise » a été appliqué lors de la construction du modèle GLM, la formule optimale obtenue pour le modèle GLM sera utilisée pour le modèle PLS.

Il n'y a pas d'autres options disponibles. La méthode de régression multivariée utilisée se nomme « kernelpls ». Le nombre de composantes à inclure dans le modèle est égal au nombre de termes de la formule de régression.

7.1.3 Modèles SVM (« Support Vector Machines »)

Les SVM sont un ensemble d'outils développés pour résoudre des problèmes très généraux d'apprentissage statistique supervisé. Il s'agit d'une famille d'algorithmes d'apprentissage mise en avant en particulier par le mathématicien Vapnik [64]. La maturité atteinte par cette théorie, sa faisabilité pratique et ses performances rendent maintenant possible son application effective dans de nombreux domaines. Originellement, les SVM étaient utilisées pour faire de la classification, et l'extension à la régression est assez récente. Aussi ces rappels théoriques commenceront-ils par le cas de la classification, puis nous verrons l'extension à la régression.

7.1.3.1 Aspects théoriques

Classifieurs à marge optimale

Dans un contexte de classification, il est possible de borner l'erreur de généralisation (*i.e.* l'erreur de prédiction) si le ratio entre la richesse de l'espace fonctionnel du modèle sur lequel on travaille (calculée par la dimension de Vapnik-Chervonenkis, cf. Vapnik [64]) et le nombre d'exemples est assez petit. Il existe aussi d'autres moyens de contrôler l'erreur de généralisation. L'un d'eux, pour les problèmes de classification, est le ratio entre la marge (distance minimale entre la surface séparatrice et les exemples d'apprentissage) et le diamètre de l'hypersphère qui englobe les vecteurs d'entrée de l'ensemble

d'apprentissage. Plus ce ratio est grand, et plus petite est la différence entre l'erreur d'apprentissage et l'erreur de généralisation.

Cas linéaire

Soit une surface de décision (i.e. la surface séparatrice de classification) linéaire, $f(x) = w \cdot x = 0$ (le biais peut être incorporé avec un élément constant de x). Quand $f(x) \geq 0$, on choisit la classe y=1, sinon la classe y=-1. Dans ce cas, pour un ensemble d'exemples D={(x_i,y_i)}, la marge est :

$$M(w, D) = \underset{i}{Min} \frac{(y_i f(x_i))}{\|w\|}$$

puisque $\frac{f(x)}{\|w\|}$ est la distance signée entre x et la surface $f(x)$=0. Cette marge a un sens quand w est choisi tel que $y_i f(x_i) > 0$ (la bonne classification) pour tous les i. Parmi les w qui ne donnent aucune erreur, on veut donc.

Notez que maximiser la marge est équivalent à minimiser la norme w^2 (toujours sous la contrainte que les exemples sont classifiés correctement).

Points de support (*support vectors*)

On peut montrer que w^* ne dépend que des (x_i,y_i) qui sont sur la marge, i.e., pour lesquels

$$\frac{y_i f(x_i)}{||w^*||} = M(w^*, D)$$

On appelle ces exemples des « points de support » (*support vectors*). Il existe un autre résultat concernant la relation entre le nombre de points de support et l'erreur de généralisation : l'erreur de généralisation espérée (sur des ensembles d'exemples D différents) est bornée par le ratio entre le nombre de points de support et le nombre d'exemples.

Re-paramétrisation astucieuse

On va utiliser le fait que la plupart des exemples ne sont pas des points de support, et que (dans le cas de la fonction de décision linéaire) la solution a forcément la forme d'une combinaison linéaire des exemples, $w = \sum_i \alpha_i x_i$, où les α_i sont des scalaires. Donc on a forcément $\alpha_i = 0$ pour les i qui ne sont pas des points de support. On peut alors réécrire la fonction de décision :

$$f(x) = w.x = \sum_i \alpha_i (x_i.x)$$

Selon cette paramétrisation, le nombre de degrés de liberté effectif est donc égal au nombre de points de support (en réalité il y a aussi le choix des points de support), et cela reste vrai même quand la dimension de w ou de x est très grande ou même infinie.

Cas non-linéaire

Il s'agit d'un développement assez récent (Guyon et al. [22]) qui a soudain rendu la théorie des SVM beaucoup plus intéressante. Il y a deux astuces : la première est de projeter les x sur un espace à plus haute dimension (éventuellement de dimension infinie),

$$f(x) = w.\phi(x)$$

avec une fonction vectorielle ϕ. On peut ainsi profiter de la paramétrisation efficace en terme des points de support :

$$f(x) = \sum_i \alpha_i \phi(x_i).\phi(x)$$

La deuxième astuce consiste à choisir des fonctions ϕ telles que le produit scalaire dans l'espace des ϕ est très peu coûteux à faire, et on écrit donc

$$f(x) = \sum_i \alpha_i K(x_i, x)$$

où $K(u,v)$ doit être une fonction symétrique satisfaisant certaines propriétés mathématiques (pour représenter un produit scalaire $\phi(u).\phi(v)$). $K(u,v)$ est appelé un noyau. Par exemple on peut prendre :

$$K(u,v) = e^{-\gamma\|u-v\|^2}$$

c'est-à-dire un noyau gaussien qui est l'un des plus utilisés.

Optimisation

Maximiser la marge, sous la contrainte d'un apprentissage correcte des exemples, est un problème d'optimisation quadratique sous contraintes. Ceci est assez complexe (en termes algorithmiques) mais donne des garanties sur le temps de convergence (à un minimum global), contrairement à l'optimisation numérique d'une fonction coût non quadratique.

Classifieurs à marge molle

Les utilisateurs se sont rendus compte en pratique que l'algorithme était instable quand les données étaient bruitées, i.e. quand la solution optimale (même quand on a un nombre infini d'exemples) ne sépare pas parfaitement les classes. Cortes & Vapnik [8] ont donc fait une modification heuristique, qui consiste à accepter certains exemples comme incorrects, en rajoutant une pénalité dans la fonction coût pour la distance entre la position x_i de l'entrée i et la marge au-delà de laquelle il devrait se trouver. Mathématiquement, le problème d'optimisation est presque le même qu'avant, sauf que la contrainte sur les α est plus forte : $0 \le \alpha_i \le C$

Les classifieurs à marge molle (« soft margin classifiers ») ont donc comme défaut qu'il faut choisir un paramètre (ici C) qui contrôle en fait la pénalité sur les exemples mal classifiés. On le choisit généralement par validation croisée. Il est à noter que l'ensemble des points pour lesquels α_i est différent de zéro inclut non seulement les « points de support » habituels sur la marge, mais aussi les exemples qui sont du mauvais côté de leur marge. Pour ceux-ci on obtiendra une valeur de $\alpha_i = C$, i.e., si on prenait une valeur de C plus grande, ces exemples pourraient devenir des exemples correctement classifiés ou des exemples sur la marge. Cette formulation est équivalente à une autre où l'on représente explicitement les « écarts à la marge » par des variables $\eta_i \ge 0$:

$$y_i(w.\phi(x_i) + b) \ge 1 - \eta_i$$

et on minimise la somme des η_i.

Extensions à la régression et à l'estimation de densité

L'algorithme a aussi récemment été étendu à la régression et à l'estimation de densité. Pour la régression, l'idée de base est que l'on considère une fonction de perte appelée *ε-insensitive* :

$$Q(y, f(x,w)) = I_{|y-f(x,w)| \le \varepsilon}\varepsilon + I_{|y-f(x,w)|>\varepsilon}|y - f(x,w)|$$

où Q est la fonction de perte, x et y sont les données, w est le vecteur de poids et ε est la valeur de l'erreur qu'on est prêt à accepter.

Cela définit ce qu'on appelle un ε-tube autour de y : tant que notre solution est à l'intérieur de ce tube l'erreur est minimale (cela correspond au cas où tous les exemples d'apprentissage sont correctement classifiés). On peut alors construire une fonction coût qui inclut la minimisation de la norme des poids (w^2), ainsi que des pénalités linéaires pour les écarts η_i du ε-tube. La minimisation de ces écarts est donc très similaire au problème des SVM à marge molle (mais on doit considérer deux sortes d'écarts séparément,

les écarts positifs et les écarts négatifs). Comme pour les classifieurs à marge molle, il faut choisir une constante *C* qui est une borne sur les coefficients de Lagrange α_i, mais il faut aussi choisir ε.

7.1.3.2 Implémentation dans SMURFER

Pour construire le modèle SVM, on utilise la fonction « svm » contenue dans le package « e1071 » de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « library(e1071) » puis « help(svm) » dans *R*.

Le modèle mis en œuvre dans SMURFER se limite pour l'instant au noyau gaussien. Trois paramètres sont alors à régler : le paramètre γ (portée du noyau gaussien), le paramètre *C* (coût) et le paramètre ε (marge de la fonction de perte). On décrit ci-dessous les différentes options liées au modèle SVM et disponibles dans la fonction SMURFER.R.

L'option SVMexplore (par défaut SVMexplore=TRUE) permet de chercher le jeu de paramètres optimal de *C*, ε et γ par une exploration systématique de certaines plages de variation. Les plages d'exploration des paramètres sont données dans les options cmin, cmax, cpas, epsmin, epsmax, epspas, gmin, gmax, gpas, qui correspondent au minimum, maximum et pas des paramètres C, ε et γ. Par défaut, on a les valeurs suivantes : cmin=1, cmax=101, cpas=50, epsmin=0.01, epsmax=0.51, epspas=0.25, gmin=0.5, gmax=10.5, gpas=5. Si SVMexplore = FALSE, alors on n'explore pas les plages de variations des paramètres et on prend *C*, ε et γ à leur valeur minimale cmin, epsmin et gmin.

L'option « critere » donne le critère sur lequel le programme se base pour obtenir le meilleur modèle dont il récupère le jeu de paramètres optimal. Par défaut, critere = R2 ; mais les autres possibilités sont R2ajuste, MSE, NMSE, SMSE, SSD, B, BN, RM, RRM ou RRE (cf. paragraphe 7.2.1).

