Table des matières

Chapitre 1

Introduction

La présente thèse s'inscrit dans le cadre de la poursuite et de la formalisation de travaux en tant qu'ingénieur de recherche et de développement au sein de l'équipe DIGIPLANTE du laboratoire de Mathématiques et Informatique pour la Complexité et les Systèmes (MICS) de l'école CentraleSupélec.

Elle consiste à réduire la complexité du domaine de la modélisation des systèmes biologiques au moyen de l'outil informatique en proposant un langage, en concevant et en développant des outils pour l'étude des systèmes dynamiques discrets stochastiques.

DIGIPLANTE travaille sur le développement, l'analyse et l'évaluation de modèles de la croissance des plantes pour l'agronomie et la foresterie. Aussi, ces modèles fourniront les cas d'application privilégiés de ces travaux. Malgré tout, la démarche est plus générale et repose simplement sur le type mathématique des modèles, i.e. des systèmes dynamiques discrets à espaces d'états continus.

La construction de tels modèles nécessite à la fois une connaissance en biologie (agronomie, pédologie, climatologie, ...), une connaissance des outils mathématiques (systèmes dynamiques, statistiques, ...) et une bonne maîtrise de l'informatique (simulation, visualisation, calcul haute-performance, ...)

C'est sur ces deux dernières disciplines qu'intervient cette thèse dont la problématique peut être formulée ainsi :

> Étant donnés des systèmes biologiques modélisables par des systèmes dynamiques stochastiques à temps discret, quels sont les outils et éléments de langage nécessaires afin de conduire une rigoureuse analyse quantitative de ces modèles tout en exploitant au mieux les ressources de calcul ?

Nous verrons ainsi les caractéristiques de ces modèles et de leur analyse ainsi que ce que cela implique en matière de modélisation informatique afin de développer un grand nombre de fonctionnalités. Nous verrons aussi comment mettre en place un flux de travail à partir des éléments développés afin de valoriser et partager la connaissance générée. Cette connaissance s'appuiera sur les données expérimentales et les modèles dans le but final d'améliorer les prévisions de rendements, la gestion des cultures, ou encore la sélection variétale.

Il est important de noter l'importance du travail en équipe et de l'échange au sein de l'équipe d'accueil et avec les chercheurs associés. La création d'une bibliothèque logicielle et d'un langage dédié étant à la fois un challenge technique, étant donné les contraintes imposées, mais aussi un challenge humain afin de rendre facile d'accès l'ensemble et qu'il soit utilisé.

Cette thèse s'organise de la sorte : le présent chapitre 1 "Introduction" présentera les pré-requis nécessaires autour la modélisation mathématique, de la simulation numérique et de l'agronomie afin de comprendre la portée générale de la problématique. Le chapitre 2 "Modèles, noyau, simulation et langage" constitue le cœur de notre recherche et sera dédié à établir la définition mathématique des modèles et de sa transformation algorithmique afin de conduire des simulations et de construire notre langage dédié. Le chapitre 3 "Analyse quantitative des modèles mécanistes" présentera le flux de travail général pour

l'analyse des modèles ainsi que l'ensemble des algorithmes utilisés dans le cadre de ce système. Nous appliquerons l'ensemble à trois études de cas afin d'illustrer comment le système d'analyse développé peut être utilisé sur plusieurs modèles, dans une démarche de bonnes pratiques de modélisation. Enfin, nous présenterons les perspectives possibles dans le cadre du langage, des outils, des algorithmes et de la communauté avant de conclure cette thèse au chapitre 4.

Dans un premier temps, nous allons discuter de l'activité de modélisation et simulation dans un cadre général avant de voir les spécificités en agronomie afin de finir sur les problématiques qui nous concernent dans le cadre de cette thèse.

1.1 Modélisation mathématique des systèmes complexes

Dans cette première section, nous regarderons les notions de modélisation et de système dans leur forme la plus large, avant de passer au cadre d'application, à savoir l'agronomie, dans la section suivante. Il faut rappeler que l'un des objectifs de la modélisation scientifique est de trouver des variables explicatives à un phénomène donné. Ainsi la capacité prédictive d'un modèle est directement liée à sa capacité explicative. Toutefois, dans cette section, nous donnerons un sens plus large de modèle en introduisant aussi la modélisation guidée par les données qui donne lieu à une "boîte noire" où le comportement importe moins que la simple capacité prédictive par rapport à une situation donnée.

1.1.1 Qu'est-ce qu'un système complexe ?

Définissons en premier lieu la notion de système, nous reprenons à ce titre deux citations que l'on peut retrouver dans (Cellier 1991) :

— *A system is what is distinguished as a system.* (Gaines 1979)
— *A system is a potential source of data.* (Zeigler 1976)

La première citation met en avant le rôle de l'observation dans la création du système. C'est en fonction de ce que l'on souhaite voir qu'implicitement le système se crée. La deuxième citation, quant à elle, nous indique que, potentiellement, nous aurons une source de données avec laquelle confronter notre abstraction du système.

De façon générale, un système se définit par rapport à une frontière. Tout ce qui est à l'intérieur de cette frontière est **le domaine** du système, tout ce qui est en dehors est **l'environnement** du système. Il y a un échange entre ce domaine et cet environnement au moyen d'**entrées** vers le domaine et de **sorties** vers l'environnement. La frontière est conditionnée par l'observation qui est faite sur le système. Autrement dit la création d'un système est liée à ce que l'on souhaite étudier sur celui-ci. La figure 1.1 est une synthèse de ceci.

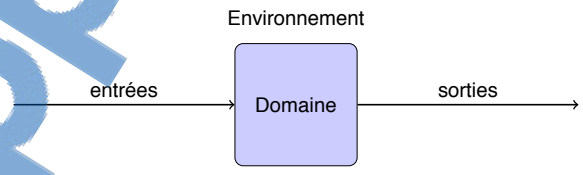


Figure 1.1 – notion de système

Ainsi, on peut résumer la notion de système à : *un système est une abstraction d'un objet réel, décomposé en un ensemble d'éléments contenus dans un domaine délimité par une frontière, la séparant de son environnement, et soumis à une observation qui produit un ensemble de données en sortie.*

Toutefois, par la suite, nous lui préférons la version qui se trouve dans (Walter et Pronzato 1994) et dont nous trouverons une représentation en figure 1.2. Cette représentation rajoute une notion supplémentaire sur les **entrées**, en précisant la notion de **bruit** ou **perturbation**. Elle rajoute aussi une notion sur les sorties, en faisant la différence entre les **variables mesurables du système**, notées ici sorties, et

les **variables cachées à la mesure**, notées ici **variables d'intérêts**. Cette vision des choses nous est plus favorable afin de prendre en compte la **stochasticité** par la notion de bruit et de prendre en compte la différence entre des **états observables** du système et des **états cachés**.

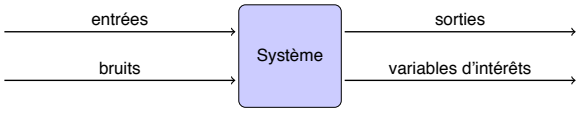


Figure 1.2 – Notion de système selon (Walter et Pronzato 1994)

On parle de système complexe lorsque l'interaction de ses éléments constitutifs fait émerger des propriétés qui dépasse celles décrites par les sous-systèmes. Ainsi, dans la figure 1.3, nous avons des éléments de notre système qui interagissent les uns avec les autres et dont la sortie n'est pas prédictible directement à partir des règles de chaque sous-système.

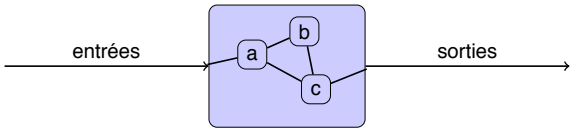


Figure 1.3 – notion de système complexe avec rétroaction

Comme nous l'avons vu précédemment, étudier un système c'est en même temps le définir car les hypothèses que l'on porte sur lui, et que l'on souhaite valider ou invalider, définissent son domaine et son environnement. Afin de comprendre son fonctionnement, nous allons solliciter le système et enregistrer sa réponse.

Cette sollication porte le nom d'expérience, tandis que la réponse est enregistrée sous forme de donnée. Nous pouvons relier ces deux éléments en reprenant la définition proposée dans (Cellier 1991) : *An experiment is the process of extracting data from a system by exerting it throught its inputs.*

Ainsi à partir d'un système réel donné et d'une problématique associée, il s'agit de réduire le périmètre de l'ensemble de telle sorte que l'on puisse décrire le problème sous la forme de variables explicatives et de variables expliquées. Si le système étudié est sous l'influence d'un autre système alors on répartira les variables explicatives en deux groupes : les variables d'état du système et les variables de contrôle du système. Ces ensembles de variables constituent le périmètre du système considéré. Une fois que ce dernier est fixé, on peut passer à la modélisation à proprement dite, à savoir quelle est la nature de la relation que l'on souhaite établir entre explicatif et expliqué. Cette relation peut se faire soit en mettant en évidence des corrélations dans la dynamique par le biais des méthodes d'apprentissage automatiques, soit en mettant en évidence un lien de causalité par le biais d'une modélisation prenant en compte les théories scientifiques. C'est que nous verrons respectivement dans les prochaines sous-sections.

1.1.2 Qu'est-ce qu'un modèle mathématique ?

La modélisation scientifique d'un système consiste à simplifier notre vue de ce dernier au moyen des théories scientifiques afin de générer une connaissance nous permettant par la suite de mieux comprendre, définir, classer, ou prédire les évolutions de celui-ci. Ceci est valable quelle que soit la nature du système étudié : matériaux, structure, population, climat, culture végétale, ...

Cette modélisation peut posséder différents caractéristiques :

- qualitative ou quantitative selon que l'on cherche à réfléchir sur la description générale d'un processus ou aux mesures de celui-ci ;
- statistique ou mécaniste en mettant en place soit des relations de corrélations, en se basant uniquement sur les observations, ou de causalités, en se basant sur les observations et la connaissance des processus sous-jacents du système considéré ;

— déterministe ou stochastique selon que l'on considère le système évoluant de manière déterminée ou aléatoire ;
— évènementiel, discret ou continu en temps selon notre représentation de l'écoulement du temps ;
— discret ou continu en espace selon notre discrétisation de l'espace des états ;

Les simulations des modèles devront être confrontées aux données issues des expériences afin de vérifier l'adéquation entre données réelles et données simulées et ainsi valider et évaluer le modèle.

Nous allons par la suite faire rapidement une distinction entre modèle réel et modèle virtuel, avant de voir les spécificités de la modélisation mécaniste en la comparant à la modélisation statistique.

1.1.2.1 Modéle physique et modèle mathématique

Il nous faut d'abord distinguer la modélisation qui peut être de nature "physique", i.e. on reproduit à plus petite échelle et avec un minimum de constituants un système réel. On parlera aussi de maquette. C'est ainsi que l'on procèdait exclusivement pour la création de voitures ou d'avions avant l'arrivé des outils mathématiques et informatiques dans ces industries. Un ensemble de maquettes étaient construites pour être testées en soufflerie afin de vérifier l'aérodynamisme de l'ensemble ou bien subissaient des essais de choc afin de tester la solidité de la structure. Dans le cadre de la biologie, nous parlerons de modèle *in vitro* ou encore de modèle *in vivo* pour qualifier une telle modélisation "physique".

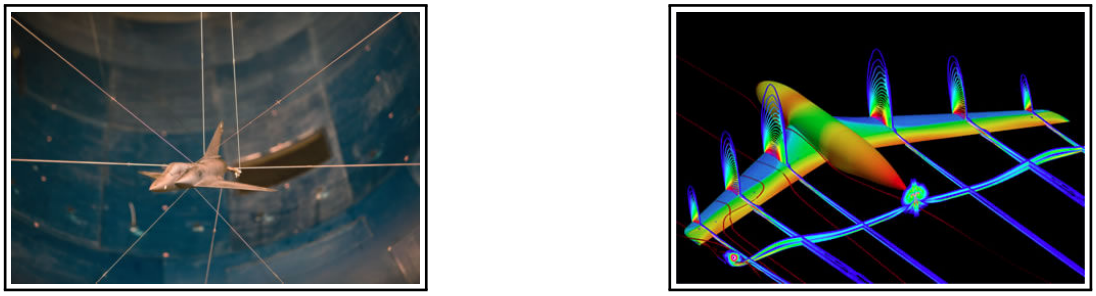


Figure 1.4 – Modélisation physique versus modélisation mathématique d'un avion dans une soufflerie (source ONERA)

Toutefois avec l'élaboration des modèles mathématiques, nous pouvons considérer des tests virtuels conduits à partir du calcul (le plus souvent informatiquement désormais) afin de réduire le nombre de maquettes construites, en éliminant virtuellement des constructions peu intéressantes par rapport à un ensemble de critères. Dans le cadre de la biologie, nous parlerons de modèles *in silico* pour qualifier la modélisation mathématique et informatique.

1.1.2.2 Modélisation mécaniste, multi-échelle, multi-formalisme, et causalité

La modélisation mécaniste cherche à mettre en relation les différentes variables par un lien de causalité qui est construit sur l'ensemble des théories scientifiques et qui permet notamment de formuler de nouvelles hypothèses de fonctionnements des processus constituant le phénomène global.

La simulation des modèles permet d'obtenir des données virtuelles et, par extension, de chercher à optimiser un comportement du système vis à vis de certaines contraintes. L'angle d'approche de cette optimisation détermine le niveau de description et par conséquence la modélisation est bornée par ce que l'on souhaite en faire.

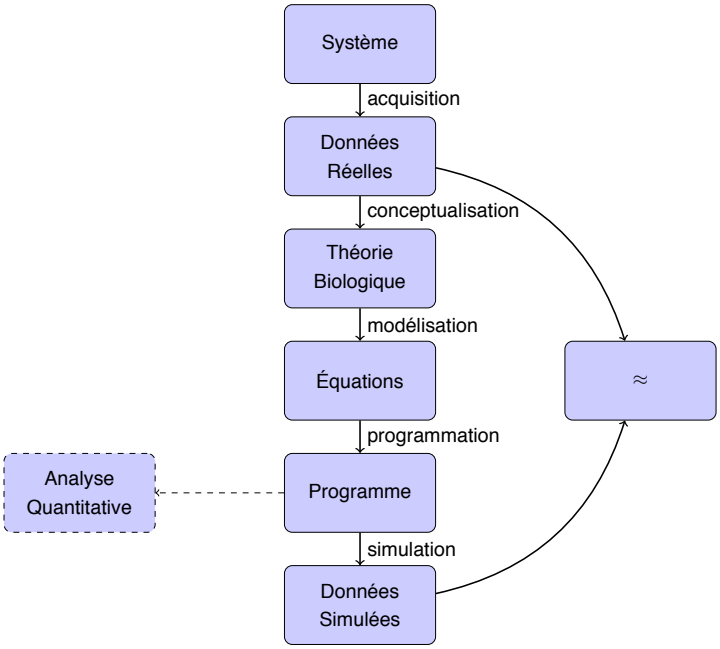


Figure 1.5 – du réel au simulé dans le cadre de la modélisation mécaniste

Certains modèles peuvent posséder des caractéristiques qui peuvent le rendre plus complexe dans sa façon de représenter le monde. Ainsi si la formulation du modèle peut être faite au travers de différents formalismes (équations aux différences, équations différentielles ordinaires, équations aux dérivées partielles, ...), nous pouvons aussi avoir à faire à des modèles possédant plusieurs formalismes en leur sein. On décrira par exemple un processus plus facilement avec des équations différentielles ordinaires puis nous utiliserons des automates a états finis pour retranscrire les sorties de sous-systèmes par exemple. Ainsi en mélangeant les formalismes dans les sous-systèmes, nous obtenons un modèle avec **multi-formalisme** ou **systèmes hybrides** (Goonatilake et Khebbal 1994 ; Blom et al. 2006).

De plus, certains modèles peuvent chercher à expliquer des phénomènes à différentes échelles d'un même système. Par exemple, nous souhaitons expliquer les variations en termes de production de biomasse sur un territoire (échelle macroscopique), en ayant celle des organes (échelle mésoscopique) et en descendant au niveau des voies métaboliques comme le cycle de la photosynthèse (échelle microscopique). Ces modèles sont qualifiés de **multi-échelles** (Naldi, Pareschi et Toscani 2010 ; Ingram, Cameron et Hangos 2004).

Nous venons de voir certaines propriétés des modèles mécanistes ainsi que le lien qu'ils entretiennent avec la relation de causalité permettant de faire des prévisions grâce à leur pouvoir explicatif. Toutefois créer de tels modèles nécessite de faire suivre toute une logique de tests et de validation afin d'obtenir des modèles mécanistes robustes permettant de comprendre et évaluer leur pouvoir prédictif. Nous verrons cela dans les sections suivantes. Pour l'instant nous allons aussi brièvement parler de la modélisation guidée par les données afin de voir en quoi elle diffère de la modélisation mécaniste et qu'est ce que qu'elle peut apporter dans l'étude des modèles mécanistes.

1.1.2.3 Modélisation guidée par les données, métamodèle, et corrélations

La modélisation guidée par les données désigne toute activité mathématique consistant à créer des estimateurs permettant de simuler les systèmes considérés sans connaissance a priori du système.

Les techniques vont de la régression linéaire simple à des méthodes d'apprentissage statistique plus sophistiquées comme les "réseaux de neurones" ou les "machines à vecteurs de support" (Hastie, Tibshirani et Friedman 2001).

La figure 1.6 illustre la chaîne de travail permettant de passer du système réel aux données simulées avec la modélisation guidée par les données. La représentation multiple au niveau des données réelles

est un choix permettant de mettre en avant le fait que certaines techniques statistiques, notamment issues du machine learning, nécessite un grand volume de données avant de pouvoir être correctement initialisées.

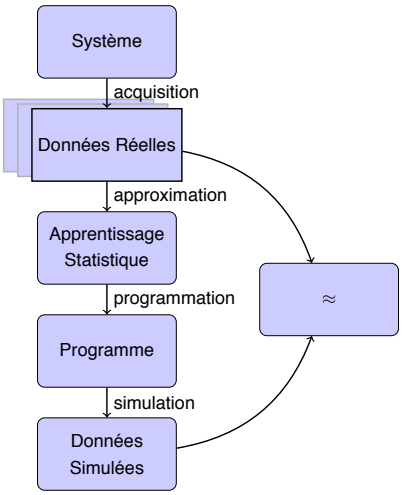


Figure 1.6 – du réel au simulé dans le cadre de la modélisation statistique

Nous pouvons aussi utiliser ce genre de modélisation statistique à partir des simulations d'un modèle mécaniste sur un échantillonage des paramètres. Ce type de modèle est appellé un **métamodèle** et permet notamment une évaluation plus rapide, lors d'analyse de sensibilité par exemple dont nous reparlerons plus loin, étant donné que la fonction du métamodèle est généralement plus simple à évaluer que l'intégration du modèle (Faivre et al. 2013). Par exemple, dans la figure 1.6, nous pourrions créer un estimateur d'un modèle existant en considérant comme "données réelles" celles en sortie du programmation de simulation dans la figure 1.5.

En ce qui concerne l'agronomie, nous pourrions obtenir des estimateurs afin de faire de la prévision et du contrôle (Kano et Nakagawa 2008) mais serions dans l'incapacité de comprendre les phénomènes régissant la croissance des plantes étant donné que ce type de modèles ne reposent sur aucune hypothèse biologique. Ainsi nous obtenons des modèles prédictifs mais non explicatifs. Ce type d'analyses a été conduit durant le déroulement de cette thèse par d'autres acteurs de l'équipe de recherche mais ne seront pas étudiés dans le cadre de cette thèse. Nous en reparlerons néanmoins au moment des perspectives.

Il faut noter que nous n'avons présenté les choses que sous l'angle de la régression afin de se comparer à la modélisation mécaniste. Cependant, la modélisation guidée par les données peut avoir une portée plus large notamment lorsqu'il s'agit d'effectuer des opérations de clustering par exemple (Hastie, Tibshirani et Friedman 2001). Ainsi nous pouvons chercher à créer des typologies sur la base de données quantitatives par exemple.

1.1.3 Comment obtenir un modèle mécaniste prédictif et robuste ?

Nous venons de voir qu'en plus de la modélisation mécaniste, nous pouvons parfois avoir recours à une modélisation guidée par les données. Ces deux types de modélisation peuvent avoir le même but, à savoir tenter de fournir des estimateurs d'un processus réel. Toutefois, les modèles purement statistiques ne possèdent pas d'explication quant au comportement interne des systèmes. Si nous souhaitons émettre des hypothèses sur le fonctionnement interne des processus, seule la modélisation mécaniste peut nous aider. Dès lors, nous devons nous poser les questions du pouvoir prédictif de ces modèles ainsi que de leur robustesse. Autrement dit, est-ce qu'un modèle est utilisable sur une large gamme de conditions et que se passe t'il lorsque nous le poussons un peu plus loin que sur ces conditions. Est-ce qu'un modèle conçu sur un jeu de données A pourra être aussi un bon estimateur sur un jeu de données

B ?

En effet, la formulation du modèle mécaniste réalisée, plusieurs passes sont nécessaires afin d'améliorer la prise en charge de certains processus en faisant des allers-retours entre différentes méthodes mathématiques d'analyse.

Ces différentes étapes et notamment le flux de travail associé sont dénommés sous le nom de "bonnes pratiques de la modélisation" (ou "good modeling practices") (Scholten et al. 2000). Nous avons simplifié et synthétisé le flux de travail reliant ces différentes étapes dans la figure 1.7.

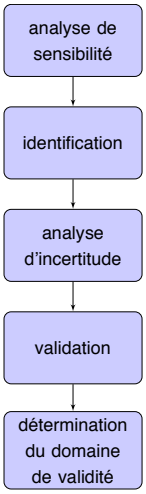


Figure 1.7 – flux de travail sur un modèle

Ces étapes peuvent être la simple simulation du modèle, l'estimation de ses paramètres ou encore son étude par analyse d'incertitude ou de sensibilité. On peut voir par exemple l'importance de certaines étapes auprès des chercheurs dans (Klipp et al. 2007) qui montre l'intêret qui est porté autour de l'estimation paramétrique et de l'analyse de sensibilité.

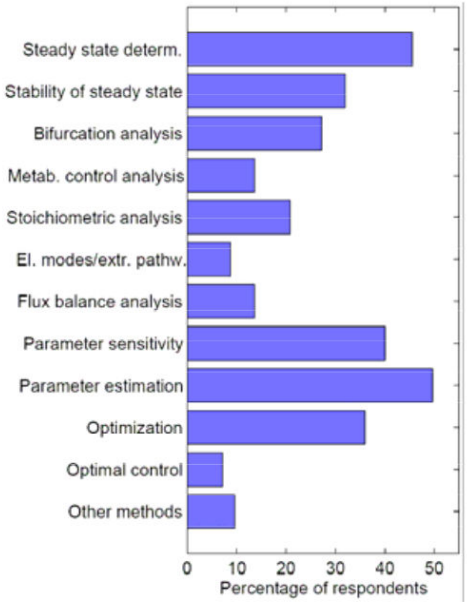


Figure 1.8 – L'estimation paramétrique et l'analyse de sensibilité, deux fonctionnalités importantes pour les modélisateurs (Klipp et al. 2007)

Toutes ces étapes ont pour but de nous renseigner sur le pouvoir prédictif et la robustesse du modèle. Le pouvoir prédictif d'un modèle consiste à avoir un modèle qui en fonction de nouvelles entrées puisse

prévoir une sortie proche de la réalité. Pour vérifier le pouvoir prédictif d'un modèle, nous pouvons appliquer d'autres données en entrées que celles qui ont servi à la calibration du modèle et générer une série de critères nous permettant d'évaluer ce pouvoir prédictif. Nous verrons cela en détail dans le chapitre 3.

La robustesse d'un modèle est une propriété qui assure que de petites perturbations au niveau des entrées d'un modèle n'ont pas d'impact trop grand sur ces sorties, ce qui invaliderait les résultats de prévision. Pour vérifier la robustesse du modèle, nous pratiquerons une analyse d'incertitude donnant lieu à des intervalles de confiance ou de crédibilité selon notre cadre statistique (fréquentiste ou bayésien). De même, nous verrons cela en détail dans le chapitre 3.

Nous avons fourni en figure 1.7, une version simple du flux de travail. Toutefois il convient de la replacer dans le contexte plus général de la modélisation, comme le montre la figure 1.9. Ainsi, le commanditaire de la modélisation est potentiellement le groupe de recherche dont le modélisateur fait partie ou un utilisateur final qui peut être, par exemple, un expérimentateur. L'activé de modélisation a été découpée en trois entités distinctes, à savoir le modélisateur, chargé de l'écriture du modèle, l'analyste, chargé de l'analyse générale du modèle, et l'ingénieur, chargé du déploiement et de la diffusion du simulateur encapsulant le modèle. Toutefois, dans nombre de cas, les trois activités sont réalisées par une même personne. Ce découpage permet de comprendre que c'est par le retour d'informations (feedback) que le modélisateur peut améliorer son ou ses modèles en vue d'un objectif de modélisation.

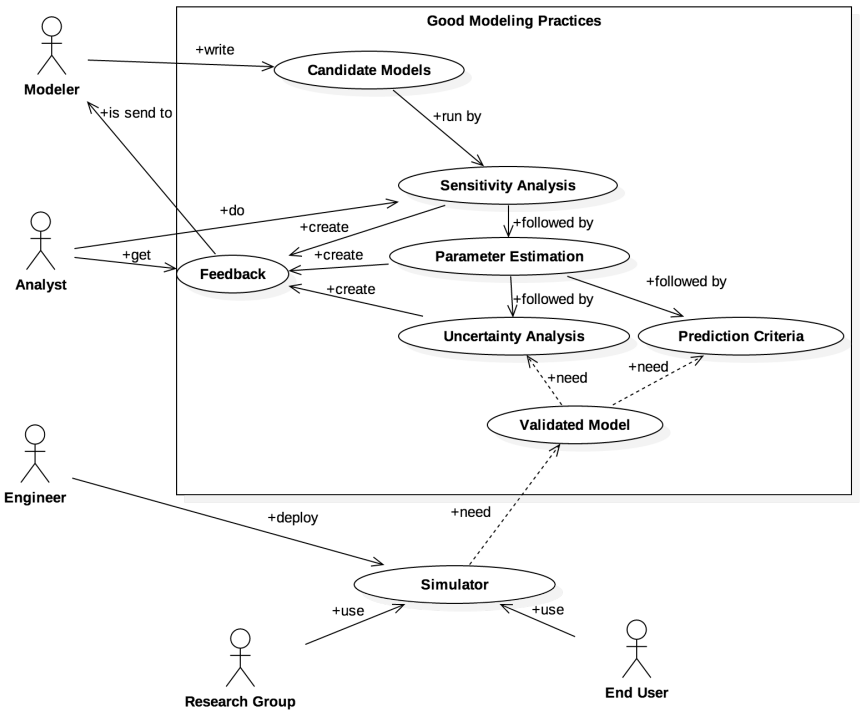


Figure 1.9 – cas d'utilisation typique de la chaîne d'analyse

La présentation de ce flux et de notre implémentation particulière dans cette thèse seront présentés lors du chapitre 3.

Nous venons de voir quelques caractéristiques du flux de travail concernant la modélisation mécaniste. Ce type de travail nécessite de réaliser un grand nombre de simulations numériques pour certains algorithmes. Il faut donc savoir à la fois coder son modèle afin d'avoir le maximum de similitude entre sa formulation et son implémentation, mais aussi pouvoir exploiter les architectures parallèles des ordinateurs modernes afin de minimiser les temps de simulation de tous les modèles candidats. Nous verrons dans la section suivante les caractéristiques de la simulation numérique.

1.1.4 Qu'est-ce qu'une simulation numérique ?

De manière générale, "une simulation consiste à appliquer des expériences sur un modèle" (Korn et Wait 1978 ; Cellier 1991) afin d'obtenir des données. La simulation numérique consiste à prendre la modélisation mathématique et de l'encoder sous format numérique par le biais d'un langage de programmation et d'un formalisme informatique adéquat afin d'obtenir des données virtuelles en effectuant le calcul sur un calculateur.

Dans cette section, nous verrons quelques précisions sur les formalismes dédiés à la modélisation et simulation des systèmes complexes ainsi que les langages de programmation qui permettent de les encoder et enfin quelques spécificités des architectures des calculateurs utilisés pour les simulation et les analyses.

1.1.4.1 Formalismes informatiques dédiés à la modélisation et à la simulation

Nous avons vu précédemment que les modèles mathématiques peuvent être écrits avec différents formalismes, et parfois ils peuvent en mêler différents entre eux afin de satisfaire des modélisations de processus soit avec des dynamiques différentes, soit avec des échelles différentes, voire les deux. Toutefois il nous reste à traduire cette modélisation mathématique en quelque chose de compréhensible pour un ordinateur afin que les calculs soient possibles. Une première étape consiste à adopter un formalisme dédié à l'informatique. Ce formalisme peut soit être déja existant ou à inventer selon les contraintes de la modélisation.

Un des formalismes les plus connus est DEVS (Zeigler et al. 1995) qui est à l'origine dédié à la modélisation des systèmes avec évènements discrets (Discrete Events Specification System). Il a connu de nombreuses extensions comme DTSS (Discrete Time Specification System) (Zeigler, Kim et Praehofer 2000), DESS (Differential Equation Specification System) (Zeigler, Kim et Praehofer 2000) (Kofman et Junco 2001), PDEVS (Parallel [1] Discrete Events Specification System) (Chow 1996), ... Il a aussi connu une réécriture en STDEVS (Stochastic Discrete Events Specification System) (Castro, Kofman et Wainer 2008) afin de favoriser la modélisation des évènements discrets stochastiques.

Figure 1.10 – visualisation des extensions de DEVS et des formalismes de modélisation mathématique sous-jacents.

Il existe une notion intéressante introduite lors des premiers travaux autour de DEVS, à savoir la notion de cadre expérimental (Zeigler, Kim et Praehofer 2000). Cette notion souligne l'importance de séparer

1. Ici, il faut prendre parallèle au sens de concurrent.

la description du modèle, i.e. les variables, les paramètres et les équations, et le cadre expérimental qui contient les objectifs de la modélisation ainsi que les valeurs admissibles pour les entrées. Ce faisant, un simulateur devrait prendre en compte un modèle et une expérience afin de faire tourner le dernier sur le premier mais aussi un cadre expérimental afin de vérifier que l'expérience virtuelle conduite est valide sur le modèle considéré. Toutefois, à notre connaissance, ce cadre a été formalisé dans le contexte du modèle DEVS classique mais pas étendu aux extensions. On pourra trouver un exemple d'utilisation dans (Cheon 2007) ou (Foures, Albert et Nketsa 2013).

Nous venons de voir quelques formalismes théoriques dédiés au champ de la modélisation et simulation en général. Étant théoriques, il ont besoin d'être encodés par le biais d'un langage de programmation pour pouvoir calculer les sorties des simulations sur une architecture matérielle. Nous verrons dans la section suivante un tour d'horizon des différents types de langages existant.

1.1.4.2 Langages de programmation

La précédente section était dédiée au formalisme informatique en modélisation et simulation. Néanmoins il n'y a pas de simulation numérique sans langage de programmation. Ces langages de programmation peuvent être décrits selon différents niveaux et certains projets de modélisation peuvent mêler différents niveaux ensemble. Quels sont les différents types de langage que l'on peut rencontrer ?

Si nous écrivons les différents niveaux de complexité dans la programmation de ces systèmes, nous pouvons créer 4 niveaux de complexités :

- on écrit un modèle et un simulateur en étant fortement couplé et sans formalisme particulier. Pour cela nous utiliserons soit un langage de programmation général (General Purpose Language - GPL) de type C++, Fortran, Java, Python, ..., soit un langage de programmation dédié (Domain-Specific Language - DSL) de type Modelica (Elmqvist et Ab 1997) ou un paradigme de programmation visuelle comme MATLAB/SIMULINK (MATLAB 2010), SCILAB/SCICOS (Campbell, Chancelier et Nikoukhah 2006a), si notre modélisation rentre dans le cadre prévue par ces langages. Nous sommes dans le cas d'une application spécifique.
- on écrit un modèle et un simulateur en découplant les deux. Ce découplage est généralement fait par la mise au point d'une interface de programmation commune (Application Programming Interface - API) aux modèles et simulateurs. On peut donc soit réutiliser une bibliothèque logicielle existante, soit en créer une afin de créer plusieurs modèles avec un même simulateur. C'est notamment le cas avec des bibliothèques logicielles comme VLE (Quesnel, Duboz et Ramat 2009) qui implémente le formalisme DEVS et certaines de ses extensions.
- on formalisme l'ensemble précédemment développé afin de créer un langage dédié embarqué (Embedded Domain-Specific Language - EDSL) (Leijen et Meijer 1999) qui s'appuyera sur le langage général utilisé pour le développement. On met au point des opérateurs et des structures de données génériques pouvant encapsuler les spécificités du modèle étudié. C'est le cas par exemple de PyCUDA ou PyOpenCL (Klöckner et al. 2009) qui sont des EDSL en python pour le lancement de calcul sur GPU, ou encore de Boost.Spirit qui est un EDSL en C++ pour la création de parser et de lexer.
- on extrait un langage avec sa propre syntaxe (DSL, Domain Specific Language) et sa propre sémantique par étude d'un domaine particulier tel que l'on puisse créer un interpréteur ou un compilateur dédié (Deursen, Klint et Visser 2000). De fait on perd la généralité des langages utilisés en amont de type C++, Fortran, C, Java, mais nous avons un langage facile à utiliser pour coder notre modèle. Toutefois nous ne pouvons pas sortir de ce cadre. C'est le cas du langage Modelica (Elmqvist et Ab 1997) par exemple.

Pour résumer :

1. **langage de programmation général - GPL** : langage permettant d'exprimer toutes fonctions. [a]
2. **interface de programmation applicative - API** : ensemble des structures de données et algorithmes permettant de développer à partir d'une bibliothèque informatique.
3. **langage dédié embarqué - EDSL (Leijen et Meijer 1999)** : ensemble de structures de données et algorithmes articulé autour d'une syntaxe formalisée s'appuyant sur la syntaxe d'un langage hôte.
4. **langage dédié - DSL (Deursen, Klint et Visser 2000)** : langage autonome du point de vue syntaxique et sémantique dédié à l'approche d'un domaine particulier.

a. fonctions calculables au sens de Turing, mais nous ne rentrerons pas dans le détail.

Nous venons de voir les différents niveaux de complexité dédié aux langages. En passant du niveau 1 au niveau 4, on réduit la complexité d'encodage des modèles et on peut plus facilement fournir des abstractions à des processus complexes. Par exemple, ce qui est long et difficile à écrire au niveau 1 pourra être abstrait par un mot dédié dans une syntaxe particulière et ainsi permettre à un utilisateur qui connait très bien le domaine mais moins la programmation de pouvoir exprimer sa modélisation d'un phénomène. Si l'on reproduit la chose un grand nombre de fois, nous pouvons simplifier l'écriture et la lecture des modèles pour des personnes ayant une forte compétence dans le domaine mais pas forcément en programmation. Toutefois, cette abstraction et cette écriture ne permet plus de descendre facilement d'un niveau pour introduire des spécificités propres. Ainsi, si on se trouve dans la situation où nous devons traiter N simulations qui prennent t secondes à être évaluées, avec respectivement N et t de l'ordre de la centaine de milliers de simulation et de plusieurs secondes, alors nous devons fournir un système exploitant au mieux les architectures modernes de calcul avec notamment le calcul parallèle.

1.1.4.3 Architecture des calculateurs

Selon le type d'algorithme, nous pouvons optimiser les calculs pour prendre en compte différentes architectures de calcul et notamment le calcul parallèle. Nous ne traiterons pas du calcul concurrent (Herlihy et Shavit 2008) dans cette thèse, ce dernier ayant pour objectif de traiter des modèles dont des évènements apparaissent au même moment. C'est une approche qui fait sens dans le cadre des systèmes biologiques mais qui sera néanmoins mise de côté par rapport à notre modélisation de ceux-ci, i.e. les modélisations mathématiques que nous avons mis en place ne prennent pas en compte ce type d'évènement.

Ainsi, au du niveau du matériel et afin de gagner en performance, on retrouve l'exploitation d'architectures matérielles comme les architectures à mémoire partagée, exploitées avec OpenMP (OpenMP Architecture Review Board 2008), les architectures à mémoire distribuée, exploitées avec MPI (Gropp, Lusk et Skjellum 1999 ; Gabriel et al. 2004), les architectures massivement parallèles, exploitées avec CUDA (Nickolls et al. 2008) ou openCL (Stone, Gohara et Shi 2010).

Dans la suite de cette thèse, nous ferons en sorte qu'au minimum l'ensemble des outils exploite les architectures à mémoire partagée étant donnée la nature parallèle de la plupart des algorithmes d'analyse et le coût calcul potentielle important relatif aux méthodes.

Nous n'exploiterons pas les architectures distribuées étant donné qu'à ce jour la limite du modèle à mémoire partagée n'est pas encore problématique pour l'exploitation des ressources et que nous avons surtout voulu donner une cohérence à l'ensemble du système plutôt que de la performance brute. Toutefois cela fait partie des éléments à regarder par la suite.

Il est important de noter l'importance des langages dédiés dans ce domaine. En effet, avec ces langages, il est possible de cacher à l'utilisateur une complexité issue de l'exploitation des architectures parallèles. On peut proposer un langage et le traduire afin de supporter une ou plusieurs de ces architectures.

Ceci est notamment possible grâce à l'avénement de bibliothèques logicielles dédiées à la compilation multi-plateforme matérielle comme c'est le cas avec LLVM (Low-Level Virtual Machine) (Lattner et Adve 2004) dont nous reparlerons plus loin.

1.1.4.4 Génération de nombres aléatoires

La génération des nombres aléatoires et particulièrement la génération des nombres aléatoires en proposant des flux paralléles indépendants est un élément important pour nous dans la construction de nos algorithmes et de la chaîne générale d'analyse des modèles. C'est un sujet de recherche en constante évolution et, à notre connaissance, il n'existe pas de consensus autour d'un algorithme en particulier mais une liste de bonnes pratiques (Mascagni et Srinivasan 2000 ; L'Ecuyer, Oreshkin et Simard 2014 ; Reuillon et al. 2012 ; L'ecuyer 1999).

Dans le cadre de notre travail, il ne s'agira pas de proposer un algorithme plus avancé que ce qui existe déjà mais de faire un tour d'horizon de l'état de l'art afin de bien prendre conscience des problématiques issues de ce domaine afin de construire un simulateur permettant une bonne utilisation des distributions au sein de nos simulations.

1.1.5 Comment diffuser la connaissance au sein d'une communauté ?

Nous avons vu, aux travers des sections précédentes, qu'un modèle mécaniste nous permet de valider ou invalider le comportement d'un processus à partir de la comparaison avec les données réelles. Cette validation ou invalidation est par essence une information et, de proche en proche, nous pouvons construire une connaissance autour de notre système à partir des différents modèles que nous avons construits. Une fois ces informations collectées et cette connaissance générée, il faut se poser la question de leurs diffusions au sein de la communauté scientifique. Cette diffusion peut être faite évidemment par le biais d'articles de recherches mais il est aussi important de la structurer afin de pouvoir raisonner sur celle-ci et l'explorer plus efficacement. C'est dans ce cadre qu'interviendront les notions d'ontologie et de standardisation.