7.1.4 Modèle BOOSTING (boosting d'arbres de régression)

Le boosting (Hastie et al. [24], Besse [2]) diffère des approches précédentes par ses origines et ses principes. L'idée initiale, en apprentissage statistique, était d'améliorer les compétences d'un *classifieur faible,* c'est-à-dire celles d'un modèle de discrimination dont la probabilité de succès sur la prédiction d'une variable qualitative est légèrement supérieure à celle d'un choix aléatoire. L'idée originale de Schapire [58] a été affinée par Freund & Schapire [20] qui ont décrit l'algorithme original *AdaBoost (Adaptative boostin*g) pour la prédiction d'une variable binaire. De nombreuses études ont ensuite été publiées pour adapter cet algorithme à d'autres situations (*k* classes, régression) et rendre compte de ses performances sur différents jeux de données (Schapire [58]). Ces études ont montré le réel intérêt pratique de ce type d'algorithme pour réduire sensiblement la variance, mais aussi le biais de prédiction comparativement à d'autres approches. Cet algorithme est même considéré comme la meilleure méthode *"off-the-shelf*" c'est-à-dire ne nécessitant pas un long prétraitement des données ni un réglage fin de paramètres lors de la procédure d'apprentissage.

Le boosting adopte le principe général suivant : construction d'une famille de modèles qui sont ensuite agrégés par une moyenne pondérée des estimations ou un vote. Il diffère nettement sur la façon de construire la famille qui est dans ce cas récurrente : chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées ou mal prédites. Intuitivement, cet algorithme concentre donc ses efforts sur les observations les plus difficiles à ajuster tandis que l'agrégation de l'ensemble des modèles permet d'échapper au surajustement. Les algorithmes de boosting proposés diffèrent par certaines caractéristiques :
• la façon de pondérer c'est-à-dire de renforcer l'importance des observations mal estimées lors de l'itération précédente ;
• leur objectif selon le type de la variable à prédire *Y* : binaire, qualitative à *k* classes, réelles ;
• la fonction perte, qui peut être choisie plus ou moins robuste aux valeurs atypiques, pour mesurer l'erreur d'ajustement ;
• la façon d'agréger, ou plutôt de pondérer, les modèles de base successifs.

La littérature sur le sujet présente donc de très nombreuses versions de cet algorithme et il est encore difficile de dire lesquelles sont les plus efficaces et si une telle diversité est bien nécessaire. A titre d'information, nous présenterons tout d'abord l'algorithme basique consacré à la classification, puis nous nous intéresserons à son extension à la régression.

Algorithme de base

Décrivons la version originale du *boosting* pour un problème de discrimination élémentaire à deux classes en notant δ la fonction de discrimination à valeurs dans {-1,1}. Dans cette version, le modèle de base retourne l'identité d'une classe, il est encore nommé Adaboost discret. Il est facile de l'adapter à des modèles retournant une valeur réelle comme une probabilité d'appartenance à une classe.

ALGORITHME: **AdaBoost** (adaptative boosting*)*
• *Soit x_0 à prévoir et*
• *$z = \{ (x_1, y_1), \ldots, (x_n, y_n) \}$ un échantillon*
• Initialiser *les poids $w = \{ w_i = 1/n ; i = 1, \ldots, n \}$.*
• Pour *m = 1* à *M*, Faire :
– *Estimer δm sur l'échantillon pondéré par w.*
– *Calculer le taux d'erreur apparent :* $\hat{\varepsilon}_p = \dfrac{\sum_{i=1}^{n} w_i 1\{\delta_m(x_i) \neq y_i\}}{\sum_{i=1}^{n} w_i}$
– *Calculer les logit :* $c_m = \log((1-\hat{\varepsilon}_p)/\hat{\varepsilon}_p)$
– *Calculer les nouvelles pondérations :*
• Fin Pour
Résultat du vote (*sgn e*st la fonction signe) :

$$\hat{\phi}_M(x_0) = sgn\left[\sum_{m=1}^{M} c_m \delta_m(x_0)\right]$$

FIN ALGORITHME

Les poids de chaque observation sont initialisés à 1/n pour l'estimation du premier modèle puis évoluent à chaque itération donc pour chaque nouvelle estimation. L'importance d'une observation w_i est inchangée si elle est bien classée, elle croît sinon proportionnellement au défaut d'ajustement du modèle. L'agrégation finale des prévisions : $\sum_{m=1}^{M} c_m \delta_m(x_0)$ est une combinaison pondérée par les qualités d'ajustement de chaque modèle. Sa valeur absolue appelée *marge* est proportionnelle à la confiance que l'on peut attribuer à son signe qui fournit le résultat de la prévision.

Ce type d'algorithme est largement utilisé avec un arbre (CART) comme modèle de base. Si le "classifieur faible" est un arbre trivial à deux feuilles *(stump*), *AdaBoost* fait mieux qu'un arbre sophistiqué pour un volume de calcul comparable : autant de feuilles dans l'arbre que d'itérations dans *AdaBoos*t. Hastie et al. [24] discutent la meilleure stratégie d'élagage applicable à chaque modèle de base.

Version aléatoire

A la suite de Freund & Schapire [20], Breiman [5] développe aussi, sous le nom d'*Arcing* (*adaptively resample and combine*), une version aléatoire, et en pratique très proche, du boosting. Elle s'adapte à des classifieurs pour lesquels il est difficile voire impossible d'intégrer une pondération des observations dans l'estimation. Ainsi plutôt que de jouer sur les pondérations, à chaque itération, un nouvel échantillon est tiré avec remise, comme pour le bootstrap, mais selon des probabilités inversement proportionnelles à la qualité d'ajustement de l'itération précédente. La présence des observations difficiles à ajuster est ainsi renforcée pour que le modèle y consacre plus d'attention. L'algorithme *adaboost* précédent est facile à adapter en ce sens en regardant celui développé ci-dessous pour la régression et qui adopte ce point de vue.

Régression et boosting

Friedman [21] propose sous l'acronyme MART *(multiple additive regression tree*s) un algorithme basé sur des arbres de régression (cf. Hastie et al. [24]) pour traiter le cas quantitatif en supposant la fonction perte seulement différentiable. Le principe de base est le même que pour *Adaboos*t, construire une séquence de modèles de sorte qu'à chaque étape, chaque modèle ajouté à la combinaison, apparaisse comme un pas vers une meilleure solution. Ce pas est franchi dans la direction du gradient, approché par un arbre de régression, de la fonction perte.

7.1.4.1 Implémentation dans SMURFER

Pour construire le modèle BOOSTING, on utilise la fonction « gbm.fit » contenue dans le package « gbm » de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « library(gbm) » puis « help(gbm) » dans *R*. Cette fonction met en œuvre l'algorithme de Friedman décrit ci-dessus. On présente ci-dessous les différentes options liées au modèle BOOSTING et disponibles dans la fonction SMURFER.R.

L'option n.trees permet de fixer le nombre d'arbres à ajuster (par défaut n.trees = 25000). L'option interaction.depth donne la profondeur maximale des interactions entre variables. Intercation.depth = 1 correspond à un modèle additif, interaction.depth = 2 (valeur par défaut) implique un modèle avec des interactions d'ordre deux, ... La distribution des résidus est, quant à elle, fixée au type gaussien.

7.1.5 Modèles FORETS ALEATOIRES (« Random Forests »)

Dans les cas spécifiques des modèles CART (arbres binaires), Breiman [5] propose une amélioration d'une autre méthode, le bagging, par l'ajout d'une randomisation. L'objectif est de rendre plus indépendants les arbres de l'agrégation en ajoutant du hasard dans le choix des variables qui interviennent dans les modèles. Cette approche semble plus particulièrement fructueuse dans des situations hautement multidimensionnelles, c'est-à-dire lorsque le nombre de variables explicatives *p* est très important. C'est le cas lorsqu'il s'agit, par exemple, de discriminer des courbes, spectres, signaux, biopuces (Besse [2]).

7.1.5.1 Aspects théoriques

ALGORITHME : **Forêts aléatoires**
• Soit x_0 à prévoir et
• $z = \{ (x_1, y_1), \ldots, (x_n, y_n) \}$ un échantillon
• Pour $b = 1$ à B, Faire :
– Tirer un échantillon bootstrap z^*_b
– Estimer un arbre sur cet échantillon avec randomisation des variables selon l'une des deux options :
  i. Si le nombre de variables explicatives est important, la recherche de chaque nœud optimal est précédé d'un tirage aléatoire d'un sous-ensemble de *q* prédicteurs.
  ii. Sinon, tirer au hasard $q_1 \approx 3$ variables explicatives puis construire q_2 "prédicteurs" par combinaisons linéaires avec des coefficients obtenus par tirages aléatoires uniformes sur [0,1].
• Fin Pour
Calculer l'estimation moyenne :

$$\hat{\phi}_{\hat{B}} = \frac{1}{B}\sum_{b=1}^{B} \hat{\phi}_{Zb}(x_0)$$

ou le résultat du vote.
FIN ALGORITHME

La stratégie d'élagage peut, dans le cas des forêts aléatoires, se limiter à des arbres de taille *q* relativement réduite voire même triviale avec $q = 2$ (*stump*). La sélection aléatoire d'un nombre réduit de prédicteurs potentiels à chaque étape de construction d'un arbre, accroît significativement la variabilité en mettant en avant nécessairement d'autres variables. Chaque modèle de base est évidemment moins performant mais, l'union faisant la force, l'agrégation conduit finalement à de bons résultats. Le nombre de variables tirées aléatoirement n'est pas un paramètre sensible ; un choix par défaut de $q = \sqrt{p}$ est suggéré par Breiman [5]. L'évaluation itérative de l'erreur « out-of-bag » prévient d'un éventuel sur-ajustement si celle-ci vient à se dégrader.

L'interprétation est ensuite facilitée par le calcul et la représentation graphique d'un indice proportionnel à l'importance de chaque variable dans l'agrégation de modèles et donc de sa participation à la régression ou à la discrimination. C'est évidemment d'autant plus utile que les variables sont très nombreuses. Plusieurs critères sont proposés par Breiman [5] pour évaluer l'importance de la $j^{ème}$ variable. Ils reposent sur une permutation aléatoire des valeurs de cette variable. L'un de ces critères consiste à calculer la moyenne sur toutes les observations de la décroissance de leur marge lorsque la variable est aléatoirement perturbée. La marge est ici la proportion de votes pour la vraie classe d'une observation moins le maximum des proportions des votes pour les autres classes.