Nous voyons une communauté comme un ensemble d'entités partageant un même langage et s'auto-organisant pour accomplir des buts. Une seule communauté pouvant être aussi composée de communautés en son sein. Par exemple dans la communauté des modélisateurs en agronomie, il existe des communautés connaissant mieux un modèle qu'un autre et de fait, développe un langage qui leur est propre autour de ce modèle. Ce type de langage est dit communautaire. Ainsi les experts d'un modèle A développe des outils et méthodes autour de A, de même avec un modèle B. Toutefois ces langages communautaires ne sont pas forcément compris de tous les agronomes. Ainsi, pouvons-nous traduire ces éléments dans un langage plus général, un langage dit véhiculaire, afin de rendre compte des éléments pris en compte dans la modélisation des processus.

1.1.5.1 Ontologies

Le terme ontologie a émergé au sein de la communauté de l'intelligence artificielle afin de faire des systèmes capables de raisonner sur un ensemble de connaissances. La définition d'une ontologie dans le domaine des sciences de l'information peut être approchée par celle donnée par Tom Gruber comme la "spécification formelle d'une conceptualisation" (Gruber 1995)

Ainsi, si l'on peut décrire dans une base de connaissances sous forme de relation "sujet-prédicat-objet" un ensemble de connaisances, c'est bien la découverte automatique (inférence) de certains liens entre les éléments, qui sont importants et qui animent la communauté de l'intelligence artificielle. On peut par exemple utiliser un raisonneur sémantique comme Hermit (Glimm et al. 2014) au sein de la plateforme Protégé (Gennari et al. 2003) afin de faire émerger de nouvelles connaissances. Par exemple, soit un système S et deux modèles $M1$ et $M2$, nous pouvons demander la traduction des termes de $M1$ en $M2$

s'il y a des équivalences, ou bien voir s'il manque des processus dans la modélisation effectuée.

1.1.5.2 Standardisation

Les standards ont pour particularité de favoriser à la fois la discussion car ils permettent de comparer rapidement des expériences et des simulations diverses dans un même référentiel, mais aussi ils permettent de favoriser la reproducibilité qui est un élément déterminant en recherche. Il n'existe pas à ce jour de standards dans le domaine des bonnes pratiques de modélisation, autrement dit un standard qui permettrait de suivre l'analyse et l'évolution d'un modèle ou d'un simulateur de sa création jusqu'à ses différentes versions de diffusion.
Cepandant, si l'on prend l'étude par Klipp et al. sur les standards en biologie computationelle (Klipp et al. 2007), nous pouvons nous rendre compte de l'engouement pour les standards au sein d'une communauté illustré par la figure 1.11. Ce sondage n'a pas été conduit sur l'ensemble de la communauté, mais on peut penser qu'une telle tendance se retrouve en général.

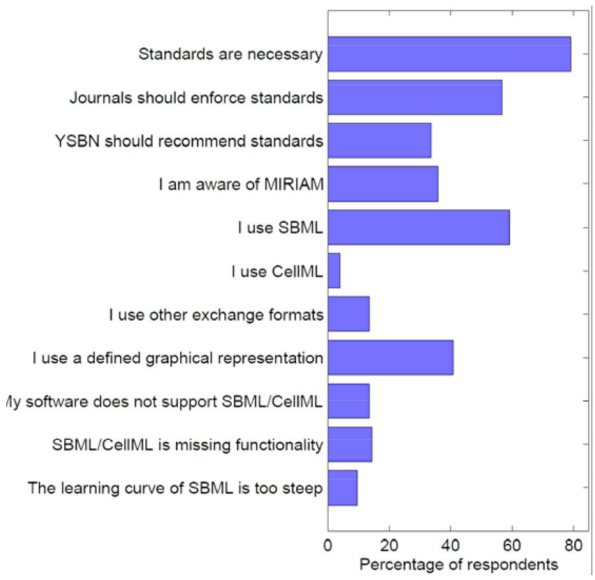


Figure 1.11 – Opinions sur les standards auprès de 125 chercheurs (Klipp et al. 2007)

Il existe un standard intéressant qui émerge auprés des chercheurs : le SBML (System Biology Markup Language) (Hucka et al. 2003). Ce format permet l'échange en XML (Bray et al. 2008) des modèles. Dans sa dernière version (Hucka et al. 2010) il permet le support des modèles hiérarchiques, des distributions, en plus des modèles compartimentaux régis par des équations. On trouvera en annexe G un exemple d'un fichier SBML pour le modèle Lotka-Volterra dont on retrouvera une formulation en annexe F. Cependant, si les équations sont échangées ce n'est pas le cas du cadre expérimental qui l'accompagne. Nous reparlerons de cet élément lors des perspectives.

Nous avons présenté le cadre général autour des notions de modélisation et simulation qui servira de support à nos travaux de recherche. Nous avons également souligné qu'un modélisateur doit manipuler un grand nombre de concepts et d'outils mathématiques et informatiques afin de conduire de façon satisfaisante une modélisation. Nous le verrons dans la section suivante, mais on peut déjà s'interroger sur le fait de vouloir réduire la complexité et la courbe d'apprentissage de tous ces éléments. Existe t'il un outil, ou un langage, permettant de manipuler tous ces éléments de façon standard afin d'aider le modélisateur à conduire au mieux ses travaux ? Avant de passer à la formulation de notre problématique, nous allons nous attarder sur le cadre biologique qui nous intéresse, à savoir la croissance des plantes. Nous ne regarderons ce cadre que sous l'angle de la modélisation mécaniste et laisserons de côté la modélisation guidée par les données.

1.2 Modélisation mécaniste de la croissance des plantes pour l'agronomie

"Quels seront les rendements de blé en France pour le mois de juillet prochain suite à la forte variation climatique subie ces derniers mois ?"

"Quel est l'apport d'eau optimal pour la culture en évitant les gaspillages ?"

"Quelle variété de maïs est la plus adaptée à des conditions de sécheresse précoce ?"

Répondre à ces questions nécessite la mise en place d'un système permettant de prédire quantitativement l'évolution des systèmes biologiques sous-jacents et notamment de mettre au point des modèles mécanistes permettant d'approcher le comportement des systèmes par la simulation. Toutefois la prédiction n'est pas l'unique but de cette approche quantitative et nous pourrions aussi chercher à partitionner des données d'origine agronomique, ou encore effectuer des classifications de données de nature qualitative.

À l'instar des disciplines (comme la finance quantitative, la biologie quantitative, ou encore les sciences sociales quantitatives) qui se sont développées autour de la modélisation mathématique dans des nouveaux champs d'application, nous nommons cette approche : *"agronomie quantitative"*.

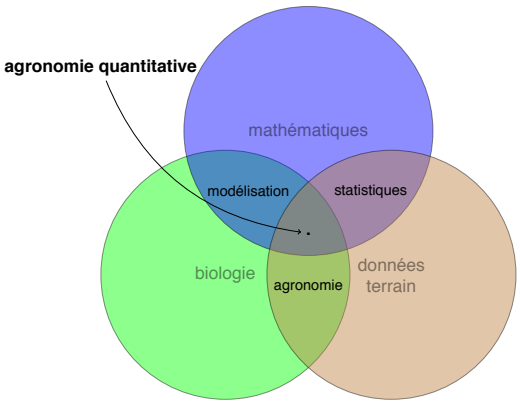


Figure 1.12 – composition de l'agronomie quantitative

Afin de bien comprendre la portée générale du travail présenté dans cette thèse, il est important de comprendre cette expression en mettant en perpective les deux mots qui la composent vis à vis de l'ensemble du vocabulaire existant dans le domaine.

L'**agronomie** consiste en la science qui étudie la naissance, la vie et la mort des plantes en interaction avec leur environnement. Elle ne doit pas être confondue avec l'agriculture qui possède une dimension sociale plus forte. En effet l'agriculture est une notion temporalisée et qui évolue en fonction des sociétés. L'agronomie a une portée plus large en ce sens qu'elle est une *scientia* ("connaissance") et non une

cultura (”honorer”, ”cultiver”, ”soigner”).

La notion **quantitative** trouve son fondement dans les méthodes quantitatives qui reflète ici l'utilisation des outils issus du calcul numérique, des statistiques et de la modélisation. Ce que nous cherchons à quantifier ce sont les surfaces foliaires, les futurs rendements, les teneurs en eau ou en azote du sol, plus généralement les caractéristiques du système de la plante ou de la culture dans son environnement. Dans le diagramme de Venn ci-dessus nous avons utilisé le mot ”statistiques” pour dénoter toutes les méthodes issues de l'apprentissage, de l'identification et évaluation, de l'analyse d'incertitudes ou de l'analyse de sensibilité des modèles mécanistes.

L'**agronomie quantitative** est donc une discipline mélant agronomie, modélisation, calcul numérique et statistiques. Son objet d'étude est en premier lieu la modélisation quel que soit son type et les caractéristiques qui en découlent. Elle n'est pas exclusivement basée sur le concept de modélisation mécaniste des systèmes dynamiques. On peut aussi approcher cette discipline au moyen de méthodes issues de l'apprentissage automatique comme avec les réseaux de neurones ou les machines à vecteurs de support. On peut aussi mélanger les différentes techniques entres elles (Fan et al. 2015). Toutefois, l'utilisation des modèles mécanistes permet de mieux approcher des phénoménes biologiques complexes dans lesquels les dynamiques des processus mis en jeu peuvent interagir et créer de nouveaux phénomènes avec leur propre dynamique. Ainsi nous pouvons formuler des hypothèses sur le fonctionnement de la culture, de l'allocation de biomasse ou de la photosynthèse par exemple.

Quelle place donnons nous à cette activité vis à vis des nomenclatures existantes ?

Comment peut-on placer l'agronomie quantitative vis à vis de l'agronomie, de la modélisation, des statistiques, des mathématiques et de l'informatique ?

Prenons l'exemple de l'étude d'un agrosystème, nous confions à une équipe de personnes le soin de créer un cadre expérimental afin de faire des relevés terrains. Une autre équipe crée à partir de ces données un estimateur, basé sur une modélisation mécaniste par exemple, afin de faire de la prévision de rendement. Une autre équipe produit à partir des données d'expérimentation un estimateur basé sur la modélisation guidée par les données.

Quel est le modèle le plus adapté pour faire de l'estimation sachant qu'un nouveau jeu de données sera acquis par la suite ? Peut-on créer une typologie des climats basée sur les données environnementales ?

Ce genre de question relève de l'agronomie quantitative. Son point de départ est de considérer le modèle comme existant, ainsi elle englobe l'activité de modélisation mais ne s'identifie pas totalement à elle. C'est la partie mathématiques et informatiques de l'agronomie qui porte sur le choix des modélisations et des modèles, qui évalue l'intêret d'une structure de modèles plutôt qu'une autre. Elle est agnostique vis à vis des processus retenus et met en compétitions les différents points de vue en confrontant les modèles uniquements sur leur capacité prédictive ou plus généralement sur leurs propriétés mathématiques et statistiques.

Ce n'est pas une nouvelle forme d'agronomie, mais plutôt une emphase sur une concentration de compétences transverses et sur l'application systématique des bonnes pratiques de modélisation à un ensemble de modèles, qui sont évalués et mis en compétition vis à vis d'un objectif donné.

En ce sens c'est l'entité qui regroupe agronomie, modélisation mécaniste, modélisation guidée par les données, statistiques, mathématiques, informatique ainsi que les liens de différentes natures que ces éléments partagent entre eux. Cette unité nous aménera par la suite à formuler l'hypothèse de la constitution d'une ontologie propre à ce domaine.

Figure 1.13 – Tournesols réels sous serre (source INRA) versus simulation numérique d'un champ par modèle GREENLAB.(source DIGIPLANTE)

1.2.1 Quel est le rôle de la modélisation mécaniste en agronomie ?

Il faut noter que cette approche est possible grâce à *la numérisation des activités agricoles* (cartes des sols, cartes d'épandage, cartes d'irrigation, relevés de rendements, images satellites, capteurs, ...). Cette numérisation peut être le résultat de la mise au point de nouveaux outils permettant de générer des données pour le domaine concerné par des acteurs extérieurs i.e. utilisation des satellites initialement non-prévus pour cette tâche spécifique ou bien le déploiement d'équipements par les équipes terrain du domaine concerné, par exemple installation de stations météorologiques dans les champs.

Dans un premier temps, nous présenterons le cadre biologique permettant de comprendre les échelles de description abordées, le cadre mathématique afin de comprendre les caractéristiques des méthodes existantes et comment ces deux volets interagissent avec les questions issues du terrain. Dans un second temps nous verrons les possibles champs d'application de ces modélisations avant de présenter un état de l'art autour des modèles, méthodes et logiciels.

Les systèmes biologiques qui nous intéressent particulièrement dans cette thèse sont :

- les **plantes à l'échelle individuelle** (individu moyen représentatif d'une espèce dans un environnement donné avec description des organes) avec une extrapolation à la **population** en faisant interagir **plusieurs instanciations de modèles** de plantes individuelles.
- les **champs à l'échelle de la parcelle**.

Nous n'aborderons pas les niveaux d'ordre inférieurs (cellulaire, ...), ni les niveaux d'ordre supérieurs (région, pays, continent, ...).

De même nous n'aborderons pas la problématique du multi-échelle qui consiste à pouvoir mélanger un niveau supérieur (parcelle) à un niveau inférieur (plante individuelle).

La démarche de la modélisation mathématique consiste, dans le cadre d'une problématique donnée, à prendre des processus biophysiques pour les traduire en équations, et généralement de les confronter à des données réelles qui sont acquises sur le système. Cette démarche est considérée comme valide si d'une part l'acquisition des données permet d'avoir un jeu de données de bonne qualité et si d'autre part les équations permettent de retrouver le comportement du système vis à vis de ces données, équations qui pourront le cas échéant avoir un domaine de validité plus grand. Il est, par conséquent, important de bien comprendre comment les processus fonctionnent mais aussi comment se fait l'acquisition des données. Étant donnée que notre compréhension de systèmes ouverts et vivants ne peut être parfaite, il y aura en jeu une notion de **bruit de modélisation** et de manière similaire nos appareils de mesures n'étant pas parfaits, nous introduirons du **bruit d'observation**.

Nous verrons deux types de modélisation. La première concerne la modélisation du champ ou de la parcelle, la seconde, concerne la plante individuelle.

1.2.2 Modélisation mécaniste à l'échelle du champ

La première étape de modélisation mathématique se construit avec un agronome. Celui-ci nous renseignera sur le fonctionnement du champ selon sa vision d'expert. Nous essaierons d'extraire de ce fonctionnement les concepts minimaux afin d'établir une première modélisation fonctionnelle. La prise en compte d'autres phénomènes et le raffinement du modèle pourront être vus par la suite. Nous illustrons sur un exemple cette démarche. Une première étape peut donner la vision suivante : "D'après l'étude du système et les données acquises on peut dire que la variation journalière de biomasse du champ est fonction du rayonnement et de la surface foliaire. Cette dernière est fonction de la variation du temps thermique."

Nous pouvons par exemple reprendre la modélisation faite dans (Brun et al. 2006) pour un modèle journalier discret (pas de temps = 1 jour) :

$$\begin{cases} \delta TT_j & = & max(\frac{TMIN_j+TMAX_j}{2} - T_{base}, 0) \\ TT_{j+1} & = & TT_j + \delta TT_j \\ \delta B_j & = & \begin{cases} RUE * PAR_j * (1 - e^{-K*LAI_j}) & TT_j \leq TT_M \\ 0 & TT_j > TT_M \end{cases} \\ B_{j+1} & = & B_j + \delta B_j \\ \delta LAI_j & = & \begin{cases} \alpha * \delta TT_j * LAI_j * max(LAI_{max} - LAI_j, 0) & TT_j \leq TT_L \\ 0 & TT_j > TT_L \end{cases} \\ LAI_{j+1} & = & LAI_j + \delta LAI_j \end{cases} \tag{1.1}$$

où l'index j signifie que la valeur est au jour j et :

- TT représente le temps thermique.
- TT_M représente le temps thermique de maturité.
- TT_L représente le temps thermique de fin de la croissance foliaire.
- $TMIN$ et $TMAX$ représente les températures minimales et maximales journalières.
- T_{base} est la température de base du système.
- B représente la biomasse.
- RUE (Radiation Use Efficiency) représente l'efficacité d'utilisation du rayonnement
- PAR (Photosynthetically Active Radiation) représente le rayonnement photosynthétiquement actif.
- K, α représentent des paramètres.
- LAI (Leaf Area Index) représente l'indice de surface foliaire.
- LAI_{max} est la surface foliaire maximale.

Il faut noter que l'expression sur le calcul de la biomasse B provient de l'interprétation des travaux de Monteith (Monteith et Moss 1977).

La modélisation proposée en (1.1) est une modélisation à base d'équations aux différences finies. Elle considère les valeurs du champ moyen sans fournir de spatialisation détaillée de l'information. Ainsi la température est une information moyennée sur la surface considérée et non pas une valeur spatialisée sur l'ensemble du champ (i.e. un maillage représentant les variations de températures) comme cela pourrait être le cas avec une version de la modélisation avec des dérivées partielles (PDE). Une illustration de cette différence se trouve en 1.14. Cela dit, une discrétisation spatiale et une résolution sur chaque cellule serait également possible.

Nous allons procéder à une réécriture de ces équations afin de regrouper les différents termes en fonction

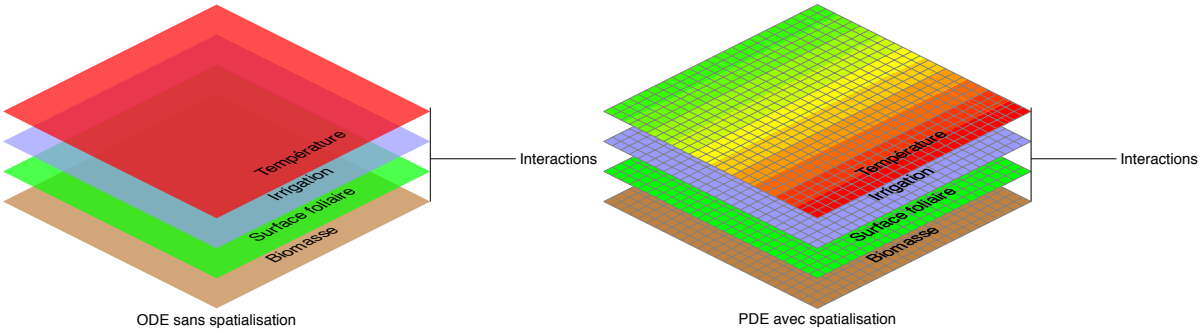


Figure 1.14 – un modèle non-spatialisé et un modèle spatialisé

de leur type. Cette réécriture donne :

$$\begin{cases} \delta TT_j & = & max(\frac{U_1(j)+U_2(j)}{2} - P_4, 0) \\ X_1(j+1) & = & X_1(j) + \delta TT_j \\ \delta B_j & = & \begin{cases} P_2 * U_3(j) * (1 - e^{-P_1 * X_3(j)}) & X_1(j) \leq P_6 \\ 0 & X_1(j) > P_6 \end{cases} \\ X_2(j+1) & = & X_2(j) + \delta B_j \\ \delta LAI_j & = & \begin{cases} P_3 * \delta TT * X_3(j) * max(P_5 - X_3(j), 0) & X_1(j) \leq P_7 \\ 0 & X_1(j) > P_7 \end{cases} \\ X_3(j+1) & = & X_3(j) + \delta LAI_j \end{cases} \tag{1.2}$$

où :

- $X = [TT, B, LAI]$
- $U = [TMIN, TMAX, PAR]$
- $P = [K, RUE, \alpha, T_{base}, LAI_{max}, TT_M, TT_L]$

TT, B et LAI sont les **variables d'état du système**, PAR, $TMIN$, $TMAX$ sont les **variables d'environnement du système**, K, RUE, T_{base}, α, LAI_{max}, TT_M sont les **paramètres du système**. Nous pourrons parfois parler de **variables externes** ou **variables de contrôle** au lieu de variables d'environnement par la suite.

Le modèle (1.2) est un modèle **déterministe**.

Néanmoins, il est plus réaliste de considérer des bruits de modélisation pour certains phénomènes. Ainsi la biomasse peut subir des variations dues à des facteurs externes inconnus. Lorsque ce formalisme est introduit, nous parlerons de modèles *stochastiques*. Nous proposons une reformulation de (1.2) en une version stochastique en (1.3) en introduisant un vecteur E d'erreur de modélisation.

$$\begin{cases} \delta TT_j & = & max(\frac{U_1(j)+U_2(j)}{2} - P_4, 0) \\ X_1(j+1) & = & X_1(j) + \delta TT_j \\ \delta B_j & = & \begin{cases} P_2 * U_3(j) * (1 - e^{-P_1 * X_3(j)}) * (1 + b_1(j)) & X_1(j) \leq P_6 \\ 0 & X_1(j) > P_6 \end{cases} \\ X_2(j+1) & = & X_2(j) + \delta B_j \\ \delta LAI_j & = & \begin{cases} P_3 * \delta TT * X_3(j) * max(P_5 - X_3(j), 0) * (1 + b_2(j)) & X_1(j) \leq P_7 \\ 0 & X_1(j) > P_7 \end{cases} \\ X_3(j+1) & = & X_3(j) + \delta LAI_j \end{cases} \tag{1.3}$$

où :

- $X = [TT, B, LAI]$
- $U = [TMIN, TMAX, PAR]$
- $P = [K, RUE, \alpha, T_{base}, LAI_{max}, TT_M, TT_L]$
- $E = [b_1 \sim N(0, \sigma_1^2), b_2 \sim N(0, \sigma_2^2)]$

Dans le modèle 1.3, b_1 et b_2 sont des bruits de modélisation suivant des distributions normales et permettant de faire varier la production de biomasse et la surface foliaire selon des phénomènes aléatoires.

Parmi les modèles mécanistes dont la production de biomasse est à l'échelle du champ (ou par unité de surface) nous pouvons citer les modèles : CERES (Jones, Kiniry et Dyke 1986), PILOTE (Mailhol, Olufayo et Ruelle 1997), STICS (Brisson 2008), SUNFLO (Casadebaig et al. 2011), LNAS (Cournède et al. 2013). Le modèle CERES se distingue par son calcul d'indice foliaire basé sur les organes et non par m^2 comme c'est le cas avec les autres. À ce titre il se rapproche du calcul fait dans le modèle GreenLab que nous décrivons dans la section suivante.

1.2.3 Modélisation mécaniste à l'échelle de la plante

Un autre type de modélisation est possible lorsque l'on cherche à comprendre et quantifier les biomasses au niveau des organes de la plante. C'est par cette volonté de compréhension de la répartition de la biomasse au sein des différents compartiments que le modèle Greenlab (De Reffye et Hu 2003) a été créé. Contrairement aux précédents modèles que l'on peut qualifier de modèles **fonctionnels**, GreenLab est un modèle dit **structure-fonction** ((Sievänen et al. 2000),(Vos et al. 2010)). C'est à dire décrivant à la fois le fonctionnement écophysiologique de la plante et son développement architectural. La structure de la plante est représentée par un graphe. Ce graphe est une version "compressée" de l'arbre (au sens arbre informatique) qui existerait si nous donnions un identifiant unique à chaque noeud de la plante considérée. En effet, il a été démontré que cette approche permet de mettre en place une factorisation des structures sous-jacentes au moyen d'une clé composée de l'âge ontologique, chronologique et physiologique. ((Cournède et al. 2006),(Smolenova, Kurth et Cournède 2012))

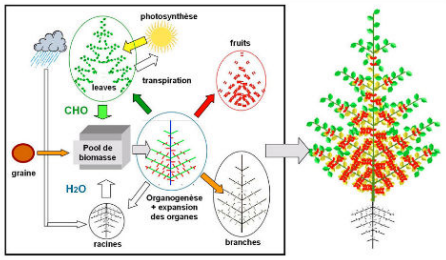


Figure 1.15 – cycle de croissance pour le modèle structure-fonction GreenLab : production de biomasse et répartition dans les différents organes. L'architecture est prédite à chaque cycle de simulation.

La figure 1.15 représente les phénomènes qui sont pris en compte à ce niveau de modélisation. On peut voir au centre la stucture de données sous forme d'arbre qui contient les informations pour les différents organes (entrenœuds, feuilles, fruits, ...).

La communauté des modèles FSPM compte un grand nombre de modèles. Citons par exemple LIGNUM (Perttunen et al. 2002), ADEL (Evers et al. 2005), ou encore MappleT (Costes et al. 2008)

En ce qui nous concerne, ce type de modèles à eu une importance particulière lors du design de la plateforme étant donné le potentiel grand nombre de noeuds du graphe lorsque l'on simule des arbres et plus encore lorsque l'on simule une population d'arbres i.e plusieurs instanciations de modèles d'arbres afin de simuler un peuplement, comme une forêt. C'est pourquoi nous avons dû introduire un mécanisme permettant de nettoyer la mémoire en fonction de la complexité du modèle. Nous verrons cela au chapitre 2.

Afin d'évaluer et valider un modèle, il est nécessaire de le confronter à des données réelles. Nous décrirons dans la section suivante les caractéristiques des données expérimentales pour les modèles de cultures ou les FSPM et les modes d'acquisition.

1.2.4 Quelles sont les propriétés des données expérimentales en agronomie ?

Des campagnes de prélévement sont organisées afin d'acquérir des données expérimentales sur ces systèmes. Ces campagnes ne sont pas régulières dans le temps (**irrégularité des données**) et ne portent pas forcément sur les mêmes données d'une date à une autre (**hétérogénéité des données**). De plus l'acquisition étant un processus coûteux, l'expérimentation n'est pas réalisée de très nombreuses

fois sur la durée de vie du système (**rareté des données**)

> Ces trois caractéristiques : **irrégularité**, **hétérogénéité**, **rareté** sont déterminantes pour la construction des fonctions d'observation du modèle.

Sur le terrain, nous pouvons acquérir des données par le biais de méthodes non-destructives (LAI-mètre, radar pluviométrique, satellites) ou destructives (prélèvements sur les plantes d'une partie ou destruction du tout pour l'analyse). Pour les modèles individu-centrés, il est important d'avoir en tête le caractère destructif qui implique que tout suivi ne pourra correspondre qu'à un modèle de plante moyenne.

Lors de l'acquisition de ces données, nous faisons des **erreurs** dans les valeurs reportées soit parce que l'instrument n'est pas assez précis, soit parce que l'opérateur commet des erreurs lors d'une campagne qui peut être à la fois longue et fastidieuse. Ces **erreurs** seront modélisées par des bruits d'observations lors de la projection sur le cadre mathématique que nous décrirons plus loin.

jour	biomasse sèche totale (g/m^2)	biomasse de la tige (g/m^2)	biomasse du grain (g/m^2)	indice de surface fiolaire
80	1.871	.	.	.
122	6.21	.	.	.
162	.	.	.	0.09
165	19.47	.	.	.
175	.	.	.	1.04
191	.	.	.	2.14
197	379.111	231.451	.	.
212	.	.	.	3.51
219	459.852	303.180	.	4.37
233	.	.	.	5.21
248	1718.137	881.778	.	.
266	1608.259	492.789	829.933	.

Table 1.1 – données blé Villamblain Raffy 2013

Le tableau 1.1 nous donne un aperçu des données agronomiques collectées. Une lecture rapide rend compte des caractéristiques énoncées plus haut. En effet nous avons 4 types de données (*hétérogénéité*), la biomasse de la tige par exemple n'est donnée que sur 4 points (*rareté*), il n'y a pas de période bien établie entre les différents jeux de données (*irrégularité*).

1.2.5 Quels objectifs afin d'améliorer les systèmes de culture et les itinéraires techniques ?

Si nous nous plaçons dans une optique de l'amélioration des systèmes de culture et des itinéraires techniques, nous pouvons nous pencher entre autre sur trois problématiques terrains :

- la prévision de rendement
- le contrôle optimal des cultures
- la sélection variétale

1.2.5.1 Prévision de rendement

La prévision de rendement est généralement faite sous une semaine, un mois ou un cycle de culture en prenant en compte l'historique des variations sur une ou plusieurs parcelles et plusieurs points dans le temps.

Cette prévision peut se faire à partir d'outils du machine learning sans modèle ou bien avec les modèles

mécanistes. La figure 1.16 montre une courbe de simulation d'un modèle mécaniste vis à vis des points de données expérimentales qui ont servi à la calibration du modèle.

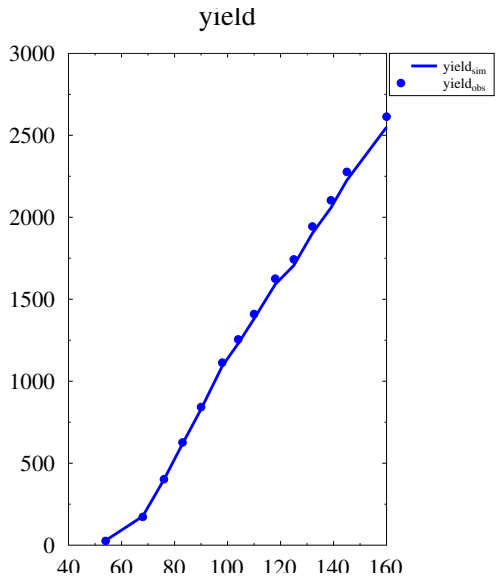


Figure 1.16 – point de données expérimentales versus courbe de simulation

On peut ainsi fournir des outils pour évaluer les risques de déficit de rendement d'une parcelle donnée par exemple.

1.2.5.2 Contrôle optimal des cultures

Le contrôle optimal des cultures consiste à trouver la politique de gestion d'intrants optimisant une certaine fonction critère. Cette fonction critère peut être le chiffre d'affaires lié au rendement diminué du coût de l'apport d'intrants.

Cette optimisation peut être réalisée sous contraintes, par exemple un quota d'eau disponible ou une limitation réglementaire sur les apports de fertilisants.

1.2.5.3 Sélection variétale

La sélection variétale est une activité qui peut être divisée en plusieurs catégories :

- utilisation de la sélection naturelle pour diffuser de nouvelles espèces jusqu'alors inconnues (cas très rare)
- utilisation de la sélection artificielle conservatrice (Pitrat et Foury 2003) avec les lignées et variétés hybrides
- utilisation de la sélection artificielle créatrice (Pitrat et Foury 2003) avec la mutagènese ou la transgénèse

Notre approche mathématique permet de simuler un grand nombre de fois les rendements pour un génotype donné en faisant varier les conditions environnementales et en conséquence de tester de nouvelles variétés à moindre coût. Cette étape nécessite des recherches plus approfondies dans le lien existant entre les paramètres des modèles de croissance et les modèles de la génétique quantitatives. (Letort et al. 2008)

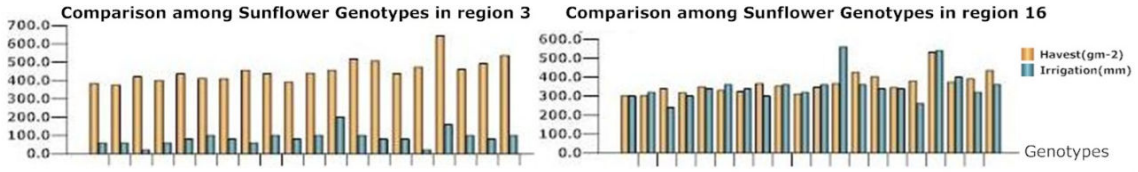


Figure 1.17 – comparaison des rendements potentiels de différents génotypes de tournesol et de la consommation en eau nécessaire dans 2 régions européennes contrastées (Kang 2013)

La figure 1.17 montre un exemple d'expérimentation numérique à partir du modèle SUNFLO (Lecoeur et al. 2011) où l'on étudie différents génotypes en fonction de la localisation pour savoir quelle est la variété la plus adaptée au sein d'une espèce par rapport à l'environnment (Kang 2013). On peut ainsi voir très clairement que certaines variétés bien qu'ayant de forts rendements en région 3 et un faible coût d'irrigation deviennent très désavantageuses dans une autre région car bien que le rendement soit important il a fallu énormément les irriguer.

Nous avons vu quelles sont les spécificités de modélisation et simulation dans le cadre de l'agronomie et quelles sont les domaines d'applications des modèles. Nous allons maintenant étudier plus en détails les problématiques de cette thèse.

1.2.6 Bilan

Après avoir vu les problématiques générales relatives à la modélisation des systèmes complexes, nous nous sommes concentrés sur les problématiques en agronomie de la modélisation mécaniste. Nous avons vu que ce type de modélisation peut être intéressante pour tout ce qui est relatif au domaine des outils d'aide à la décision, notamment en mettant en place de la prévision de rendement, de l'optimisation des pratiques culturales ou encore de la sélection variétale. Cependant, pour qu'un modèle passe du simple cadre de l'hypothèse à un outil utilisé en production, il faut prévoir tout un cadre d'analyse nécessitant une batterie d'algorithmes afin de pouvoir vérifier et valider un ensemble de modèles vis à vis d'une problématique. Nous allons voir dans la section suivante quel est l'état de l'art en matière de modélisation de ces systèmes ainsi que de leur analyse avant de formuler une problématique dont nous présenterons une réponse dans la suite de ces chapitres.

1.3 Problématiques

1.3.1 Contexte

Comme nous l'avons vu au cours de ce chapitre, notre champ d'étude réside dans un domaine où interagissent la biologie, les mathématiques et l'informatique.

En fonction des espèces (que ce soit le maïs, l'arabidopsis ou le caféier) ou des utilisations (prévision de rendement, gestion des stress, caractérisation génotypique, ...) de nombreux modèles sont développés par les biologistes et mathématiciens. Nous pouvons formaliser ceci en disant que nous avons N modèles produits par la communauté scientifique.

De plus ces modèles doivent être analysés grâce aux bonnes pratiques de la modélisation afin de les évaluer et de garantir leur domaine de validité. Pour mettre en place cette bonne pratique de la modélisation la communauté scientifique a créé M algorithmes comme l'analyse de sensibilité ou l'estimation de paramètres.

L'objectif est le développement d'un outil permettant d'utiliser ces M algorithmes pour n'importe lequel de ces N modèles.

Par exemple, le logiciel Digiplant, développé précédemment dans l'équipe, (Cournède et al. 2006,Cournède et al. 2011) est un logiciel assez performant pour la simulation et différentes méthodes d'estimation paramètrique, mais s'applique au seul modèle GreenLab. D'un autre côté, les logiciels OpenAlea (Pradal et al. 2015) ou GroIMP (Kniemeyer et Kurth 2008) permettent de développer des modèles de type FSPM de façon relativement simple, mais ne permettent que leur simulation. Nous cherchons donc à concevoir un système nous permettant de gérer $N + M$ éléments au lieu de $N * M$ éléments pour l'analyse des modèles.

À cela, s'ajoute la question de la complexité de l'outil informatique afin de mettre en place quelque chose de simple d'utilisation mais aussi performant en exploitant à minima les architectures multi-threadés voire distribuées.

Nous trouvons à ce jour plusieurs méthodes ou logiciels permettant d'aborder ces différentes problématique.

1.3.1.1 Systèmes informatiques pour la modélisation des systèmes complexes

L'ensemble des systèmes informatiques que nous présentons par la suite ne représente qu'une partie des logiciels existants. Nous présentons ici des candidats potentiels qui ont été évalués sur la base de critère concernant l'installation, la configuration, les paradigmes pris en comptes dans le cadre du développement des algorithmes nécessaires à l'analyse.

Cependant, nous pourrions citer des logiciels comme MATLAB/SIMULINK (MATLAB 2010), SCICOS (Campbell, Chancelier et Nikoukhah 2006a) qui sont présents dans la communauté de la modélisation et simulation des systèmes. Toutefois, notre cadre stochastique ainsi que la complexité d'écriture des modèles ne nous permet pas de prendre ces outils de référence comme point de départ.

Les outils qui ont été considérés sont donnés ci-dessous :

- MODELICA (Elmqvist et Ab 1997) est un langage de modélisation et une plateforme de modélisation dédiés en premier lieu aux systèmes physiques continus. On y trouve un grand nombre de modules notamment pour les systèmes industriels. Il n'a pas été choisi en raison de son approche sur la stochasticité (à notre connaissance il n'est pas possible en l'état de coder de tels systèmes avec le langage d'origine) et le fait de ne pouvoir coder facilement des structures sous forme de graphes nécessaires au développement de modèles de type FSPM. Il est développé par une organisation indépendante. Une version commerciale est maintenue notamment par DASSAULT SYSTÈMES sous le nom DYMOLA.
- RVLE (Quesnel, Duboz et Ramat 2009) qui s'appuie sur le formalisme DEVS (Zeigler et al. 1995) et le logiciel de statistique R (R Core Team 2013). C'est la base de la solution développée à l'INRA connue sous le nom de RECORD (Bergez et al. 2013). Cette solution n'a pas été retenue car d'une part elle est apparue pendant les développements et d'autre part le formalisme de modélisation n'est pas facilement adaptable aux méthodes mathématiques que nous développons (cf. CPF en section 3). Un autre défaut de la plateforme concerne les performances de R (Viaud et al. 2015) ainsi que la complexité de mise en œuvre de l'ensemble au démarrage du projet.
- GROIMP (Kniemeyer et Kurth 2008) est une plateforme qui permet de développer des modèles structure-fonction. Un partenariat avec l'équipe de recherche a permis la création d'un modèle de type GreenLab dans cette plateforme. Elle est développée par l'université de Göttingen.
- OPENALEA (Pradal et al. 2015) est une plateforme logicielle écrite en python et reposant sur le paradigme de programmation visuelle afin d'aider les modélisateurs à construire leurs modèles. Cette plateforme permet d'exploiter des packages python pour la modélisation, simulation et visualisation. Elle est développée par l'INRIA, le CIRAD et l'INRA.

1.3.1.2 Systèmes informatiques pour l'analyse des modèles mathématiques

De même que pour la modélisation et simulation des systèmes complexes, nous présentons ici la liste des bibliothèques et/ou programme qui ont été considérés afin de répondre à notre problématique.

Cependant, cette liste pourrait être étendue à des outils comme JMODELICA (Åkesson et al. 2010), OPT++ (Meza et al. 2007), IPOPT (Wächter et Biegler 2006) qui sont des bibliothéques et outils du domaine de l'optimisation.

La liste des bibliothèques et outils considérés est :

— URANIE (**5283986**) est une bibliothèque C++ d'analyse d'incertitude et de sensibilité construite autour de ROOT (Brun et Rademakers 1996) développée par le CEA.
— OPENTURNS (Baudin et al. 2015) est une bibliothèque permettant de conduire des analyses d'incertitude développée par EDF.
— VERDANDI (Chapelle et al. 2013) est une bibliothèque C++ permettant de faire de l'assimilation de données développée par l'INRIA.

Ces différents logiciels permettent plus ou moins facilement d'utiliser leurs méthodes d'analyse sur des modèles du type qui nous intéresse (système dynamique discret stochastique), mais ne couvre chacun qu'une partie restreinte de la chaîne de bonnes pratiques de modélisation que nous souhaitons mettre en place. (cf. figure 1.7)

1.3.1.3 Systèmes informatiques pour le flux des bonnes pratiques de modélisation

À ce jour nous avons trouvé trois plateformes qui ont pour objectif de regrouper les différentes étapes des bonnes pratiques de modélisation.