7.1.5.2 Implémentation dans SMURFER

Pour construire le modèle FORETS ALEATOIRES, on utilise la fonction « randomForest » contenue dans le package « randomForest » de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « library(randomForest) » puis « help(randomForest) » dans *R*. L'option liée au modèle FORETS ALEATOIRES et disponible dans la fonction SMURFER.R concerne l'option ntree qui permet de fixer le nombre d'arbres à ajuster (par défaut ntree = 2000).

7.1.6 Modèles RESEAUX de NEURONES

7.1.6.1 Aspects théoriques

Les réseaux de neurones (Dreyfus et al. [13]) procurent un outil flexible pour généraliser les fonctions de régression linéaire. Ce sont des modèles de régression non linéaires (en les paramètres), avec autant de paramètres souhaités, de telle sorte qu'ils sont capables d'approximer n'importe quelle fonction lisse. Les métamodèles par réseau de neurones ont souvent été utilisées au CEA/DEN grâce à l'outil NeMo (Martinez & Arnaud [45]) basé sur la librairie SNNS de l'Université de Stutgart.

Un réseau de neurones peut être vu comme un réseau de couches possédant chacune des nœuds (ou neurones) : une couche en entrée (les nœuds sont les entrées), des couches cachées reliant les nœuds des différentes couches entre eux, et une couche en sortie (les nœuds sont les sorties). Par exemple, une sortie *y* d'un réseau de neurones avec une couche cachée s'exprime de la manière suivante :

$$y = f_0\left(a_0 + \sum_h w_h f_h\left(a_h + \sum_i w_{ih} x_i\right)\right)$$

Les entrées x_i sont distribuées sur les noeuds d'une couche cachée. Chaque nœud *h* pondère avec w_{ih} ses entrées puis les somme. La constante a_h est ajoutée au résultat, puis une fonction d'activation f_h lui est appliquée. Les nœuds de la couche cachée sont ensuite distribués sur le nœud en sortie qui opère la même opération. Les fonctions d'activation f_h peuvent être des fonctions sigmoïdes, logistiques, radiales, … Les poids sont choisis en minimisant un critère d'ajustement, par exemple les moindres carrés, mais d'autres mesures ont été proposées.

Le nombre de couches et le nombre de nœuds par couche constituent donc la complexité d'un réseau de neurones. Il existe un important risque de surapprentissage quand le nombre de poids à ajuster devient trop élevé : le réseau tendra à donner des réponses correctes sur la base d'apprentissage, mais mauvaises sur une base de test. Une solution est de trouver le nombre de cycles d'apprentissage optimal, ainsi que le nombre de nœuds optimal qui permettent d'avoir le meilleur compromis biais-variance afin de minimiser l'erreur de généralisation. Une autre solution est de pénaliser le critère des moindres carrés avec un terme de régularisation.

7.1.6.2 Implémentation dans SMURFER

Pour construire le modèle RESEAU de NEURONES, on utilise la fonction « nnet » contenue dans le package « nnet » de *R*. Pour plus de détails sur cette fonction de *R*, il suffit de taper « library(nnet) » puis « help(nnet) » dans *R*. Cette fonction n'autorise qu'une seule couche cachée dans le réseau de neurones, ce qui est suffisant dans la plupart des cas. On décrit ci-dessous les différentes options liées au modèle RESEAU de NEURONES et disponibles dans la fonction SMURFER.R.

L'option maxitRN donne le nombre maximal d'itérations pour l'apprentissage du réseau de neurones (par défaut maxitRN = 1000). L'option nunits donne le nombre de nœuds dans la couche cachée (par défaut nunits = 5).

7.1.7 Modèles Additifs Généralisés (GAM)

Les modèles additifs sont une extension des modèles linéaires classiques. Le modèle est supposé être une somme de termes (les effets additifs) qui sont estimés de manière non paramétrique (Hastie & Tibshirani [23]). Dans SMURFER, on utilise le package « mgcv » qui considère des splines pénalisées pour les termes additifs des GAMs.

7.1.8 Modèles Processus Gaussiens par la librairie mlegp (PG)

Le modèle des processus gaussiens est à présent bien connu en modélisation des expériences numériques (Santner et al. [57][16], Kleijnen [38]). Il est basé sur la même formulation que le krigeage en géostatistique. Il est défini par une composante régression à laquelle on additionne un processus stochastique gaussien caractérisé par sa fonction de covariance. En grande dimension, la difficulté est d'estimer correctement les paramètres de cette fonction de covariance (souvent appelé hyperparamètres). Pour la modélisation des expériences numériques, le modèle des processus gaussiens présente l'avantage d'être un interpolateur exact, ce qui est important car les codes de calcul étudiés sont la plupart du temps déterministes. D'autre part, son prédicteur est rapide à évaluer, il offre une formulation analytique interprétable et une estimation de son erreur en chaque nouveau point de prédiction.

On laisse le lecteur se référer à la bibliographie pour plus de détails sur ce métamodèle. Marrel et al. [43] ont souligné la difficulté de ce métamodèle pour les problèmes complexes de grande dimension (plusieurs dizaines de paramètres d'entrée). Ils ont proposé une solution par approche séquentielle de l'estimation des hyperparamètres.

Dans SMURFER, la méthode n°8 correspond au processus gaussien créé à l'aide du package « mlegp » . Celui-ci possède des méthodes d'optimisation relativement efficaces (simplex et BFGS) mais est limité à des processus gaussiens à covariance gaussienne. Il est possible également d'y insérer une partie régression linéaire en plus de la partie processus gaussien. Ce package offre également un certain nombre de fonctions d'analyse et de diagnostics très utiles. Ce package est relativement récent et nous espérons qu'il s'améliorera rapidement.

7.1.9 Modèles Processus Gaussiens par la librairie DiceKriging (PGdice)

Dans SMURFER, la méthode n° 9 correspond au processus gaussien créé à l'aide du package « DiceKriging ». Celui-ci possède des méthodes d'optimisation très efficaces (BFGS et algorithme génétique issu du package « rgenoud »). D'autre part, il utilise la covariance exponentielle généralisée qui offre toute une gamme de variation de la régularité du processus gaussien grâce au paramètre puissance (Marrel et al. [43]). La partie déterministe est une formule de régression quelconque : la méthode PGdice utilise la même formule polynomiale que dans la méthode GLM. Les options liées au modèle PGdice et disponibles dans la fonction SMURFER.R sont les mêmes que celles de la méthode GLM. Si l'algorithme « stepwise » a été appliqué lors de la construction du modèle GLM, la formule optimale obtenue pour le modèle GLM sera utilisée pour le modèle PGdice.

7.1.10 Quelques conseils heuristiques

Le modèle GLM est à tester en priorité, avec tout d'abord le modèle le plus simple : le modèle linéaire gaussien classique. On applique la formule de degré un avec tous les paramètres d'entrée, puis on augmente le degré du polynôme, en y ajoutant également des interactions. On cherche ensuite à simplifier la formule par sélection automatique de termes. Une analyse des résidus, non implémentée dans la fonction SMURFER.R, peut aussi permettre de voir s'il n'est pas préférable d'utiliser d'autres types de distributions et de fonctions lien.

Le modèle GAM est une extension naturelle du GLM et peut s'adapter à de fortes non linéarités. Il possède les mêmes outils d'analyse de variance que le GLM et permet une visualisation claire des termes dans la régression.

Le modèle PLS est à préférer au modèle GLM s'il y a de fortes colinéarités entre les paramètres d'entrée ou si le nombre de données est insuffisant par rapport au nombre de coefficients à ajuster dans la formule de régression.

SMURFER donne un outil pour explorer le modèle SVM car les trois paramètres de ce modèle sont difficiles à ajuster. Cependant, même avec cet outil, il faut choisir les bornes de variation des paramètres, ce qui est délicat. Le modèle SVM est donc très puissant, mais également complexe à ajuster.

En revanche, le modèle BOOSTING ne nécessite pas de réglages compliqués. C'est donc une méthode très rapide à mettre en œuvre. Sur nos cas les plus complexes, à nombre de paramètres d'entrée assez élevé (plusieurs dizaines), c'est la méthode qui a donné les meilleurs résultats.

La méthode des FORETS ALEATOIRES est adaptée pour les cas hautement multidimensionnels (plusieurs centaines de paramètres d'entrée), mais nous n'avons pas encore rencontré de telles applications. Par contre, cette méthode s'est révélée insuffisante par rapport aux autres méthodes lors de comparaisons réalisées sur quelques applications. Cependant, elle présente un intérêt dans sa possibilité d'interprétation du modèle (non encore mis en œuvre dans SMURFER).

Le modèle RESEAU de NEURONES implémenté dans SMURFER (et dans *R*) n'offre qu'une couche cachée à ajuster. Il est vivement conseillé de standardiser les variables avant de les utiliser. Lors de son utilisation sur quelques-unes de nos applications, nous n'avons pas encore réussi à rendre ce modèle aussi efficace que les réseaux de neurones mis en œuvre dans d'autres logiciels (comme par exemple dans le logiciel NeMo, Martinez & Arnaud [45]).

Pour les modèles BOOSTING, FORETS ALEATOIRES et RESEAU de NEURONES, il n'y a pas de procédure de sélection automatique des paramètres à ajuster, car il n'y en a qu'un ou deux. En revanche, SMURFER fournit a l'écran les diagnostics de ces méthodes afin d'orienter les réglages manuels.

Il est conseillé d'utiliser le modèle PROCESSUS GAUSSIEN quand il n'y a pas plus de 1000 données dans la base d'apprentissage. Au-delà, l'inversion de la matrice de covariance devient extrêmement coûteuse, voire impossible sous *R*.

7.2 VALIDATION STATISTIQUE DU METAMODELE

Avant d'utiliser un métamodèle, il est nécessaire de le valider pour l'utilisation prévue. En général, on attend d'un métamodèle des qualités d'approximation et de prédiction. La qualité d'approximation est déterminée à partir d'une analyse statistique sur la base de construction, alors que la qualité de prédiction est estimée à partir de l'analyse de points n'appartenant pas à la base de construction.

SMURFER donne tout d'abord les valeurs de critères de qualité calculées à l'aide des observations ayant servi à construire le métamodèle. Cela permet d'estimer rapidement si ce métamodèle est une bonne approximation de la valeur exacte. Pour aller plus loin, une analyse graphique des résidus s'avère indispensable ; celle-ci n'a pas encore été mise en œuvre dans SMURFER mais peut être faite indépendamment, en R.

Dans SMURFER, il est possible de définir une base de test séparément de la base de construction. Cependant, si l'on veut une validation plus robuste du métamodèle, il est préférable d'utiliser les techniques de validation croisée et/ou de bootstrap. Cette section explicite tout d'abord les différents critères de validation définis dans SMURFER, puis explique les techniques de leave-one-out, de validation croisée et de bootstrap.

7.2.1 Les critères de qualité

On a défini onze critères de qualité (Pair & Iooss [48]) exposés ci-dessous. Dans toute la suite, on appelle y_i (*1 ≤ i ≤ n*) les *n* observations de la variable de sortie et y_i' (*1 ≤ i ≤ n*) les *n* valeurs prédites par le modèle correspondantes. On appelle $\bar{y}$ et σ_y la moyenne et l'écart type des y_i. On appelle $\bar{y}'$ et σ_y' la moyenne et l'écart type des y_i'.

7.2.1.1 Le R²

Le R^2 (coefficient de détermination) équivaut à l'erreur quadratique moyenne divisée par la variance de *y*. Il est l'un des critères les plus utilisés et est défini par (Saporta [56]) :

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - y_i')^2}{\sum_{i=1}^{n} (\bar{y} - y_i)^2}$$

Il représente donc la fraction de la variation par rapport à la moyenne expliquée par le modèle de régression, c'est-à-dire le pourcentage de réponses expliquées par le métamodèle. Plus R^2 est proche de 1, plus la proximité des données par rapport au modèle est bonne. Lorsque ce coefficient est calculé sur une base de test, il est souvent nommé coefficient de prédictivité et noté Q_2.

7.2.1.2 Le R² ajusté

Dans le cas de la régression multiple, le R^2 étant un estimateur biaisé, certains auteurs utilisent le R^2 ajusté (coefficient de détermination ajusté), qui est défini par (Saporta [56]) :

$$\hat{R}^2 = \frac{(n-1)R^2 - p}{n-p-1}$$

où p est le nombre de termes dans la régression. La statistique du R^2 ajusté correspond à la correction du R^2 par rapport au nombre de degrés de liberté du modèle ($p-1$). Elle est plus adaptée que celle du R^2 pour comparer des modèles comportant des nombres différents de variables explicatives.

7.2.1.3 Le MSE

Le MSE (Mean Square Error), qui est l'erreur quadratique moyenne, est défini par :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2$$

Le MSE est la moyenne des erreurs de prédiction au carré (correspond à la norme L_2). Plus il est faible, plus l'ajustement est bon. Le MSE n'est pas très représentatif lorsque les y_i diffèrent de plusieurs ordres de grandeur entre eux. C'est pourquoi on utilise alors les résidus relatifs en moyenne et en écart type (voir paragraphes 7.2.1.10 et 7.2.1.11).

7.2.1.4 Le NMSE

Le NMSE (*Normalized Mean Square Error*), 'erreur quadratique moyenne normalisée, est défini par :

$$NMSE = \frac{1}{n}\frac{\sum_{i=1}^{n}(y_i' - y_i)^2}{\bar{y}'\,\bar{y}}$$

Le NMSE est le MSE normalisé, c'est-à-dire divisé par le produit des moyennes des y_i et des y_i'. Il faut faire attention à cette statistique qu'il convient d'utiliser si $\bar{y}$ et $\bar{y}'$ ne sont pas proches de 0.

7.2.1.5 Le SMSE

Le SMSE (*Standardized Mean Square Error*), qui est l'erreur quadratique moyenne standardisée, est défini par :

$$SMSE = \frac{1}{n}\frac{\sum_{i=1}^{n}(y_i' - y_i)^2}{\sigma_y'\,\sigma_y}$$

Le SMSE est le MSE standardisé, c'est-à-dire divisé par le produit des écarts types des y_i et des y_i'. Cette statistique est intéressante si le phénomène que l'on ajuste est plutôt fluctuant (σ_y grand).

7.2.1.6 Le MAE

Le MAE (Mean Absolute Error) est défini par :

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i' - y_i |$$

Le MAE est donc la moyenne des valeurs absolues des résidus (correspond à la norme L_1). Par rapport au MSE, il est plus robuste par rapport à quelques valeurs élevées de résidus.

7.2.1.7 Le biais

Le biais est défini par :

$$B = \bar{y} - \bar{y}'$$

Il permet de voir si le métamodèle n'est pas biaisé, signe d'un problème dans la régression.

7.2.1.8 Le biais normalisé

Le biais normalisé est défini par :

$$BN = \frac{\bar{y} - \bar{y}'}{\bar{y}}$$

Il permet de voir si il y a un biais faible ou important par rapport aux valeurs des observations.

7.2.1.9 Le résidu maximal

Le résidu maximal est défini par :

$$RM = \max_i (abs(y_i - y_i'))$$

Il correspond à la norme L^{∞} et permet de vérifier si toutes les observations ont été correctement prédites.

7.2.1.10 Le résidu relatif en moyenne

Le résidu relatif en moyenne est défini par :

$$RRM = \frac{1}{n}\sum_{i=1}^{n} \frac{(y_i - y_i')}{y_i}$$

Il s'agit en fait du biais des résidus relatifs.

7.2.1.11 Le résidu relatif en écart type

Le résidu relatif en écart type est défini par :

$$RRE = \sqrt{Var((y - y')/y)}$$

Ces deux derniers critères ont un intérêt quand la variable y prend des valeurs sur plusieurs ordres de grandeur. Dans ce cas-là, ces statistiques permettent d'homogénéiser les résidus et de constater si y' a été correctement ajusté sur tout son domaine de variation. Plus le RRM et le RRE sont faibles, plus l'ajustement est bon.

7.2.2 L'analyse et la visualisation des résidus

Après une régression, il est indispensable d'analyser finement les résidus de celle-ci pour détecter d'éventuels problèmes (biais, corrélation des résidus, hétéroscédasticité, données aberrantes, données très influentes par rapport à d'autres, …). Il n'y a pas d'outils automatiques pour réaliser cette étape dans SMURFER. Néanmoins, la plupart des objets construits par régression admette les fonctions *R* «summary() » et plot() » :

- *summary(modeleGLM) ; plot(modeleGLM)*

- *summary(modelePLS) ; plot(modelePLS)*
- *summary(modeleSVM)*
- *summary(modeleBOOSTING) ; plot(modeleBOOSTING)*
- *summary(modeleRF) ; plot(modeleRF)*
- *summary(modeleRN) ; plot(modeleRN)*
- *summary(modeleGAM) ; plot(modeleGAM)*
- *summary(modelePG) ; plot(modelePG)*
- *summary(modelePGdice) ; x11() ; plot.km(modelePGdice)*

Pour connaître exactement ce que produisent ces graphes, la fonction « help() » est indispensable (par exemple *help(plot.randomForest)*. Par ailleurs, la fonction « names() » permet d'obtenir tous les attributs de l'objet *modele*.

7.2.3 Le leave-one-out

Cette technique, également appelée jacknife (Hastie et al. [24]), consiste à estimer les qualités en prédiction d'un modèle en enlevant un point de la base d'apprentissage, à construire le modèle sur cette nouvelle base, puis à comparer la prédiction en ce point avec sa vraie valeur. En répétant cette procédure pour tous les points, on obtient au final *n* erreurs de prédiction (où *n* est le nombre de points). On appelle PRESS la somme des carrés des résidus dont la moyenne correspond au MSE en prédiction. Le coefficient de prédictivité, équivalent du R^2 en prédiction, est noté quant à lui Q_2.

Grâce à cette technique de leave-one-out, SMURFER donne ainsi une valeur pour tous les critères de qualité en prédiction. Par ailleurs, pour chaque échantillon d'apprentissage (il y en a *n*), les critères sont également calculés. SMURFER donne donc également la moyenne, l'écart-type, le minimum et le maximum de chaque critère de qualité en apprentissage.

Cette méthode devient vite coûteuse quand *n* augmente car *n* modèles doivent être construits. D'autre part, si *n* est faible, on n'a aucune garantie que les critères en prédiction soient corrects, et aucune incertitude n'y est associée.

7.2.4 La validation croisée

La validation croisée est une méthode permettant d'évaluer les qualités de prédiction du modèle avec l'estimation de leur précision. Elle est conseillée quand on a plus de 100 observations. L'idée est d'itérer l'estimation de l'erreur sur plusieurs échantillons de validation puis d'en calculer la moyenne (Besse [2]).

SMURFER divise donc aléatoirement les observations en *N* parties à peu près égales (en général, on prend *N = 10*). Il forme ensuite un échantillon de test avec une des parties et un échantillon d'apprentissage avec les *N–1* parties restantes. SMURFER calcule le modèle à l'aide de l'échantillon d'apprentissage. A partir de ce modèle et des valeurs d'entrée des observations de l'échantillon d'apprentissage, il prédit les valeurs de sortie correspondantes. Il calcule ensuite les critères de qualité à partir des valeurs réelles observées et des valeurs prédites de l'échantillon d'apprentissage (R^2, MSE, …). Il effectue ensuite la même opération avec l'échantillon de test : à partir de ce modèle et des valeurs d'entrée des observations de l'échantillon de test, il prédit les valeurs de sortie correspondantes. Il calcule ensuite les critères de qualité à partir des valeurs réelles observées et des valeurs prédites de l'échantillon de test (Q_2, MSE, …).

On effectue cette opération pour chacune des *N* parties et on fait la moyenne des *N* résultats obtenus pour les critères de qualité. On affiche également l'écart-type, le minimum et le maximum de la série de *N* résultats pour chacun des critères de qualité. Cela permet de savoir si les résultats sont dispersés suivant les échantillons que l'on prend.