— OPENMOLE (Reuillon, Leclaire et Rey-Coyrehourcq 2013) est un logiciel permettant de faire tourner des plans d'expérience sur un cluster pour conduire des analyses d'incertitude et de sensibilité. Il est développé initialement par l'institut des systèmes complexes de Paris.
— FME (Soetaert et Petzoldt 2010) est un paquet pour le logiciel R qui se veut générique afin de faire de l'estimation paramétrique, de l'analyse de sensibilité et de l'analyse d'incertitude. Il est développé par une équipe du "Royal Netherlands Institute for Sea Research"
— DAKOTA (Eldred et al. 2007) est une bibliothéque C++ et un ensemble d'outils dédiés aussi à cette chaîne. Il possède notamment un système de fichiers de configuration qui permet de manipuler l'ensemble avec un simulateur de type boîte noire, ce système permet donc d'étudier des modèles non prévus par la plateforme. Cependant, à notre connaissance, il ne permet pas d'étudier les états cachés par exemple étant donné la construction de la plateforme. L'ensemble est développé par "Sandia National Laboratories".

1.3.2 Formulation

D'une façon très générale, nous pouvons formuler notre problématique de la façon suivante :

> Comment améliorer les modèles et le processus de modélisation afin de valider un modèle, vérifier sa robustesse, valider son utilisation dans un cadre expérimental précis, et sélectionner le modèle mécaniste le plus pertinent vis à vis d'un objectif d'utilisation tout en maximisant, à la fois la diffusion au sein de la communauté scientifique, mais aussi en permettant d'exploiter les ressources de calcul de manière simple ?

Dans le cadre de la modélisation de la croissance des plantes, nous nous restreignons à certains types de modèles (discrets en temps, continus en espace, déterministes ou stochastiques) et que la validation et vérification se fait au moyen des méthodes issues des bonnes pratiques de modélisation telles que définies dans (Scholten et al. 2000) et composé de l'analyse de sensibilité, l'analyse d'incertitude, l'estimation paramétique, ...

Cette problématique générale peut alors se décomposer en sous-problématiques :

- ”Quels sont les éléments de vocabulaire, la sémantique et l’arbre de syntaxe nécessaires à la création d’un langage dédié pour la simulation et l’analyse quantitative des modèles ?”
- ”Comment coupler N modèles à M algorithmes afin d’éviter de redévelopper chaque algorithme pour chaque modèle, ce qui est souvent la norme dans les développements actuels ?”
- ”Comment gérer l’aléatoire dans le cadre des modèles et des algorithmes ?”
- ”Comment mettre en place un flux de travail permettant de sélectionner le bon modèle vis à vis d’une problématique tout en garantissant son domaine de validité ?”
- ”Comment réduire la courbe d’apprentissage et l’opérationnalité d’un nouveau modélisateur ?”
- ”Comment exploiter les architectures modernes de calcul afin de pouvoir gérer la montée en charge et ainsi renforcer la performance et par extension la robustesse de la chaîne de travail ?”

Par sa nature même, la problématique mêle à la fois mathématiques appliquées et développement informatiques afin d’établir un pont entre le cadre formel et son utilisation pratique dans un langage informatique.

Nous pouvons retrouver dans la figure 1.18 l’ensemble des concepts présentés ci-dessus tout en ayant une visualisation de la problématique par rapport à l’espace de recherche global de l’équipe Digiplante, et plus généralement de la communauté scientifique en modélisation de la croissance des plantes.

Nous allons répondre à cette problématique en présentant nos travaux sur la formalisation et le développement d’une bibliothèque C++. Cette bibliothèque a été nommée PYGMALION au sein de l’équipe. Ce nom est l’acronyme de ”**P**lant **G**rowth **M**odelling **A**na**L**ysis **I**dentification **O**ptimization Co**N**trol”. Nous le verrons par la suite mais cette bibliothèque ne s’applique pas uniquement aux modèles de croissance des plantes. Cet objet a permis de fédérer le travail de l’équipe autour des concepts et des outils qui vous seront proposés par la suite permettant la mise en place des bonnes pratiques de modélisation.

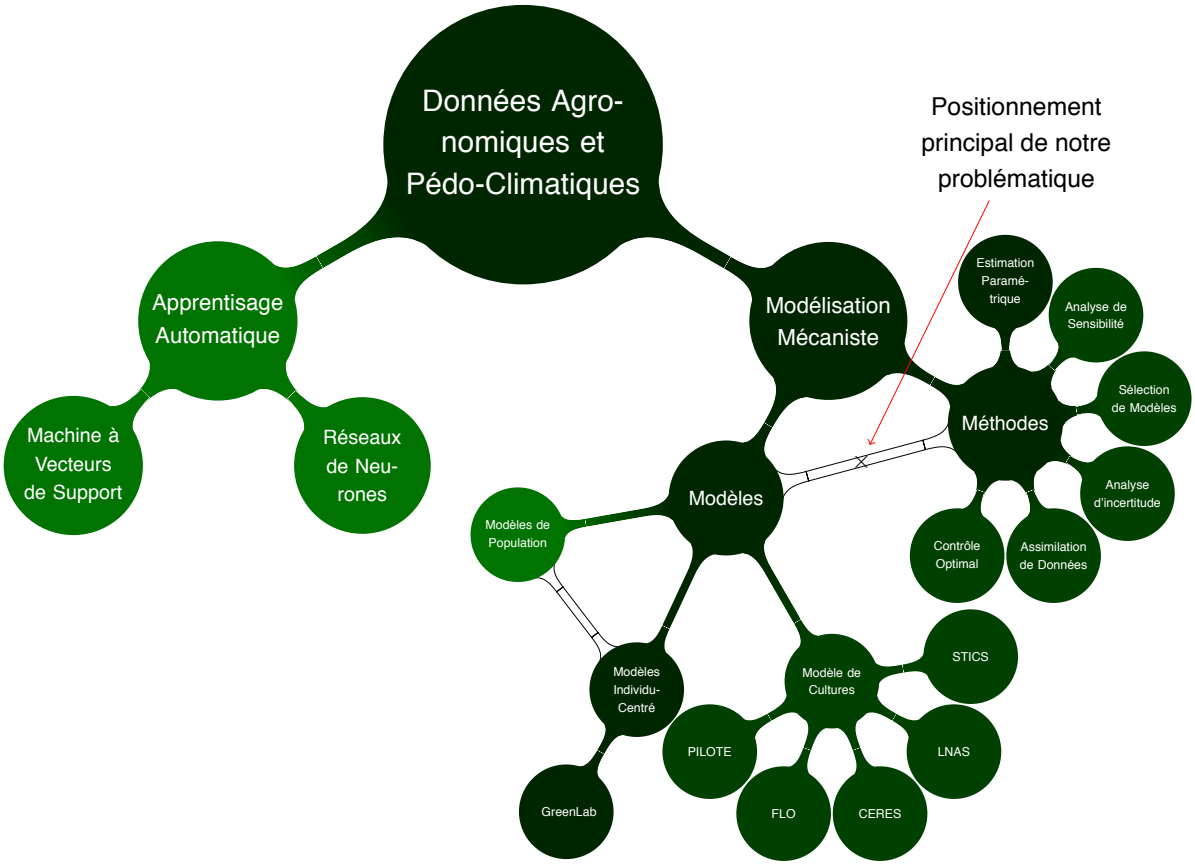


Figure 1.18 – visualisation des concepts rencontrés dans le domaine de l'agronomie quantitative et sujets explorés au sein de l'équipe Digiplante.

1.4 Conclusion

Dans ce chapitre, nous avons posé les bases de la modélisation mathématique des systèmes complexes. Nous avons vu que l'on peut approcher la chose soit par une modélisation purement statistique, soit par une modélisation mécaniste. Nous avons restreint notre champ d'étude à la modélisation mécaniste étant donné que c'est le seul type de modélisation qui permet de formuler des hypothèses vis à vis du fonctionnement interne des systèmes.

Nous avons aussi vu que dans le cadre de la modélisation des plantes, nous pouvons avoir affaire à différents types de modélisation selon la nature du système considéré. Toutefois, dans le cadre de nos travaux de recherche nous ne sommes pas encore penchés sur les notions telles que la modélisation multi-échelle ou multi-formalisme afin de nous concentrer sur les bonnes pratiques de modélisation. Nous considèrerons uniquement les systèmes dynamiques discrets à espaces d'état continus, stochastiques ou déterministes afin de pouvoir formuler un cadre de travail permettant d'apporter une solution à la problématique de recherche que nous avons précisée. Toutefois, ce sont des questions à garder en mémoire pour formuler des hypothèses quant aux perspectives de ces travaux.

Les travaux concernant notre cadre de recherche sont multiples car nous touchons à la fois à des notions telles que la modélisation et simulation des systèmes, l'analyse des modèles mécanistes et la stochasticité.

Nous avons présenté des approches qui ont alimenté notre cadre de réflexion quant à la modélisation et simulation, ainsi que l'analyse et nous verrons par la suite l'état de l'art concernant la stochasticité lorsque nous mettrons en place le cadre conceptuel concernant les simulations.

Chapitre 2

Modèles, noyau, simulation et langage

Dans ce chapitre nous décrirons les modèles utilisés d'un point de vue mathématique et informatique afin d'introduire le vocabulaire et la sémantique nécessaires au développement d'un langage de modélisation. Puis nous effectuerons quelques rappels sur la notion de stochasticité et ce que cela implique en termes de générateur aléatoire, pour finalement voir comment est implémenté le noyau et l'algorithme principal de simulation au sein de la bibliothèque C++.

2.1 Modèle mathématique

Dans cette section nous discutons du niveau de modélisation choisi pour nos systèmes biologiques ainsi que du cadre mathématique propice au développement des algorithmes d'analyse. Nous finirons par présenter le cadre informatique qui découle de toutes ces considérations en fin de section.

2.1.1 Différentes approches

La modélisation des systèmes biologiques peut se faire selon différentes approches en fonction de la nature des données et des processus mis en jeu mais aussi en fonction des objectifs de la modélisation. L'approche mécaniste permet de mettre en avant certaines hypothèses de fonctionnement et d'organisation dans les plantes. La communauté de la modélisation de la croissance des plantes développe des modèles à dfférentes échelles (Prusinkiewicz 2004) :

- échelle cellulaire (Bessonov, Crauste et Volpert 2011)
- échelle de la plante individuelle (Reffye et al. 2008)
- échelle de la population, vue comme un ensemble d'individu interagissant entre eux (Fournier et Andrieu 1999), (Cournède et al. 2008) ou comme population statistique (Baey et al. 2012)
- échelle du champ (Brisson 2008)

Nous restreignons notre étude aux 3 derniers niveaux en considérant le cadre des modèles de Markov cachés. Ce cadre nous permet de développer un ensemble de méthodes statistiques. Il faut noter que les variables entrant en compte dans les différentes interactions ne sont pas toutes de nature observable et nous aurons l'occasion par la suite d'introduire le concept de variables d'états cachés pour rendre compte de ce phénomène. Par exemple, nous pouvons avoir accès à la masse totale d'une plante (et encore, seulement par mesures destructives) mais par contre la production journalière de biomasse est plus délicate à observer.

2.1.2 Représentation d'état

Soient $(t_n)_{0\leq n\leq N}$, la suite des temps de discrétisation du système étudié (par exemple une discrétisation journalière), une première formulation des modèles peut être faite grâce à sa représentation d'état qui

est donnée en (2.1) :

$$\begin{cases} X_0 \text{ donné} \\ X_{n+1} = F_n\left(X_n, U_n, P, \varepsilon_n\right) \\ Y_n = G_n\left(X_n, P, \varepsilon'_n\right) \end{cases} \tag{2.1}$$

où :

- X_n est l'ensemble des variables d'états du système au temps t_n, $X_n \in \mathbb{R}^q$
- U_n est l'ensemble des variables de contrôle du système au temps t_n, $U_n \in \mathbb{R}^m$
- P est l'ensemble des paramètres du système, $P \in \mathbb{R}^p$
- ε_n est une variable stochastique correspondant au bruit de modélisation $\varepsilon = \{\epsilon_n\}_{1 \leq n \leq N}$ avec $\epsilon \in \mathbb{R}^d$
- F_n est la fonction de transition du système au temps t_n
- ε'_n est une variable stochastique correspondant au bruit d'observation $\varepsilon' = \{\epsilon'_n\}_{1 \leq n \leq N}$ avec $\epsilon' \in \mathbb{R}^{d'}$
- G_n est la fonction d'observation du système au temps t_n
- Y_n le vecteur des observations du système à valeurs dans $\mathbb{R}^y$.

Comme évoqué dans la problématique en 1.3.2, les systèmes biologiques que nous étudions ont la particularité que leurs observations sont :

- hétérogènes puisque nous pouvons observer autant le rendement que la surface foliaire,
- non-périodiques car nous n'avons pas des données de façon régulière,
- rares car en plus du manque de régularité le volume est faible.

La fonction globale d'observation du système G est composée de k fonction d'observations g_i dont les exécutions sont contraintes par la fonction Λ_{τ_i}. τ_i représente une timeline (ligne de temps) qui spécifie à quel temps nous devons activer g_i.

Nous réécrivons dés lors la fonction d'observation de la forme :

$$\begin{cases} G_n\left(X_n, P, \varepsilon'_n\right) & = & \left(g_i(X_n, P, \varepsilon'_n)\Lambda_{\tau_i}(n)\right)_{1 \leq i \leq k} \\ \Lambda_{\tau_i}(n) & = & \begin{cases} 1, & \text{si } n \in \tau_i \\ 0, & \text{sinon} \end{cases} \end{cases} \tag{2.2}$$

Cette représentation d'état est utile pour transmettre aux nouveaux modélisateurs le cadre dans lequel ils vont travailler et ainsi à la fois découpler et relier les différents compartiments du modèle.

2.1.3 Modèle de Markov caché

La description du modèle de Markov caché peut se retrouver dans (Trevezas et Cournède 2013) et (Cournède et al. 2013) et est reproduite ci-dessous en (2.3) :

$$\begin{cases} X_0 & \sim & \mu_p(.) \\ X_{n+1} | (X_n = x) & \sim & f_{n,P,U_n}(.|x) \\ Y_n | (X_n = x) & \sim & g_{n,P}(.|x) \end{cases} \tag{2.3}$$

avec :

- $(X_n)_n$ le vecteur des variables d'états au temps n (biomasse, surface foliaire, ...) considéré comme une variable aléatoire à valeur dans $\mathbb{R}^q$
- $P \in \mathbb{R}^p$ le vecteur des paramètres du système qui peuvent être d'origine génétique ((Tardieu 2003), (Letort 2008))
- $(U_n)_n$, les variables de contrôle dans $\mathbb{R}^u$, qui reprèsentent l'environnement comme la température, le niveau d'irrigation
- Y_n le vecteur des observations du système, variable aléatoire à valeurs dans $\mathbb{R}^y$.

Cette formulation a été montrée comme équivalente à la formulation en représentation d'état pour GreenLab dans (Trevezas et Cournède 2013), ou pour LNAS (Chen 2014).

La fonction d'observation et la transition principale du système sont stochastiques afin de représenter les différentes pertubations qui ne peuvent pas être approchées par des processus déterministes comme les bruits de modélisation ou d'observation.

Cette formulation est pratique pour la mise au point d'algorithmes statistiques. Cependant, dans le cadre de l'élaboration d'un modèle et de son évaluation par un ordinateur, on lui préferera sa représentation d'état.

Dans la prochaine section, nous montrerons les différentes étapes qui ont mené du modèle mathématique général à la création de la formulation informatique et de l'ensemble du langage qui en découle.

2.2 Modèle et noyau informatique

Dans cette section, nous introduisons les concepts nécessaires à la manipulation du modèle informatique précédent mais sous un angle plus pratique. Nous détaillons comment a été faite l'implémentation du noyau des fonctions nécessaires à l'écriture et au bon fonctionnement des modèles et des algorithmes.

Par noyau nous entendons donc :

- le cadre de développment d'un modèle
- les fonctions qui permettent de modifier ses structures de données associées
- les fonctions qui permettent d'accéder aux observations faites sur les systèmes que ceux-ci soient réels ou virtuels
- la fonction de simulation qui permet de récupérer les observations faites sur les systèmes virtuels durant leur évolution. La présentation de celle-ci sera déléguée à la section suivante étant donnée l'introduction de nouveaux concepts permettant de penser les simulations dans leur ensemble.

Les notions de vocabulaire et d'arbre de syntaxe sont présentées dans les sections 2.2.4 et 2.2.5 à des fins pédagogiques et de fluidité de la lecture. Il faut noter que ces deux sections pourraient être aussi lues en dehors de toute considération avec le langage c++.

Le but fondamental du système est à la fois de réduire la quantité de modèles et d'algorithmes à implémenter en faisant en sorte que le couplage soit fait par des interfaces et en même temps de pouvoir écrire le tout directement en c++ pour éviter d'une part la multiplicité des langages et en même temps être directement sous un langage performant. [1]

2.2.1 Vers la conceptualisation du modèle et du noyau informatique

Le passage de la formulation mathématique précédente à un cadre informatique générique permettant l'écriture des modèles, leurs simulation et leurs analyses s'est déroulé en plusieurs étapes.

Il y a eu plusieurs itérations qui ont permis un raffinement du modèle informatique en le confrontant à la fois aux besoins scientifiques et techniques (type d'algorithme, calcul parallèle, scripting, ...) mais aussi aux retours humains sur l'interprétation des mots décrivant l'ensemble du système.

Il faut simplement savoir pour la lecture de cette section que les développements ont commencé avec la norme "C++03" (ISO 2003), qu'ils ont continué et fortement évolué par l'utilisation d'Haskell [2], qui est un langage de programmation fonctionelle pure, et l'introduction de la norme "C++11" [3] (ISO 2011) qui met une emphase sur les paradigmes de programmation fonctionelle et de programmation générique.

Nous pouvons distinguer cinq phases dans la conception :

1. Évidemment la connaissance du langage dans la recherche de performance est une étape nécessaire
2. On trouvera notamment en annexe A le code présentant le noyau sous Haskell
3. Nous faisons un rappel rapide de certaines notions de cette norme en annexe E

phase	description
0	prototype avec la programmation orienté objet
1	plateforme avec la programmation orienté objet et quelques éléments de programmation générique
2	prototype d'un noyau de simulation minimal avec Haskell
3	prototype d'un noyau de simulation mninimal avec C++11 reprenant les concepts développés dans le prototype précédent en Haskell
4	plateforme en C++11 utilisant un mélange de fonctionel, générique, et orienté objet

La conceptualisation durant cette dernière phase a pu également être validée par l'introduction d'une plateforme écrite en Julia reprenant les mêmes éléments mais développée et maintenue par un autre membre de l'équipe.

Dans la phase 0, nous avons utilisé le paradigme de programmation orienté objet en décrivant l'ensemble du projet par classes. Ainsi comme le montre le diagramme de classes 2.1 nous avons une classe ”Model”, ”Parameters”, ”Control”, ”State”, ”Simulation”, ”SimulationList”, ...

Dans la phase 1, nous avons réécrit les éléments développés en phase 0 mais en supprimant certains éléments de la programmation orienté objet en privilégiant une approche générique. Ainsi on ne se retrouve plus à manipuler un objet dérivant d'une classe Model mais un type abstrait T qui possède des propriétés propres aux modèles (sous-types pour les variables d'état, paramètres, ...). Cette approche est celle utilisée par la bibliothèque standard de C++.

La phase 2 consiste en l'utilisation du langage de programmation Haskell et du paradigme de programmation fonctionelle pure qui lui est associé. Ce fut un bon exercice afin de simplifier l'ensemble des structures de données et concepts qui furent développés dans les phases précédentes.

Durant la phase 3, nous avons réimplémenté ces éléments conceptualisés précédemment dans le langage C++.

Finalement, dans la dernière phase, nous avons repris peu ou prou l'ensemble des élèments constitutifs de la première plateforme afin de fournir tous les outils nécessaires à la conduite de la chaîne de modélisation.

Il faut noter que l'écriture d'un langage et de tels outils ne peut pas être de nature linéaire. Ainsi, durant chacune de ces phases il y a eu de nombreux allers-retours avec l'ensemble des personnes travaillant soit sur les modèles, sinon sur les algorithmes afin de simplifier et raffiner l'ensemble à chaque fois. Nous allons passer en revue les deux architectures principales qui sont issues des phases 1 et 4 afin de compendre la différence en d'organisation.

2.2.1.1 Première architecture de la plateforme

Cette première architecture, représentée sur la figure 2.1, utilise les principes de la programmation orientée objets (Meyer 1988) ainsi que les patrons de conceptions classiques (Gamma et al. 1995) de ce domaine. La plupart des éléments constituant l'espace de modélisation et simulation sont représentés par des classes et l'héritage est mis en avant concernant la création d'un modèle. Ainsi tout modèle dérive de la classe mère ”Model”.

Cette architecture a permis notamment de mettre en oeuvre les premiers algorithmes et les premières études.

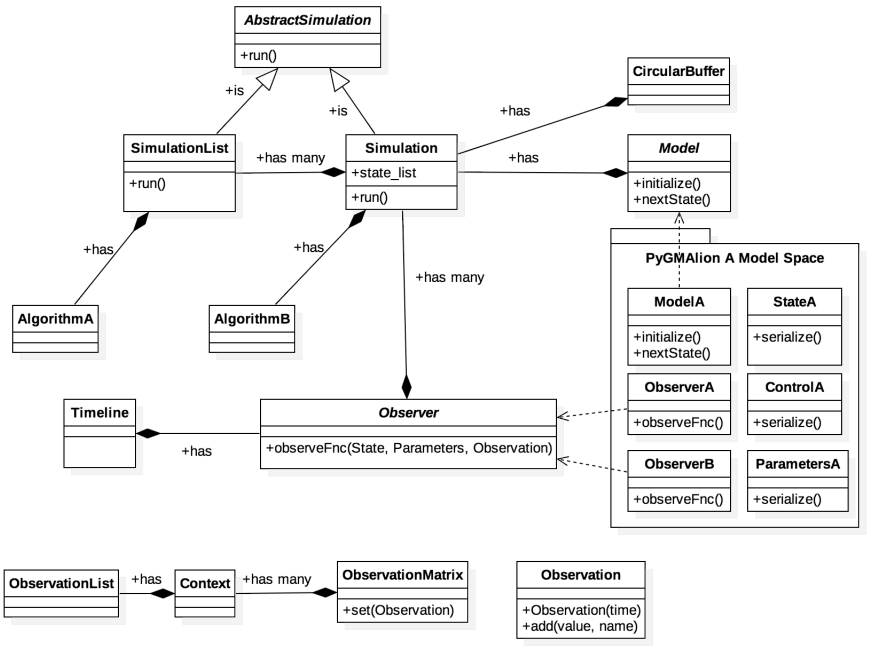


Figure 2.1 – première version de l'architecture uniquement orienté objet

Toutefois cette architecture possède des limites en terme de courbe d'apprentissage et de performance. Il n'est pas aisé de faire face à la complexité interne pour les modélisateurs et les concepteurs d'algorithmes de la plateforme par le grand nombre de liens entre les éléments. Une refonte de l'ensemble a donné naissance à la deuxième architecture en se basant plus fortement sur les notions de programmation fonctionelle (Hughes 1989). On peut trouver une version du simulateur déterministe en Haskell en annexe A.

Ce type de programmation s'est montré bien plus adapté dans notre cadre de modélisation d'un simulateur que la programmation orientée objets car cette dernière faisait mettre en place une trop grande complexité au niveau architectural.

Le simulateur est vu comme une simple fonction qui prend en entrée un modèle, une ou des expériences et un ou des jeux de paramètres. C'est cette fonction, qui cache en interne le mécanisme de parallélisation, qui sera utilisée par tous les algorithmes par la suite. Il y aura néanmoins une exception pour les algorithmes de prédiction-correction dont nous reparlerons au chapitre suivant.

2.2.1.2 Deuxième architecture de la plateforme

Sur la figure 2.2 il y a deux éléments centraux qui nous intéressent. Premièrement, la mise en place d'une interface de programmation pour la notion de modèle. En effet, nous avons voulu, pour cette deuxième architecture, pouvoir exécuter du code C++ compilé sous format binaire mais aussi de générer du code à la volée (Aycock 2003) à partir d'un langage de programmation dédié basé sur le langage que nous décrirons dans la suite de ce chapitre. Pour ce faire, nous avons mis en place une interface qui abstrait la notion de modèle et qui est utilisée par les concepteurs d'algorithmes dans l'ensemble de la plateforme.

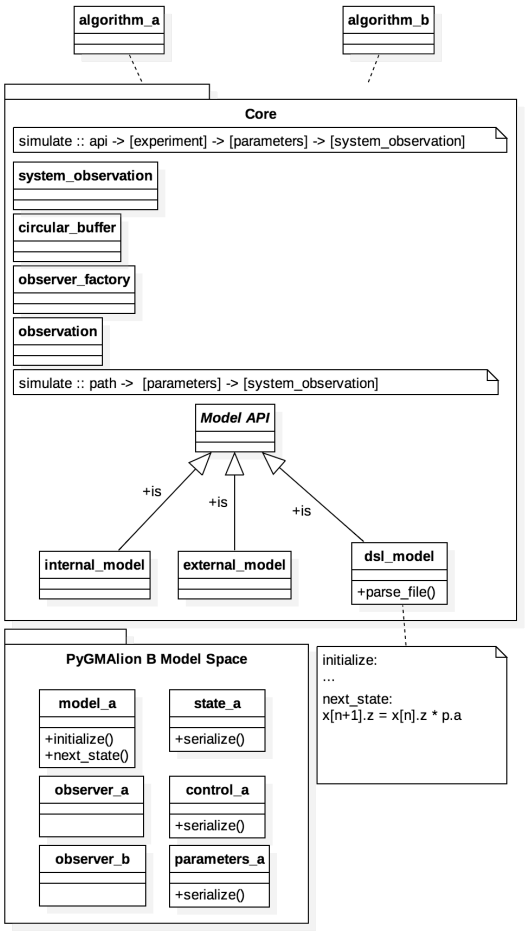


Figure 2.2 – deuxième version de l'architecture mélant objet, fonctionnel et générique

La plupart des éléments étaient déjà présents dans la première version. C'est uniquement la réécriture du noyau de simulation et de l'interfaçage des modèles par une API qui a permis de créer un effet de levier dans les développements concernant les modèles et les algorithmes, ainsi que l'utilisation générale du système.

Nous venons d'abord de présenter les différentes étapes ayant constitué la démarche de modélisation d'un système informatique permettant de conduire des études autour des bonnes pratiques de modélisation, avant de donner un aperçu de l'architecture générale de la plateforme actuellement utilisée qui est une instanciation de l'effort de conceptualisation fait autour de la représentation d'état des modèles mathématiques.

Nous allons voir plus finement par la suite à la fois le modèle informatique ainsi que la conception du noyau de simulation informatique.

2.2.2 Modèle informatique

Il faut rappeler ici que nous cherchons à créer un cadre pour améliorer l'interactivité de l'utilisateur avec le système plutôt que de chercher la composabilité en termes de temps ou de modules tels qu'elle est présente dans de nombreux formalismes existants (Elmqvist et Ab 1997 ; Zeigler et al. 1995). Nous voulons une cohérence de bout en bout de la chaîne de modélisation. Nous attaquons donc notre système par son interface avec les futurs utilisateurs et développeurs plutôt que par la structure même du calcul qui permettrait de faire de la gestion évènementielle ou de coupler différents systèmes de bases de temps différentes comme c'est le cas avec DEVS ou dans (Le Chevalier et Jaeger 2010).

2.2.2.1 Première formulation

En s'appuyant sur (2.1) nous proposons le formalisme explicite (Bayol, Chen et Cournède 2013 ; Bayol et al. 2015) suivant pour représenter une simulation

$$S = (T, X, U, P, Y, \varepsilon, \varepsilon', I, F, G) \tag{2.4}$$

où :

- T est la base de temps, dans notre cas cela représente les entiers
- X est l'ensemble des états
- U est l'ensemble des variables externes
- P est l'ensemble des paramètres
- Y est l'ensemble des valeurs de sorties
- ε est l'ensemble des bruits de modélisation tels que $\varepsilon = \{\epsilon_n\}_{1 \leq n \leq N}$ avec $\epsilon_n \in \mathbb{R}^d$
- ε' est l'ensemble des bruits d'observation tels que $\varepsilon' = \{\epsilon'_n\}_{1 \leq n \leq N}$ avec $\epsilon'_n \in \mathbb{R}^{d'}$
- $I : X \times P \to X$ est la fonction d'initialisation
- $F : T \times X \times U \times P \times \varepsilon \to X$ est la fonction de transition
- $G : X \times P \times \varepsilon' \to Y$ est la fonction d'observation

2.2.2.2 Modularité de la fonction de transition et d'observation

Si l'on peut se satisfaire de la lecture mathématique de la formulation (2.4), elle cache néanmoins des soucis d'ordre pratique. D'une part, les modèles décrits sont potentiellement composés de dizaines d'équations d'état et d'autre part, on souhaite obtenir le comportement du système à travers différentes fonctions d'observation en termes de variables observées ou de temps d'observation. Ainsi intervient une décomposition des deux grandes fonctions afin de faciliter la gestion de la complexité dans le cadre de la fonction de transition et de permettre la mise en place des fonctions d'observation du système de façon à mimer l'obtention de données rares, hétérogènes et irrégulières :

- la fonction de transition est elle-même décomposée en M sous-fonctions de même signature telle que $F = \{f_i : T \times X \times U \times P \to X | 1 \leq i \leq M\}$
- la fonction d'observation est décomposée en k "modèles d'observation" notés OM telle que $G = \{OM_i | 1 \leq i \leq k\}$ avec $OM_i = (\{g_i, \tau_i\})$ où g_i est la i-ème fonction d'observation de signature $X \times P \times \varepsilon' \to Y$. La partie modèle d'observation permet de gérer l'hétérogénéité tandis que la partie timeline permet de gérer le côté irrégulier et rare.

2.2.2.3 Formulation finale

Finalement on obtient :

$$S = (T, X, U, P, Y, \varepsilon, \varepsilon', I, F, G) \tag{2.5}$$

où :

- T est la base de temps
- X est l'ensemble des états
- U est l'ensemble des variables externes
- P est l'ensemble des paramètres
- Y est l'ensemble des valeurs de sorties
- ε est l'ensemble des bruits de modélisation tels que $\varepsilon = \{\epsilon_n\}_{1 \leq n \leq N}$ avec $\epsilon_n \in \mathbb{R}^d$
- ε' est l'ensemble des bruits d'observation tels que $\varepsilon' = \{\epsilon'_n\}_{1 \leq n \leq N}$ avec $\epsilon'_n \in \mathbb{R}^{d'}$
- $I : X \times P \to X$ est la fonction d'initialisation
- $F = \{f_i : T \times X \times U \times P \times \varepsilon \to X | 1 \leq i \leq M\}$ est la fonction de transition avec M modules

— $G = \{OM_i | 1 \leq i \leq k\} = \{(g_i, \tau_i) | 1 \leq i \leq k\}$ est la fonction d'observation

Nous venons donc de présenter le modèle informatique que nous utilisons dans l'ensemble du système. Nous allons maintenant voir les composants du noyau C++ qui est développé autour de ces concepts.

2.2.3 Pourquoi définir un langage dédié ? Quel est son périmètre ?

Dans la suite des travaux qui vous seront présentés, il y aura deux niveaux de lectures concernant les langages. Premièrement, nous avons reconstruit autour de la définition formelle de la représentation d'état le vocabulaire nécessaire à la manipulation de l'ensemble des éléments présents en modélisation et simulation à des fins de manipulation pour l'estimation paramétrique, l'analyse de sensibilité ou encore l'analyse d'incertitudes. Ensuite, nous avons pris un sous-ensemble de ces éléments de langages afin de fournir un langage dédié à la modélisation des systèmes dynamiques discrets stochatiques qui permet notament de manipuler plus facilement les définitions de lois de probabilités lors de la création des processus.

Ainsi, l'ensemble du noyau et des algorithmes ont été développés autour du vocabulaire et des types qui vous seront présentés dans la section suivante. Toute personne [4] qui a dû penser et développer un algorithme dans ce cadre a dû communiquer avec les autres sur la base de ce langage afin de réaliser les implémentations ou échanger les résultats d'expériences.

Le langage dédié de modélisation reprend ce vocabulaire afin de faciliter l'articulation de l'ensemble.

Pour résumer :

> Nous aurons affaire à un même langage dérivant du formalisme de la représentation en état et qui a été décliné en :
> — un langage dédié embarqué en C++ afin de simplifier la manipulation des structures de données et des fonctions de simulation lors de la rédaction des algorithmes.
> — un langage dédié à la modélisation afin de créer plus rapidement des modèles supportés par le formalisme de système dynamique à temps discret sous-jacent.
> — un langage dédié pour l'interface des programmes afin de manipuler un ensemble cohérent d'instructions quel que soit le type d'algorithme manipulé.

2.2.4 Vocabulaire, types et signature

Au fur et à mesure des itérations, nous avons affiné notre langage commun dans l'équipe pour ne finalement retenir que quelques éléments de vocabulaire et leur sens associé. Dans le tableau 2.1 nous regroupons les principaux éléments de langage. Ces éléments permettent de lever les ambiguités et seront utilisés lors du développement du langage de modélisation dédié en section 4.2.4.1.

Il faut noter que la principale approche dans l'élaboration du système informatique a été de bien définir les types, structures de données, et fonctions nécessaires au bon déroulement de l'ensemble. Ainsi nous avons essayé de respecter au mieux les hypothèses de base, à savoir :

— nous n'avons aucune connaissance a priori des types de données utilisées dans la construction des espaces d'états, de contrôles ou de paramètres. Cela peut être autant des réels, des vecteurs, ou des matrices, voire des graphes.
— Les algorithmes sont de différentes natures et on doit pouvoir utiliser la simulation de façon transparente vis à vis des données, i.e la signature de la fonction de simulation se doit d'être simple.

Les élements sont décomposés en 5 zones que l'on peut retrouver dans la table 2.1 :

— la zone relative aux types primitifs (en orange)
— la zone relative aux modèles de transition du système (en bleu)

4. L'ensemble des personnes qui ont été en contact direct avec la manipulation du langage sur l'ensemble de la thèse est d'environ 30 personnes

— la zone relative aux modèles d'observation du système (en vert)
— la zone relative à la simulation et aux nombres aléatoires (en rouge)
— la zone relative aux algorithmes de haut-niveau (en marron)

La table 2.1 est en anglais car c'est la langue utilisée pour écrire les éléments de vocabulaire ainsi que la spécification associée.

long-name	short-name	description
tuple	()	
list	[]	
integer	int	
real	real	
vector	vec	
matrix	mat	
string	str	
time	n	int
state	x	set of state variables
control	u	set of external variables
parameters	p	set of model and observation function parameters
noises	eps	set of model and observation noises
state-list	xl	[x]
initialize	i	initialization function x0(p) (x*u*p->x*u*p)
next-state	f	transition function of the dynamical system (n*xl*u*p*eps->x)
model	m	(initialize,next-state)=((x*u*p->x*u*p),(n*xl*u*p*eps->x))
observation	obs	list of values observed at n
observe		(x*p*eps->obs)
observation-model	om	(name,observe) = (str,(x*p*eps->obs)) ; the str is the name of the observation-model
timeline	tml	[int]
observer	o	(om,tml)
observer-list	ol	[o]
observation-function	of	ol
random-generator	g	once initialized it is a function ()->real
initial-state	x0	
initial-parameters	p0	
distribution	dist	(g->real) or (g->int) or (g->vec) ; concept of classical statistical distribution like uniform or normal
sampling-rule	sr	(str,dist) ; link the name of a parameter with a distribution and its parameters like ”a” and normal(0,0.05)
sampling-rule-list	srl	[sr] ; sometimes is noted ”univariate-rule-list” or ”multivariate-rule-list”
parameters-list	pl	[p]
pseudo-random-sampler	prs	prs ::p->[sr]->[p]
quasi-random-sampler	qrs	qrs ::p->[sr]->[p]
system-observation	so	data structure that stores observation per its properties. It can be seen as a [obs] but it has a more complex storage mechanism.

long-name	short-name	description
context	c	(x0,u) ; word that makes a link between ”experimental context” and ”initial conditions”
experiment	e	(c,of) ; common word for ”real experiment” and ”virtual experiment”
experimental-data	ed	(c,so) ; common word for ”experimental-data from real experiment” and ”experimental-data from virtual experiment aka simulation”
simulation	s	simulate :: m * [e] * [p] -> [so]
optimization-function	opt-fn	
weight-matrix	w	
configuration-file	cfg-file	
application-programming-interface	api	contains several C functions to create, read, update, delete x u p and to run i, f, g
configuration	C	configuration space of an algorithm (more related to the algorithm space , maybe needs an algorithm expert to be changed)
domain	D	domain space of an algorithm (more related to the dynamical system than the configuration, can be understood by an analyst)
result	R	result space of an algorithm
algorithm	A	C*D->R
functionnality	F	[A]
parameter-estimation	pe	F
sensitivity-analysis	sa	F
uncertainty-analysis	ua	F
model-selection	ms	F
data-assimilation	da	F
optimal-control	oc	F
command-line-interface	cli	interaction within a terminal
application	app	cli using A - generic
program	prg	cli using [A] - specific

Table 2.1 – table de vocabulaire

Au final nous pourrions dénoter `virtual_system` le tuple `(m,e,p)` de sorte que :

[system_observation] = simulate [virtual_system]

C’est aussi une façon de voir la grille ou cube de simulation qui vous sera présenté dans la section suivante. Cela permet aussi de voir directement le rapport entre système étudié ou réel et système virtuel étant donné que l’on a aussi :

[system_observation] = get_system_observation([experimental_data])

Dans la section suivante nous allons effectuer une visualisation de l’ensemble grâce à la visualisation de certains arbres syntaxiques abstraits.

2.2.5 Arbre syntaxique abstrait

Bien que nous ayons défini le vocabulaire et les liens entre les termes par le truchement des signatures de fonctions, listes et tuples, il peut être pratique de visualiser l’arbre syntaxique abstrait [5] généré dans

5. Nous utilisons ici une notion ”simplifiée” d’arbre syntaxique car nous nous concentrons uniquement sur la syntaxe du langage dédié sans faire intervenir la syntaxe du langage hôte, ici le C++

le cadre de certains exemples de notre activité.

Pour ce faire, nous proposons de regarder les éléments mis en jeu dans un code qui consiste à simuler 1000 systèmes virtuels :

```
1  //chargement d'un jeu de données pour les paramètres initiaux
2  auto p0 = load("p0.dat")
3
4  //création d'une expérience avec extraction de la fonction d'observation
5  //depuis les données expérimentales y_exp
6  auto el = create_experiment(c,y_exp);
7
8  //échantillonnage de 1000 jeux de paramètres
9  auto pl = sample(p0, {{"a","uniform",[1,2]}},{"b","uniform",[3,4]}, 1000);
10
11 //simulation de 1000 systèmes virtuels
12 auto yl = simulate(f,el,pl);
```

Code 2.1 – code de simulation de 1000 systèmes virtuels avec échantillonnage

Ce code met en jeu trois fonctions que sont `create_experiment`, `sample`, et `simulate`. On utilise aussi trois structures de données, à savoir les `system_observations` qui stockent les valeurs de `y_exp`, et des règles d'échantillonnage pour la fonction `sample` et enfin la notion de liste de systèmes virtuels pour la fonction `simulate`. Les trois fonctions présentées s'enchainent afin de créer les jeux de données nécessaires pour la simulation et stockent le résultat dans une valeur `yl`. Il faut noter que les variables suffixées d'un `l` dénotent une liste.