Remarque (choix des paramètres de la validation croisée) : Si l'échantillon de test ainsi formé contient zéro ou une seule valeur, cet échantillon est supprimé. De plus, il peut arriver que l'échantillon d'apprentissage contienne trop peu de valeurs pour pouvoir calculer tous les coefficients du modèle. Dans ce cas, cet échantillon est supprimé. Il peut arriver ainsi que tous les échantillons soient supprimés. Dans ce cas, on ne peut pas effectuer la validation croisée. Il faut alors changer la valeur de N ou diminuer le nombre de paramètres d'entrée du modèle de régression.

7.2.5 Le bootstrap

La deuxième méthode utilisée pour évaluer les qualités de prédiction du modèle, avec l'estimation de leur précision, est le bootstrap (Efron & Tibshirani [15]). Il peut être utilisé quand on a des échantillons plus petits car il ne nécessite pas de couper les observations en *N* parties. Il nécessite par contre beaucoup plus de calculs que la validation croisée mais présente l'avantage de fournir des résultats plus complets. En effet, il ne permet pas seulement d'évaluer les critères de qualité de prédiction du modèle, mais donne aussi des indications sur les coefficients du modèle.

L'idée du bootstrap est d'approcher par simulation la distribution d'un estimateur lorsqu'on ne connaît pas la loi de l'échantillon ou, plus souvent, lorsqu'on ne peut pas supposer qu'elle est gaussienne (Besse [2]). Le principe fondamental de cette technique de reéchantillonnage est de substituer, à la distribution de probabilité inconnue *F*, dont est issu l'échantillon d'apprentissage, la distribution empirique F_n qui donne un poids *1/n* à chaque réalisation. Ainsi on obtient un échantillon de taille *n* dit é*chantillon bootstrap* selon la distribution empirique F_n par *n* tirages aléatoires avec remise parmi les *n* observations initiales. Il faut noter cependant que le bootstrap est une méthode généralement valide pour calculer des statistiques de type moyenne. En revanche, il reste encore à démontrer sa validité pour le calcul de quantiles. De plus, il faut garder en tête que souvent un estimateur bootstrap est biaisé.

SMURFER forme donc un échantillon bootstrap à partir des *n* observations de départ en effectuant *n* tirages avec remise. Ce nouvel échantillon forme un échantillon d'apprentissage (appelé aussi réplique) tandis que les observations qui n'ont pas été utilisées, du fait du tirage avec remise, forment un échantillon de test. On génère ainsi un grand nombre de répliques (50 par défaut dans SMURFER).

Pour le modèle GLM, on utilise le bootstrap lors de l'estimation des coefficients de la régression. On calcule ces coefficients pour chacun des échantillons d'apprentissage, puis on calcule la moyenne, l'écart type et les quantiles d'ordre 0.05 et 0.95 de chaque coefficient. Le calcul des estimations, des écarts type et des intervalles de confiance à 90 % n'est pas redondant avec les résultats du programme principal. En effet, les résultats du programme principal ne sont valables que lorsque les termes d'erreur respectent un certain nombre d'hypothèses de départ. Ces hypothèses ne sont pas forcément respectées dans le jeu de données que l'on utilise. Le bootstrap permet donc d'évaluer l'écart type et les intervalles de confiance des coefficients quand on n'est pas sûr de la validité des hypothèses de départ sur les termes d'erreur.

De même, lors de la construction d'un métamodèle, pour calculer l'estimation, l'écart type et l'intervalle de confiance à 90 % des différents critères de qualité, on procède de la manière suivante :

- Pour chaque réplique, on reprend le modèle calculé précédemment à l'aide de l'échantillon d'apprentissage.
- On considère alors les observations de l'échantillon de test et on utilise le modèle pour prédire les valeurs de la variable de sortie.
- A partir des valeurs observées et des valeurs prédites, on calcule les critères de qualité.
- On effectue cette opération pour chaque réplique puis on calcule la moyenne, l'écart type et les quantiles d'ordre 0.05 et 0.95 de la série de résultats obtenue.

Par défaut, SMURFER effectue 50 répliques. C'est en général suffisant pour obtenir une bonne estimation de la moyenne et de l'écart type d'un estimateur. Cependant, si on veut obtenir une bonne estimation de l'intervalle de confiance à 90 % de l'estimateur, il vaut mieux effectuer 500 répliques. Cela a bien sûr l'inconvénient d'allonger le temps de calcul.

Si beaucoup d'observations sont répétées dans l'échantillon d'apprentissage, il arrive qu'on ne puisse pas calculer tous les coefficients de la régression. Dans ce cas, l'échantillon est éliminé.

Il se peut également qu'un des coefficients de la régression ou un des critères de qualité ne soit pas disponible (NA : Not Available) ou soit infini. Par exemple, dans le cas où l'échantillon de départ est très petit, il peut arriver que l'échantillon de test ne contienne qu'une seule valeur. On obtiendra donc un R^2 infini du fait de la division par 0. Cette valeur est complètement faussée et, dans ce cas, l'échantillon sera éliminé.

Enfin, quand l'échantillon de départ est petit, certains échantillons d'apprentissage peuvent ne pas être représentatifs de l'échantillon de départ et donner un modèle très mauvais. Les valeurs prédites seront donc très différentes des valeurs observées. On souhaite donc ne pas tenir compte de ces échantillons. On a donc la possibilité, avec l'option elimination = TRUE, d'enlever les échantillons pour lesquels $Q_2 < 0$. En

effet, il est possible de trouver un Q_2 négatif car la relation $Q_2 \geq 0$ n'est vraie que dans le cas où on a utilisé les mêmes observations pour déterminer le modèle et pour calculer les valeurs prédites. Or, dans le cas du bootstrap (et des autres méthodes), on n'utilise pas le même échantillon d'observations pour déterminer le modèle et pour calculer le Q_2 à partir des valeurs prédites.

7.3 LE PROGRAMME SMURFER.R

7.3.1 Synopsis

La fonction utilisée peut prendre une grande quantité d'arguments en entrée. Ceux-ci vont être explicités dans ce paragraphe. Les valeurs par défaut de ces arguments sont données dans la définition de la fonction ci-dessous :

```
# SMURFER Version 1.0 - 10/01/2010

smurfer<-function(x,y,
        methodes=c(rep(TRUE,9)),
        logx=rep(FALSE,dim(x)[2]),logy=FALSE,affiche_file=TRUE,
        affiche=TRUE,loo=FALSE,vc=FALSE,N=10,bootstrap=FALSE,B=50,
        elimination=FALSE,xtest=NULL,ytest=NULL,sobol=FALSE,S=10000,
        R=1,Sij=TRUE,LHS=FALSE,lois=rep(0,dim(x)[2]),
        paramlois=array(0,dim=c(4,dim(x)[2])),
        tronq=rep(F,dim(x)[2]),paramtronq=array(0,dim=c(2,dim(x)[2])),
        cmin=1,cmax=101,cpas=50,epsmin=0.01,epsmax=0.51,epspas=0.25,
        gmin=0.5,gmax=10.5,gpas=5,SVMexplore=TRUE,critere="R2",
        n.trees=25000,interaction.depth=2,ntree=2000,
        maxitRN=1000,nunits=5,
        min.nugget=0,simplex.ntries=5,constantPG=0,
        nugget.estim=T,optim.method="BFGS",
        forme="creationformule",degreformule=1,famille=gaussian,
        stepwise=FALSE,penaltyAIC=2,trace=0,iteration=FALSE,niter=1,
        intercept=TRUE,
        tabA1=rep("no",dim(x)[2]),tabA2=rep("no",dim(x)[2]),
        tabA3=rep("no",dim(x)[2]),tabA4=rep("no",dim(x)[2]),
        tabB1=array("no",dim=c(dim(x)[2],dim(x)[2])),
        tabC1=array("no",dim=c(dim(x)[2],dim(x)[2],dim(x)[2])),
        tabC2=array("no",dim=c(dim(x)[2],dim(x)[2])),
        tabD1=array("no",dim=rep(dim(x)[2],4)),
        tabD2=array("no",dim=c(dim(x)[2],dim(x)[2])),
        tabD3=array("no",dim=c(dim(x)[2],dim(x)[2],dim(x)[2])),
        tabD4=array("no",dim=c(dim(x)[2],dim(x)[2])),
        tabAgam=rep("no",dim(x)[2]),
        tabBgam=array("no",dim=c(dim(x)[2],dim(x)[2])),
        tabCgam=array("no",dim=c(dim(x)[2],dim(x)[2],dim(x)[2])),
        tabDgam=array("no",dim=rep(dim(x)[2],4))
                  ){

#-------------------------------------------------------------------
#Fonction applicant diverses mehodes de regression sur un jeu de    |
#donnees afin de creer une surface de reponse. Les methodes sont :  |
#  GLM (forme polynomiale jusqu'au 4eme degre), regression PLS, SVM,|
#  boosting, forets aleatoires, reseaux de neurones (RN), GAM, PG   |
#Les performances sont evaluees par leave-one-out, validation       |
#croisee et/ou bootstrap                                            |
#                                                                   |
#Les indices de sensibilite (Sobol) peuvent etre calcules a l'aide  |
#des surfaces de reponse.                                           |
#                                                                   |
#-------------------------------------------------------------------|
```

7.3.2 Description des arguments en entrée

Voici la liste des arguments en entrée de la fonction telle qu'elle se présente dans SMURFER.R :

```
#-------------------------------------------------------------------|
#Arguments                                                          |
#obligatoires : x : matrice des variables d'entree                  |
#               y : vecteur de la variable de sortie                |
#                                                                   |
#Arguments facultatifs :                                            |
#      methodes: Tableau de booleens indiquant les methodes a       |
#        utiliser ou pas : GLM, PLS, SVM, BOOSTING, RF, RN, GAM, PG |
#                          PGdice                                   |
#      logx : si logx=c(TRUE,TRUE,...,TRUE), le programme           |
#        transformera les variables d'entree Vi en log(Vi)          |
#      logy : si logy=TRUE, le programme transformera la variable   |
#        de sortie y en log(y)                                      |
#      affiche_file : si TRUE, affiche automatiquement les fichiers |
#        de sortie "res-SURFER.out" et res-SOBOL.out" a l'ecran     |
#      affiche : si TRUE, cree les graphes de la validation croisee |
#        et du bootstrap                                            |
#        Sous Unix, gv est lance pour les afficher automatiquement  |
#        Sous Windows, l'affichage n'est pas automatique            |
#      loo : si loo=TRUE, le leave-one-out sera applique pour       |
#        evaluer les criteres de qualite de la surface de reponse   |
#      vc : si vc=FALSE, le programme n'effectuera pas la           |
#        validation croisee                                         |
#      N : nombre de repetitions de la validation croisee           |
#        si N est négatif, on intervertit les rôles des bases de    |
#        test et d'apprentissage (en conservant |N| intervalles)    |
#      bootstrap : si bootstrap=FALSE, le programme n'effectuera    |
#        pas le bootstrap                                           |
#      B : le nombre de repetitions du bootstrap                    |
#      elimination : si TRUE, permet d'eliminer dans le bootstrap   |
#        les echantillons pour lesquels R2 < 0                      |
#      xtest : matrice des variables d'entree de la base de test    |
#      ytest : vecteur de la variable de sortie de la base de test  |
#      sobol : si TRUE, calcul des indices de Sobol                 |
#      S : nombres de simulations effectuees pour calculer les      |
#        indices de sensibilite de Sobol                            |
#      R : nombre de repetitions du calcul des indices de Sobol     |
#      Sij : TRUE pour calculer les indices d'ordre 2, FALSE sinon  |
#      LHS : TRUE pour utiliser un echantillonnage LHS lors du      |
#        calcul des indices de Sobol                                |
#        Par defaut : FALSE pour un echantillonnage simple          |
#      lois : vecteur contenant les types de distribution de proba  |
#        pour chaque entree                                         |
#        0=uniforme ; 1=normale ; 2=lognormale ; 3=weibull          |
#        4=exponentielle ; 5=beta ; 6=triangulaire ; 7=trapezoidale |
#        10=gumbel                                                  |
#        Par defaut, on prend la loi uniforme                       |
#      paramlois : tableau avec les parametres de chaque loi (max=4)|
#        pour chaque entree (range par colonne) :                   |
#        (min,max,0,0) pour uniforme (par défaut : min=0, max=1)    |
#        (moy,ecart-type,0,0) pour normale,                         |
#        (moy du log, ecart-type du log,0,0) pour lognormale,       |
#        (forme,echelle,0,0) pour Weibull,                          |
#        (lambda,0,0,0) pour exponentielle,                         |
#        (shape1, shape2,0,0) pour beta,                            |
#        (min,mode,max,0) pour triangulaire,                        |
#        (min,mode1,mode2,max) pour trapezoidale,                   |
#        (mode,echelle,0,0) pour Gumbel                             |
```

```
#       Par defaut, on a la loi uniforme avec (min,max) calcules a |
#       partir des parametres d'entree x                           |
#     tronq : vecteur pour selectionner ou non une loi tronquee    |
#       TRUE pour loi tronquee, FALSE sinon (par défaut)           |
#     paramtronq : tableau avec les parametres de troncature de    |
#       chaque loi : (min,max) range par colonne                   |
#     cmin,cmax,cpas,epsmin,epsmax,epspas,gmin,gmax,gpas :         |
#       les plages d'exploration des SVM : cout, marge et portee   |
#     SVMexplore: Si FALSE, alors on n'explore pas les plages des  |
#       parametres. On prend les valeurs minimales                 |
#     critere: Critere pour la selection des parametres SVM:       |
#       R2,R2ajuste,MSE,NMSE,SMSE,MAE,B,BN,RM,RRM ou RRE           |
#     n.trees, interaction.depth : les parametres du boosting      |
#       (nombre d'arbres et ordre des interactions)                |
#     ntree : nombre d'arbres de la foret aleatoire                |
#     maxitRN : nombre max. d'iterations pour l'apprentissage du RN|
#     nunits : nombre de noeuds dans la couche cachee du RN        |
#     min.nugget : valeur minimum de la pepite pour PG (mlegp)     |
#     simplex.ntries : nb d'optim par simplex pour PG (mlegp)      |
#     constantPG : 0 (defaut) = partie regress lineaire PG (mlegp) |
#                  1 : partie deterministe constante pour PG       |
#     nugget.estim : T (defaut) = effet de pepite estime PGdice    |
#     optim.method : "BFGS" (defaut) : optimisation PGdice par BFGS|
#                    "gen" : algorithme genetique de rgenoud       |
#     forme : sert a choisir la facon dont est cree la formule de  |
#       regression pour les GLM, PLS, GAM, PGdice                  |
#       On peut choisir "creationformule" ou "maformule"           |
#     degreformule : le degre de la formule dans le cas ou on      |
#       choisit "creationformule"                                  |
#     famille : pour GLM et GAM le programme suppose que la        |
#       fonction de densite de y est de type "famille"             |
#       Par defaut, on suppose qu'elle est gaussienne de fonction  |
#        lien l'identite (famille=gaussian(link="identity"))       |
#     stepwise : pour GLM, si TRUE, permet de construire un modele |
#       optimal en eliminant certains termes. Dans ce cas, on ne   |
#       pourra calculer ni les contributions des effets, ni Sobol  |
#       Si stepwise est utilise pour GLM, on ne peut pas appliquer |
#       sobol pour PLS et PGdice (car meme formule)                |
#     penaltyAIC : multiple du nb de degre de liberte pour AIC dans|
#       le stepwise (par defaut = 2 ; si > 2 -> selection + dure   |
#     trace : si trace = 1, le programme affiche les etapes de     |
#       l'algorithme stepwise. Par defaut, trace = 0 et le         |
#       programme n'affiche rien                                   |
#     iteration : indique si on choisit le nombre d'iterations dans|
#       la regression. Si iteration = FALSE, le programme effectue |
#       les iterations jusqu'a ce qu'il y ait convergence          |
#     niter : indique le nombre maximal d'iterations a effectuer   |
#     intercept : si intercept=TRUE, il y a une constante dans la  |
#       formule de regression (GLM, GAM, PGdice) si on a choisi    |
#                                                  "maformule"     |
#     tabA1 : permet d'inclure les Vi dans "maformule"             |
#     tabA2 : permet d'inclure les Vi^2 dans "maformule"           |
#     tabA3 : permet d'inclure les Vi^3 dans "maformule"           |
#     tabA4 : permet d'inclure les Vi^4 dans "maformule"           |
#     tabB1 : permet d'inclure les ViVj dans "maformule"           |
#     tabC1 : permet d'inclure les ViVjVk dans "maformule"         |
#     tabC2 : permet d'inclure les Vi^2Vj dans "maformule"         |
#     tabD1 : permet d'inclure les ViVjVkVl dans "maformule"       |
#     tabD2 : permet d'inclure les Vi^3Vj dans "maformule"         |
#     tabD3 : permet d'inclure les Vi^2VjVk dans "maformule"       |
#     tabD4 : permet d'inclure les Vi^2Vj^2 dans "maformule"       |
#     tabAgam : pour GAM inclut les s(Vi) dans "maformule"         |
```

```
#      tabBgam : pour GAM inclut les s(Vi,Vj) dans "maformule"      |
#      tabCgam : pour GAM inclut les s(Vi,Vj,Vk) dans "maformule"   |
#      tabDgam : pour GAM inclut les s(Vi,Vj,Vk,Vl) dans "maformule"|
#                                                                   |
#Par defaut, la formule de regression utilisee par le programme     |
#        sera y ~ 1                                                 |
#-------------------------------------------------------------------|
```

Options obligatoires

Les seules options obligatoires de la fonction SMURFER.R sont les suivantes :

- x : la matrice des variables d'entrée,
- y : le vecteur de la variable de sortie.

Quand on utilise cette fonction, on doit toujours commencer par entrer d'abord la variable x puis la variable y.

Choix des métamodèles

Le choix des métamodèles à construire se fait par l'argument « methodes », qui est un vecteur de booléens. Pour utiliser un métamodèle, il faut mettre TRUE à sa position dans ce vecteur. Cette position vaut 1, 2, 3, 4 ou 5, valeur fixée d'après l'ordre des métamodèles :

1. GLM,
2. PLS,
3. SVM,
4. Boosting,
5. Forêts aléatoires,
6. Réseaux de neurones,
7. GAM,
8. Processus Gaussien (package mlegp),
9. Processus Gaussien (package DiceKriging).

Par exemple, si on veut que SMURFER ajuste un GLM et un modèle de boosting, il faudra assigner à l'option methodes le vecteur suivant :

methodes=c(TRUE,FALSE,FALSE,TRUE,FALSE,FALSE,FALSE,FALSE,FALSE)

Remarque : Il est possible de mettre T à la place de TRUE et F à la place de FALSE.

Options pour passer au logarithme

Il est possible de transformer en logarithme les *x* ou *y* avant d'ajuster les métamodèles :

- logx : vecteur de booléens pour indiquer quels sont les paramètres d'entrée à transformer. Par exemple, si on a trois variables, il faut taper logx = c (TRUE , TRUE , FALSE) pour transformer les variables X_1 et X_2 et laisser inchangée la variable X_3.
- logy : booléen permettant de passer la variable de sortie en logarithme.

Si l'une des variables à passer en logarithme possède des valeurs nulles ou négatives, cette variable n'est pas passée en logarithme et la fonction continue son exécution après un message d'avertissement.

Options pour estimer les critères de qualité

Le leave-one-out est mis en œuvre en passant l'option loo=TRUE.

La validation croisée est mise en œuvre en passant l'option vc=TRUE ; N permet de définir le nombre d'intervalles de la validation croisée (par défaut 10).

Le bootstrap est mis en œuvre en passant l'option bootstrap=TRUE ; B permet de définir le nombre d'échantillons bootstrap créés (par défaut 50). Si elimination=TRUE, on élimine dans le bootstrap les échantillons pour lesquels $R^2<0$.

Il est possible de rentrer dans la fonction une base de test, en passant en options la matrice xtest (paramètres d'entrée de la base de test) et ytest (réponses correspondantes). Les critères de qualité sont alors effectués sur cette base de test. Si l'option ytest est omise, les réponses du métamodèle à xtest sont mises dans des fichiers "resGLM-test.dat", "resPLS-test.dat", …, correspondant aux modèles appliqués.