La figure 2.3 est une représentation des éléments mis en jeu lors de la création des paramètres à l'exécution de la ligne 1 du code présenté en 2.1

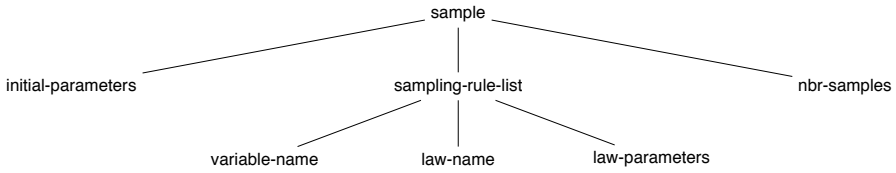


Figure 2.3 – arbre de syntaxe de la fonction sample

La figure 2.4 est une représentation des éléments mis en jeu lors de la simulation d'un modèle sur un ensemble d'expériences et de paramètres. Dans notre cas le `parameters-list` dans l'arbre est le résulat de la fonction `sample`.

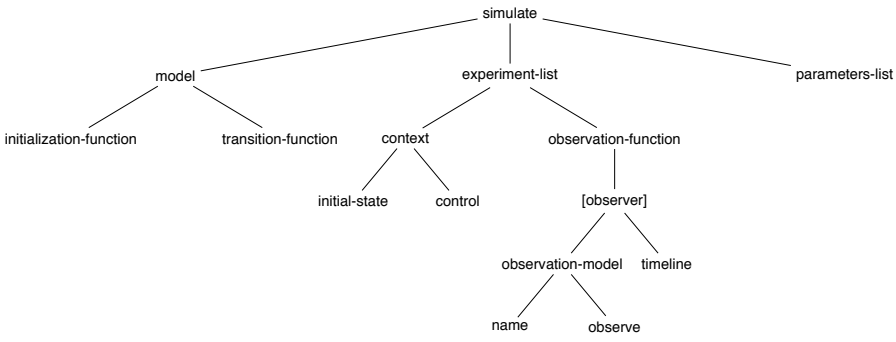


Figure 2.4 – arbre de syntaxe de la fonction simulate

Enfin nous retrouvons les éléments qui concernent la structure de données des observations du système.

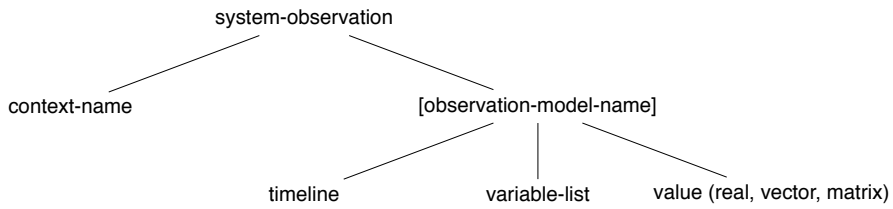


Figure 2.5 – abre de syntaxe de la structure de données pour les observations du système

2.2.6 Langage dédié embarqué en c++11

Nous avons vu dans la section 2.1 une définition des modèles que nous étudions. Si nous appliquons la formulation finale 2.2.2.3 au modèle classique Lotka-Volterra[6] (Lotka 1925 ; Volterra 1926 ; Volterra 1931) nous obtenons les écritures suivantes pour les structures d'états (code 2.2) , de variables externes (contrôles) (code 2.3) et de paramètres (code 2.4).

On notera de suite que la volonté de ce cadre de modélisation est de réduire l'ambiguité dans l'écriture des modèles en faisant en sorte de ne pas utiliser des vecteurs génériques de type `std::vector<double>` afin d'éviter des écritures du type `x[0] * p[42] - min(u[5], u[6])`. En effet le nombre de variables, paramètres et équations dans nos modèles d'intérêt étant potentiellement important, nous ne voulons pas augmenter l'ambiguité. Toutefois, les mécanismes introduits dans la bibliothèque permettent ce genre de formulation au moyen du polymorphisme statique introduit par les "types traits"[7] propres à la programmation en C++.

Traduisons les éléments présentés à partir du modèle Lotka-Volterra (AMS 1968). Définissons dans un premier temps les espaces d'états, de contrôles et de paramètres :

```cpp
struct state
{
  double x;
  double y;
};
//
//
```

Code 2.2 – structure d'états

```cpp
struct control
{
  //no control
};
//
//
```

Code 2.3 – structure de contrôles

```cpp
struct parameters
{
  double a;
  double b;
  double c;
  double d;
};
```

Code 2.4 – structure de paramètres

À partir de ces structures de données, nous pouvons définir la fonction d'initialisation et de transition principale du modèle (code 2.5). Dans ce code nous pouvons voir l'introduction des "traits" du modèle au moyen des `typedef` en début de structure. Ces éléments seront accessibles aussi au travers d'une classe d'interfaçage grâce à une structure de données `model_traits`. Les signatures des deux fonctions principales diffèrent par leur nature en termes de domaine d'arrivée. En effet, la fonction d'initialisation peut modifier tous les éléments qui lui sont fournis car ceux-ci ne seront pas `const` et de plus passés par référence. À l'inverse la fonction de transition respecte plus classiquement la notion de fonction mathématique car tous ses éléments sont `const` à l'exception faite du nouvel élément qui stockera le nouvel état du système.

6. voir annexe F
7. propriété des types. L'expression "types traits" est consacrée dans la communauté C++

```
struct model
{
  typedef state state_type;
  typedef control control_type;
  typedef parameters parameters_type;

  static void initialize(state_type & x0, control_type & u, parameters & p)
  {
    //no initialization
  }

  static void next_state(const int n,
  const state_list_type & xl,
  const control_type & u,
  const parameters_type & p,
  state_type & xnplus1)
  {
    xnplus1.x = ( p.a + 1) * xl[n]->x - p.b * xl[n]->x * xl[n]->y;
    xnplus1.y = (-p.c + 1) * xl[n]->y + p.d * xl[n]->x * xl[n]->y;
  }
};
```

Code 2.5 – modèle Lotka-Volterra sous pygmalion-c++

Il s'agit maintenant de donner des modèles d'observation afin de pouvoir regarder comment évolue le système. Nous avons vu dans la section précédente comment un modèle d'observation est une composante de la fonction d'observation du système. Il ne faut pas les confondre. Le modèle d'observation est dédié à une tâche et une évolution particulière dans le temps. Ainsi la fonction d'observation peut être dédiée à la fois à l'observation de la variable x et de la variable y mais à des temps différents. De plus l'observation de la variable x peut être bruitée par exemple c'est pourquoi nous avons préféré la notion de modèle pour ces éléments car cela véhicule l'idée de modèle statistique.

```
struct om1
{
  typedef state state_type;
  typedef parameters parameters_type;

  static void observe(const state_type & xn,
  const parameters_type & p,
  observation & obs)
  {
    obs.add("x", xn.x);
  }
};
```

Code 2.6 – modèle d'observation de la variable x pour Lotka-Volterra

2.2.7 Manipulation des données du modèle

Étant donné que nous sommes a priori incapables de connaître la structure de données associée à un modèle il fallait fournir un mécanisme permettant d'extraire des données de ces types (états, contrôles, paramètres) afin de les modifier en cours de simulation pour effectuer des estimations paramétriques par

exemple. De plus la projection de ces données n'est pas totale et seul un sous-ensemble doit pouvoir être sélectionné.

De ce fait, nous avons introduit un mécanisme qui s'inspire de la notion "Modèle-Vue-Contrôleur" (Krasner et Pope 1988) que l'on peut retrouver en programmation orientée objet.

La chose la plus importante à comprendre est que dans la notion "Modèle-Vue-Contrôleur" employée ici, la notion de "Modèle" n'est pas la même que celle employée plus haut afin de désigner les modèles de systèmes dynamiques. Ici la notion de "Modèle" se réfère à un modèle de données et correspond à la description des éléments présents dans les structures de données composant les modèles. Ainsi, il y a le modèle des états ou encore le modèle des paramètres. Il n'y a qu'un seul type de contrôleur pour l'instant qui met sous forme de vecteur une structure de données C++ munie d'une fonction de sérialisation.

Prenons l'exemple d'une estimation paramétrique pour laquelle nous souhaitons effectuer nos modifications uniquement sur 2 paramètres parmi les 4 existants. Nous avons donc une structure de données ayant 4 éléments :

```
struct P
{
  double a = 0.1;
  double b = 0.02;
  double c = 0.15;
  double d = 0.03;
};
```

Code 2.7 – structure de données à 4 éléments

Nous voulons faire une estimation sur "a" et "c". Nous allons donc récupérer une vue c'est à dire un sous-ensemble en lecture/écriture sur le jeu de données de la façon suivante :

```
//création d'un jeu de paramètres
auto data = parameters();

//création de la vue
auto view = create_view(data, {"a","c"});

//interrogation de la vue
std::cout << view[0].name << " " << view[0].value << std::endl;
std::cout << view[1].name << " " << view[1].value << std::endl;
```

Code 2.8 – affichage de deux éléments dans une structure de donnée

Le code 2.8 est en réalité incomplet car il y a un détail à préciser pour pouvoir faire la sélection qui consiste à introduire une fonction de sérialisation afin de renseigner quels sont les éléments qui peuvent être atteints par la fonction `create_view`. Ainsi la définition complète d'une structure de données se trouve être :

```
struct P
{
  double a = 0.1;
  double b = 0.02;
  double c = 0.15;
  double d = 0.03;
};

template <typename archive_type> auto serialize(archive_type & ar, data_name & x) -> void
{
  ar & make_property("a", x.a);
  ar & make_property("b", x.b);
  ar & make_property("c", x.c);
  ar & make_property("d", x.d);
}
```

Code 2.9 – déclaration complète d'une structure de données à 4 éléments

La deuxième fonction `serialize` est une fonction de sérialisation qui permet de mettre à plat la structure de données associée ainsi que de pouvoir munir l'ensemble de la plateforme d'un mécanisme mimant la réflexion. Toutefois, cette sérialisation peut être sélective et ne pas concerner l'ensemble des éléments constituant l'espace en cours.

C'est grâce à cette fonction que nous pouvons définir à l'exécution les paramètres qui seront utilisés lors des estimations ou des analyses.

2.2.8 Manipulation des données d'observation

Si la première version du logiciel s'est efforcée de respecter la manipulation des données à travers un système hiérarchique de type orienté objet, la dernière version s'inspire des syntaxes de types LINQ (Language-Integrated Query) (Cheney, Lindley et Wadler 2013) afin de fournir à l'utilisateur la notion de :

- requête
- fonction de recherche
- itérateur à travers un mécanisme de "range"

L'implémentation actuelle se borne à la syntaxe utilisateur puisque les classes sous-jacentes ne bénéficient pas d'un moteur complet d'interrogation des données.

Ainsi une requête prendra comme entrées :

- le nom d'un modèle d'observation *et/ou*
- un temps d'observation *et/ou*
- une variable observée

Prenons l'exemple d'une sortie de simulation d'un modèle Lotka-Volterra entre le temps 10 et 100 par pas de 10.

Si nous souhaitons récupérer toutes les valeurs nous ferons :

```
//loading model and creating data
auto api = load_model(...);
auto el = create_experiment_list(...);
auto pl = sample(...);

//simulating and getting all data
auto y = simulate(api, el, pl);
auto r = find(y, query());
```

Code 2.10 – récuperer toutes les données d'une observation d'un système

Si nous souhaitons récupérer une valeur d'un modèle d'observation précis sur une variable précise à un temps donné nous aurons cette formulation :

```
//loading model and creating data
auto api = load_model(...);
auto el = create_experiment_list(...);
auto pl = sample(...);

//simulating and getting specific data
auto y = simulate(api, el, pl);
auto r = find(y, query("om1", 4, "x"));
std::cout << r.first.value << std::endl;
```

Code 2.11 – affichage de la valeur "x" au temps 4 pour le modèle d'observation "om1"

La manipulation de la variable `r` se fait comme pour un vecteur au moyen d'itérateurs sauf que les bornes sont délimitées par les mots-clés `first` pour `begin()` et `second` pour `end()`.[8] Ainsi si nous voulons afficher tous les résultats, nous pouvons faire :

```
auto r = find(y, query());
for (auto it = r.first ; it != r.second ; ++it)
{
  cout << it->position << " ";
  cout << it->observation_model <<  " ";
  cout << it->time << " ";
  cout << it->variable << " ";
  cout << it->value << std::endl;
}
```

Code 2.12 – affichage de tous les éléments de la simulation

2.2.9 Entrées-sorties du système

En ce qui concerne les entrée-sorties du système, nous avons privilégié la manipulation par un opérateur humain plutôt que par un autre programme. Nous avons donc des formats textuels et non binaire, nous avons aussi, pour l'instant, proscrit l'usage du XML étant donné sa verbosité. Toutefois, ce dernier format sera pris en charge naturellement lors du passage à l'échelle de la plateforme par souci d'efficacité, notamment pour la communication entre des processus externes et la plateforme.
Nous avons trois formats de fichiers pris en compte :

8. l'utilisation de first et second provient de l'utilisation d'un `std::pair` dont nous n'avons pas surchargé les noms d'accès.

— un format DAT destiné à la sauvegarde des paramètres, des variables d'états ou de contrôle et qui s'apparente à un fichier de type "clé-valeur" avec le signe = comme séparateur. Il permet de stocker des scalaires, des vecteurs et des matrices dans des formats similaires à ceux disponible en Matlab ou en Scilab. Un exemple de ce format est donnée en annexe D.1. Toutefois, la plupart du temps nous manipulerons la version CSV de ce format et nous trouverons un exemple en annexe D.2.
— un format OBS destiné à la sauvegarde des observations du système et dont le contenu est réparti tel que représenté en annexe D.3. Il y a aussi une version CSV de ce format qui est disponible en annexe D.4.

Cette section a présenté les types de modèles mathématiques qui ont donné naissance à notre conceptualisation.
Les modélisateurs utiliseront la représentation d'états afin de coder les modèles correspondant sur ordinateur grâce à un langage dédié embarqué qui est construit sur l'analyse des types, signatures et éléments de syntaxe découlant de cette représentation d'états.
Les personnes en charge de la création des algorithmes de traitement statistique, quant à eux, utiliseront la notion de modèle Markov caché afin d'échanger et de construire les instructions requises. Dans la pratique, ils coderont leur algorithmes en utilisant le même langage dédié embarqué que les modélisateurs. Nous verrons ces éléments dans le chapitre suivant.
Cette conceptualisation est le résultat à la fois de l'analyse du domaine mais aussi du retour d'expérience sur les différents essais de réalisation du coeur de simulation et de manipulation des données.
Toutefois, nous avons mis de côté une question très importante du point de vue calculatoire qui est la génération des nombres aléatoires, c'est ce point qui va nous occuper dans la section suivante avant de voir comme nous réalisons conceptuellement les différentes simulations de nos systèmes. Cette notion de simulation sera par la suite abstraite afin d'être étendue à tout type de simulateur existant.

2.3 Simulation

Nous introduirons dans un premier temps comment nous voyons la simulation au travers d'un "cube de simulations" et définirons une classification en "types de simulations". Ensuite nous décrirons le cadre général pour l'écriture d'un algorithme avant de présenter l'implémentation. Ce cadre sera utilisé à la fois pour décrire la simulation ainsi que les fonctionnalités (algorithmes statistiques) du chapitre 3. Toutefois, la version pour la fonction de la simulation est la plus simple. Nous introduirons donc des précisions lors du chapitre 3.

2.3.1 Cube de simulations

2.3.1.1 Pourquoi visualiser l'ensemble sur un cube ?

Comme nous le verrons dans le chapitre suivant, l'ensemble des algorithmes qui composent la chaîne des bonnes pratiques de modélisation utilise intensivement la simulation avec parfois plusieurs centaines de milliers de simulations à faire d'une étape à une autre. Nous avons voulu chercher le moyen de discuter de ces notions de simulations autour d'un objet commun à la fois pour comprendre la relation entre les simulations et les algorithmes, notamment dans le cadre des algorithmes de type prédiction-correction, mais aussi car il y a la volonté de former un cadre conceptuel à la programmation massivement parallèle.

Pour certains algorithmes, il s'agit de faire une simple simulation pour obtenir un vecteur d'observations du temps 0 au temps maximal. Pour d'autres, il s'agit d'aller de temps d'observation en temps d'observation afin de corriger les jeux de paramètres ou les variables d'état. La notion de cube (Bayol, Chen et Cournède 2013) permet de se mettre dans un cadre où, à la fois, nous pouvons discuter du domaine

particulier des comportements des algorithmes et où nous pouvons mettre en perspective les notions de parallèlisme et de granularité en terme de simulation.

2.3.1.2 Description du cube

Nous pouvons projeter l'ensemble de nos besoins en termes de simulations sur le cube [9] illustré en figure 2.6. L'abscisse du cube correspond aux expériences tandis que les ordonnées correspondent aux différents jeux de paramètres.

Les expériences sont définies selon la définition formelle qui a été donnée précédemment, à savoir un couple entre une fonction d'observation global du système et un contexte. Cette fonction d'observation est elle-même composée d'une liste de modèles d'observation, eux-mêmes composés d'une timeline et d'une référence syntaxique vers une fonction d'observation simple. Le contexte, quant à lui, est un couple composé des variables d'état initial et des variables externes du système. À une coordonnée (i, j), nous avons une particule d'information qui est elle-même constituée des différents éléments constituant l'état en cours du système $xl[n]$, les paramètres p, les variables de contrôle u, ainsi que l'état des différents générateurs g_{mi} et g_{oi} afin de produire les observations y du système.

La troisième dimension du cube de simulation est celle du temps de simulation et chaque ligne rouge, ou grille, représente un temps d'observation. Ce temps d'observation est le même pour toutes les particules car, par définition, nous appliquons la même fonction d'observation générale à toutes les expériences, mais pas forcément les mêmes états initiaux ou les mêmes environnements. Ainsi à chaque temps d'observation, nous obtenons un vecteur ou une matrice d'information, composé des observations des différentes simulation au temps $t_{current}$. On pourra aussi parler de "slice" d'information.

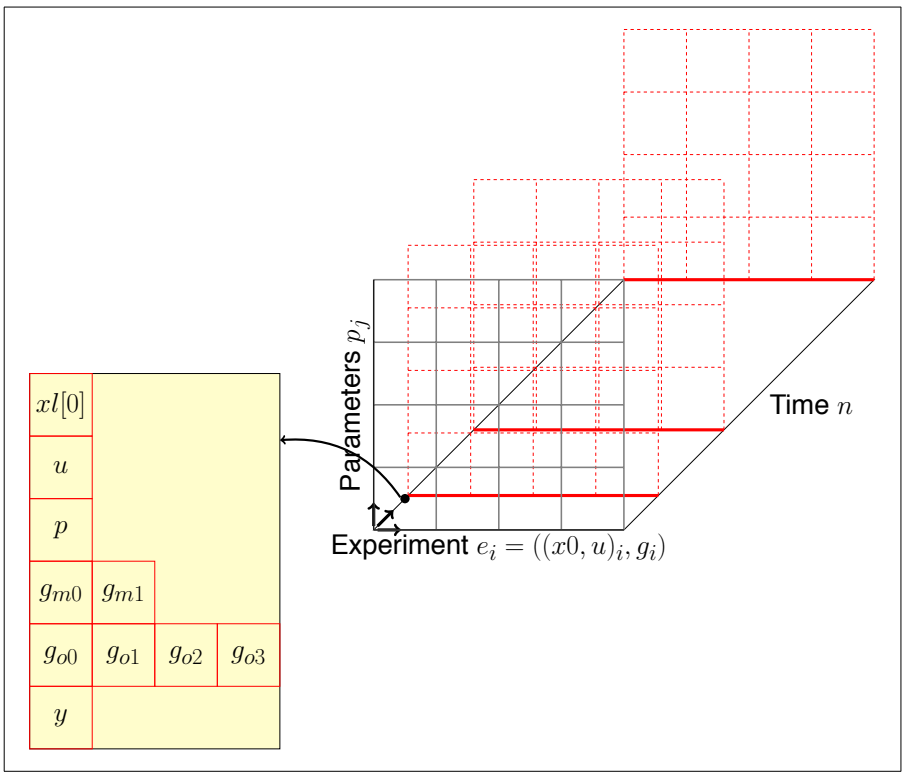


Figure 2.6 – visualisation du cube et des constituants d'une case (ou particule)

Ce cube que nous utiliserons plus tard pour décrire le fonctionnement de chacun de nos algorithmes peut être mis sous forme de listes pour faciliter la simulation de type multithreadée ou sur architecture distribuée. Nous allons donc voir comment passer de la notion de cube à celle de liste qui permettra de

9. Si l'on pense en 2D nous avons affaire à une grille à t0, si nous allons jusqu'à t_{max} cela sera un parallépipède. J'ai préféré le mot cube car plus court et qui transmet bien l'idée générale bien que cela ne soit donc pas formellement un cube

répartir les calculs lorsque nous voudrons exploiter les architectures parallèles.

La figure 2.7 est une visualisation de la répartition des éléments du cube sous forme de liste et notamment par rapport à la notion d'expérience (”experiment”) e_i et de paramètres p_j de façon à ce que des observations y_j^i soient le résultat des simulations $e_i \times p_j$. En jaune, nous retrouvons toutes les simulations sous forme de liste. La première face du cube permet de discuter de la répartition entre les expériences et les jeux de paramètres, tandis que la liste jaune permet de visualiser la liste qui sera découpée en paquets pour le calcul parallèle.

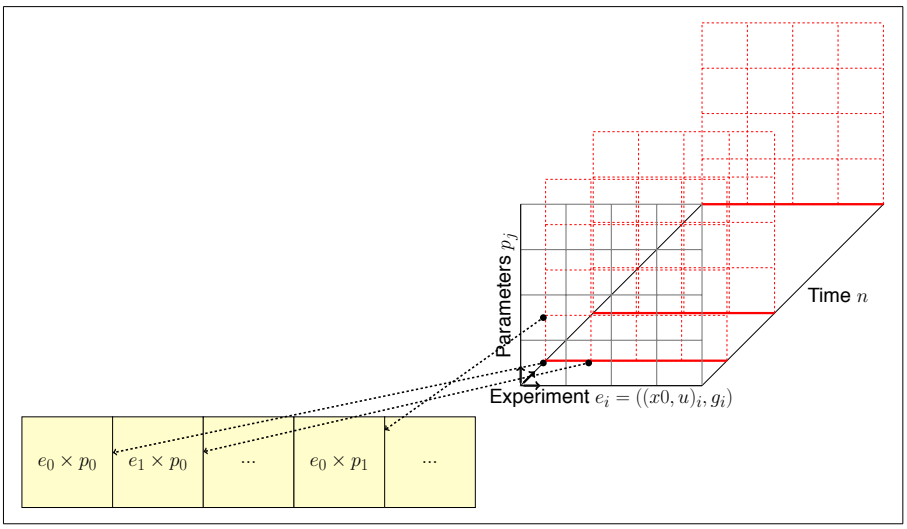


Figure 2.7 – visualisation des couples (p,e) correspondant aux entrées des simulations du cube sour forme de liste

Une fois que cette répartition est effectuée, nous pouvons enfin lancer la simulation en tant que telle. Si nous simplifions notre problème et considérons que nous avons une seule fonction à appliquer, au lieu de deux fonctions comme c'est le cas pour nos modèles (fonction d'initialisation et fonction de transition), alors cela consiste à faire un mapping de cette fonction sur une liste constituée des expériences virtuelles à conduire. C'est ce que nous pouvons visualiser sur la figure 2.8. À la première étape nous avons notre liste de couples ”expériences et paramètres” puis nous répartissons sur différents groupes pour avoir un thread de calcul par groupe, puis nous mappons la fonction de transition que nous appliquerons N fois (où N est le nombre de temps de simulation) et par conséquent nous récupérons dans chaque groupe les observations du système que nous réduisons dans une dernière étape afin d'obtenir Y.

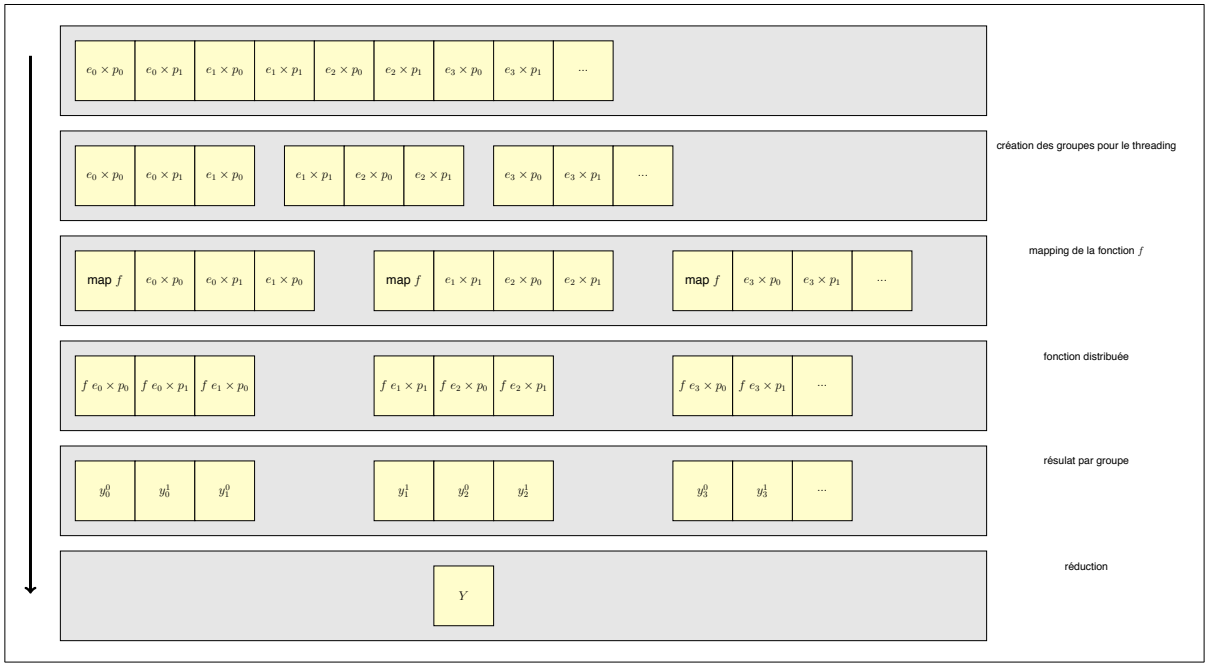


Figure 2.8 – visualisation de la répartition de la liste lors des calculs

La figure 2.9 permet de se représenter les trajectoires des particules en termes d'observation des systèmes simulés. Nous avons, à droite, l'état du cube avant une simulation avec 4 points noir dans la colonne 1. Autrement dit nous allons appliquer sur une même expérience, i.e. même état initial, même environnement, et même fonction d'observation globale, différents jeux de paramètres, ici 4. Sur la gauche, nous avons les 4 simulations finies avec sur chaque ligne rouge la valeur d'une variable d'état observée. On peut voir qu'il y a eu pour chaque particule (ou simulation, ou système virtuel) trois moments d'observation. Chaque observation concerne deux variables d'état que sont la valeur de biomasse totale et un indice de stress hydrique.

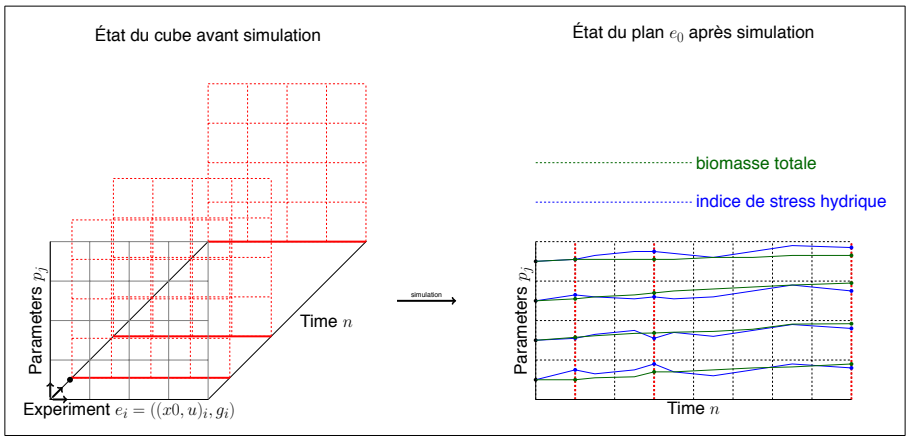


Figure 2.9 – état du cube avant simulation et état d'un plan après simulation et observation

Nous avons vu comment construire le cube, voyons maintenant comment classer nos algorithmes en fonction du parcours de ce cube.

2.3.2 Typologie des simulations

Les figures 2.10, 2.11, 2.12, 2.13 donnent une représentation graphique des différents types de simulations utilisées par les différents algorithmes. Cela permet de classifier les algorithmes par rapport à la structure de la simulation. Ainsi la typologie des simulations est basée sur le type de parcours du cube de

simulation. Cette typologie est composée de quatre éléments : simulation simple, simulation de plusieurs expériences, simulation de Monte-Carlo, simulation régulée par un mécanisme de prédiction-correction.

2.3.2.1 Simulation simple

Dans la figure 2.10 nous appliquons la fonction `simulate` à une seule expérience et un seul jeu de paramètres. Il faut noter que nous pourrions appeler ce couple ”simulation” ou ”système virtuel” de telle sorte que les `system-observations` soient le résultat de la fonction `simulate` sur les systèmes virtuels.

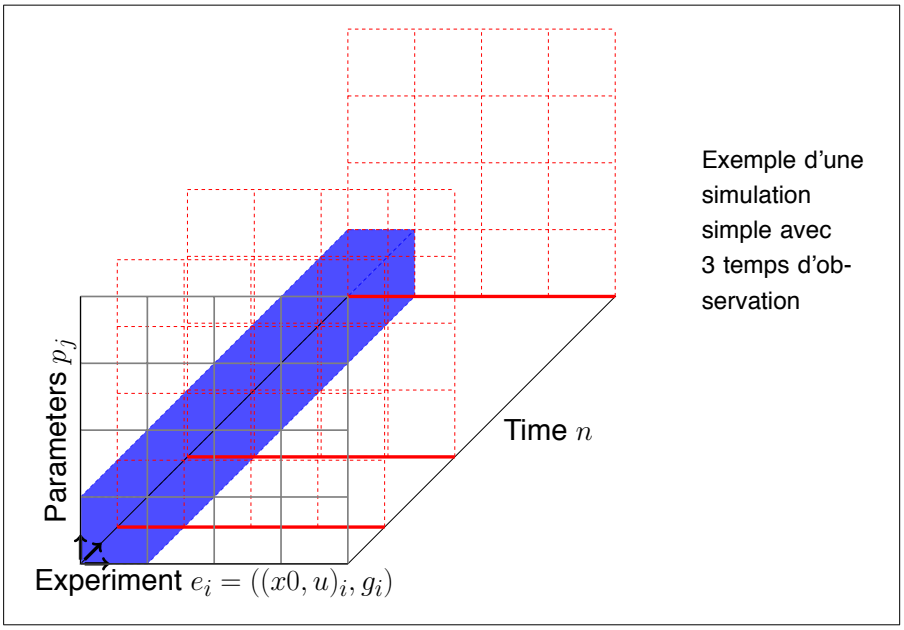


Figure 2.10 – visualisation du cube et d'une ligne de simulation

La simulation d'un modèle consiste à appliquer la fonction de transition jusqu'à atteindre le temps d'observation maximal. À chaque application de la fonction de transition, nous devons ou non appliquer la fonction d'observation du système afin de collecter des informations sur le système.

Notre cadre mathématique permet d'avoir une fonction de simulation simple car ne résidant que sur l'incrémentation du temps courant par une valeur unitaire. Le pseudo code de cette fonction de simulation est le suivant :

Algorithm 1 simulation simple

```
1 : procedure simulate(m,e,p)
2 :     done ← False
3 :     current_time = 0
4 :     xn ← e.c.x0
5 :     xn ← m.initialize(xn,e.c.u,p)
6 :     while not done do
7 :         for each om in e.of do
8 :             if current_time in om.tml then
9 :                 y.add(current_time,observe(xn,p,e))
10 :        xn ← m.next_state(xn,u,p,e)
11 :        current_time ← current_time + 1
12 :        if current_time = t_max then
13 :            done ← True
    return y
```

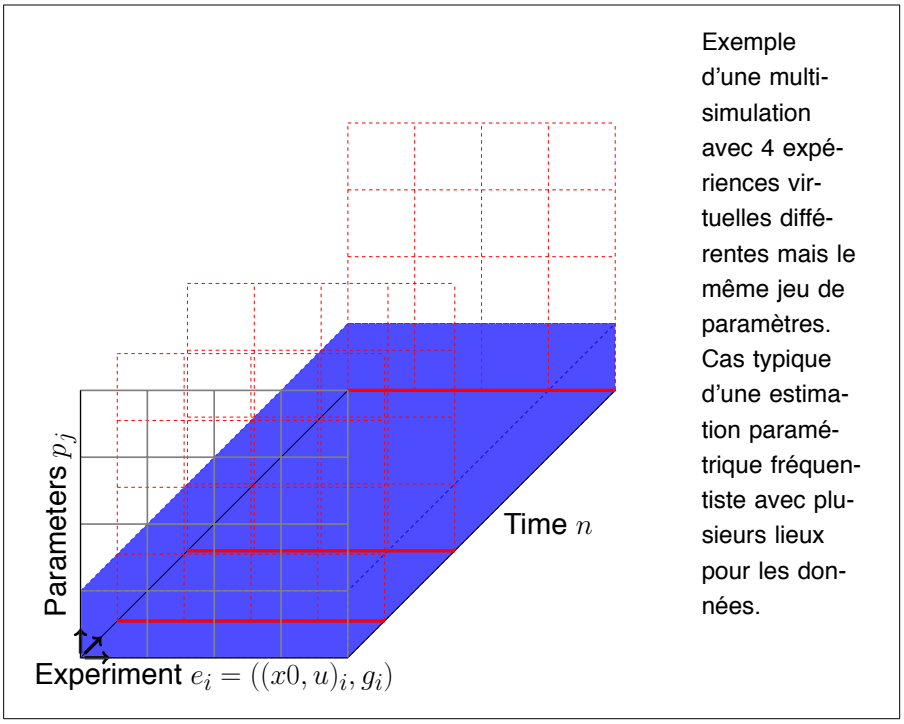
Un programme autonome a été créé au sein de la plateforme et se nomme ”simulate”.

```
simulate --model lv.so
--observation-function '[("x",[1,4,5,6])]'
```

Code 2.13 – commande de lancement d'une simulation simple avec observation de la variable x aux temps 1,4,5 et 6.

2.3.2.2 Simulation de plusieurs expériences

La figure 2.11 permet de se faire une représentation de plusieurs expériences avec un seul jeu de paramètres. C'est une configuration qui est utilisée lors de l'estimation paramétrique avec des jeux de données venant de parcelles sur plusieurs localisations par exemple, ou issus de modalités culturales différentes (cas irrigué, cas non irrigué par exemple).



Exemple d'une multi-simulation avec 4 expériences virtuelles différentes mais le même jeu de paramètres. Cas typique d'une estimation paramétrique fréquentiste avec plusieurs lieux pour les données.

Figure 2.11 – visualisation de simulations multiples

2.3.2.3 Simulation de Monte-Carlo

La figure 2.12 présente la configuration du cube utilisée lors des analyses d'incertitudes ou de sensibilité.

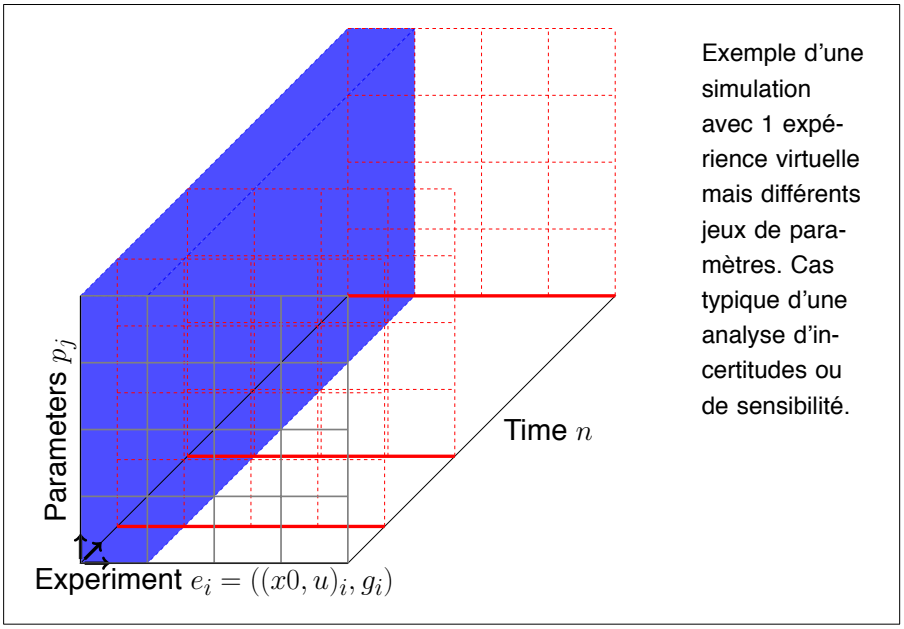


Figure 2.12 – visualisation d'une simulation avec multiples paramètres

C'est aussi cette configuration qui est utilisée lors d'une simulation de Monte-Carlo qui consiste à simuler un modèle pour un nombre N de jeux de paramètres.

Algorithm 2 simulation de Monte-Carlo

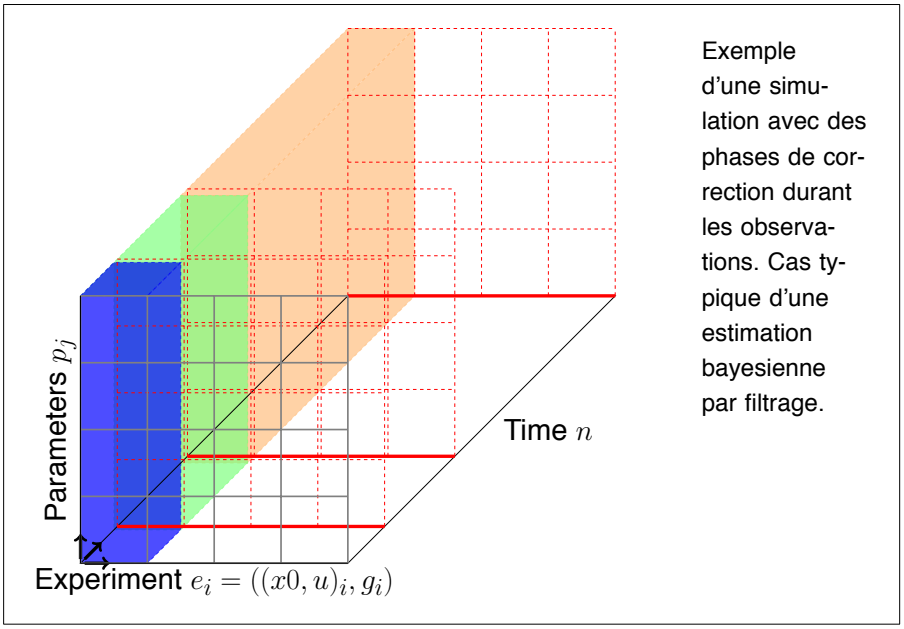
```
1 : procedure simulate-monte-carlo(m,e,p0,rl)
2 :     pl ← pseudo_random_sampler(p0,rl)
3 :     for i from 0 to size(pl) do
4 :         yl[i] ← simulate(m,e,pl[i])
    return yl
```

```
monte-carlo --model lv.so
--observation-function '[("x",[1,4,5,6])]'
--univariate-rule-list [("a", uniform, [3.2,4.2])]
--nbr-samples 1000
```

Code 2.14 – commande de lancement d'une simulation de Monte-Carlo sur 1000 échantillons

2.3.2.4 Simulation régulée par un mécanisme de prédiction-correction

Enfin la figure 2.13 présente l'utilisation du cadre général lors d'une estimation bayésienne par filtrage et plus particulièrement dans le cas ou une phase de correction des distributions intervient à chaque temps d'observation du système virtuel.



Exemple d'une simulation avec des phases de correction durant les observations. Cas typique d'une estimation bayesienne par filtrage.

Figure 2.13 – visualisation d'une simulation avec multiples paramètres et phases de correction. Chaque changement de couleurs correspond à une mise à jour des distributions de paramètres et d'états cachés (Chen 2014)

La figure 2.14 permet de représenter ce qu'il se passe entre le temps t_O, le temps où l'observation est effectuée, et la phase t_O^+ (ou t_c), le temps où la phase de correction a été faite, qui donne lieu au temps t_c^p où nous appliquons une légère pertubation après la phase de correction. (voir par exemple le convolution particle filter (Chen et Cournède 2014))

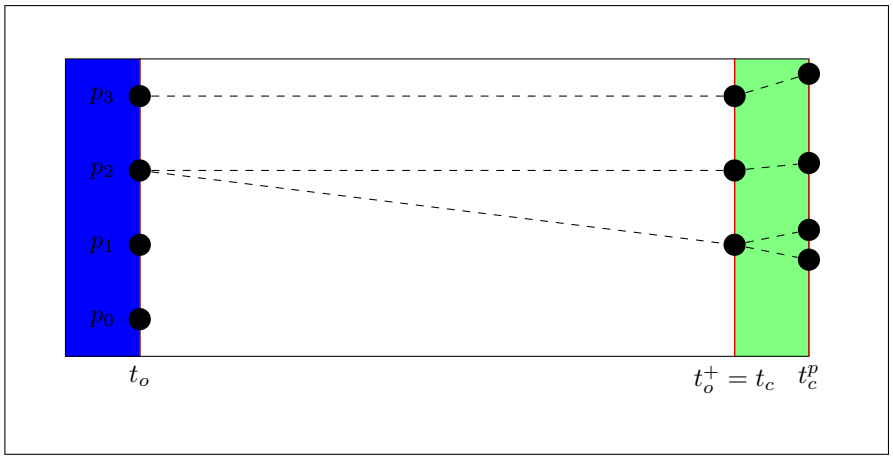


Figure 2.14 – phase de rééchantillonnage par sélection qui s'accompane d'une phase de perturbation

2.3.3 Implémentation

Aujourd'hui l'ensemble de la plateforme utilise OpenMP (OpenMP Architecture Review Board 2008) lorsque cela est possible afin d'avoir au moins les simulations parallèles calculées sur plusieurs cœurs. Nous avons rejeté le SIMD (Flynn 1972) car la granularité et la complexité de ce type d'approche est trop forte pour les modèles mis en jeu. Toutefois, nous verrons que par l'utilisation de LLVM (Lattner et Adve 2004) lors de la section 2.5 (page 70) nous pourrons profiter des opérations de vectorisation automatique dans le langage dédié.

MPI (Forum 1994) est une bibliothèque que nous avons utilisée pour faire quelques tests mais pour l'instant son utilisation n'est pas standard dans la bibliothèque. Elle serait utile afin de répartir les cal-

culs sur un plus grand nombre de particules [10] et ainsi réduire la consommation mémoire par nœud en répartissant la mémoire totale nécessaire.

La programmation sur GPU a aussi été envisagée, mais pour l'instant nous préférons voir comment évolue des outils comme le PTX pour LLVM qui permet de générer du code pour GPU directement depuis un langage dédié, ou encore des outils comme OpenACC (Herdman et al. 2014). Ils pourraient permettre de générer les modèles directement à partir du langage dédié pour ce genre de plateforme matérielle.

> Nous venons dans cette section de construire un cadre conceptuel autour de la notion de cube qui nous permet de penser le fonctionnement de nos algorithmes quel que soit leurs types. Ce cadre permet à la fois l'échange entre les différents chercheurs de l'équipe mais aussi de guide quant à l'implémentation des fonctions de simulations générales. Pour l'instant nous l'avons implémenté pour être compatible avec les architectures à mémoire partagée uniquement.

Nous allons maintenant voir les spécificités propres à la gestion des nombres aléatoires et comment nous devons les gérer pour nos bruits de modélisation et d'observation.

2.4 Simulations et stochasticité

Nous avons omis pour l'instant la gestion de la production des nombres aléatoires au sein des simulations et donc pour chaque particule. Ces nombres aléatoires seront utilisés par l'ensemble des distributions attribuées aux bruits.

Il convient de déterminer une solution souple pour permettre l'écriture de systèmes déterministes ou stochastiques ainsi qu'une implémentation permettant de palier les problèmes courants avec les générateurs aléatoires (mauvaise initialisation, corrélation, ...) (Coddington 1997 ; L'Ecuyer, Oreshkin et Simard 2014).

Par ailleurs, comme nous le verrons dans la prochaine section, de nombreux algorithmes d'analyse statistique (analyse de sensibilité, Monte-Carlo séquentiel, Monte-Carlo par chaîne de Markov, ...) reposent sur la méthode de Monte-Carlo, et donc un échantillonnage dans l'espace des paramètres. Là aussi, la génération des suites aléatoires est importante. Par contre, une exploration optimisée de l'espace pourra être préférée à une réalité probabiliste, ce qui pourra modifier la nature des générateurs aléatoires (cf. section 2.4.1.4 sur les générateurs quasi-aléatoires).

C'est pour toutes ces raisons que nous avons étudié la façon de gérer la stochasticité au sein du système. On pourra notamment retrouver une étude approfondie de ces éléments pour le calcul distribué sur grille dans (Reuillon 2008) ou encore dans (Hill 2014)

Nous allons voir, dans un premier temps, comment générer des suites de nombres aléatoires sur ordinateur et, par la suite, nous verrons quelle est la stratégie que nous avons retenue parmi l'ensemble des stratégies disponibles.

2.4.1 Comment mimer l'aléatoire sur une machine déterministe ?

Dans cette partie, nous ferons d'abord un rappel sur la notion d'aléatoire et de suite aléatoire avant de voir comment calculer sur machine des valeurs de distributions de type uniforme ou normale.

2.4.1.1 Historique

Avant l'avènement des ordinateurs et des algorithmes de génération de nombres aléatoires, la tâche consistant à introduire des nombres aléatoires était confiée à des êtres humains lors des calculs. Ces derniers devaient se référer à des tables de calcul éditées dans des livres de nombres aléatoires. Ces

10. Dans notre cadre, une particule est un concept équivalent à celui de simulation.

nombres étaient choisis par une méthode basée sur l'interrogation d'êtres humains. Toutefois, cette méthode, qui a fonctionné jusque dans les années 50 avec notamment l'édition finale de (Rand_Corporation 1955), introduit de nombreux biais dans la suite produite.

Dans les années 50, Von Neumann s'est penché sur le problème de la génération des nombres aléatoires et a notamment proposé un algorithme basé sur la congruence. Cet algorithme dénommé "middle-square algorithm" n'est pas bon statistiquement (Von Neumann 1951) mais par l'importance du sujet et par la notoriété de Von Neumann cela a permis de fédérer autour de certaines notions. C'est l'un des premiers algorithmes et il n'est guère plus utilisé que pour ses valeurs pédagogiques. Au même moment, Derrick Henry Lehmer (Lehmer 1951) a créé la première version du générateur linéaire congruentiel ("linear congruential generator") dont une description sera donnée plus loin. C'est un générateur que l'on retrouve notamment au sein de la famille des langages C/C++ comme générateur par défaut.

D'autres générateurs ont fait leur apparition et on s'est naturellement posé la question de la performance de ces différents générateurs, i.e lequel faut-il utiliser ou plutôt quelles sont les conditions favorables à l'exploitation d'un générateur plutôt qu'un autre ?

Cette question donnera lieu à l'exploration des tests théoriques et empiriques que nous décrirons dans la section 2.4.2.3.

Il faudra attendre les années 90 avec le générateur "Mersenne Twister" (Matsumoto et Nishimura 1998b) pour que la communauté scientifique utilisant des générateurs de nombres aléatoires se fédère autour de celui-ci et de ses différentes caractéristiques pour la modélisation en système biologique.

Il faut noter l'importance du langage dans l'écriture et l'utilisation de ces algorithmes.

Certains langages font beaucoup de choses pour l'utilisateur (Matlab, Python, ...), d'autres laissent plus de libertés (C, C++, ...) et ces derniers exigent une bonne compréhension des couches basses

Après ce bref historique nous présentons les notions entourant le concept de générateur pseudo-aléatoire. [11]

2.4.1.2 Suite aléatoire et générateur aléatoire

Avant de déterminer des algorithmes permettant de générer des suites aléatoires, il faut se poser la question des caractéristiques d'une suite considérée comme aléatoire.

Nous considérons les suites aléatoires indépendantes identiquement distribuées.

Si la définition théorique peut parfois être vérifiée, nous verrons dans la section 2.4.2.3 qu'elle a aussi ses limites et que dans la pratique nous testerons les algorithmes de génération grâce à des tests expérimentaux dont le contenu à été défini au fil des années et a été notamment consigné par les travaux de (L'Ecuyer et Simard 2007)

Si nous souhaitons établir une suite **vraiment aléatoire**, alors plusieurs phénomènes physiques peuvent être observés, comme la radioactivité ou le bruit thermique. Il existe aussi des cartes matérielles pour ordinateur qui observent les changements thermiques et électriques sur celles-ci afin d'obtenir des nombres aléatoires.

L'acquisition de nombres **vraiment aléatoires** par des moyens physiques possède plusieurs défauts dans notre cadre :

- le coût d'acquisition en termes de temps et de mémoire puisqu'il faut prévoir des communications entre les cartes d'acquisition et le processeur.
- la non-reproducibilité des expériences qui demanderait d'enregistrer chaque suite en mémoire afin d'être rejouée pour chaque simulation.

C'est pour ces raisons que nous utiliserons des nombres **pseudo aléatoires**. Ils sont peu coûteux relativement aux autres et les suites sont reproductibles, ce qui est une qualité intéressante en termes de tests numériques ou de recherche de bugs dans nos programmes.

11. Nous présenterons aussi d'autres générateurs comme les quasi-aléatoires par la suite

2.4.1.3 Générateurs pseudo-aléatoires

Nous utiliserons donc des générateurs pseudo-aléatoires dans la suite de cette section. Nous décrirons brièvement le générateur congruentiel linéaire (LCG) afin d'avoir une idée générale du concept et d'introduire les autres types de générateurs existants.

Générateur congruentiel linéaire Le générateur congruentiel linéaire (*Linear Congruential Generator - LCG*) (Knuth 1997) est le générateur pseudo-aléatoire utilisé par défaut dans le langage C++ à travers sa commande *rand()*. Son espace d'états est relativement petit ce qui lui permet notamment d'être embarqué sur des plateformes où la taille mémoire est contrainte comme les GPU par exemple. Il est primordial de voir ce générateur car il reste l'un des plus abordables en termes de compréhension. Il se base sur l'utilisation de la fonction modulo et d'une fonction linéaire tel que :

$$X_{n+1} = (a * X_n + b)\,[m] \tag{2.6}$$

où

- X_n est l'élément n de la suite aléatoire.
- a est le multiplicateur
- b est l'incrément
- m est la période

Les nombres obtenus sont compris entre 1 et RAND_MAX qui est la période m du générateur. Nous pouvons utiliser ce générateur afin de générer des nombres suivant une distribution uniforme dans l'intervalle $[bottom, top]$ tel que montré dans le code 2.15.

```
double uniform_lcg(double bottom, double top)
{
    return std::rand() / RAND_MAX * (top - bottom) + bottom;
}

int main(int argc, char ** argv)
{
    //set seed to 0 for the incoming sequence
    std::srand(0);
    double u;
    //get first uniform real between 0 and 1 using sequence 0 of lcg
    u = uniform_lcg(0,1);
    //get second uniform real between 0 and 1 using sequence 0 of lcg
    u = uniform_lcg(0,1);
    //set seed to 42 for the incoming sequence
    std::srand(42);
    //get first uniform real between 0 and 1 using sequence 42 of lcg
    u = uniform_lcg(0,1);
}
```

Code 2.15 – c++ générant des réels selon une distribution uniforme de paramètres 0 et 1 avec le lcg

Nous nous aperçevons de l'importance du terme de **graine** dans la création des suites puisque si l'on met deux fois la même **graine** nous obtiendrons la même suite évidemment. Par la nature de sa construction (congruence), cet algorithme possède une **période**. Si nous générons un nombre de variables au delà de cette période nous ferons boucler le générateur.

Le choix de la graine et la période d'un générateur pseudo-aléatoires sont deux notions fondamentales dans la manipulation des générateurs pseudo-aléatoires.

Dans la prochaine section nous décrirons quelques caractéristiques du générateur ”Mersenne Twister”.

Mersenne Twister Le ”Mersenne Twister” (Matsumoto et Nishimura 1998b) est un générateur fréquemment utilisé dans le domaine de la simulation biologique. Il est, de plus, présent par défaut dans de nombreux langages comme R, Python ou MATLAB.

Dans sa version la plus utilisée, il possède une grande période à savoir 2^{19937}. Cette période et ses autres attributs peuvent être changés en le paramétrant différement.

C'est un générateur qui passe de nombreux tests [12] correctement sauf ceux dédiés à la cryptographie.

Autres générateurs (Xorwow, WELL, MTDC, MTGP, ...) Outre ces deux générateurs connus car LCG (Knuth 1997) est implémenté en standard en C/C++ depuis longtemps et MT19937 (Matsumoto et Nishimura 1998b) est utilisé et bien diffusé depuis 2000 par de nombreuses applications scientifiques, sa longue période et sa réputation en matière de qualité statistique, il existe d'autres générateurs qui possèdent des caractéristiques intéressantes :

- xorwow (Marsaglia 2003)
- WELL (Panneton, L'ecuyer et Matsumoto 2006) qui a la réputation d'avoir une meilleure qualité statistique que le Mersenne Twister
- MGRK32a (L'ecuyer 1999) qui peut être découpé pour les calculs parallèles (L'ecuyer et al. 2002)
- MTDC (Matsumoto et Nishimura 1998a) qui est un algorithme permettant de chercher des générateurs Mersenne Twister indépendant en cherchant des configurations de paramètres indépendants.
- ...

Il existe aussi des générateurs qui sont par nature parallèles car créés pour des environnements massivement parallèle (GPGPU). C'est le cas notamment de MTGP (Saito 2010) ou des générateurs dans des bibliothèques tel que CURAND (NVIDIA 2010).

2.4.1.4 Autres types de générateurs (quasi-aléatoire, latin hypercube, ...)

Avant de voir les notions de distributions et d'échantillonnage nous pouvons nous attarder sur l'existence d'autres types de générateurs. Nous avons décrit l'ensemble des éléments autour de la notion de générateur pseudo-aléatoire. Cependant il existe d'autres types de générateurs, à savoir les générateurs quasi-aléatoires, générateurs de type latin hypercube, les générateurs à base de sigma-points.

Ces différents générateurs ont des caractéristiques permettant d'explorer les espaces des paramètres d'une façon peut être moins coûteuse ou plus régulière que les générateurs pseudo-aléatoires.

Au sein du système, nous avons réimplémenté différents générateurs.

La figure 2.16 permet de se rendre compte de la différence en termes d'exploration des données par rapport à la figure 2.15.

12. Nous verrons quelques caractéristiques des tests dans la section 2.4.2.3

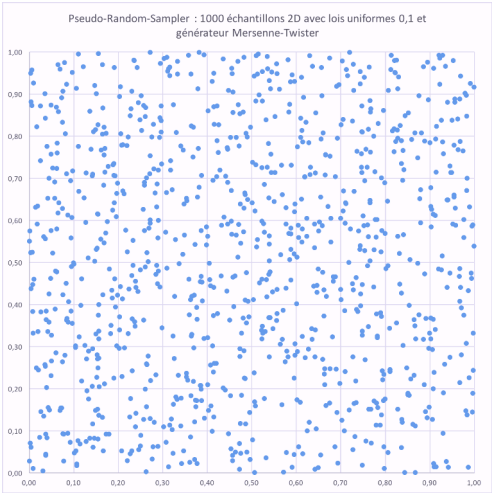


Figure 2.15 – échantillonnage par pseudo-random-sampler

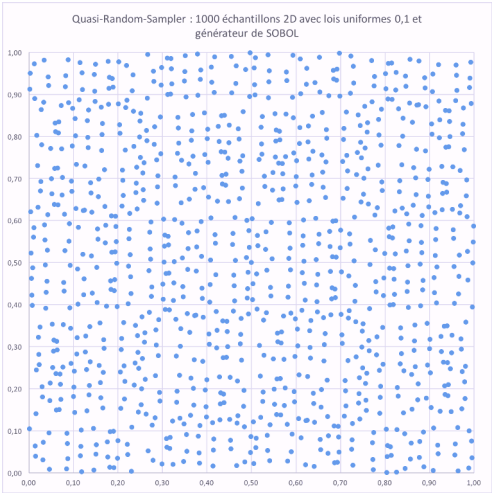


Figure 2.16 – échantillonnage par quasi-random-sampler

On peut voir que le pseudo-random cherche à ”mimer” un aléatoire naturel tandis que le quasi-random est plus ordonné dans son balayage de l'espace. Ces caractéristiques différentes peuvent être exploitées en fonction des algorithmes.

2.4.1.5 Distributions et échantillonnage

Dans cette section nous allons revenir sur les notions de variables aléatoires, variables aléatoires multi-dimensionnelles, de distributions univariées, distributions multivariées et enfin d'échantillonnage sur un espace.

Variables aléatoires multidimensionelles Nous reprenons ici les définitions classiques de variable aléatoire et variable aléatoire multidimensionnelle (ou vecteur aléatoire).

On peut approcher l'aléatoire de deux façons sur des variables multidimensionelle. Soit on considère le vecteur comme un ensemble de variables aléatoires unidimensionelles univariés (”n-univariées”) soit on peut voir l'ensemble comme étant ”multivarié”. Si l'on prend l'approche ”n-univariées” alors on considère qu'il n'y a pas de corrélation entre les différentes composantes du vecteurs tandis que dans l'approche ”multivariée” nous allons introduire des corrélations entre les différents éléments du vecteur par le biais d'une matrice de covariance.

Dans le cadre de notre modélisation nous utilisons un certains nombre de distributions classiques permettant d'approcher certains processus ou bien d'en bruiter d'autres. Il est important de comprendre que le choix d'une distribution crée une hypothèse forte selon que l'on considère certains processus suivant une loi uniforme, une loi normale, ... Cette même importance se manifeste selon que nous allons approcher le problème par une distribution univariée (ou ”n-univariées”) ou par une distribution ”multivariée”.

Distributions univariées Un certain nombre de distributions ont été introduites pour modéliser les bruits de modélisation et d'observation, comme les distributions uniforme, normale ou log-normale.

Ces distributions sont utilisées au moyen de bruits multiplicatifs afin d'avoir des pertubations dans des ordres de grandeur identiques aux variables bruitées, ou additifs.

Distributions multivariées Les distributions multivariées permettent de générer des vecteurs aléatoires dont les composantes sont liées par une matrice de covariance.

Les distributions ayant été introduites sont les distributions normale et log-normale. Nous envisageaons également d'implémenter des distributions multivariées caractérisées par des copules.

Échantillonnage L'échantillonnage ("*sampling*") consiste à obtenir N jeux de données à partir d'un ensemble de distributions qui peuvent être univariées ou multivariées et combinées à des générateurs aléatoires de différents types.

Ainsi l'échantillonnage qui nous intéresse met en lien les différents types de générateurs (pseudo, quasi, latin-hypercube, sigma, ...) avec un ensemble de distributions (uniforme, normal, ...) sur certaines données sources. Ainsi on retrouve l'échantillonnage avec les *générateurs pseudo-aléatoires*, l'échantillonnage avec les *générateurs quasi-aléatoires*, l'échantillonnage avec les *cubes hyperlatins*, l'échantillonnage avec les *générateurs de sigma-points*.

Ces différents types d'échantillonnage auront des propriétés différentes quant au balayage de l'espace considéré.

L'échantillonnage avec une loi normale multivariée de corrélation nulle est représentée en figure 2.17. Il est équivalent à un échantillonnage avec deux lois normales univariées. Chaque loi est associée à une dimension du plan. L'échantillonnage avec une loi normale multivariée et de corrélation non-nulle est représentée en figure 2.19.

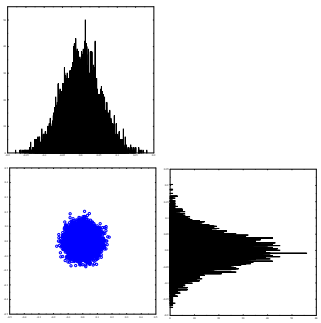


Figure 2.17 – échantillonnage avec 1 loi normale multivariée et corrélation nulle

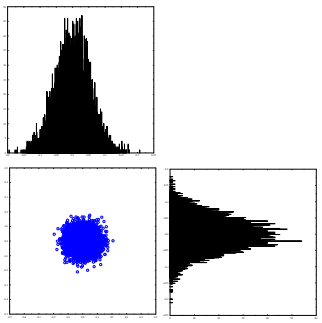


Figure 2.18 – échantillonnage avec 2 lois normales

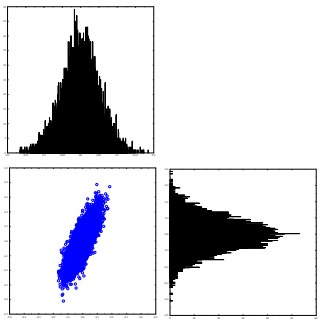


Figure 2.19 – échantillonnage avec 1 loi normale multivariée et une corrélation non-nulle

Nous allons maintenant voir quelles sont les contraintes relatives à cette notion dans le cadre du cube de simulations.

2.4.2 Différentes stratégies pour la gestion des générateurs

La question principale qui va nous préoccuper est la suivante : Si nous associons un seul générateur à tout le cube, qu'en est-il des corrélations et de l'indépendance entre les bruits générés ? Par exemple si un modèle utilise 4 bruits de modélisation et 4 bruits d'observations est-ce que les suites pseudo-aléatoires extraites gardent toujours leurs propriétés aléatoires ? Par exemple nous pouvons avoir un bruit ϵ_2 dont l'usage est conditionné par la valeur d'un bruit ϵ_1. Est-ce que la suite utilisée par ϵ_2 garde toujours les propriétés du générateur initial ? On revient donc à analyser une sous-suite de la suite. Une bonne discussion autour du sujet des techniques d'initialisation et ses problématiques en général se trouve dans (L'Ecuyer, Oreshkin et Simard 2014). On peut visualiser ces différences sur les figures 2.26 et 2.27.

Nous avons donc conduit deux études pour nous mettre dans notre cadre de simulation. Premièrement une étude sur la corrélation et deuxièment une étude sur la qualité des sous-suites aléatoires grâce à la bibliothèque TestU01 (L'Ecuyer et Simard 2007). Nous verrons cela par la suite, mais pour l'instant nous passons en revue les différentes stratégies d'initialisation.

2.4.2.1 Stratégie d'initialisation

Quels sont les choix qui s'offrent à nous quant à l'initialisation de la valeur X_0 ou **graine** pour la suite aléatoire ?

Cette question est d'autant plus stratégique lorsque l'on multiplie les générateurs au sein d'un programme. En effet nous ne voudrions pas nous retrouver avec deux fois le même générateur sur des bruits différents ou pire le même générateur pour toutes les simulations, entraînant de fait la simulation de la même chose N fois.

Nous avons distingué trois approches qui peuvent être résumées en :

- initialisation par rapport au temps sur 1 générateur
- initialisation par rapport au temps sur N générateurs
- initialisation par rapport au temps sur 1 générateur afin d'alimenter N générateurs

Ces trois approches ont pour but de distribuer des nombres pseudo-aléatoires à des distributions quelconques, ici au nombre de deux.

Nous proposons ci-dessous une visualisation de ces trois approches sur les figures 2.20, 2.21, 2.22 :

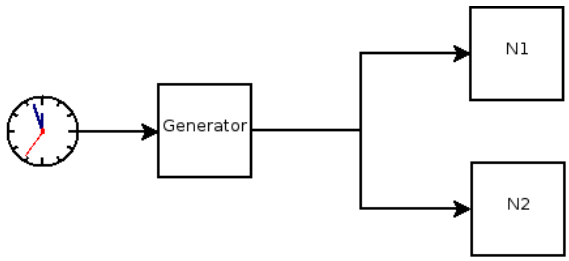


Figure 2.20 – initialisation par rapport au temps sur 1 générateur

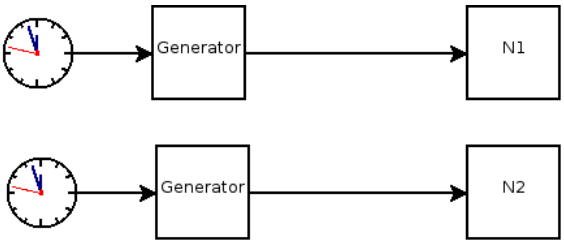


Figure 2.21 – initialisation par rapport au temps sur N générateurs

On peut noter sur 2.21 qu'il peut facilement y avoir un problème d'initialisation identique lors du lancement sur calcul distribué.

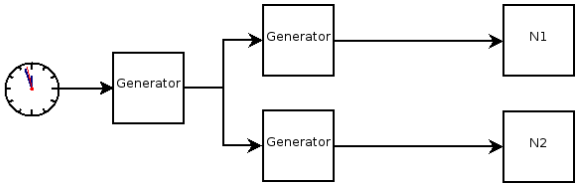


Figure 2.22 – initialisation par rapport au temps sur 1 générateur afin d'alimenter N générateurs

Il faut noter que nous avons mis en entrée de la figure 2.22 une horloge pour décider de la graine initiale du premier générateur, toutefois pour des soucis de reproducibilité nous pourrions mettre une valeur choisie par une personne afin que l'expérience totale soit facilement refaite.

2.4.2.2 Génération de n suites à partir de l'extraction de sous-suites d'une suite

Une question intéressante consiste à étudier le caractère aléatoire des sous-suites extraites de la suite aléatoire principale qui alimente les distributions N_1 et N_2. Est-ce que la sous-suite donnée à N_1 a conservé son caractère aléatoire ? Autrement dit, est-ce que le bruit introduit n'est pas biaisé ?

En effet les modèles étudiés peuvent posséder un nombre non-négligeable de variables aléatoires et dont le calcul peut-être conditionné par la valeur d'une autre. Par exemple, en fonction de la valeur d'une variable on lance le calcul ou non d'une autre variable. Qu'en est-il alors du caractère uniforme et indépendant des sous-suites extraites ?

La figure 2.23 représente les termes X_0, X_1, X_2, ... d'une suite aléatoire.

Figure 2.23 – une suite aléatoire

Si nous associons certains éléments de cette suite à des bruits spécifiques (b1, b2, b3), nous obtenons la figure 2.24 qui montre la répartition des éléments de la suite en fonction de certaines variables aléatoires d'un modèle quelconque.

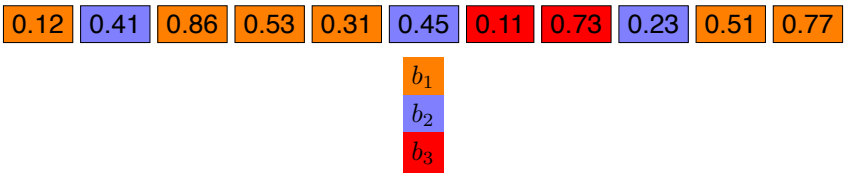


Figure 2.24 – utilisation d'un générateur aléatoire

La figure 2.25 montre cette répartition mais en réorganisant le tout par rapport à chaque variable afin de mieux visualiser le problème.

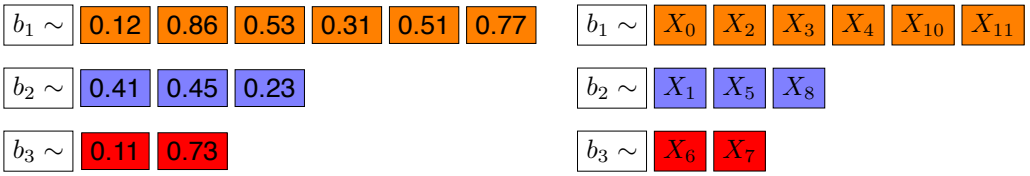


Figure 2.25 – une suite aléatoire répartie par variable

Ce problème est abordé dans la littérature (L'Ecuyer, Oreshkin et Simard 2014) qui en fait un état de l'art et une discussion, et propose plusieurs solutions :

- un générateur unique pour tous les flux mais cela consisterait en un grand goulot d'étranglement en terme computationnel.
- un générateur différent pour chaque flux, i.e. un générateur de même type mais avec des paramètres différents. C'est ce que fait l'algorithme MTDC (Matsumoto et Nishimura 1998a) par exemple, mais ça a le défaut d'être lent en terme d'exécution en temps réel ou alors il faut appliquer un mécanisme de sauvegarde de ces paramètres en faisant tourner constamment un programme stockant les paramètres de nouveaux générateurs en mémoire.
- l'algorithme dit du "saute-mouton" (leapfrog) qui consiste à diviser le flux d'un seul générateur aléatoire en prenant des éléments de la suite les uns après les autres en ayant un décalage.
- l'algorithme qui consiste à séparer un flux en N flux de taille égale (sequence splitting)
- un seul générateur aléatoire avec une graine aléatoire pour chaque flux qui est une alternative lorsque l'une des deux dernières solutions n'est pas viable (indexed sequences). L'Ecuyer, Ore-

shkin et Simard 2014 mentionnent le fait qu'il y a une possibilité d'overlapping mais qu'elle est négligeable. Cette graine aléatoire est idéalement choisi en sortie d'un autre générateur aléatoire.
— utiliser une bibliothéque dédiée comme SPRNG (Mascagni et Srinivasan 2000).

Nous avons rejeté le premier algorithme étant donné que cela aurait pénalisé la simulation en parallèle. Pour ce qui du deuxième algorithme, il y a une forte lenteur computationnelle, par exemple nous n'avions pas trouvé 10 000 générateurs différents après 20 min de calcul. Cela demande donc de faire un effort conséquent au niveau de l'ingénierie logicielle afin de chercher, sauvegarder et charger ces paramètres de générateurs à des fins de simulations. Cela aussi doit être mis en perspective vis à vis du fait que nous utilisons parfois plus de 500 000 particules (simulations) dans certains algorithmes qui sont elle-même composées de plusieurs bruits de modélisation et d'observation. En ce qui concerne les algorithmes du leapfrog et du sequence splitting, cela demande de bien connaitre les algorithmes sous-jacents et les techniques relatives aux générateurs de nombres aléatoires. La technique du sequence splitting est particulièrement problématique lorsque la période du générateur est très longue (L'Ecuyer, Oreshkin et Simard 2014). Pour ces différentes raisons, nous nous sommes orientés en premier lieu vers la dernière solution.

Toutefois il s'agit maintenant de chercher la granularité au niveau de la parallélisation des générateurs. Est-ce qu'un générateur par simulation est suffisant ou est-ce que nous devons descendre au niveau d'un générateur par bruit. Pour ce faire, nous allons utiliser des outils permettant d'étudier les propriétés de ces suites aléatoires. Par sa qualité statistique et sa disponibilité dans les bibliothèques logicielles classiques en C++, nous avons choisi de tester ces stratégies avec le Mersenne Twister.

2.4.2.3 Tests

Les tests se découpent en deux types : soit on procède à une étude analytique, soit on procéde à des tests expérimentaux en se livrant à une analyse des suites générées par ces générateurs.

Tests théoriques Les tests théoriques sur les générateurs portent sur la notion d'indépendance et d'uniformité et sont difficilement applicables sur des cas réels ne serait-ce qu'avec un simple générateur comme le lcg. On a ainsi recours à des études empiriques telles celles définies dans (Knuth 1997) et reprises par L'Ecuyer lorsqu'il a compilé les différents tests empiriques existants au travers de sa bibliothèque TestU01 (L'Ecuyer et Simard 2007).

Tests expérimentaux Les tests expérimentaux ont émergé au fil des diverses utilisations, ils sont une vision simplifiée de certains problémes poussant les générateurs dans leurs limites quant à la qualité statistique des suites produites.

Au lieu d'utiliser un seul test, la communauté a pour habitude de faire passer une batterie de tests et de s'y référer comme un "standard". Ainsi on retrouve les batterie de tests "DIEHARD", "Small Crush", "Crush", "Big Crush" (L'Ecuyer et Simard 2007) et d'autres.

On pourrait se contenter du "Small Crush" mais certains générateurs n'ont de difficultés qu'à partir du "Big Crush". Il faut bien intégrer la notion fondamentale et limitante des tests qui stipule que bien que ceux-ci peuvent être validés cela ne veut pas dire pour autant que les suites sont exemptes de défauts. Les tests expérimentaux permettent simplement de renforcer la confiance en certains générateurs en diminuant l'incertitude les accompagnant.

Nous allons maintenant visualiser la problématique des sous-suites dans le cadre de notre cube de simulation.

2.4.2.4 Visualisation des stratégies

La figure 2.26 nous permet de visualiser la problématique des interactions entre les différents éléments de la suite aléatoire. En effet dans ce premier graphique nous avons mis un générateur par particule.

Ainsi les alternances orange, bleu, rouge se servent toutes d'une source d'aléatoire pour une particule. À l'inverse dans la figure 2.27 nous pouvons voir qu'en ce qui concerne une particule, chaque couleur (et donc chaque bruit) possède une source unique aléatoire. Ainsi nous avons chaque bruit, bien que configuré de manière identique au niveau de leurs paramètres, qui peuvent maintenant utiliser le caractère aléatoire du générateur pseudo-aléatoire associé.

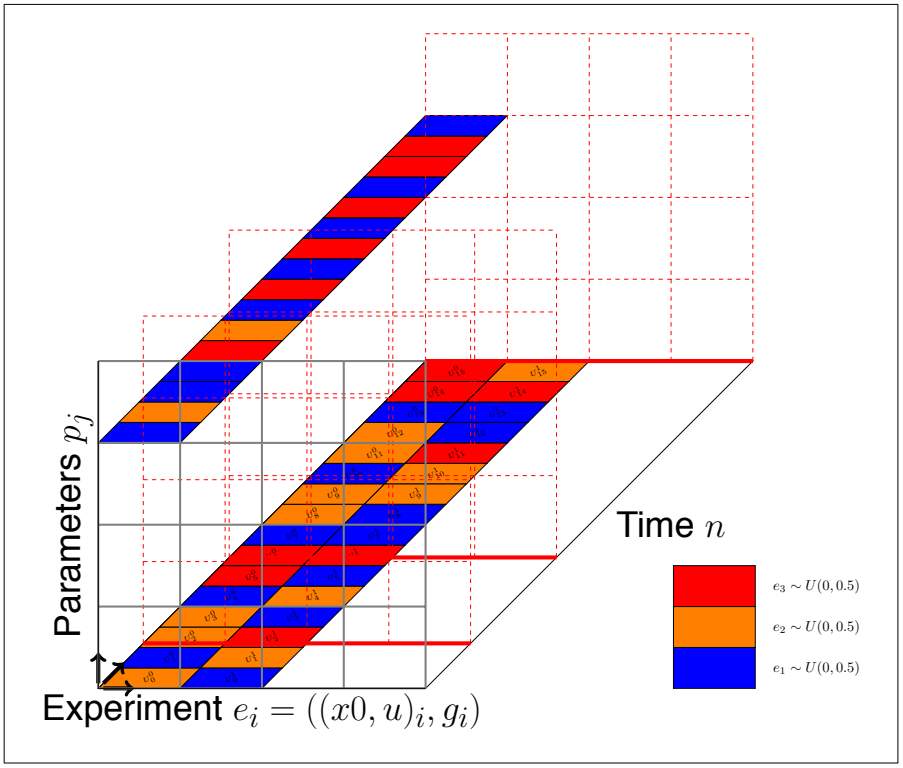


Figure 2.26 – visualisation des répartitions des bruits lors des simulations - 1 générateur par particule/simulation/-système virtuel

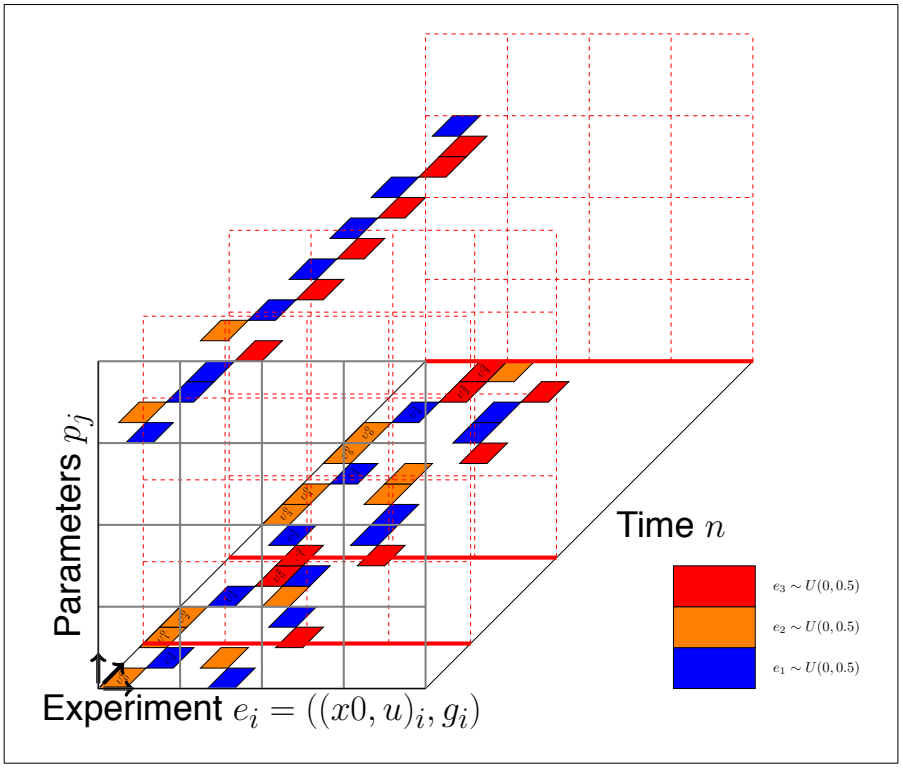


Figure 2.27 – visualisation des répartitions des bruits lors des simulations - 1 générateur par bruit par particule

2.4.2.5 Test de corrélation

Le test de corrélation consiste à produire autant de larges suites qu'il y a de bruits et de calculer la corrélation entre les colonnes produites par la matrice contenant toutes les données.
Les études que nous avons menées ne montrent pas de corrélation significative entre les sous-suites extraites. Toutefois nous allons voir dans la section suivante qu'en complexifiant les tests nous pouvons obtenir des comportements nous permettant de douter de la stratégie utilisant un seul générateur.