Options d'affichage graphique

L'option affiche=TRUE permet d'avoir des graphes de comparaison entre les valeurs observées et les valeurs prédites par les modèles. S'il n'y a pas eu de validation croisée ni de bootstrap, il n'y a qu'un seul fichier produit, qui regroupe les graphes sur les bases de construction de chaque métamodèle construit. S'il y a eu une validation croisée ou un bootstrap, il y a un fichier par métamodèle, qui montre les graphes sur les bases de construction et les bases de test.

Options pour les indices de Sobol

Le calcul des indices de Sobol est facultatif. Si sobol=TRUE, il est activé et la fonction sobolS.R est exécutée (cf section 6.3.2). Les options disponibles dans SMURFER.R sont les suivantes :

- S : nombre de calculs pour l'estimation,
- logx, logy : si les données ont été passées au logarithme avant l'ajustement du métamodèle,
- modele, methode : pour définir quel type de modèle est utilisé,
- S : nombre de calculs pour l'estimation,
- R : nombre de répétitions du calcul des indices,
- Sij : booléen pour calculer les indices d'ordre deux,
- LHS : booléen pour évaluer les indices avec la méthode LHS,
- lois : vecteur contenant les types de distribution de probabilité pour chaque paramètre d'entrée,
- paramlois : tableau contenant les 4 paramètres de chaque loi pour chaque paramètre d'entrée.

Toutes les options facultatives peuvent être rentrées dans n'importe quel ordre. Il faut faire attention à bien respecter les majuscules et les minuscules.

7.3.3 Les sorties de la fonction

Voici la liste des sorties à l'issue de l'exécution de SMURFER.R :

```
#-------------------------------------------------------------------|
#Sortie : Pour chaque methode, la fonction renvoie les  parametres  |
#         des modeles obtenus et les criteres de qualite du modele :|
#         R2: coefficicient de determination                        |
#         R2ajuste : R2 non biaise (corrige du nombre de ddl)       |
#           mesure valide pour GLM, PLS et GAM                      |
#         MSE: Mean Square Error                                    |
#         NMSE: Normalized Mean Square Error                        |
#         SMSE: Standardized Mean Square Error                      |
#         MAE: Mean Absolute Error                                  |
#         B: biais,                                                 |
#         BN: biais normalise                                       |
#         RM: residu maximum                                        |
#         RRM: residu relatif en moyenne                            |
#         RRE: residu relatif en ecart-type                         |
#                                                                   |
#         Par leave-one-out, on a des criteres moyens de prediction |
#         et des statistiques pour les criteres d'apprentissage     |
#         La valeur du Q2 est mise dans une variable externe Q2     |
#         Par validation croisee, on a la moyenne, l'ecart-type et  |
#         les min et max des criteres de qualite obtenus pour       |
#         l'apprentissage et la prediction                          |
#         Par bootstrap, on a la moyenne, l'ecart-type et           |
#         l'intervalle de confiance a 90% des criteres de qualite   |
#         pour la prediction                                        |
#         La fct peut calculer les criteres sur une base de test    |
#         Si on n'a pas les resultats de la base de test, les       |
#         fichiers "resGLM-test.dat', "resPLS-test.dat", ...        |
#         contiennent les predictions des modeles sur la base test  |
#                                                                   |
#         La fonction trace egalement les valeurs predites en fct   |
```

```
#         des valeurs observees pour les echantillons de test et    |
#         d'apprentissage pour le leave-one-out, la validation      |
#         croisee et le bootstrap                                   |
#                                                                   |
#         Pour le modele GLM, on a en plus :                        |
#            - les coefficients de la regression : Estimation,      |
#              Ecart-type, t-value et Pr(>|t|)                      |
#            - la variance totale et son degre de liberte           |
#            - la variance residuelle et son degre de liberte       |
#            - les contributions des effets et des residus          |
#            - l'indice de conditionnement de la matrice de         |
#              correlation des effets                               |
#            - la moyenne, l'ecart-type et l'intervalle de confiance|
#              a 90% des coefficients de la regression par bootstrap|
#         Pour SVM, si on explore les plages des parametres SVM, la |
#         fct enregistre les criteres (r2m,msem,nmsem,smsem,ssdm,bn,|
#         bnm,rmm,mrrm,srrm) dans le fichier "res-exploreSVM.Rdata" |
#                                                                   |
#         Tous ces resultats sont dans un fichier "res-SURFER.out"  |
#         (fichier ecrase) dans le repertoire de travail            |
#         Les sorties graphiques se trouvent dans les fichiers      |
#         compar*.ps : compar.ps si pas de vc ni de bootstrap,      |
#           sinon compar1.ps, compar2.ps, ..., compar8.ps           |
#                                                                   |
#         De plus, les modeles sont renvoyes dans des objets R      |
#         modeleGLM, modelePLS, modeleSVM, modeleBOOSTING, modeleRF |
#         modeleRN, modeleGAM, modelePG, modelePGdice               |
#         Ceux-ci sont utilisables sous la meme session R           |
#         Avant de quitter la session R, les objets peuvent etre    |
#         sauvegardes dans un fichier a l'aide de la commande "save"|
#         Pour les recuperer dans une autre session, utiliser "load"|
#         Les parametres dim_x, dim1_x, dimnames_x, dimnames1_x     |
#         sont aussi conserves                                      |
#                                                                   |
#         Les resultats des calculs Sobol sont dans le fichier      |
#         "res-SOBOL.out" (fichier ecrase) du repertoire de travail |
#-------------------------------------------------------------------|
#       Auteur : B. Iooss et P-M. Pair                              |
#-------------------------------------------------------------------|
```

Il faut retenir que les fichiers « res-SURFER.out » et « res-SOBOL.out » contiennent les résultats de la construction des différents métamodèles et le calcul des indices de Sobol sur celles-ci.

Un autre élément important concerne la récupération des objets modeleGLM, modelePLS, modeleSVM, modeleBOOSTING, modeleRF, modeleRN, modeleGAM, modelePG, modelePGdice qui contiennent les métamodèles sous format *R* et que l'on peut réutiliser. Pour les effacer, il suffit de taper « rm(modeleGLM) » par exemple.

8. PERSPECTIVES

Ce rapport a présenté l'outil SMURFER V1.0 développé à partir du logiciel de statistiques *R*. Ce logiciel a la vocation d'être une boite à outils facile à utiliser et, dans laquelle on peut tester de nombreuses stratégies et intégrer de nouvelles méthodes rapidement.

En conclusion, nous allons lister quelques compléments qu'il conviendrait d'intégrer aux principales fonctionnalités de SMURFER :

- Concernant les métamodèles, plusieurs améliorations et extensions sont à réaliser :
  - mettre la fonction noyau en option de SMURFER.R pour les SVM,
  - inclure de nouveaux modèles, par exemple la « ridge regression », les méthodes polymars, les méthodes de régression non paramétriques (Hastie et al. [24], Storlie & Helton [62]), …
  - inclure le modèle GAM joint (Iooss & Ribatet [33]) qui permet de modéliser des codes de calcul non déterministes (un métamodèle pour la moyenne et un pour la dispersion). La fonction sobolS_joint.R permet alors d'estimer des indices de Sobol sur chacun des métamodèles.

  L'amélioration majeure concernant les métamodèles résidera dans la prise en compte de plusieurs variables de sortie simultanément, pour effectuer des régressions multivariées. Cette amélioration est cependant conditionnée à sa faisabilité dans les modèles de régression de *R*.

- Concernant l'analyse de sensibilité, il serait utile d'intégrer d'autres algorithmes pour l'estimation des indices de Sobol : méthodes quasi-Monte Carlo, RBD, Sobol amélioré (Saltelli et al. [54]), estimateur efficace (Da Veiga & Gamboa [9]).

- Concernant l'échantillonnage de l'espace des paramètres d'entrées, il serait intéressant d'autres plans également bien adaptés à la construction de métamodèle comme les techniques déterministes (quasi Monte-Carlo, suites à discrépance faible, …) décrites par exemple dans Feuillard [18] et Franco [19] (package « DiceDesign »).

- D'autre part, le package *R,* nommé « sensitivity », regroupe plusieurs outils d'analyse de sensibilité (méthodes de régression linéaire, de régression des rangs, de Morris, Sobol et FAST). Pour l'instant, SMURFER le reprend tel quel pour offrir les fonctionnalités FAST et Morris qu'il ne possède pas. Il serait intéressant d'intégrer ce package plus en profondeur dans SMURFER (en faisant appel à ses fonctions de la même manière qu'une fonction SMURFER).

- SMURFER n'est pas destiné à être utilisé pour faire du calcul intensif, mais il serait possible et utile de la paralléliser à moindre coût. Le langage *R* n'est pas adapté à la parallélisation, mais une surcouche en Python ou en LSF (langage de soumission de processus parallèles) est tout à fait envisageable.

- Finalement, une modernisation de l'architecture de SMURFER, dans un esprit de programmation objet et modulaire, serait la bienvenue. L'objectif à terme serait d'obtenir plusieurs modules :
  - un module d'analyse statistique,
  - un module de fiabilité,
  - un module d'échantillonnage,
  - un module d'analyse de sensibilité,
  - un module de métamodèles,
  - une fonction de haut niveau appelant les modules précédents avec prise en charge de l'utilisateur par une interface conviviale.

9. BIBLIOGRAPHIE

[1] D. Barthel, *Etude, mise en œuvre et développement de méthodes d'analyse de sensibilité basées sur des tests statistiques,* Rapport de stage 5ème année GMM-MMS de l'INSA Toulouse, 2008.

[2] P. Besse. *Data mining II. Modélisation statistique & Apprentissage*. Publication du Laboratoire de Statistique et Probabilités, Université Paul Sabatier, Toulouse, 2003. <http://www.lsp.ups-tlse.fr/Besse/enseignement.html>.

[3] P. Besse. *Pratique de la modélisation statistique*. Publication du Laboratoire de Statistique et Probabilités, Université Paul Sabatier, Toulouse, 2003. <http://www.lsp.ups-tlse.fr/Besse/enseignement.html>.

[4] G.E.P. Box and N.R. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, New York, 1987.