2.4.2.6 Test de qualité pseudo-aléatoire

Nous allons lancer plusieurs tests qui vont simuler les différentes stratégies. Soit nous avons un seul générateur, soit nous avons un générateur par bruit.

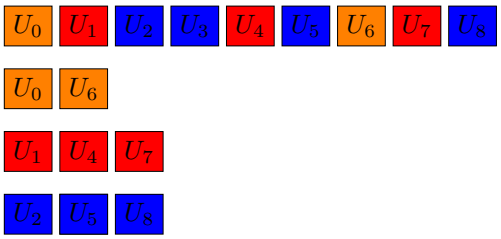


Figure 2.28 – visualisation des suites utilisées avec un seul générateur

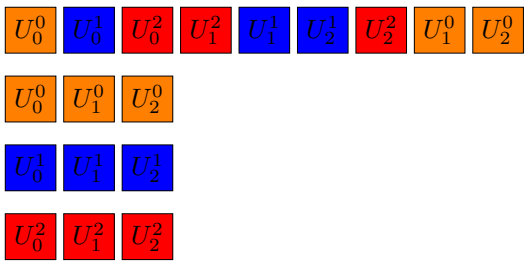


Figure 2.29 – visualisation des suites avec N générateurs

Le Mersenne Twister est pris en défaut sur le test "Big Crush" lorsque l'on met un seul générateur. En effet certains des tests qui ne lui sont pas problématiques quant il est utilisé par une seule suite le deviennent avec plusieurs suites extraites. On peut se référer à l'annexe B qui montre différents résultats de ces tests. On constate une variabilité dans les résultats vis à vis des tests qui ne passent pas. Normalement il est censé uniquement échouer sur un test dédié à la cryptographie, alors que là nous avons des tests en dehors de ce cadre qui des fois passent et des fois ne passent pas.

C'est pour cette raison que nous avons privilégié un algorithme de simulation qui associe un générateur par bruit de modélisation ou d'observation plutôt qu'un générateur par simulation.

Nous avons vu qu'il existe une grande variété de générateurs de nombres pseudo-aléatoires. La nécéssité d'évaluer à la fois un grand nombre de bruits de modélisation et d'observations par simulation et un grand nombre de simulations exige l'évaluation de flux de nombres aléatoires en parallèle. Cette génération en parallèle peut être faite techniquement par le biais de diverses techniques comme le "sequence splitting", le "sequence indexing", le "leap frogging", l'attribution d'une "graine aléatoire" par flux (Coddington 1997 ; L'Ecuyer, Oreshkin et Simard 2014).

Nous avons choisi la dernière option. L'implémentation actuelle pourrait être cepedant améliorée vis à vis de la gestion de cette initialisation. Toutefois, ce choix a été fait par la contrainte du design d'algorithmes tel que le CPF (Chen et al. 2012) qui peut voir son nombre de particules augmenter ou descendre en cours de réalisation. Nous avons testé notre approche pour rechercher d'éventuelles corrélations ou problèmes avec les tests classiques du domaine et n'avons pas trouvé d'indications de problèmes majeurs.

Il existe le problème d'overlapping qui peut être présent mais qui est considéré comme négligeable dans ce cas (L'Ecuyer, Oreshkin et Simard 2014).

L'utilisation d'une librairie externe n'a pas été envisagée à cause de la complexité de cette solution vis à vis de l'objectif principal de cette thèse. Il faut bien garder à l'esprit que l'objectif final reste l'articulation et le développement d'un ensemble formel autour des notions d'analyse et de calibration des modèles.

Bien entendu, il serait bon de revenir sur cette partie maintenant que la structure générale de l'ensemble est présente en ajoutant notamment un support garantissant la reproducibilité des expériences.

Nous allons voir maintenant quel est le langage dédié à la modélisation.

2.5 Langage dédié à la modélisation

Dans cette section, nous présentons les spécificités d'un langage dédié (Bayol et al. 2015). Ce langage a été implémenté au moyen de LLVM (Lattner et Adve 2004) afin de générer du code interprétable par la machine.

Un langage dédié est un système formel restreint syntaxiquement et sémantiquement à un domaine particulier. Ils sont à mettre en contraste avec les langages généraux permettant de développer n'importe

quel type de programme, à l'instar de C++, Java ou Python.

L'une des formulations les plus connues concernant l'importance de ces langages dédiés se trouve dans (Landin 1966) qui prévoit une forte augmentation de ce type de langage étant donnés les besoins de communications entre experts. On retrouve dans (Mernik, Heering et Sloane 2005) et (Spinellis 2001) une analyse du développement d'un langage dédié au moyen de patrons de conception.

2.5.1 Historique

Dans cette section nous ferons un rapide tour des langages dédiés à la modélisation des systèmes complexes et ceux dédiés aux probabilités et à la manipulation des distributions, étant donné que le langage décrit met en jeu les deux aspects.

En effet, nous considérons à la fois des transitions de systèmes dynamiques mais aussi la possibilité de définir des distributions afin d'ajouter des bruits de modélisation ou d'observations.

Parmis les langages dédiés à la modélisation des systèmes complexes, on retrouve principalement la famille Modelica/Dymola (Fritzson et Bunus 2002) qui propose un outil exploitant à la fois du code source et une visualisation sous forme de schéma-bloc. Il existe bien sûr d'autres outils à paradigme visuel tel que Simulink (Beucher 2006) ou Scicos (Campbell, Chancelier et Nikoukhah 2006b), qui sont performants quant à la simulation mais ne possèdent pas, à notre connaissance, de traduction du schéma-bloc vers un code source dans un langage dédié à la modélisation, bien que traduisibles vers un langage de type C.

Nous donnons dans le code 2.16 un aperçu du modèle Lotka-Volterra dans le langage Modelica. Il faut noter qu'à la différence du langage dédié présenté ici, il est orienté vers les équations différentielles avec un mécanisme de résolution d'équations. À l'inverse nous avons un langage qui considère seulement des systèmes discrets.

```
model LotkaVolterra
parameter Real alpha=0.1 "Reproduction rate of prey";
parameter Real beta=0.02 "Mortality rate of predator per prey";
parameter Real gamma=0.4 "Mortality rate of predator";
parameter Real delta=0.02 "Reproduction rate of predator per prey";
parameter Real x0=10 "Initial prey population";
parameter Real y0=10 "Initial predator population";
Real x(start=x0) "Prey population";
Real y(start=y0) "Predator population";
initial equation
x = x0;
y = y0;
equation
der(x) = x*(alpha-beta*y);
der(y) = y*(delta*x-gamma);
end ClassicModelInitialEquations;
```

Code 2.16 – code source Modelica

Parmi les langages dédiés aux probabilités, nous pouvons retrouver les initiatives de Venture (Mansinghka, Selsam et Perov 2014) ou encore BLOG (Getoor et Taskar 2007).

Ces langages mettent une emphase sur la notion d'opérateur pour les distributions, souvent dénoté ~, permettant d'associer des distributions aux variables. Ils permettent ainsi de formuler des requêtes pour inférer des valeurs plus "facilement" qu'avec un langage général. Un aperçu des fonctionnalités de tels langages est donné dans le code 2.17.

L'idée générale du langage dédié développé dans cette thèse est de proposer à la fois une vue rapide

```
mu ~ uniform(-10, 10)
x ~ normal(mu, sigma = 1)
observe(x = 5)
infer(mu)
```

Code 2.17 – code source d'un langage dédié aux probabilités

sur le modèle comme ce que permet Modelica mais aussi de proposer des opérateurs permettant de manipuler le cadre statistique plus facilement notamment en créant des raccourcis vers leurs noms pour les estimations paramétriques.

2.5.2 Prérequis sur LLVM

Afin de bien comprendre comment le langage dédié a été implémenté, il est nécessaire de faire un point technique sur LLVM.

"Low-Level Virtual Machine" ou "LLVM" (Lattner et Adve 2004) est une bibliothèque C++ qui contient les éléments nécessaires à la construction d'un compilateur pour un langage quelconque. Elle a émergé dans les années 2000 à l'université de l'Illinois et a été depuis soutenue par des acteurs majeurs de l'industrie du logiciel (Google, Apple, ...). On la retrouve à la base du compilateur Clang. La communauté est large et très active notamment au moyen de sa mailing-list.

L'une des caractéristiques de LLVM est d'introduire un langage intermédiaire de représentation des données qui permet de découpler facilement la partie "front-end" du compilateur (exploration de l'arbre syntaxique abstrait) et sa partie back-end (génération du code pour une architecture matérielle particulière). Sur la figure 2.30 nous donnons un exemple des différents niveaux.

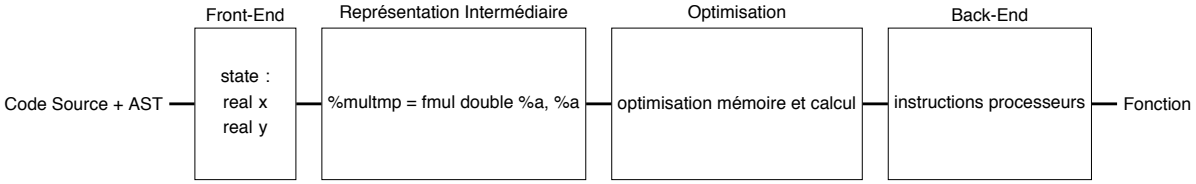


Figure 2.30 – découpage Front-End - IR - Back-End de llvm

Autrement dit à partir du code source et de la construction d'un arbre syntaxique abstrait nous pouvons générer à la volée du code machine qui sera interprété par notre fonction de simulation.

Ainsi auparavant nous écrivions notre modèle au moyen du c++ et procédions à une phase de compilation avant de pouvoir exploiter une librairie dynamique dorénavant, nous pouvons nous concentrer sur l'écriture du modèle dans un langage simplifié sans se soucier à la fois de la complexité du langage hôte et des intermédiaires techniques. Cette différence est exprimée dans les codes 2.18 et 2.19.

```
edit model.cpp
compile-model model.cpp -o model.so
simulate --model model.so --observation-function [...]
```

Code 2.18 – processus de simulation à partir du c++

Cela permet notamment d'améliorer la vitesse de développement et par conséquence de formuler plus facilement de nouvelles hypothèses quant au fonctionnement de certains processus existants.

```
edit model.dsl
simulate --model model.dsl --observation-function [...]
```

Code 2.19 – processus de simulation à partir du dsl

2.5.3 Implémentation

Notre langage source sera le langage décrit grâce aux éléments de langage définis dans la table 2.1. Le code 2.20 est une implémentation du modèle Lotka-Volterra avec le langage dédié que nous avons créé. On reconnait bien les structures de données relatives aux variables d'états (**state**) et de paramètres (**parameters**), ainsi que les bruits de modélisation et les bruits d'observations (**noises**). Il existe aussi une section relative aux variables externes (**control**) qui n'est pas illustrée étant donné que le modèle n'en possède pas.

```
dynamical_model: lotka_volterra
 state:
  real pred = 0.11
  real prey = 0.2

 parameters:
  real a = 0.01
  real b = 0.2
  real c = 0.05
  real d = 0.1

 noises:
  alpha ~ normal(0., 0.05)
  beta ~ normal(0., 0.05)

 #in equations we have symbols that represent state list, parameters ...
 # xl[n] -> state list at time n
 # u -> control
 # p -> parameters
 # e -> noises
 equations:
  x[n+1].pred = (p.a * x[n].pred - p.b * x[n].pred * x[n].prey) * (1 + e.alpha())
  x[n+1].prey = (p.d * x[n].pred * x[n].prey - x[n].prey * p.c) * (1 + e.beta())

observation_model: pop
 equations:
  y["pop."] = xn.pred + xn.prey

observation_model: noised_pop
 noises:
  gamma ~ normal(0., 0.05)
 equations:
  y["noised pop."] = (xn.pred + xn.prey) * (1 + e.gamma())
```

Code 2.20 – modèle Lotka-Volterra avec le langage dédié

En ce qui concerne les performances actuelles de l'implémentation il y a plusieurs éléments que nous n'avons pas implémentés tel que l'alignement mémoire en fonction des structures de données ou encore l'exploitation de la vectorisation.

Nous pensons que cette approche par langage dédié permettra de créer un effet de levier sur la création et l'analyse des modèles. Pour l'instant, le périmètre des langages possibles avec ce langage dédié est restreint à des modèles avec compartiments simples. Les compartiments vectoriels, matriciels ou encore sous forme de graphes ne sont pas encore supportés. Toutefois, il reste une étape à approfondir qui consiste à introduire le modèle dans l'espace de complexité propre au domaine d'étude, à savoir l'analyse quantitative.

Nous avons vu au cours de ce chapitre le type de modèle que nous prenions en compte, notre noyau de modélisation et simulation articulé autour du langage dédié embarqué en C++ ainsi que notre langage dédié. Nous allons maintenant voir comment ouvrir nos algorithmes à d'autres types de modèles et de simulateurs par le biais des simulateurs externes.

2.6 Simulateur externe

Pour rappel, la signature générale de nos simulations en utilisant le vocabulaire défini en 2.1 se formule ainsi :

$$\begin{aligned} &simulate :: m \to [e] \to [p] \to [so] \\ &simulate :: (i, f) \to [((x, u), g)] \to [p] \to [so] \\ &simulate :: ((x, u, p) \to (x, u, p), (t, [x], u, p, \epsilon) \to obs) \to [((x, u), g)] \to [p] \to [so] \end{aligned} \tag{2.7}$$

où

- m est un modèle composé des fonctions d'initialisation et de récurrence
- x est la structure de données contenant les variables d'états
- u est la structure de données contenant les variables de contrôle
- p est la structure de données contenant les paramètres
- ϵ est la structure de données contenant les bruits de modélisation
- obs est une structure de données temporaire contenant une observation à un temps t
- g est la fonction d'observation globale du système
- so est la structure de données contenant les différentes observations du modèle.

Si la notion de simulation telle que nous l'avons vue, au sens où une simulation est une composante du vecteur $[so]$ obtenu après le run, est celle qui est déployée au sein des routines de simulation de PYGMALION, nous avons aussi la possibilité d'abstraire tout type de simulateur existant en considérant une fonction qui prend en entrée un jeu de paramètres et qui fournit en sortie un ensemble d'observations.

Ainsi un simulateur est vu tel que : *simulate : parameters* $\to$ *system_observation* ce qui correspond à une vision plus classique d'un simulateur en boîte noire tel que *simulate :: input* $\to$ *output*.

Cette propriété découle naturellement de la formulation introduite en 2.2.2.3 lorsque nous définissions la simulation comme *simulate :: m* $\to$ *[e]* $\to$ *[p]* $\to$ *[so]*.

En effet, nous considérons qu'un simulateur externe possède déjà un modèle préconfiguré ainsi que le jeu d'expérience associé permettant la création des observations sur le système virtuel considéré. Ainsi nous nous retrouvons avec la formulation :

$$\begin{aligned} &external\_simulate : [p] \to [so] \\ &external\_simulate = simulate^{m \times [e]} \end{aligned} \tag{2.8}$$

Sur ce principe, nous avons donc construit un cadre permettant l'appel à un simulateur externe dans nos algorithmes. Nous pouvons voir la différence d'appel de nos différents algorithmes entre le code 2.21 et le code 2.22. Les options sont les mêmes sauf pour les états initiaux et les variables externes qui sont pris en charge pour le simulateur. Le simulateur doit être pour l'instant un script bash nommé `run.sh` avec comme premier paramètre le chemin vers le fichier d'entrées et comme second paramètre le chemin vers le fichier de sorties. C'est une solution qui pour l'instant est expérimentale et qui doit être améliorée,

notamment pour ce qui est des entrées-sorties étant donné que c'est un goulot d'étranglement en termes de temps calcul.

```
./bin/algorithm --model ./model_list/liblv.so
                --initial-state x0.dat
                --control u.dat
                --initial-parameters p0.dat
                --selected-parameters ["a","b"]
                --target y_exp.csv
```

Code 2.21 – exemple de commande pour les moindres carrés généralisés avec un modèle de la plateforme

```
#there are no initial-state and control since they are embedded in the simulator (its context)
./bin/algorithm-external --simulator ./simulator_list/simulator1/run.sh
                         --initial-parameters p0.dat
                         --selected-parameters ["a","b"]
                         --target y_exp.csv
```

Code 2.22 – exemple de commande pour les moindres carrés généralisés avec un simulateur externe

Évidemment, les entrées-sorties seront compatibles en termes de formats de fichiers dont nous pouvons trouver des exemples dans l'annexe D.

Nous allons maintenant voir quelle est la signature type associée aux algorithmes avant de conclure ce chapitre.

2.7 Algorithmes

Nous avons retenu comme définition formelle d'un algorithme la signature suivante :

$$A : C \times D \to R \tag{2.9}$$

où nous noterons C l'espace de **C**onfigurations de l'algorithme, D son espace de **D**épart qui correspond aux entrées propres à l'espace du modèle étudié et enfin R son espace de **R**ésultats.

On voit que l'on peut ainsi définir non seulement un algorithme mais aussi un ensemble d'algorithmes relativement à son espace de configurations. Par exemple nous avons $A^{C_1} : D \to R$ ou $A^{C_2} : D \to R$ où C_1 et C_2 sont des instanciations particulières de C. On peut donc avoir des versions différentes des algorithmes permettant la reprise sur erreur plus facilement au cas où des paramètres seraient trop stricts pour la recherche d'une solution.

Dans le cas de la simulation discrète, il n'y a pas de configuration particulière à prendre en compte, d'où son côté plus simple. Il est à noter que si nous voulions gérer la simulation en temps continu, il faudrait fournir des paramètres de la configuration permettant de prendre en charge l'algorithme d'intégration en temps (de type Runge-Kutta 4 par exemple).

2.8 Conclusion

Nous avons vu, le long de ce chapitre, l'ensemble des éléments ayant permis la construction du langage qui a été implémenté au sein d'une plateforme logicielle.

Nous sommes partis de la représentation d'état et, en passant par un système informatique minimal, avons pu développer l'ensemble des éléménts nécessaires à la construction de la chaîne d'évaluation et d'exploration des modèles que nous utiliserons par la suite. Nous avons aussi abstrait notre système de départ pour le confronter à tout type de simulateur et avons obtenu une écriture qui dépasse le premier cadre de modélisation.

Nous allons maintenant montrer les éléments qui composent le flux de travail général autour de l'analyse quantitative des modèles mécanistes.

Chapitre 3

Analyse quantitative des modèles mécanistes

L'analyse quantitative correspond à l'utilisation des mathématiques issues du champ des statistiques afin de mieux quantifier les états pris au cours du temps par un système virtuel. Cette quantification permet de mieux comprendre ce système et ainsi d'améliorer les prises de décisions quant à son écriture. Si nous avons une grande confiance, après calibration et validation, dans le système virtuel comme représentation du système réel, alors ces études nous permettent aussi de mieux comprendre le système réel et d'utiliser le système virtuel afin de prédire les évolutions réelles.

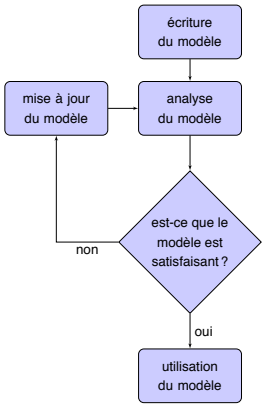


Figure 3.1 – flux général

Par analyse, nous entendons un sens large qui comprend à la fois l'analyse des sorties en fonction des entrées du système mais aussi la recherche des paramètres permettant d'approcher les données expérimentales, ainsi que l'étude simultanée de plusieurs structures de modèles afin de sélectionner la meilleure par rapport à une étude particulière.

Dans ce chapitre, nous verrons dans un premier temps quels sont les grands principes nous permettant de comprendre le flux de travail général ainsi que les problématiques liées au domaine de validité d'un modèle. Dans un second temps, nous présenterons les grands principes et algorithmes composant les différentes fonctionnalités, à savoir : analyse de sensibilité, estimation paramétrique, analyse d'incertitudes, et sélection de modèles.

Rappelons qu'à la fin des années 90, des chercheurs de l'université de Wageningen se sont réunis afin de présenter dans un ouvrage (Scholten et al. 2000) les bonnes pratiques de modélisation à destination des modèles de gestion de l'eau. C'est ce document que nous avons pris comme référence afin de visualiser l'ensemble des processus interagissant dans la modélisation mécaniste de nos problèmes. Nous pourrions compléter les éléments présentés en se reférant à des ouvrages et articles tel que (Pavé 2006).

On trouvera en annexe H des extraits des représentations graphiques des flux de modélisation suivant les bonnes pratiques de modélisation et ayant inspiré et guidé nos travaux. Toutefois, ces flux ne donnent pas de conseils concernant les algorithmes à utiliser étant donné que ceux-ci doivent être choisis en fonction de la modélisation sous-jacente.

Nous allons voir ci-dessous la méthodologie générale avant de passer en revue les algorithmes implémentés pour chaque fonctionnalité.

3.1 Méthodologie générale

Avant de rentrer plus en détail dans les composants de la chaîne des bonnes pratiques de modélisations (Cournède et al. 2013), nous allons voir un cadre général qui conduit à l'utilisation de ce flux de travail.

La figure 3.1 présente la boucle de rétroaction générale que l'on peut trouver entre la première écriture du modèle et son utilisation lorsqu'il a été validé. La principale difficulté réside dans la définition de ce que l'on appelle satisfaction et validation. Dans notre cas, nous ferons passer le modèle calibré par une série de critères de sélection afin de positionner le modèle par rapport à un modèle de référence par exemple.

3.1.1 Flux de travail général

La figure 3.2 est un schéma montrant la chaîne d'analyse des modèles qui est possible avec le système créé.

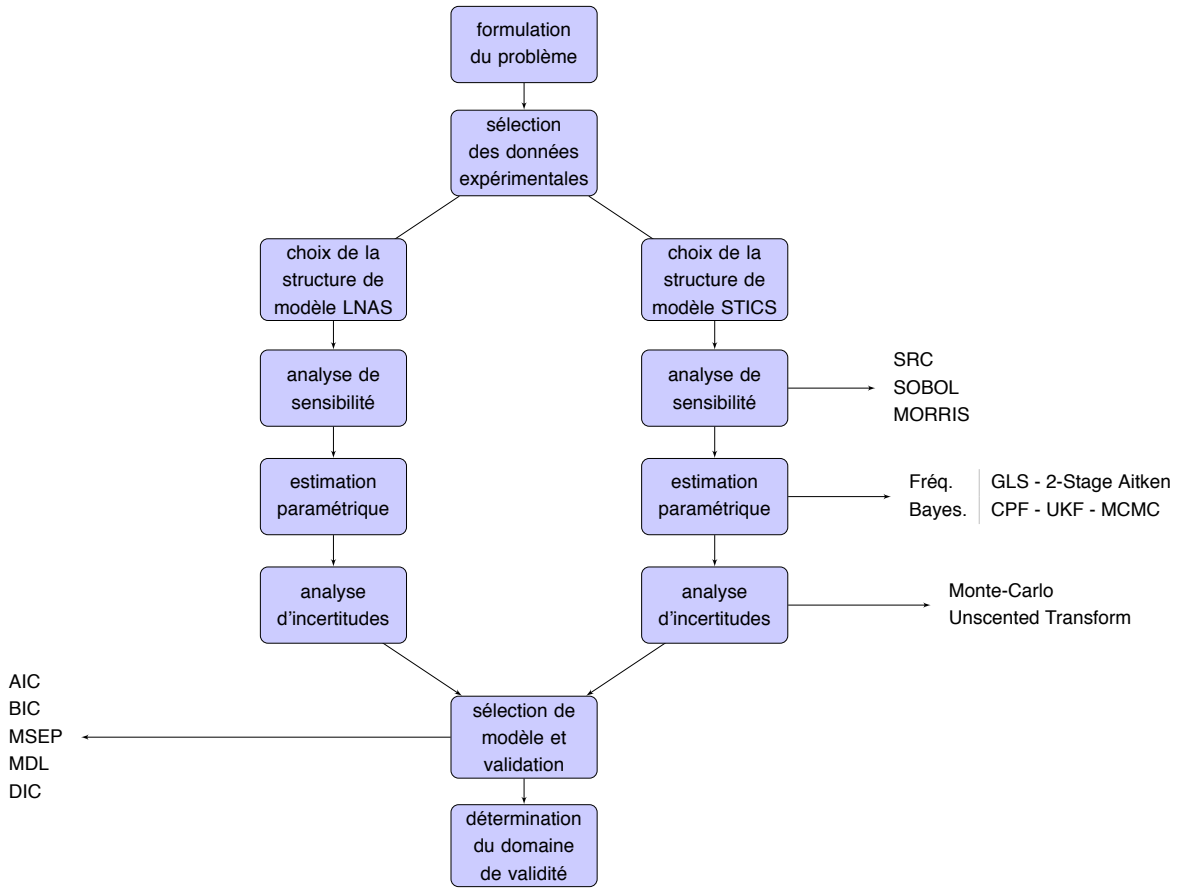


Figure 3.2 – flux des fonctionnalités

Après la formulation du problème et l'acquisition des données nécessaires à son étude nous devons choisir une structure de modèle. Cette structure nous est donnée par les théories biologiques utilisées

durant la première phase de création du modèle. Ainsi, certains processus seront décrits finement ou non, et certains fonctionnements seront approchés de telle ou telle manière. Le choix de la structure n'est pas évident et on peut se retrouver avec deux ou trois modèles ayant une structure plus ou moins équivalente, ou permettant de répondre à une certaine problématique. Comment choisir le meilleur par rapport à une situation donnée ?

Dans un premier temps, nous conduisons une **analyse de sensibilité** afin de comprendre les interactions entre paramètres et trouver quels sont ceux qui ont le plus de poids dans la simulation des variables d'intêret. Ce travail est suivi de la spécification du modèle pour un système donné, c'est à dire sa paramétrisation. À partir de données expérimentales, il s'agit de réaliser une **estimation** pour trouver les valeurs ponctuelles (approche fréquentiste) ou la distribution sur les paramètres (approche bayésienne).

Une fois cette estimation des paramètres, on peut faire une **analyse d'incertitudes** afin de caractériser les incertitudes sur la sortie en fonction de celles sur les paramètres, ou plus généralement sur les entrées.

Ces étapes menées à bien, nous avons un modèle candidat à la résolution de la problématique donnée.

Il peut par conséquent rentrer en compétition avec d'autres modèles en termes de structures et d'hypothéses formulées sur le fonctionnement. Si l'on conduit une étude sur le modèle STICS et une étude sur le modèle LNAS, on peut se poser la question de savoir quel modèle est le plus adapté (i.e. celui qui fera les meilleures prédictions). Cela nous mène par conséquent à la phase de sélection de modèle.

Il se peut qu'une fois le modèle sélectionné et validé, il ne soit plus juste. Par exemple, à cause de modifications trop importantes durant la vie du système dues à des évolutions climatiques. Nous aurons ainsi recours à **l'assimilation de données** afin de recalibrer le modèle à la volée au cours de son exploitation.

Il est important de ne pas oublier que l'ensemble des moyens mis en œuvre consiste à pouvoir fournir à la communauté scientifique des modèles qui ont été validés au regard de certaines données expérimentales.

Aujourd'hui, l'ensemble des fonctionnalités présentées s'axent plus sur la sélection des modèles afin de répondre à la question : *Quel est le modèle le mieux adapté par rapport à une problématique donnée ?* Cette question en amène une autre (ou la précède selon le point de vue) qui est : *Est-ce que mon modèle est valide par rapport à ma problématique ? Quel est son domaine de validité ? Dans quelles conditions est-il exploitable ?* Cette question est importante à plus d'un titre car c'est en allant vers sa résolution que l'on peut permettre d'éviter de mauvaises utilisations des modèles, par exemple faire des conjectures en dehors du domaine de validité. Pour parvenir à cette fin, il est nécessaire de poursuivre les études sur les analyses de résidus par exemple.

Dans les sections suivantes, nous présenterons l'ensemble des fonctionnalités et algorithmes qui ont été étudiés dans le cadre de cette thèse. Il faut comprendre que cette thèse architecture le développement de ces méthodes au sein d'une même plateforme. Ainsi, il est important de noter que certaines implémentations ne sont pas uniquement de notre fait mais ont été réalisées avec le concours de collégues ou autres dans un travail d'équipe.

L'ensemble des algorithmes bénéficient de l'exploitation du multithreading sur la génération des données étant donné le caractère vectoriel de la simulation [1]. De plus certains passages des algorithmes auront eux aussi un support du multithreading.

Dans le chapitre suivant, nous ne sélectionnerons qu'un sous-ensemble des algorithmes permettant d'illustrer les différentes étapes du processes d'analyse quantitative du modèle comme présenté dans la figure 3.2.

1. le terme vectoriel ici ne fait pas référence au SIMD et à ce type d'optimisation mais plutôt à OpenMP et MPI

3.1.2 Typologie des études

À notre connaissance, il n'existe pas d'études standardisées et répandues dans notre domaine mais un ensemble de tests et de recommandations que l'on peut trouver dans certains ouvrages de référence comme (Brun et al. 2006) ou (Faivre et al. 2013). Il est à noter que ce travail fait suite à celui présenté dans (Baey et al. 2012) qui permet de donner un premier aperçu d'une étude comparative multi-modèles en agronomie. Dans notre cas, nous nous contenterons de faire des études de cas séparées.

Nous détaillons ci-dessous les deux types d'études que sont :

— les études pas à pas qui correspondent à une analyse générale du modèle, une validation des algorithmes d'estimation choisis sur données simulées, une analyse de sensibilité pour l'estimation paramètrique, une estimation paramètrique et la caractérisation des incertitudes.

— les études automatisées qui permettent une exploration "à l'aveugle" du modèle. C'est à dire qu'à chaque mise à jour du modèle (ajout d'équations, changement de paramètres), et afin de faire émerger des comportements types du modèle, nous allons lui faire passer une batterie de tests. Ces tests permettront de voir son comportement vis à vis du nombre de particules nécessaires à une bonne analyse ou encore de trouver des intervalles de variation pouvant présenter des caractéristiques intéressantes. Cela peut aussi servir de suites de tests pour la résolution des conflits lors de la conception des modèles.

Nous détaillons les étapes et possibilités fournies par ces deux types d'études. Par la suite, nous concentrerons nos applications sur le premier type d'étude.

3.1.2.1 Études pas à pas

Ce genre d'étude est conduite soit dans le cadre du développement du modèle par un modélisateur afin d'avoir un retour d'information sur le comportement du modèle, soit en fin de modélisation afin de construire un programme de prévision qui sera déployé en environnement de production.

Nous devons distinguer trois types de données qui seront utilisées durant cette étude : les données simulées qui sont produites par le modèle, les données réelles dites "d'apprentissage" afin d'estimer un jeu de paramètres et les données réelles dites de "validation" afin de regarder la qualité de prévision après estimation.

L'étude pas à pas consiste en premier lieu à une analyse générale du modèle où nous utiliserons des méthodes d'analyse de sensibilité pour comprendre où se situent les phases linéaires et non-linéaires du modèle. Puis nous essaierons d'identifier quels sont les paramètres les plus influents lors de l'estimation en utilisant l'algorithme "sobol-gls". Par la suite nous validerons le fonctionnement de l'estimation sur données simulées. Une fois cette étape validée, nous réaliserons une estimation paramètrique d'un système réel à partir des données réelles, décrites en section 3.2.2.1. Cette estimation sera faite soit avec un algorithme de type MCMC dans le cadre d'un modèle stochastique, soit avec un estimateur d'Aitken dans le cas d'un modèle déterministe. Enfin, nous ferons une analyse d'incertitudes du modèle en sortie de l'estimation. Cela nous permettra de voir si le modèle calibré est robuste et peut être utilisé pour de la prévision sur un jeu de données de validation. Finalement, nous calculerons quelques critères de prévision afin de quantifier la qualité des modèles calibrés.

3.1.2.2 Automatisation du pas à pas

D'une part, et c'est une évidence, nous pouvons automatiser les études pas à pas que nous avons décrites précédemment. Cela permet notamment de faire repasser les études et produire des rapports automatiquement lorsqu'un modèle subit une mise en jour en termes de structure ou de données.

3.1.2.3 Automatisation pour la détection de bugs

D'autre part, nous pouvons créer des analyses plus systématiques qui peuvent être interprétées comme de l'exploration automatisée. Le but de l'exploration automatisée est de donner des intervalles de variations larges pour certains paramètres afin de pousser les modèles dans leurs retranchements ou sur des cas non prévus initialement par le modélisateur afin d'obtenir de l'information sur son comportement. On peut aussi l'utiliser de sorte à avoir un premier instantané du modèle pour un nouveau modélisateur ou bien comme un moyen d'avoir de l'information suite à la mise à jour d'un modèle dans une base centralisée. En effet, au delà de l'aspect purement scientifique dans la rédaction des équations d'un modèle, il est nécessaire d'avoir des implémentations informatiques. C'est la différence que l'on peut faire entre la conceptualisation de la structure du modèle (ensemble d'équations) et l'implémentation de la structure du modèle (implémentation informatique). L'utilisateur a besoin d'avoir confiance en cette implémentation. De fait, si l'implémentation d'un modèle est parfois sujette à l'introduction de bugs due à la complexité du domaine, il faut se poser la question de la détection des bugs ou du moins de la conformité à une série de tests garantissant non pas l'absence de bugs mais plutôt le bon déroulement vis à vis d'une série de fonctions dont les sorties sont attendues. La standardisation du langage des entrées/sorties au sein du système nous permet de mettre en place des études automatiques vis à vis d'un certain modèle et d'une certaine problématique.

3.1.2.4 Automatisation pour l'exploration et l'émergence de phénomènes

Nous avons, par exemple, une illustration en figure 3.3 permettant de voir l'impact sur l'indice de premier ordre des lois d'échantillonnage pour un paramètre ("s_init") du modèle LNAS pour le rendement. C'est un point important : la conclusion de l'analyse de sensibilité dépend des distributions sur les paramètres. Ici l'indice de premier ordre pour s_init varie de 0 à 0.25 selon l'intervalle support de la distribution uniforme. Sur la figure 3.3 le point (x,y) représente les bornes inférieures et supérieures de la distribution uniforme associée au paramètre. Ainsi si l'on échantillonne s_init entre (0.5,1.0) l'indice de premier ordre sera plus grand que si l'on échantillonne entre (0.7,0.9).

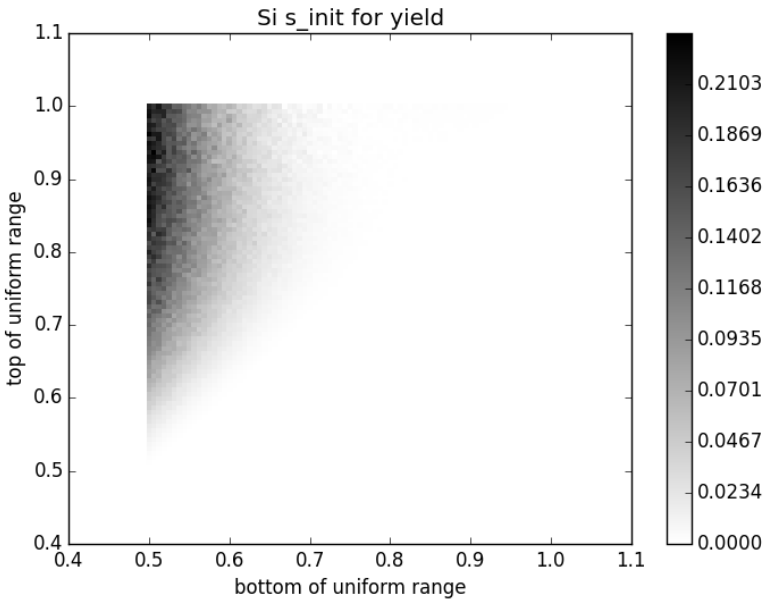


Figure 3.3 – variation de l'indice de premier ordre pour s_init sur LNAS selon l'échantillonnage choisi pour SOBOL lors de l'étude de type 1.

Cela permet aussi de mettre en place des visualisations dynamiques lors de l'exploration des modèles

permettant ainsi de faire varier dans un premier temps une dimension relative à l'algorithme pour s'assurer que les résultats sont cohérents et par la suite de se concentrer sur les dimensions du modèle. Par exemple, on peut faire évoluer le nombre de particules afin de trouver un nombre permettant d'avoir des résultats stables et par la suite faire évoluer les paramètres relatifs à l'échantillonnage des paramètres afin d'explorer de façon plus large les incertitudes qui se développent autour des valeurs produites.

3.1.2.5 Automatisation pour l'ajustement de la consommation

L'exploration automatisée, lorsqu'elle porte sur les paramètres des algorithmes, permet aussi d'explorer plus facilement certaines dimensions relatives aux algorithmes, comme par exemple le nombre de particules nécessaires pour la bonne quantification des indices de sensibilité ou d'incertitudes. Ainsi par consommation, nous entendons la consommation de ressources de calcul qui augmente nécessairement avec la quantité d'informations produites. Cette automatisation nous permet d'enregistrer l'activité et ainsi de suggérer des ajustements.
Toute cette automatisation et visualisation permet de renforcer la confiance auprès du modélisateur et de l'utilisateur quant à la robustesse du modèle.

3.1.3 Analyse de sensibilité

3.1.3.1 Principes généraux et algorithmes

L'analyse de sensibilité paramétrique vise à expliquer l'influence des différents paramètres sur les variations des sorties du modèle. Certaines méthodes permettent également d'analyser les interactions entre paramètres. Plus le degré de non-linéarité du système est important, plus ces interactions sont fortes.
Pour un système dynamique, si l'analyse de sensibilité est réalisée pour des sorties variant avec le temps, alors l'évolution des indices de sensibilité ou du degré de non-linéarité peut également permettre de mettre en évidence des phases dans la vie du système (Wu 2012 ; Wu et al. 2013).
(Iooss 2011) distingue trois grandes classes de méthodes pour l'analyse de sensibilité : le criblage, les mesures d'importance, et les outils d'exploration du modèle. Nous nous concentrerons sur les outils d'exploration du modèle, ceux-ci étant mieux adaptés à nos modèles d'étude avec un grand nombre de paramètres. Dans la figure 3.4, nous avons choisi d'explorer en priorité les méthodes qui sont situées en haut à droite de la grille de sélection proposée dans (Iooss et Lemaître 2015). Les autres méthodes étant peu adaptées à nos types de modèles.

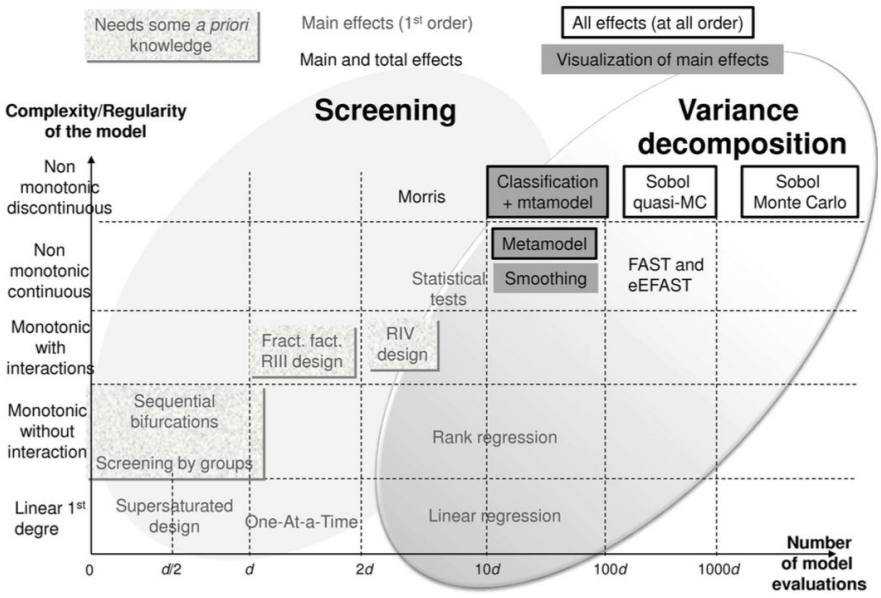


Figure 3.4 – Grille de sélection d'un algorithme en fonction de la complexité du modèle et du nombre d'évaluation du modèle (looss et Lemaître 2015)

Par ailleurs, l'analyse de sensibilité peut être utilisée dans la phase d'estimation paramétrique afin de sélectionner les paramètres à estimer en fixant les paramètres les moins importants à des valeurs de référence (phase de "parameter screening", cf. (Borgonovo 2007))

Pour ce faire, la sortie du modèle par rapport à laquelle est réalisée l'analyse de sensibilité est le critère utilisé pour l'estimation, en particulier le critère GLS. Cette analyse a été implémentée dans la méthode "SOBOL-GLS"

Trois algorithmes ont été implémentés dans la plateforme :

- SRC (Standard Regression Coefficient)
- SOBOL
- MORRIS (uniquement dans l'ancienne version de la plateforme, car finalement la méthode donnait des informations moins intéressantes que la méthode de SOBOL)

Selon le contexte nous utiliserons un algorithme ou l'autre afin d'avoir des informations pertinentes sur notre système. Ainsi, si nous souhaitons savoir si le modèle est fortement linéaire ou non, et plus précisément quelles sont les phases de non-linéarité, alors nous utiliserons SRC. Si nous souhaitons connaître l'importance des paramètres en vue d'une estimation, alors nous pouvons utiliser SOBOL-GLS. Enfin si nous voulons avoir une idée des interactions entre les paramètres, alors nous pouvons utiliser SOBOL pour les variables clés de la croissance, au cours du temps.

De plus pour SOBOL nous pouvons aussi faire une analyse module par module en regroupant les paramètres afin d'étudier leurs interactions conjointes (Wu 2012 ; Wu et Cournède 2014).

3.1.3.2 Implémentation

En ce qui concerne l'implémentation vis à vis du cube de simulations montré en section 2.3, nous sommes sur un parcours d'une colonne de 0 à t_{max} avec N jeux de paramètres où N est le nombre d'échantillons donné par l'utilisateur. En réalité, ce nombre sera peut-être plus grand selon les méthodes. Par exemple SOBOL nécessite 4 matrices pour calculer les variances. De fait il y a au final $4 * N$ jeux de paramètres pour la méthode de SOBOL. La méthode d'estimation implémentée est celle de (Wu 2012).

Au niveau de l'implémentation des programmes associés, nous avons retenu comme arguments :

model	bibliothéque du modèle
initial-state	fichier d'état initial
control	fichier d'environnement initial
parameters	fichier de paramètres
sampling-rule-list	régles d'échantillonnage pour les paramètres
group-list	indication concernant les regroupements de paramètres du modèle pour l'étude module par module (concerne uniquement SOBOL)
nbr-samples	taille de l'échantillon
observation-function	fonction d'observation du système pour les variables de sorties du modèle sur lesquelles portera l'analyse

Figure 3.5 – paramètres de l'analyse de sensibilité

Le code 3.1 est un exemple de commande pour connaître les phases linéaires ou non d'un modèle de Lotka-Volterra (programmé dans la librairie liblv.so)

```
#standard regression coefficient
./bin/src --model ./model_list/liblv.so
          --initial-state x0.dat
          --control u.dat
          --observation-function '[("x",[50,60,70,80,90,100]),("y", [15,25,35,45,55,65])]'
          --initial-parameters p0.dat
          --univariate-rule-list '[("a","normal",[0,0.05]),("b","normal",[0,0.3])]'
          --nbr-samples 10000
          --output-directory ./
```

Code 3.1 – exemple pour src

Le code 3.2 est un exemple pour obtenir les paramètres importants en vue d'une calibration.

```
#sobol for finding parameters to estimate
./bin/sobol-gls --model ./model_list/liblv.so
                --initial-state x0.dat
                --control u.dat
                --initial-parameters p0.dat
                --univariate-rule-list '[("a","normal",[0,0.05]),
                                         ("b","normal",[0.2,0.02]),
                                         ("c","uniform",[0.1,0.2]),
                                         ("d","uniform",[0.4,0.5])]'
                --target y_exp.csv
                --nbr-samples 10000
                --output-directory ./
```

Code 3.2 – exemple de commande pour l'analyse de sensibilité avec sobol_gls

Les sorties seront étudiées dans la section 3.2.

En utilisant sobol-gls nous obtenons donc le classement des paramètres les plus importants en vue d'une estimation.

3.1.3.3 Bilan

Dans le cadre de nos modèles, et aussi pour les simulateurs externes, nous avons développé l'accès à trois algorithmes d'analyse de sensibilité. Le premier SRC est basé sur l'approximation linéaire de la sortie, le deuxième SOBOL nous permet d'avoir une visualisation des indices de premier ordre et d'ordre total, enfin, le troisième SOBOL-GLS, nous permet de classer les paramètres selon leur influence en vue d'une estimation paramètrique. Nous avons pour l'instant laissé de côté la réécriture de la méthode MORRIS étant donné que les modélisateurs lui préférent la méthode de SOBOL, qui donne des informations plus pertinentes.

Dans la section suivante nous décrivons les algorithmes possibles pour une estimation paramétrique.

3.1.4 Estimation paramétrique

3.1.4.1 Principes généraux et algorithmes

Une fois la structure du modèle établie, l'estimation paramétrique consiste en l'évaluation des paramètres inconnus du modèle, à partir de donnnées expérimentales.

Concrètement, un cadre statistique est construit et les paramètres estimés sont les plus vraisemblabes étant données les observations expérimentales et éventuellement notre connaissance a priori sur ces paramètres.

Les données expérimentales peuvent provenir de plusieurs lieux différents et représenter des expériences différentes.

3.1.4.2 Fréquentiste ou Bayésien ?

Dans la plupart des cas, les algorithmes de type moindres carrés généralisés donnent de bons résultats par rapport à notre problème. Nous pouvons donc le choisir comme cadre par défaut. Toutefois, cette méthode n'est statistiquement satisfaisante que dans le cas où il n'y a pas de bruits de modélisation. Nous pouvons utiliser trois algorithmes d'optimisation pour minimiser le critère GLS (Wiberg 1983). Ces trois algorithmes sont : Gauss-Newton (Walter et Pronzato 1994) qui est un algorithme de type Quasi-Newton et converge vers le plus proche minimum local, optimisation par essaim de particules (Shi et Eberhart 1998), recuit simulé (Laarhoven et Aarts 1987) qui sont des algorithmes heuristiques censés pouvoir converger vers le minimum global.

Si nous souhaitons prendre en compte notre connaissance a priori sur les paramètres, nous pouvons utiliser l'estimation bayesienne avec les techniques de "Markov Chain Monte-Carlo".

Enfin, dans le cas où nous souhaitons recalibrer le modèle à la volée en fonction de nouvelles données qui arriveraient par un flux d'information, nous souhaitons utiliser les techniques d'estimation bayesienne dites "Sequential Monte-Carlo" , comme les filtres de Kalman sans parfum (Julier, Uhlmann et Durrant-Whyte 2000), Ensemble Kalman Filter (Kalman 1960) et filtres particulaires régularisés (Campillo et Rossi 2009).

Dans tous les cas, il est à noter que l'estimation s'accompagne d'une évaluation de l'incertitude d'estimation.

3.1.4.3 Assimilation de données

Un des intêrets des méthodes de types SMC est le caractère "online" qui nous permet de mettre en place des techniques d'assimilation de données permettant de recalibrer un modèle en prenant en compte de nouvelles données. Nous présentons rapidement ci-dessous des résultats obtenus par la première version de la plateforme et se trouvant dans (Chen et Cournède 2014).

3.1.4.4 Implémentation

Comme nous l'avons évoqué dans les sous-sections précédentes nous avons introduit différents types d'algorithmes que l'on peut répartir en algorithmes fréquentistes et algorithmes bayésiens. Plutôt que de vanter les mérites de l'un sur l'autre il est à mon sens plus important de comprendre la complémentarité de l'un avec l'autre.

Algorithmes fréquentistes Nous avons principalement travaillé autour de la méthode des moindre carrés généralisés correspondant au maximum de vraisemblance pour les cas où la fonction de transition est déterministe.
Des méthodes de type EM (Expectation-Maximization) ont également été testées dans le cas où la fonction de transition est stochastique (Trevezas et Cournède 2013) mais n'ont pas encore été rendues génériques et applicables pour tous les modèles. Au niveau de l'implémentation des programmes associés, nous avons retenu comme arguments :

model	bibliothéque du modèle
state	fichier d'état initial
control	fichier d'environnement
initial-parameters	fichier de paramètres
target	fonction d'observation du système
optimization-function	fonction d'optimisation
optimization-function-cfg	configuration de la fonction d'optimisation
covariance-matrix	matrice de covariance sur les erreurs d'observation

Figure 3.6 – arguments des moindres carrés généralisés

Nous donnons dans le code 3.3 un exemple de commande :

```
#aitken
./bin/aitken --model ./model_list/liblv.so
             --initial-state x0.dat
             --control u.dat
             --initial-parameters p0.dat
             --selected-parameters ["a","b"]
             --target y_exp.csv
             #we have several optimization function
             --optimization-function "gauss-newton"
             #create an error covariance matrix based on the variances of experimental data
             --covariance-matrix "variance-per-observation-model"
             --output-directory ./
```

Code 3.3 – exemple de commande pour les moindres carrés généralisés

Méthode de Monte-Carlo par chaîne de Markov Nous allons simuler des échantillons de paramètres et de variables d'états en générant une chaîne de Markov dont la distribution stationnaire est la distribution a posteriori des paramètres et des états étant données les observations expérimentales. La version implémentée est décrite dans (Chen 2014 ; Viaud et al. 2015)
Au niveau de l'implémentation des programmes associés, les paramètres retenus sont donnés dans la figure 3.7 :

model	bibliothéque du modèle
initial-state	fichier d'état initial
control	fichier d'environnement
parameters	fichier de paramètres
sampling-rule-list	régles d'échantillonnage pour les paramètres correspondant aux distributions a priori sur les paramètres
nbr-samples	taille de la chaine
observation-function	fonction d'observation du système

Figure 3.7 – paramètres de MCMC

```
#markov chain monte carlo
./bin/mcmc --model ./model_list/liblv.so
           --initial-state x0.dat
           --control u.dat
           --target y_exp.csv
           --observation-function '[("x_noised",[50,80,90,100]),("y_noised", [15,25,55,65])]'
           --univariate-rule-list '[("a","normal",[0,0.05]),("b","normal",[0,0.03])]'
           --nbr-iterations-parameters 10000
           --size-burnin-parameters 1000
           --output-directory ./
```

Code 3.4 – exemple de commande pour Markov-Chain Monte-Carlo

Filtres particulaires La principale méthode sur laquelle nous avons travaillé est le filtre particulaire régularisé ou convolé (Chen 2014 ; Chen et al. 2012).

Pour l'appel de ces méthodes, nous avons besoin des arguments suivants :

model	bibliothéque du modèle
initial-state	fichier d'état initial
control	fichier d'environnement initial
sampling-rule-list	ditributions a priori
nbr-samples	nombre de particules
observation-function	fonction d'observation du système

Figure 3.8 – arguments des programmes pour le filtre particulaire convolé

```
#convolution particle filter
./bin/cpf --model ./model_list/liblnas.so
          --control u.dat
          --experimental-data y_exp.csv
          --initial-parameters p.dat
          --initial-state x0.dat
          --output-directory ./
          --observations-label-list ["noised_dry_green_leaf_mass","noised_yield"]
          --parameters-sampling-rule-list [("rue","normal",[3.6, 0.5]), \
                                           ("s_init","normal",[0.6, 0.2]), \
                                           ("mu_alloc","normal",[550.0, 80.0])]
          --observation-function
  ↪  [("dry_green_leaf_mass",[100,105,110,115,120,125,130,135,140,145,150]),\
                                ("yield",[100,105,110,115,120,125,130,135,140,145,150])]
          --number-of-particles 50000
```

Code 3.5 – exemple de commande pour le filtre particulaire convolé

3.1.4.5 Bilan

Afin d'estimer les paramètres de nos modèles, nous avons développé dans notre cadre de simulation et du langage associé différents algorithmes exploitant à la fois le cadre fréquentiste et le cadre bayésien. À ce jour, le cadre du filtre particulaire diffère des autres par sa nature exploitant un algorithme de prédiction-correction. De fait, cet algorithme n'est pas accessible aux simulateurs externes pour l'instant, étant donné qu'il utilise la fonction de transition uniquement et non pas la totalité de la simulation. Nous avons, dans le cadre fréquentiste, un accès à une méthode basée sur l'estimateur d'AITKEN et nous pouvons utiliser soit l'algorithme de GAUSS-NEWTON pour la phase d'optimisation sinon des heuristiques de type Particle Swarm Optimisation ou Simulated Annealing dans le cadre de problèmes non-convexes.

3.1.5 Analyse d'incertitudes

3.1.5.1 Principes généraux et algorithmes

L'analyse d'incertitudes permet d'étudier les incertitudes sur les sorties en fonction des incertitudes sur les variables d'entrées. Pour rappel les principales sources d'incertitudes sont : le "manque de connaissance", les "erreurs de mesures ou d'échantillonnage", la "variabilité des caractéristiques du système", les "erreurs de modélisation du phénomène". (Faivre et al. 2013) À ce jour deux implémentations ont été faites dans le système : l'analyse par Monte-Carlo et l'analyse par la méthode dite "unscented transform". (Julier, Uhlmann et Durrant-Whyte 2000). Ce genre d'étude permet d'étudier la capacité prédictive du modèle. En effet si une simple variation de quelques pourcents sur les paramètres modifie grandement la sortie, alors le modèle n'est pas très stable et sa capacité de prédiction n'est pas bonne. En particulier, si nous avons une distribution sur les paramètres estimés (par une méthode bayésienne par exemple), alors nous pouvons étudier par analyse d'incertitudes la précision de la prédiction par le modèle ainsi calibré.

L'analyse d'incertitudes par Monte-Carlo consiste à lancer sur un grand nombre de jeux de paramètres des simulations et de quantifier les moyennes et variances des sorties en fonctions des paramètres d'entrées. L'analyse par Unscented Transform (Uhlmann 1995) consiste à prendre un type d'échantillonneur particulier : le sigma point sampler.

3.1.5.2 Implémentation

Nous retrouvons ci-dessous les arguments nous permettant de lancer les programmes d'incertitudes :

model	chemin vers la bibliothéque du modèle
nbr-samples	nombre d'échantillons à utiliser pour les paramètres
observation-function	fonction d'observation du système
initial-parameters	fichier de paramètres initiaux i.e. avant sampling
sampling-rule-list	règles d'échantillonnages à appliquer sur les paramètres
initial-state	fichier d'état initial
control	fichier d'environnement
output-filename	fichier de sortie

Figure 3.9 – arguments des programmes d'analyse d'incertitudes

Voici quelques commandes qui illustrent le fonctionnement de ce programme.

```
#uncertainty analysis with monte-carlo
./bin/uamc --model ./model_list/liblv.so
           --observation-function '[("x",[50,60,70,80,90,100]), \
                                   ("y", [15,25,35,45,55,65])]'
           --univariate-rule-list '[("a","normal",[0,0.05]), \
                                   ("b","normal",[0.0,0.03])]'
           --nbr-samples 10000
           --output-directory ./
```

Code 3.6 – exemple de commande pour l'analyse d'incertitudes par Monte-Carlo

3.1.5.3 Bilan

Dans le cadre de l'analyse d'incertitude, nous avons un programme qui permet d'exploiter les configurations d'échantillonnages trouvés sur la base des erreurs d'estimation lors de la phase d'estimation paramétrique. De même que la plupart des autres programmes, il est disponible pour les simulateurs externes également.

3.1.6 Critères de sélection et de prévision

3.1.6.1 Principes généraux et algorithmes

La sélection de modèles est une fonctionnalité qui met en jeu des critères dont le but est de quantifier la qualité d'un modèle calibré. La principale famille de critères combinent un critère d'ajustement aux données expérimentales pénalisé par une fonction croissante du nombre de paramètres (AIC(Akaike Information Criterion), BIC (Bayesian Information Criterion)).

Dans le cadre bayésien, DIC (Deviance Information Criterion) combine aussi une fonction d'ajustement en pénalisant par une fonction l'incertitude sur les paramètres.

Un autre critère important est le MSEP (Mean Square Error Prediction) qui permet d'étudier la capacité prédictive d'un modèle.

Les enjeux de la sélection de modèles peuvent se retrouver dans (Baey et al. 2013b) où d'autres critères sont mis en jeu.

Quelques critères qui ont été implémentés au sein de la plateforme : AIC (Akaike 1973), AICc (Hurvich et

Tsai 1989), BIC (Schwarz 1978), DIC (Spiegelhalter et al. 1998), RMSEP (Root Mean Square Prediction) et MDL (Minimum Description Length).

3.1.6.2 Implémentation

Les indices de sélection étant basés sur le calcul de la vraisemblance nous les calculons à la fin des algorithmes d'estimation.

3.1.6.3 Bilan

Les critères que nous avons implémentés nous permettent pour l'instant de mener à bien les différentes sélections et comparaisons de modèles vis à vis d'un objectif de modélisation.
On retrouvera notamment une comparaison et sélection avancée de différents modèles au sein de (Baey et al. 2013b).
Toutefois, les critères présentés sont relatifs uniquement à la validation et comparaison vis à vis d'un objectif. Nous pourrions étudier plus précisement la validation d'un modèle en particulier en intégrant des méthodes comme l'analyse des résidus (Walter et Pronzato 1994 ; Straume et Johnson 2010).

3.1.7 Bilan

Les bonnes pratiques de modélisation correspondent à un flux de travail composé de plusieurs fonctionnalités qui doivent s'enchainer les unes avec les autres afin de permettre la conception et le développement d'un modèle à la fois validé et vérifié sur un jeu de données d'apprentissage et un jeu de données de validation.
Ces fonctionnalités sont composées de plusieurs algorithmes et nous en avons implémenté quelques uns qui nous permettent de conduire notre modélisation correctement dans le domaine de la croissance des plantes avec des modèles dynamiques discrets stochastiques.
Dans la section suivante, nous montrerons deux études complètes pas à pas sur des modèles de blé et de betterave et une étude courte sur un modèle de pin mais avec un simulateur externe.

3.2 Applications et résultats

Dans cette section, nous allons appliquer des études pas à pas sur différents modèles, à savoir : LNAS, STICS et LIGNUM.
Ce chapitre permet de montrer comment appliquer les algorithmes d'analyse de modèle décrits dans la section 3 au moyen de la plateforme conçue à partir des concepts développés au chapitre 2.
Le but de ce chapitre n'est pas tant d'arriver à estimer et produire une application de prévision parfaite, mais de montrer les études qui sont dorénavant possibles à moindre coût par une standardisation des outils au moyen du vocabulaire décrit en chapitre 2. Nous verrons ainsi comment est mis en pratique le flux de travail présenté au chapitre 3 pour la mise en œuvre d'une démarche de bonne pratique de modélisation tout en discutant sur les ajustements nécessaires en fonction des modèles et des données que nous avons à disposition.
Nous étudierons en particulier deux modèles : LNAS (Cournède et al. 2013) pour la betterave et STICS (Brisson et al. 1998) pour le blé. Pour l'ensemble des études, nous pouvons considérer que les implémentations des modèles sont dans un état "stable" i.e. les développements sont considérés par le modélisateur et l'expert satisfaisants pour la prise en charge des processus.

3.2.1 Descriptions des machines de tests

Les études se sont déroulées sur un macbook pro 15 pouces avec un core i7 à 2.5 Ghz en fixant la valeur OMP_NUM_THREADS de la librairie OpenMP (OpenMP Architecture Review Board 2008) à 4 pour n'avoir que 4 threads [2]. La mémoire vive est de 16 Go à 1600 Mhz.

Nous avons aussi effectué les tests sur le mésocentre de l'école CentraleSupélec avec des travaux exploitant de 1 à 12 cœurs et des consommations mémoire variant de 2Go à 40Go selon certaines études.

3.2.2 Étude pas à pas pour LNAS avec simulation sous PYGMALION

L'étude pas à pas consistera en :

- une analyse de sensibilité générale par SRC ou SOBOL
- une analyse de sensibilité pour l'estimation paramètrique par SOBOL-GLS
- une estimation paramètrique par MCMC ou AITKEN
- une sélection par les critères AIC, AICc, BIC, ou DIC
- une caractérisation des incertitudes par la méthode de Monte-Carlo
- une caractérisation de la capacité prédictive du modèle calibré

Par rapport à ce schéma genéral, il peut y avoir des aller-retours nécessaires entre certaines parties.

3.2.2.1 Description des données pour la betterave sucrière

Nous présentons les données qui serviront de données d'apprentissage et celles qui serviront de données de tests afin de faire la validation de l'estimation dans les études pas à pas ci-dessous.

Nous utilisons également les données climatiques journalières en température, rayonnement, et évapostranspiration potentielle et précipitations issues de stations météorologiques proches des parcelles.

Les données ont été fournies par l'institut technique de la betterave (ITB) pour l'année 2008 et l'année 2010, voir pour plus de détails sur l'expérimentation (Lemaire et al. 2008) et (Baey et al. 2013a). Elles contiennent la biomasse sèche des feuilles ainsi que la masse sèche des racines (rendement).

jour	biomasse sèche des feuilles (g/m^2)	rendement (masse sèche des racines) (g/m^2)
23	5.4	0.6
45	147.4	81.3
52	218.6	169.5
60	229.3	301.2
73	314.7	588.5
107	373.5	1559.1
143	380.6	2327.7

Table 3.1 – données betterave sucrière pour l'année 2008

Année 2008

2. L'hyperthreading n'a pas été désactivé physiquement

jour	biomasse sèche des feuilles (g/m^2)	rendement (masse sèche des racines) (g/m^2)
54	85.2533	23.1276
68	372.924	199.796
76	447.603	302.348
83	440.814	409.197
90	620.444	709.23
98	523.781	768.099
104	541.356	863.854
110	620.218	1232.48
118	627.507	1498.83
125	757.617	1770.17
132	760.463	1878.17
139	598.344	1913.74
145	670.674	2118.39
160	628.394	2274.72

Table 3.2 – données betterave sucrière pour l'année 2010

Année 2010

3.2.2.2 Description du modèle LNAS

Le modèle LNAS (Cournède et al. 2013), où "Log-Normal Allocation Senescence", est un modèle simplifiant le modèle GreenLab (De Reffye et Hu 2003) et qui définit l'allocation comme étant globale et non pas feuille par feuille comme c'est le cas avec un modèle structure-fonction dont l'architecture est connue.

Les équations principales sont :

Processus	Équations
production de biomasse Q par mètre carrés	$Q_{det}(t) = rue * PAR(t) * (1 - exp(-k_B.\frac{Q_g(t)}{e}))$
allocation aux feuilles	$Q_l(t+1) = Q_l(t) + \gamma(t).Q(t)$
allocation aux racines	$Q_r(t+1) = Q_r(t) + (1-\gamma(t)).Q(t)$
proportion de feuilles non-senescente	$Q_g(t) = (1 - G_s(\tau(t))).Q_l(t)$
fonction γ	$\gamma_{det}(t) = \gamma_0 + (\gamma_f - \gamma_0).G_a(\tau(t))$
fonction cumulative G_a	log-normal pour l'allocation
fonction cumulative G_s	log-normal pour la sénescence
fonction τ	temps thermique

Table 3.3 – liste des équations de LNAS

L'ensemble des éléments présents dans les équations peuvent être regroupés et traduits dans une table d'équivalence entre la version mathématique et l'implémentation informatique.

nom mathématique	nom informatique	type	description
Q_{det}	determinist_produced_biomass	variable d'état	biomasse non-bruitée
Q_{sto}	stochastic_produced_biomass	variable d'état	biomasse bruitée
Q_l	dry_total_leaf_mass	variable d'état	biomasse totales des feuilles
Q_r	yield	variable d'état	rendement (biomasse de la racine pour la betterave)
Q_g	dry_green_leaf_mass	variable d'état	biomasse des feuilles vertes
PAR	PAR	variable de contrôle	rayonnement
rue	rue	paramètre	efficacité du rayonnement
k_B	k_b	paramètre	coefficient de la loi de Beer-Lambert
e	e	paramètre	masse surfacique foliaire
γ_0	s_init	paramètre	proportion initiale de biomasse allouée aux feuilles
γ_f	s_end	paramètre	proportion finale de biomasse allouée aux feuilles
μ_a	mu_alloc	paramètre	paramètre de la loi log-normale G_a
s_a	s_alloc	paramètre	paramètre de la loi log-normale G_a
μ_{sen}	mu_sen	paramètre	paramètre de la loi log-normale G_s
s_{sen}	s_sen	paramètre	paramètre de la loi log-normale G_s

Table 3.4 – table de traduction entre paramètres mathématiques et paramètres informatiques

Il est à noter que c'est un modèle qui existe en version stochastique. Nous pouvons donc insérer à la fois des bruits de modélisation et des bruits d'observation. Ces bruits sont répartis sur la production de biomasse de sorte que $Q_{sto}(t) = Q_{det}(t) * (1 + \eta_Q(t))$ ainsi que sur la fonction d'allocatio γ tel que $\gamma_{sto}(t) = \gamma_{det}(t) * (1 + \eta_\gamma(t))$ où les lois η sont des lois normales centrées en zéro.

3.2.2.3 Analyse de sensibilité générale

Nous conduirons une analyse au moyen des méthodes SRC et SOBOL afin de voir les différentes phases du modèle ainsi que les interactions entre paramètres.

Nous observerons la linéarité du modèle par rapport aux variables de rendement et de biomasse sèche des feuilles du 1^{er} cycle au 159^{me} avec les paramètres du modèle suivant les lois uniformes suivantes pour 10000 échantillons :

paramètre	intervalle de la loi uniforme associée
rue	$[3, 4]$
e	$[50, 70]$

s_init	$[0.5, 0.8]$
s_end	$[0.0, 0.3]$
mu_alloc	$[400, 800]$
s_alloc	$[200, 2000]$
mu_sen	$[2000, 3000]$
s_sen	$[500, 6000]$

Table 3.5 – échantillonnage des paramètres de LNAS

Les bornes des lois sont choisies soit d'après la littérature sinon d'après l'expertise du modélisateur, comme des intervalles raisonnables pour la variation des paramètres.
Nous traduirons cette configuration par les lignes suivantes [3] afin de lancer SRC :

```
./bin/src --model ./model_list/liblnas.so
          --control "./u.txt"
          --initial-parameters "./p.txt"
          --sampling-rule-list '[("rue","uniform",[3,4]),
                                 ("e","uniform",[50,70]),
                                 ("s_init","uniform",[0.7,1.0]),
                                 ("s_end","uniform",[0.0,0.3]),
                                 ("mu_alloc","uniform",[400,800]),
                                 ("s_alloc","uniform",[200,2000]),
                                 ("mu_sen","uniform",[2000,3000]),
                                 ("s_sen","uniform",[500,6000])]'
--observation-function '[("yield",[1...159]),("dry_green_leaf_mass",[1...159])]'
--nbr-samples 10000
--output-directory "./"
```

Code 3.7 – lancement de SRC

Pour SRC les résultats sont donnés dans les figures 3.10 et 3.11 :

3. Nous donnons cet exemple afin d'illustrer un lancement complet d'un algorithme sur le système. Par la suite nous ne donnerons plus les commandes dans leurs totalités.

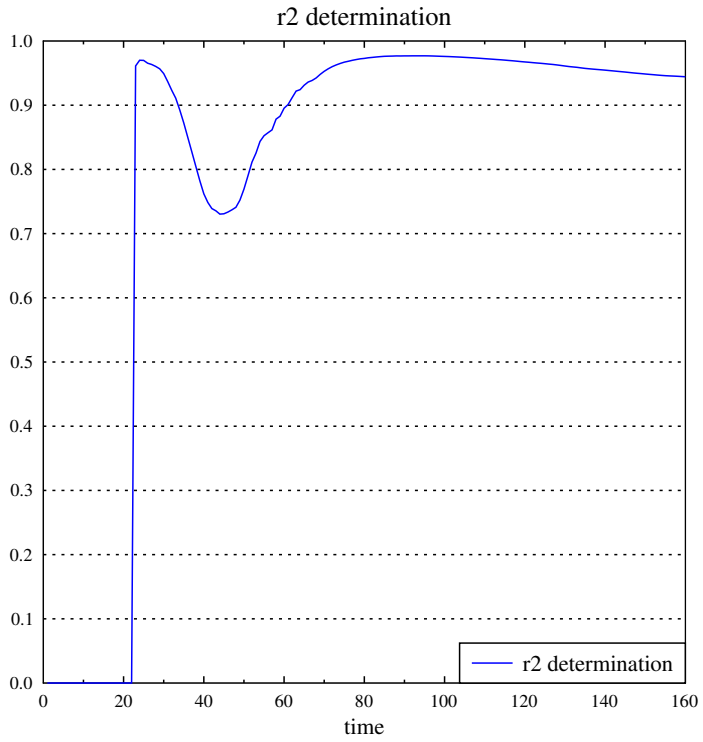


Figure 3.10 – coefficient de détermination pour la variable ”dry_green_leaf_mass”

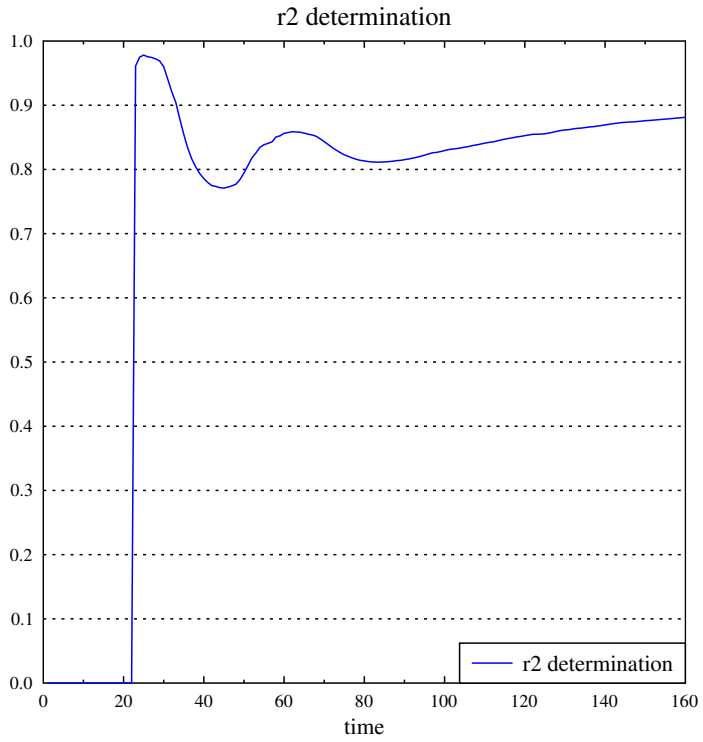


Figure 3.11 – coefficient de détermination pour la variable ”yield”

Nous pouvons constater que le modèle a un comportement globalement linéaire même s'il possède une phase non-linéaire au moment du changement de prédominance dans la stratégie d'allocation (initialement, les feuilles sont privilégiées, puis cela devient la racine) que ce soit pour la biomasse des feuilles vertes et la biomasse de la racine. Il y aussi une autre phase non-linéaire qui apparaît lorsque l'on regarde uniquement la biomasse de la racine au début de la sénescence. Nous pouvons corroborer ce

comportement lorsque l'on regarde les indices SRC un à un.

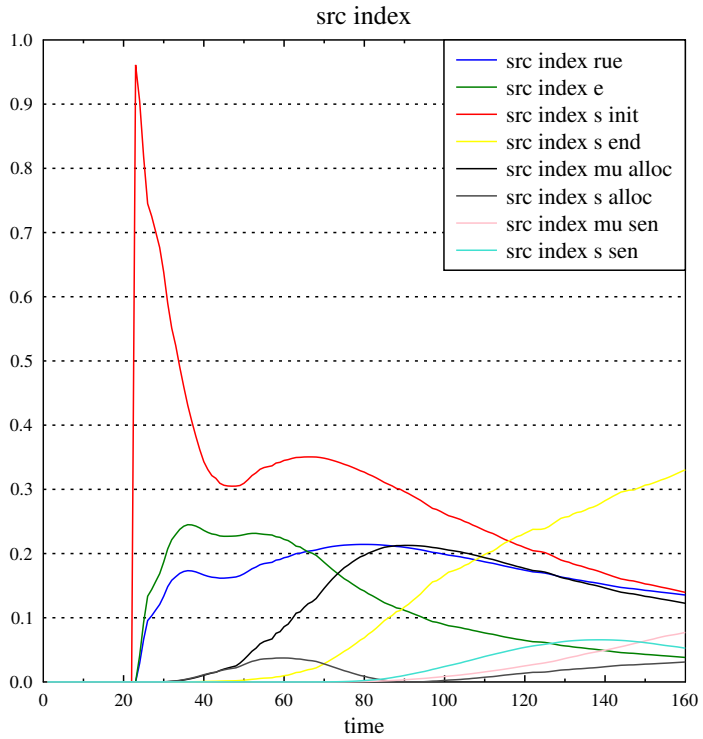


Figure 3.12 – indices src pour la variable "dry_green_leaf_mass"

Nous voyons nettement sur la figure 3.12 l'importance de "s_init" (proportion initiale de biomasse allouée aux feuilles) à l'émergence. Par la suite nous avons les contributions de chaque paramètre qui tendent à s'équilibrer et finissent par s'écraser devant "s_end" qui est la proportion finale de biomasse allouée aux feuilles.

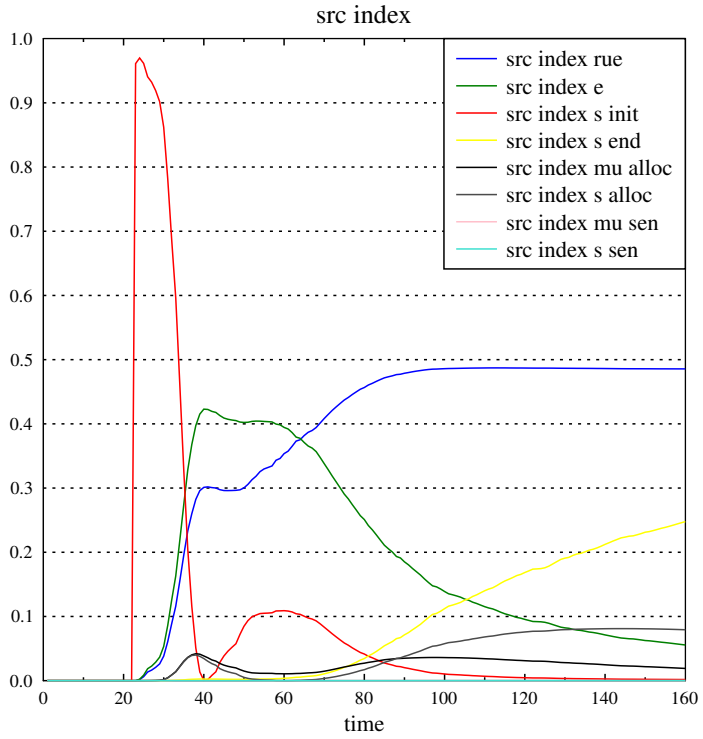


Figure 3.13 – indices src pour la variable "yield"

De même lorsque l'on regarde le rendement nous avons une forte contribution de s_init à l'émergence et une augmentation de la linéarité par rapport à rue et s_end au cours du temps.

Toutefois SRC ne reste qu'une première approche tant l'hypothèse de linéarité par rapport aux paramètres du modèle est une hypothèse forte. Notamment nous ne pouvons pas conclure sur les interactions entre les paramètres.

Ainsi nous allons conduire une analyse de SOBOL afin de quantifier la différence entre les indices de sensibilités totaux et ceux de premier ordre en utilisant la même configuration 3.2.2.3 que pour SRC.

Les résultats de cette analyse sont donnés dans les figures 3.14 et 3.15 par la somme des indices de sensibilité de premier ordre.

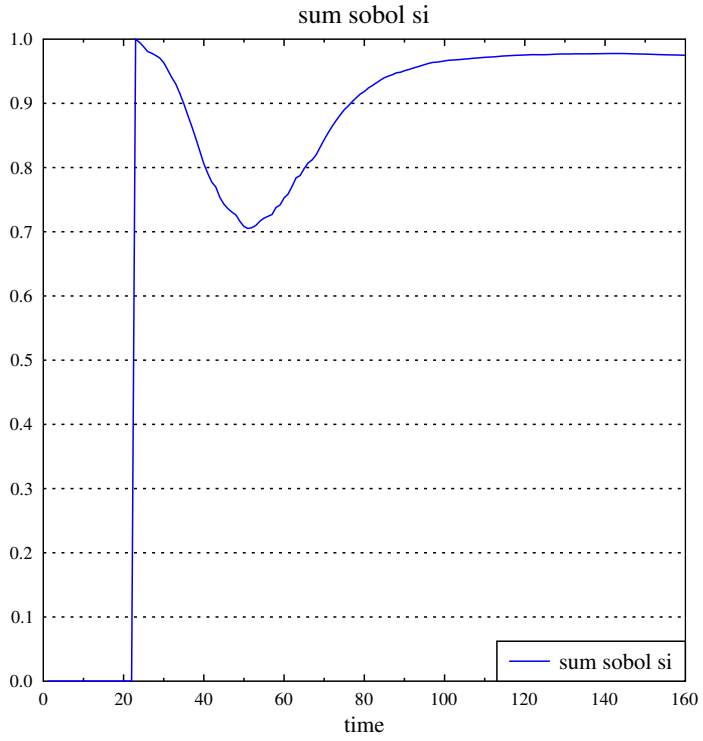


Figure 3.14 – accumulation des paramètres de premier ordre par rapport à "dry_green_leaf_mass"

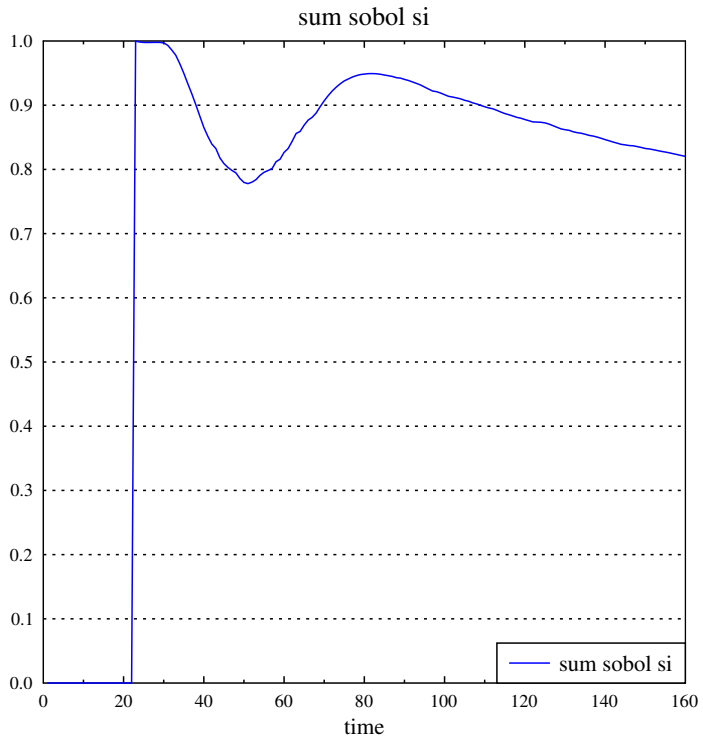


Figure 3.15 – accumulation des paramètres de premier ordre par rapport à ”yield”

Lorsque la somme tend vers 1, alors les paramètres ont peu d'interactions entre eux, à l'inverse une somme très inférieure à 1 tend à montrer qu'il y a de fortes interactions entre les différents paramètres.