[5] L. Breiman. Random forests, *Machine learning*, 45, 5-32, 2001.

[6] D.G. Cacuci and M. Ionescu-Bujor. A comparative review of sensitivity and uncertainty analysis of large-scale systems. *Nuclear science and Engineering,* 147:204-217, 2004.

[7] W.J. Conover. *Practical non parametric statistics, Third edition,* Wiley, New-York, 1999.

[8] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20, 1-25, 1995.

[9] S. Da Veiga and F. Gamboa. Efficient estimation of nonlinear conditional functionals of a density, *The Annals of Statistics,* submitted, 2007.

[10] H. A. David and H. N. Nagaraja*. Order statistics*, Wiley, 2003.

[11] A. Dean and S. Lewis (eds), *Screening. Methods for experimentation in industry, drug discovery, and genetics,* Springer, 2006.

[12] E. De Rocquigny, N. Devictor and S. Tarantola (eds). *Uncertainty in industrial practice,* Wiley, in press, 2008.

[13] G. Dreyfus, J-M. Martinez, M. Samuelides, M.B. Gordon, F. Badran, S. Thiria and L. Hérault. *Réseaux de neurones - Méthodologie et applications.* Eyrolles, 2002.

[14] A. Ducellier. *Régressions polynomiales à plusieurs variables d'entrée (SURFER v0.5) et analyse de sensibilité : Manuel utilisateur et dossier de validation.* Note technique CEA DEN/GRE/DER/SSTH/LDAS/2005-004, 2005.

[15] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, 1993.

[16] K-T. Fang, R. Li and A. Sudjianto, *Design and modeling for computer experiments,* Chapman & Hall/CRC, 2006.

[17] J.J. Faraway. *Practical Regression and Anova using R*. 2002. <www.r-project.org>.

[18] V. Feuillard. *Analyse d'une base de données pour la calibration d'un code de calcul*, Thèse de l'Université Paris VI, 2007.

[19] J. Franco, *Planification d'expériences numériques en phase exploratoire pour la simulation des phénomènes complexes,* Thèse de l'Ecole Nationale Supérieure des Mines de Saint-Etienne, 2008.

[20] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the thirteenth Conf. Machine Learning*, 148-156, 1996.

[21] J. Friedman. Stochastic gradient boosting. *Comput. Statistics and Data Analysis,* 38(4),367-378, 2002.

[22] I. Guyon, B. Boser and V. Vapnik. *Automatic capacity tuning of very large VC-dimension classifiers*. Neural information processing systems 5, p. 147-155, Morgan Kaufman, S. Mateo, CA, 1993.

[23] T. Hastie and R. Tibshirani. *Generalized additive models.* Chapman & Hall/CRC, 1990.

[24] T. Hastie, R. Tibshirani and J. Friedman. *The elements of statistical learning*. Springer, 2001.

[25] J.C. Helton and F.J. Davis. Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering and System Safety,* 81:23-69, 2003.

[26] J.C. Helton, J.D. Johnson, C.J. Salaberry and C.B. Storlie, Survey of sampling-based methods for uncertainty and sensitivity analysis, *Reliability Engineering and System Safety*, 91:1175-1209, 2006.

[27] P. Heyraud, B. Iooss, A. Bouloré and C. De Bellis, *Gestion des incertitudes dans la base de données CRACO,* Note technique CEA DEN/CAD/DEC/SESC/LSC 05-040, 2006.

[28] T. Homma and A. Saltelli. Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering and System Safety*, 52 : 1-17, 1996.

[29] R. Ihaka and R. Gentleman. R : a language for data analysis and graphics. *Journal of Computational and Graphical Statistics,* 5 : 299-314, 1996.

[30] R.L. Iman and W.J. Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics,* B11(3):311-334, 1982.

[31] B. Iooss, *Manuel utilisateur du logiciel SMURFER V1.2 : programmes en R d'analyses d'incertitudes, de sensibilités, et de construction de surfaces de réponse,* Note Technique CEA/DEN/CAD/DER/SESI/LCFR/NT DO 6 08/03/06, 2006.

[32] B. Iooss and N. Pérot, *Quelques outils statistiques pour étudier la représentativité d'un échantillon de données de faible taille,* Note Technique CEA DEN/CAD/DER/SESI/LCFR/NT DO 13 20/09/07, 2007.
[33] B. Iooss and M. Ribatet, Global sensitivity analysis of computer models with functional inputs*, Reliability Engineering and System Safety*, 94:1194-1204, 2009.
[34] B. Iooss, F. Van Dorpe and N. Devictor, Response surfaces and sensitivity analysis for an environmental model of dose calculations, *Reliability Engineering and System Safety*, 91:1241-1251, 2006.
[35] J. Jacques. *Contributions à l'analyse de sensibilité et à l'analyse discriminante généralisée*, Thèse de l'Université Joseph Fourier - Grenoble 1, 2005.
[36] J. Jacques, C. Lavergne and N. Devictor. Sensitivity analysis in presence of model uncertainty and correlated inputs. *Reliability Engineering and System Safety*, 91:1105-1108, 2006.
[37] M. Kadiri. *Analyse critique de la formule de Wilks*, Technical report, Ligeron SA, 04-0265-A/R01/MK, 2004.
[38] J.P.C. Kleijnen, *Design and analysis of simulation experiments,* Springer, 2008.
[39] J. Klejjnen and R.G. Sargent. A methodology for fitting and validating metamodels in simulation. *European Journal of operational Research,* 120:14-29, 2000.
[40] D. Kurowica and R. Cooke. *Uncertainty analysis,* Wiley, 2006.
[41] P. Lemaître, *Développement et intégration d'algorithmes d'analyses d'incertitudes dans un logiciel en R,* Rapport de stage CEA/DEN/CAD/DER/SESI/LCFR DO 38 07/09/09, 2009.
[42] J.S. Liu, *Monte Carlo strategies in scientific computing.* Springer, 2001.
[43] A. Marrel, B. Iooss, F. Van Dorpe and E. Volkova. An efficient methodology for modelling complex computer codes with Gaussian processes. *Computational Statistics and Data Analysis,* 52:4731-4744, 2008.
[44] A. Marrel, B. Iooss, B. Laurent and O. Roustant. Calculations of Sobol indices for the Gaussian process metamodel. *Reliability Engineering and System Safety*, 94:741-752, 2009.
[45] J-M. Martinez and G. Arnaud. *NeMo V1.1, manuel d'utilisation*, Rapport technique CEA DEN/SAC/DM2S/SFME/LETR/RT/02-031/A, 2002.
[46] P. McCullagh and J.A. Nelder. *Generalized linear models.* Chapman and Hall, 2nd Edition, 1989.
[47] M.D. McKay, R.L. Iman and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics,* 21:239-245, 1979.
[48] P-M. Pair and B. Iooss. *Rapport de stage - Construction de surface de réponse non linéaires : étude comparative de nouvelles méthodes de régression*. Note technique CEA DEN/CAD/DER/SESI/LCFR/NT DO 21 17/06/04, 2004.
[49] E. Paradis. *R pour les débutants*. 2002. <www.r-project.org>.
[50] M. Petelet, B. Iooss, O. Asserin and A. Loredo. Latin hypercube sampling with inequality constraints, *submitted.* HAL: http://fr.arxiv.org/abs/0909.0329.
[51] G. Pujol and C. Cannamela. *Analyse de sensibilité du code de calcul ATLAS.* Note technique CEA DEN/CAD/DEC/SESC/LSC 05-034, 2005.
[52] J. Sacks, W. Welch, T. Mitchell and H. Wynn. Design and analysis of computer experiments. *Statistical science,* 4:409-435, 1989.
[53] A. Saltelli, K. Chan and E.M. Scott, editors. *Sensitivity analysis*. Wiley Series in Probability and Statistics, Wiley, 2000.
[54] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola, *Global sensitivity analysis, the primer,* Wiley, 2008
[55] A. Saltelli, S. Tarantola, F. Campolongo and M. Ratto. *Sensitivity analysis in practice: a guide to assessing scientific models.* Wiley, 2004.
[56] G. Saporta. *Probabilités, analyse de données et statistique*. Editions Technip, 2ème edition, 2006.
[57] T.J. Santner, B.J. Williams and W.I. Notz. *The design and analysis of computer experiments,* Springer, 2002.
[58] R. Schapire. The strength of weak learnability. *Machine Learning*, 5, 197-226, 1990.
[59] SIER (Société Internationale d'Etudes et de Réalisation Pour les Organismes Publics et Privés). *Informations et techniques,* 13, Juin 1998.
[60] I.M. Sobol, Sensitivity estimates for non linear mathematical models, *Mathematical Modelling and Computational Experiments*, 1, 407-414, 1993.
[61] M. Stein, Large sample properties of simulations using Latin hypercube sampling, *Technometrics,* 29;143-151, 1987.
[62] Storlie CB, Helton JC, Multiple predictor smoothing methods for sensitivity analysis: Description of techniques, *Reliability Engineering and System Safety,* 93:28-54, 2007.
[63] M. Tenenhaus. *La régression PLS. Théorie et pratique*. Editions Technip, Paris, 1998.
[64] V. Vapnik. *The nature of statistical theory*. Springer, 1995.

[65] W.N. Venables, D.M. Smith and the R Development Core Team. *An introduction to R. Notes on R : A Programming Environment for Analysis and Graphics, Version 1.5.0*. April 2002. <www.r-project.org>.
[66] W.N. Venables and B.B. Ripley. *Modern applied statistics with S.* Springer, fourth edition, 2002.
[67] A. Vivier. Formation de base en statistique de mesure, Tome 1 : Incertitudes en mesures physiques, CEA SACLAY/INSTN/UESMS.
[68] J. Verzani, *Using R for Introductory Statistics*, Chapman & Hall/CRC, Boca Raton, FL, 2005.
[69] E. Volkova, B. Iooss and F. Van Dorpe. Global sensitivity analysis for a numerical model of radionuclide migration from the “RRC Kurchatov Institute” waste disposal site. *Stochastic Environmental Research and Risk Assessment,* 22:17-31, 2008.
[70] E. Wegman. Hyperdimensional data anlysis using parallel coordinates, *journal of the American Statistical Association,* 85:664-675, 1990.