Nous pouvons voir que les interactions entre les paramètres se font au moment du changement de stratégie d'allocation (autour du jour 50). Ces interactions entre paramètres se font plus faibles à mesure que l'on avance dans le temps. La non-linéarité reste toutefois non négligeable pour la variable rendement en fin de cycle, ce qui n'apparaissait pas avec la méthode SRC.

Cette différence de comportement peut se constater notamment lorsque l'on regarde les graphiques 3.16 et 3.17 qui montrent la différence entre l'indice d'ordre total et l'indice d'ordre premier pour les indices de SOBOL, qui représente le degré d'interaction associé au paramètre.

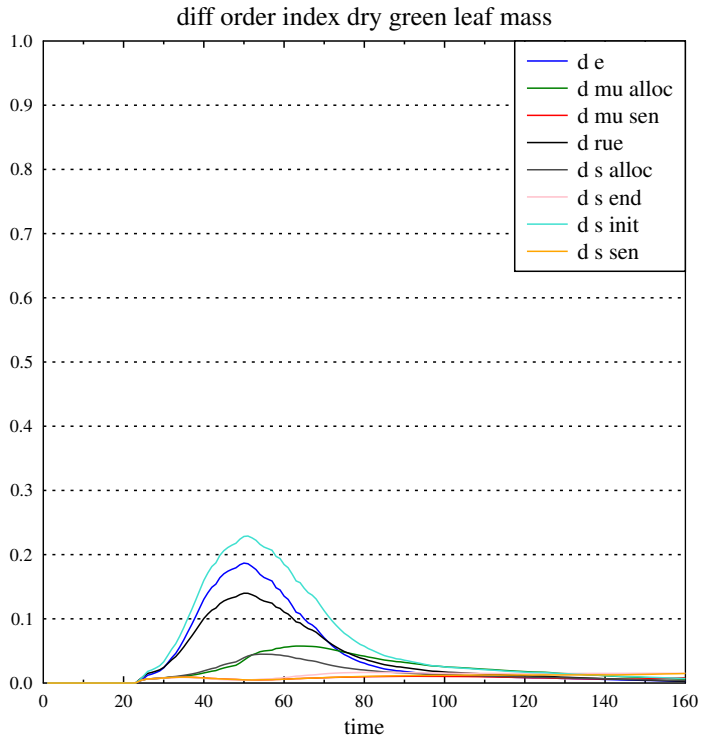


Figure 3.16 – différences entre ordre total et premier ordre ”dry_green_leaf_mass”

Pour rappel l'indice de premier ordre quantifie la part de variance due au paramètre p seul, et l'indice d'ordre total quantifie la part de variance due au paramètre p dans tous les termes où p apparaît. Par conséquent la différence entre l'indice d'ordre total et l'indice d'ordre premier (noté ici ”diff order”) quantifie toutes les non-linéarités dans lesquelles intervient p.

On constate que les interactions d'ordre supérieur se manifestent principalement autour du changement de stratégie d'allocation pour les paramètres rue, e, et s_init lorsque l'on observe la biomasse des feuilles vertes.

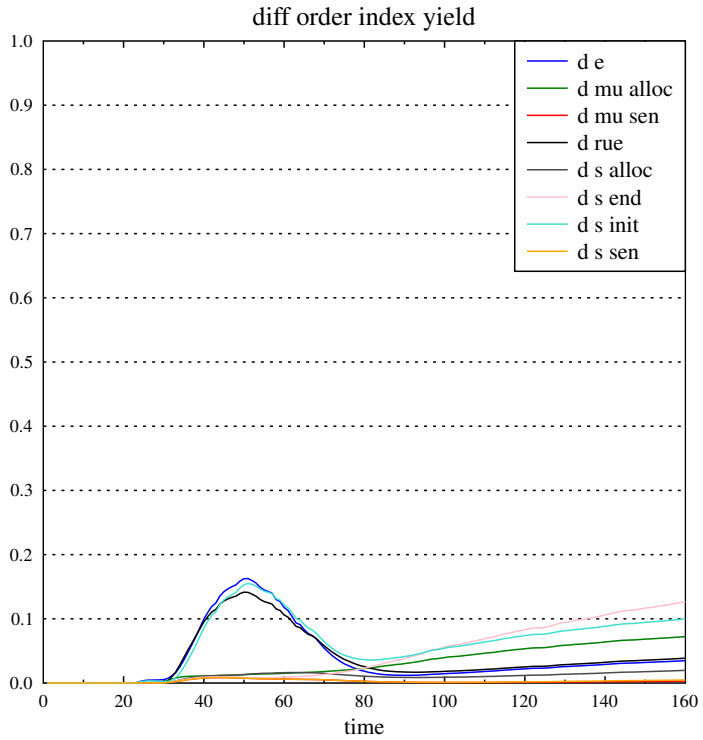


Figure 3.17 – différences entre ordre total et premier ordre pour la variable ”yield”

En ce qui concerne le rendement nous avons le même type d'interactions que pour la biomasse de feuilles vertes mais on remarque que les interactions entre paramètres ne cessent pas et augmentent vers la fin ce qui explique la décroissance observée sur la figure 3.15.

3.2.2.4 Analyse de sensibilité pour l'estimation paramètrique

Nous avons une meilleure idée du comportement du modèle vis à vis de ses paramètres, ainsi que de leurs interactions. Nous pouvons maintenant procéder à une sélection des paramètres les plus influents en vue d'une estimation paramètrique par rapport à des données expérimentales. Pour cela nous allons utiliser l'algorithme dit de sobol-gls qui permet d'étudier non plus le modèle en lui-même mais le critère de minimisation utilisé dans les algorithmes d'estimation. Ce critère correspond au critère de type moindres-carrés généralisés utilisé dans l'estimateur d'Aitken. Estimer les paramètres θ à partir d'un jeu de données expérimentales Y_{exp} revient à rechercher $\hat{\theta}$ tel que :

$$\hat{\theta} = arg\,min(Y_{exp} - Y_\theta)'\Sigma^{-1}(Y_{exp} - Y_\theta)$$

avec Y_θ la sortie du modèle pour le paramètre θ et Σ la matrice de covariance des erreurs d'observation. Nous ferons cette analyse avec les données présentées en table 3.2 et en échantillonnant 50 000 jeux de paramètres de la même manière que sur le code 3.7. En ce qui concerne la fonction d'observation, elle sera reconstruite à partir des données présentes en table 3.2 afin que le système étudié puisse faire la différence entre les données réelles et les données simulées.

Les résultats de l'analyse de sensibilité en vue de l'estimation paramètrique sont :

paramètre	S_i	S_{ti}
s_init	0.30513	0.364894
s_end	0.214936	0.285881
rue	0.14768	0.185065
e	0.110485	0.154177
mu_alloc	0.104416	0.137408
mu_sen	0.00718271	0.019596
s_alloc	0.00381358	0.0171588
s_sen	0.00102846	0.00907435

Table 3.6 – données sobol-gls sur lnas 2010

Les résultats de cette analyse sont représentés dans la figure suivante :

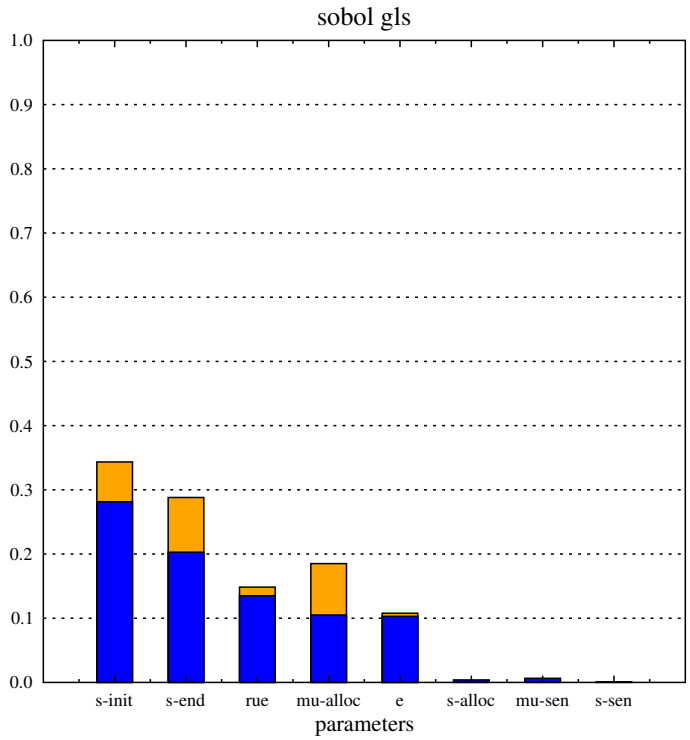


Figure 3.18 – visualisation sobol-gls lnas 2010 (en bleu = premier ordre, en orange = ordre total - ordre premier

À partir du tableau 3.6 nous voyons que les 5 paramètres les plus influents sont : s_init, s_end, rue, e, mu_alloc afin de faire une estimation des paramètres.
Toutefois d'après (Chen et Cournède 2014) nous restreignons notre estimation à "s_init", "rue" et "mu_alloc" car il a été montré que le DIC était meilleur avec ces paramètres là. Cette démonstration a été faite avec l'ancienne version de la plateforme et les calculs nécessaires à l'évaluation du DIC n'ont pas été encore injectés dans la nouvelle plateforme car ils ont besoin d'être généralisés.

3.2.2.5 Validation de l'estimation sur données simulées

Dans cette partie nous nous assurerons de la validité de l'algorithme MCMC qui sera utilisé pour l'estimation.

Nous simulerons des données à partir d'un point p_0 puis nous donnerons des distributions *a priori* pour caractériser les intervalles de variation sur les paramètres à estimer afin de voir si l'on converge vers une distribution dont la moyenne se situe autour de p_0 en sortie de l'algorithme, la moyenne finale dépend du prior donc ne noit pas être exactement p_0.

En sortie de l'estimation nous récupérerons la moyenne et l'écart-type pour chaque distribution univariée de paramètre obtenue à l'aide de la chaîne de Markov.

paramètre	p_0	distribution a priori
rue	3.56e+0	$N(3.6, 0.1)$
s_init	6.12e-1	$N(0.75, 0.08)$
mu_alloc	4.70e+2	$N(600, 20.0)$

Table 3.7 – point p_0 et distributions a priori

Nous choisirons uniquement trois composantes que sont : "s_init", "rue" et "e".

Nous effectuerons une chaîne de Markov de longueur 500 000 avec une période de rodage de 50 000 afin d'estimer la distribution a posteriori sur les paramètres. Notre algorithme sera donc configuré de la manière suivante :

```
target = y_sim.csv
univariate-rule-list = [("rue","normal",[3.5,0.5]),
                        ("s_init","normal",[0.65,0.08]),
                        ("mu_alloc","normal",[500,40.0])]
nbr-iterations-parameters = 500000
size-burnin-parameters = 50000
```

Code 3.8 – échantillonnage de LNAS pour analyse de sensibilité

Nous obtenons comme point moyen et écart-type :

paramètre	moyenne	écart-type	valeur réelle
rue	3.56	0.06	3.56
s_init	0.59	0.014	0.612
mu_alloc	483	12.79	470

Table 3.8 – point $p_{\hat{\theta}}$

Les simulations à partir du point moyen trouvé sont représentées par les figures 3.19 et 3.20. Les données simulées symbolisés par des points sont des états cachés (pas les observations).

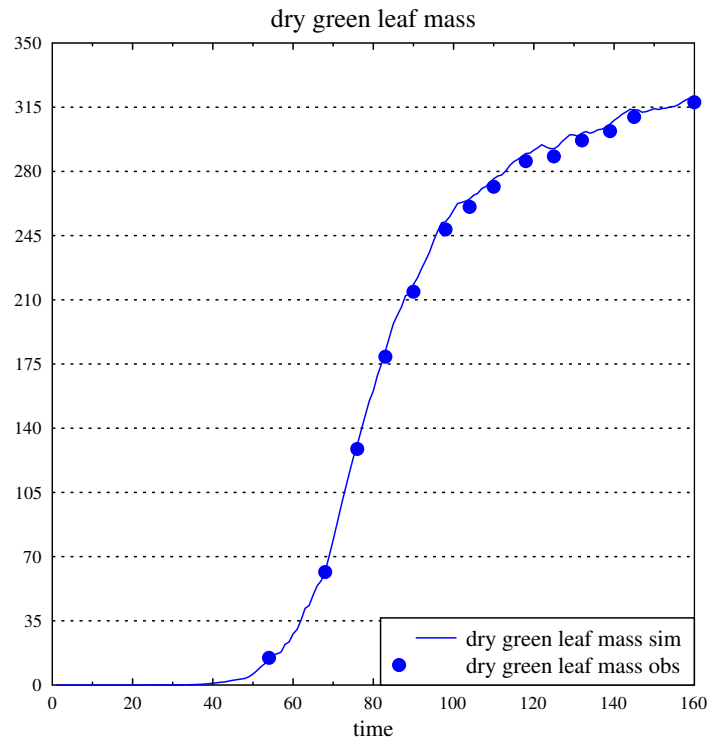


Figure 3.19 – simulation avec les paramètres estimés versus les données simulées pour ”dry_green_leaf_mass”

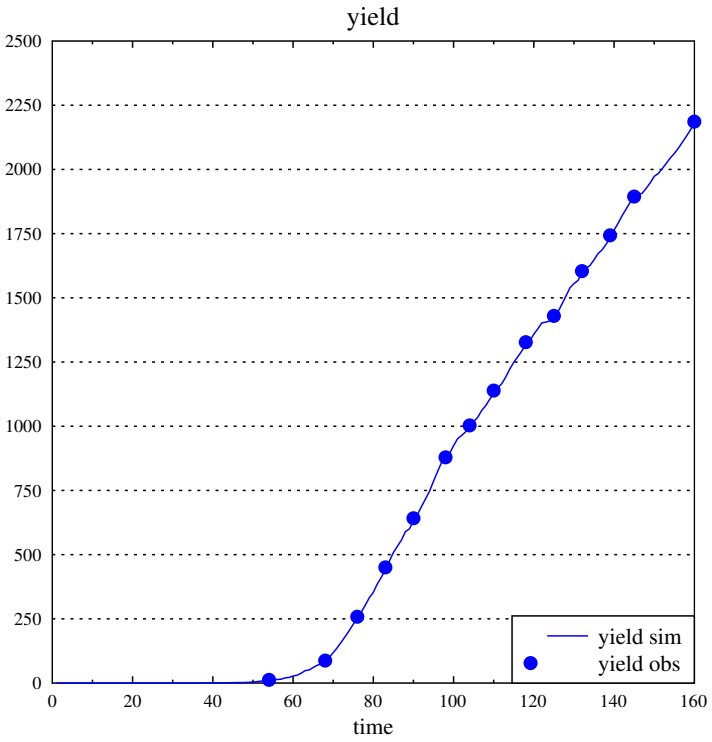


Figure 3.20 – simulation avec les paramètres estimés versus les données simulées pour ”yield”

Discussion Nous avons maintenant une vue générale sur le modèle et avons confirmation que sur au moins quelques données simulées il n’y a pas de problème à l’estimation. Nous allons maintenant faire estimer les valeurs des paramètres du modèle sur données réelles.

3.2.2.6 Estimation paramètrique sur un jeu de données d'apprentissage

Nous procédons à une estimation des paramètres avec la méthode de Monte-Carlo par chaîne de Markov. (MCMC), pour les données 2010 (cf. 3.2).

Notre algorithme sera configuré de la façon suivante :

```
target = y_exp.csv
univariate-rule-list = [("rue","uniform",[3.4,0.2]),
  ("s_init","uniform",[0.9,0.1]),
  ("mu_alloc","uniform",[650,40.0])]
nbr-iterations-parameters = 500000
size-burnin-parameters = 50000
```

Code 3.9 – paramètres du programme MCMC pour estimation paramètrique sur données réelles

Ce qui se traduit par un prior avec des lois normales sur les composantes de θ ainsi que 500 000 itérations pour trouver des paramètres satisfaisant en prenant un burnin de 50 000 itérations i.e. nous les laissons de côté pour le calcul de la moyenne et de l'écart-type étant donné que l'algorithme nécessite un certain temps d'initialisation avant de converger.

Nous obtenons comme point :

paramètre	moyenne	écart-type
rue	3.5847	0.057
s_init	0.8508	0.0246
mu_alloc	667	20.12

Table 3.9 – point $p_{\hat{\theta}}$

Nous obtenons les résultats de simulation suivants en comparaison aux données expérimentales :

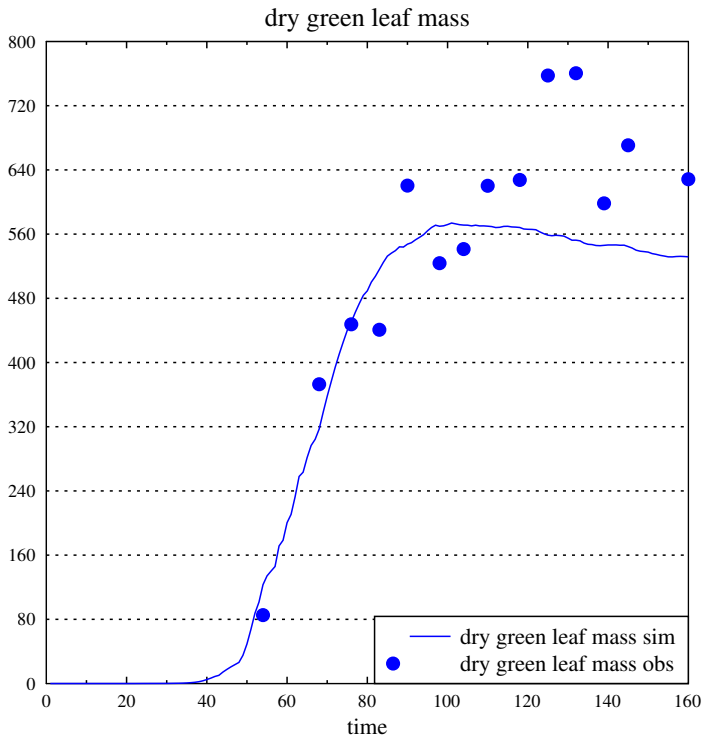


Figure 3.21 – données expérimentales versus données simulées sur le rendement (données réelles)

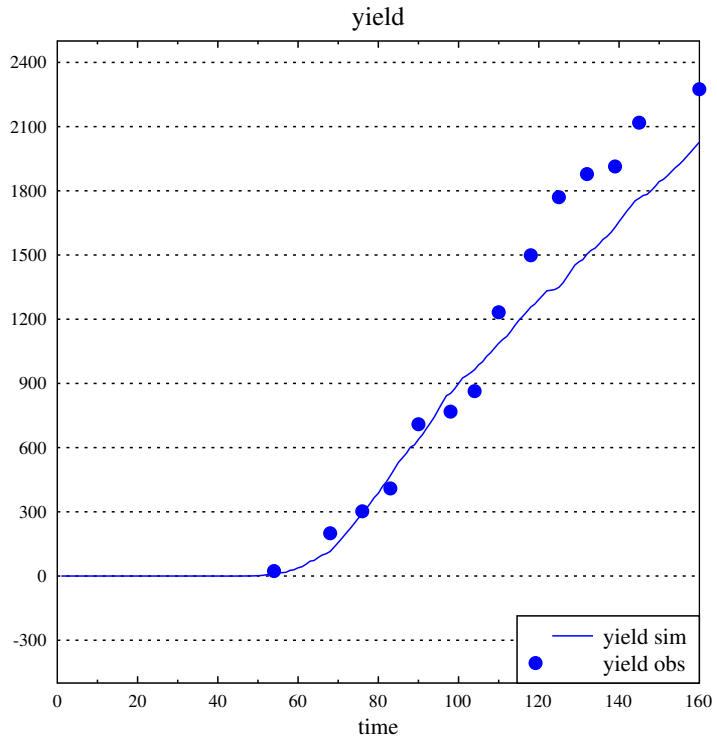


Figure 3.22 – données expérimentales versus données simulées sur la masse sèche (données réelles)

Sur ces courbes les points sont des données réelles tandis que la courbe représente des états cachés autrement dit des données non-bruitées. Il est par conséquent normal que la courbe ne passe pas par les points de données expérimentales.

3.2.2.7 Caractérisation des incertitudes sur les paramètres et la prévision

Nous utiliserons trois types de caractérisation des incertitudes :

- incertitudes sur les paramètres estimés et utilisation du coefficient de variation. En utilisant le rapport entre l'écart type et la valeur du paramètre nous pouvons voir la stabilité du paramètre. Plus ce rapport est petit et plus la valeur trouvé est "stable".
- incertitudes sur la valeur moyenne théorique et utilisation de l'intervalle de crédibilité à 95%. Nous voulons à cette étape nous assurer que le point moyen théorique est un point stable si on échantillonne selon la distribution associée.
- incertitudes sur les observations et utilisation de l'intervalle de prévision à 95%. Nous voulons vérifier quel est l'intervalle des valeurs prises pour les variables observées si l'on recommence l'expérience un grand nombre de fois.

Autrement dit si nous avons des coefficients de variation bas, un intervalle de crédibilité relativement étroit et que l'intervalle de prévision est relativement autour de la moyenne alors nous pouvons avoir confiance dans l'estimation qui a été faite.

Intervalle de prévision à 95% Lorsque nous effectuons un invertalle de confiance à 95% pour la prévision, nous obtenons des valeurs qui peuvent être représentées par les graphiques suivants où les courbes noires représentent les bornes inférieures et supérieures de l'intervalle de prévision :

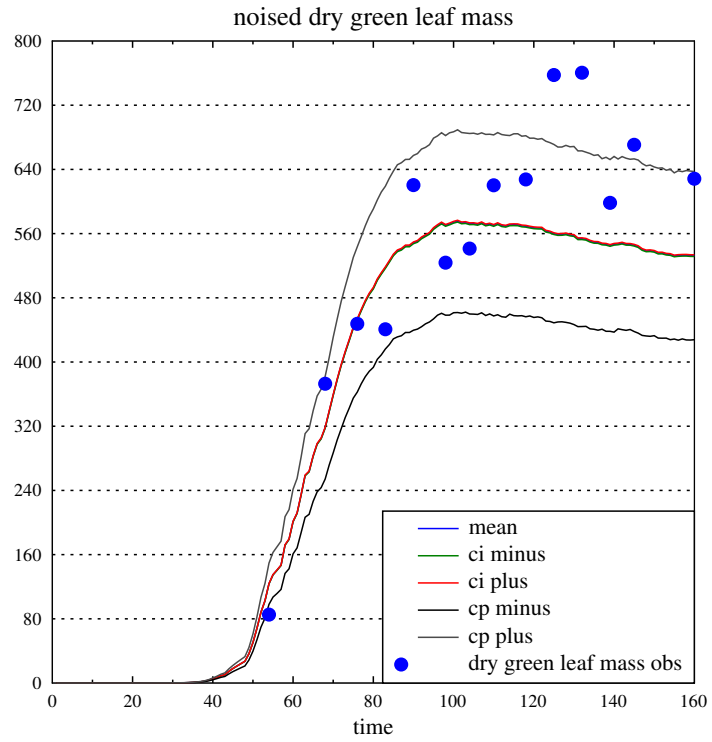


Figure 3.23 – analyse d'incertitudes sur la masse sèche

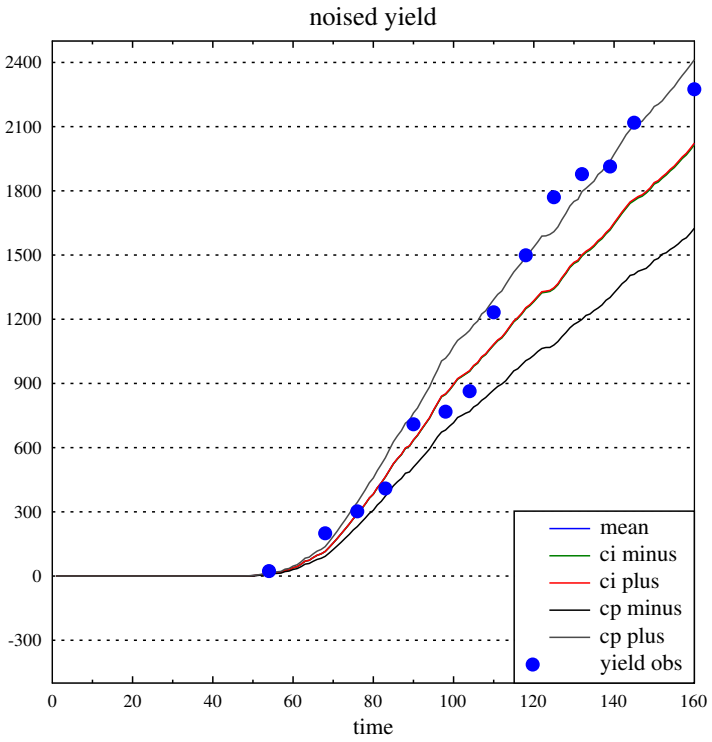


Figure 3.24 – analyse d'incertitudes sur le rendement

Discussion On peut voir que nous avons en termes de prévision un intervalle à 95% qui contient peu ou prou l'ensemble des donnnées expérimentales.

3.2.2.8 Prévision sur un jeu de données de validation

En ce qui concerne les critères de prévision nous étudierons le RMSEP (racine carré de la moyenne de l'erreur quadratique de prédiction) et EF (efficience de modélisation) sur les données présentes dans la table 3.1 (page 91), correspondant à l'année 2008 (calibration réalisée sur l'année 2010).
Nous obtenons comme valeurs :

variable observée	RMSEP	EF
dry_green_leaf_mass	109.862	0.668777
yield	179.557	0.911718

Table 3.10 – critères de prévision

Discussion On peut s'apercevoir que le modèle calibré ainsi à une bonne efficience en ce qui concerne la prévision de rendement, moins sur la masse foliaire.

3.2.3 Étude pas à pas pour STICS avec simulation sous PYGMALION

Pour rappel l'étude pas à pas consiste en :

- une analyse générale par SRC ou SOBOL
- une analyse pour l'estimation paramètrique par SOBOL-GLS
- une estimation paramètrique par MCMC ou AITKEN
- une sélection par les critères AIC, AICc, BIC, ou DIC
- une caractérisation des incertitudes par la méthode de Monte-Carlo
- une caractérisation de la capacité prédictive du modèle calibré

Par rapport à ce schéma général il peut y avoir des aller-retours nécessaires entre certaines parties.

3.2.3.1 Description des données pour le blé

Les données d'études sont extraites d'une campagne de collecte sur des champs de blé de variété Raffy localisé à Villamblain pour l'année 2012 et l'année 2013 (cf. Chen et Cournède 2014). On peut voir que ces données ont bien les propriétés énoncées dans le chapitre 1 à savoir : hétérogénéité, rareté et irrégularité. Elles ont été collectées par l'INRA Orléans.

jour	biomasse feuille verte (g/m^2)	biomasse sèche totale (g/m^2)	biomasse du grain (g/m^2)
155	86.4	155.9	.
185	177.5	397.9	.
213	.	999.7	.
239	.	1525	.
269	.	1802.3	811.3

Table 3.11 – données blé pour l'année 2012

Année 2012

jour	biomasse sèche totale (g/m^2)	biomasse du grain (g/m^2)	indice de surface fiolaire
80	1.871	.	.
122	6.21	.	.

jour	biomasse sèche totale (g/m^2)	biomasse du grain (g/m^2)	indice de surface fiolaire
162	.	.	0.09
165	19.47	.	.
175	.	.	1.04
191	.	.	2.14
197	379.111	.	.
212	.	.	3.51
219	459.852	.	4.37
233	.	.	5.21
248	1718.137	.	.
266	1608.259	829.933	.

Table 3.12 – données blé pour l'année 2013

Année 2013

3.2.3.2 Description du modèle STICS

Le modèle STICS est un modèle de référence issu de la collaboration entre des chercheurs de l'INRA dont une description se trouve en (Brisson et al. 1998). C'est un modèle générique organisé en plusieurs modules qui représentent différents processus écophysiologiques. Nous avons pris dans cette étude une version minimale afin de décrire l'évolution du blé. Il n'y a pas de prise en compte du module sol par exemple.

On trouvera une description détaillée de la version du modèle STICS utilisé dans (Baey 2014). Cette description porte sur la croissance foliaire, la production de biomasse, l'allocation au grain et la sénescence.

3.2.3.3 Analyse de sensibilité générale

Nous conduirons directement une analyse par SOBOL en prenant comme échantillonnage :

paramètre	intervalle
STLEVAMFV	$[223.25, 257.25]$
STAMFLAXV	$[247.0, 357.0]$
STLEVDRPV	$[657.4, 878.85]$
STDRPMATV	$[665.0, 735.0]$
VLAIMAXP	$[2.09, 2.31]$
PENTLAIMAXP	$[5.225, 5.775]$
DLAIMAXBRUTP	$[0.000418, 0.000462]$
EXTINP	$[0.475, 0.525]$
VITIRCARBP	$[0.01045, 0.01155]$
EFFICIENCE	$[3.8, 4.2]$
SLAMAXP	$[332.5, 367.5]$
CROIRACV	$[0.114, 0.145]$
PGRAINMAXIV	$[0.03686, 0.05019]$
TIGEFEUILLEP	$[0.475, 0.525]$

Table 3.13 – échantillonnage des paramètres de STICS

```
observation-function = '[("MAGRAIN",[1...269]),("MASEC",[1...269]),("LAI",[1...269])]'
univariate-rule-list = '[("STLEVAMFV","uniform",[223.25,257.25]),
  ("STAMFLAXV","uniform",[247.0,357.0]),
  ("STLEVDRPV","uniform",[657.4,878.85]),
  ("STDRPMATV","uniform",[665.0,735.0]),
  ("VLAIMAXP","uniform",[2.09,2.31]),
  ("PENTLAIMAXP","uniform",[5.225,5.775]),
  ("DLAIMAXBRUTP","uniform",[0.000418,0.000462]),
  ("EXTINP","uniform",[0.475,0.525]),
  ("VITIRCARBP","uniform",[0.01045,0.01155]),
  ("EFFICIENCE","uniform",[3.8,4.2]),
  ("SLAMAXP","uniform",[332.5,367.5]),
  ("CROIRACV","uniform",[0.114,0.145]),
  ("PGRAINMAXIV","uniform",[0.03686,0.05019]),
  ("TIGEFEUILLEP","uniform",[0.475,0.525])]'
nbr-samples = 10000
```

Code 3.10 – échantillonnage de STICS pour analyse de sensibilité

L'évolution de la somme des indices de premier ordre pour SOBOL nous donne :

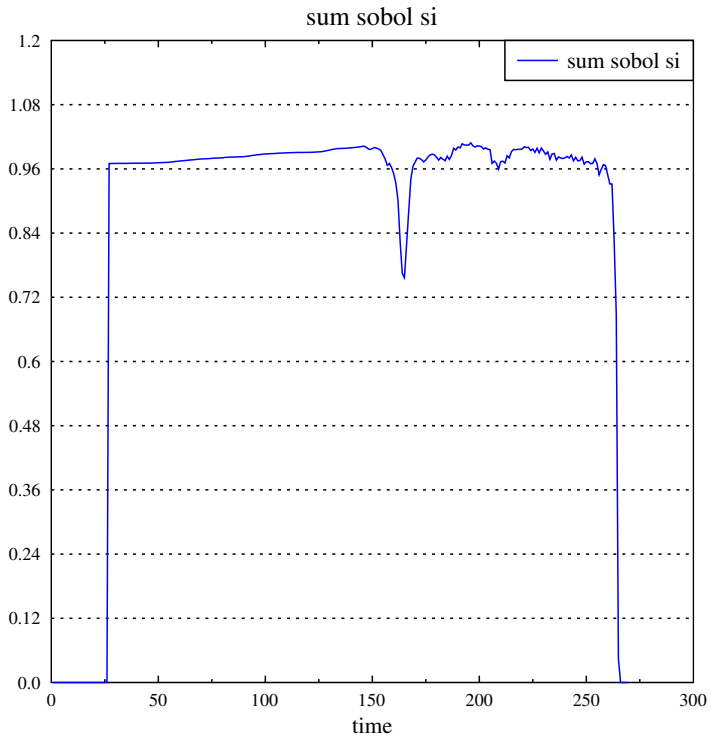


Figure 3.25 – somme des indices de SOBOL de premier ordre pour la variable "LAI"

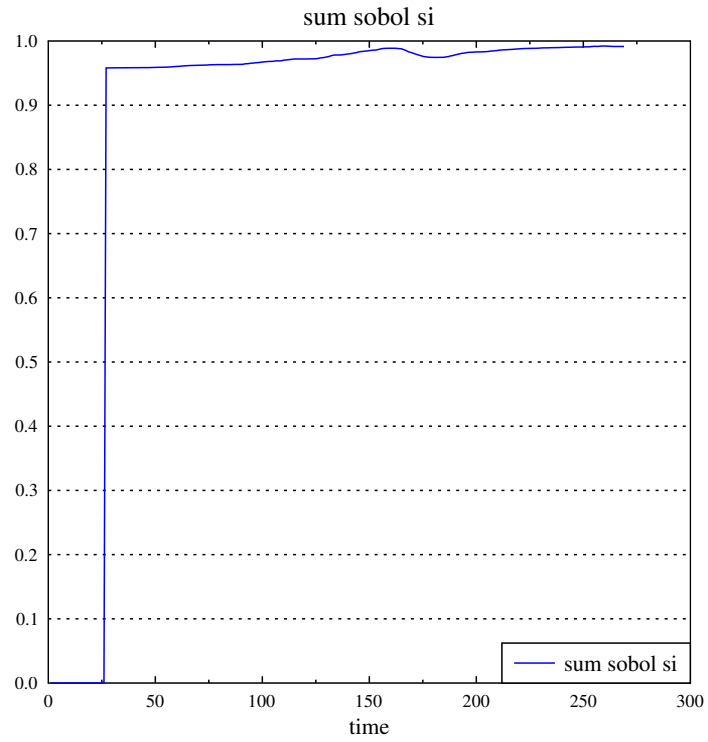


Figure 3.26 – somme des indices de SOBOL de premier ordre pour la variable ”MASEC”

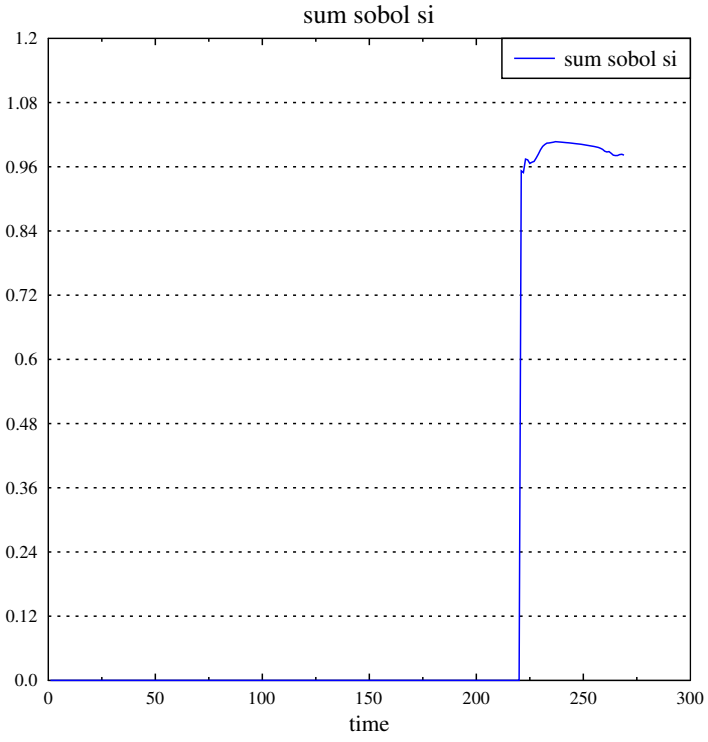


Figure 3.27 – somme des indices de SOBOL de premier ordre pour la variable ”MAGRAIN”

Nous pouvons nous apercevoir que selon les variables observées le comportement est plus ou moins linéaire. En effet si la masse sèche et la masse du grain sont très linéaires par rapport aux paramètres, l'indice de surface foliaire est quant à lui sujet à au moins deux phases un peu plus non-linéaires.

La différence entre les indices d'ordre total et les ordres du premier ordre au cours du temps sont donnés dans les figures 3.28, 3.29 et 3.30.

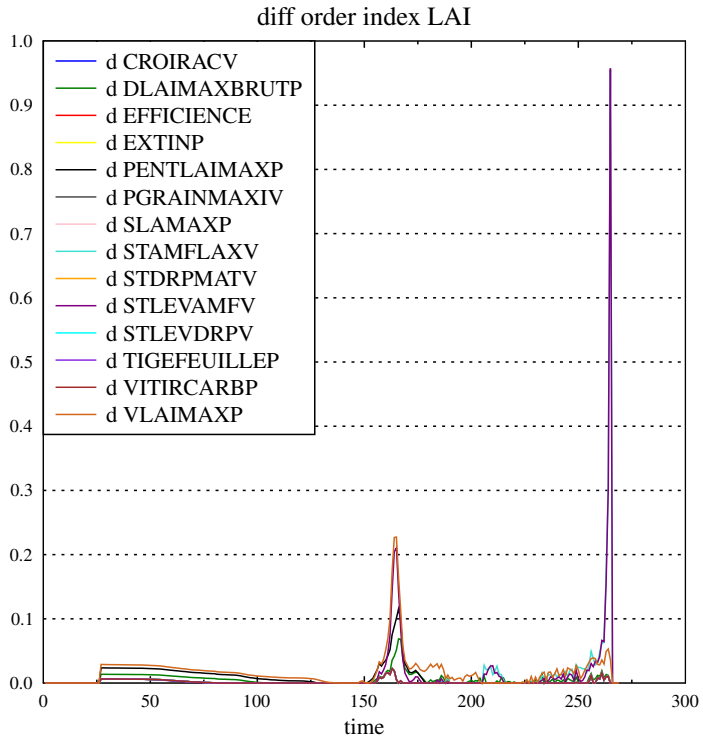


Figure 3.28 – différences entre ordre totaux et premier ordre pour la variable ”LAI”

Discussion On peut s'apercevoir que pour le LAI il y a deux phases qui vont se distinguer avec notamment une forte contribution de STLEVAMFV et STLEVDRPV qui sont des paramètres correspondant à des stades phénologiques de la culture marqués par des changements importants dans le modèle.

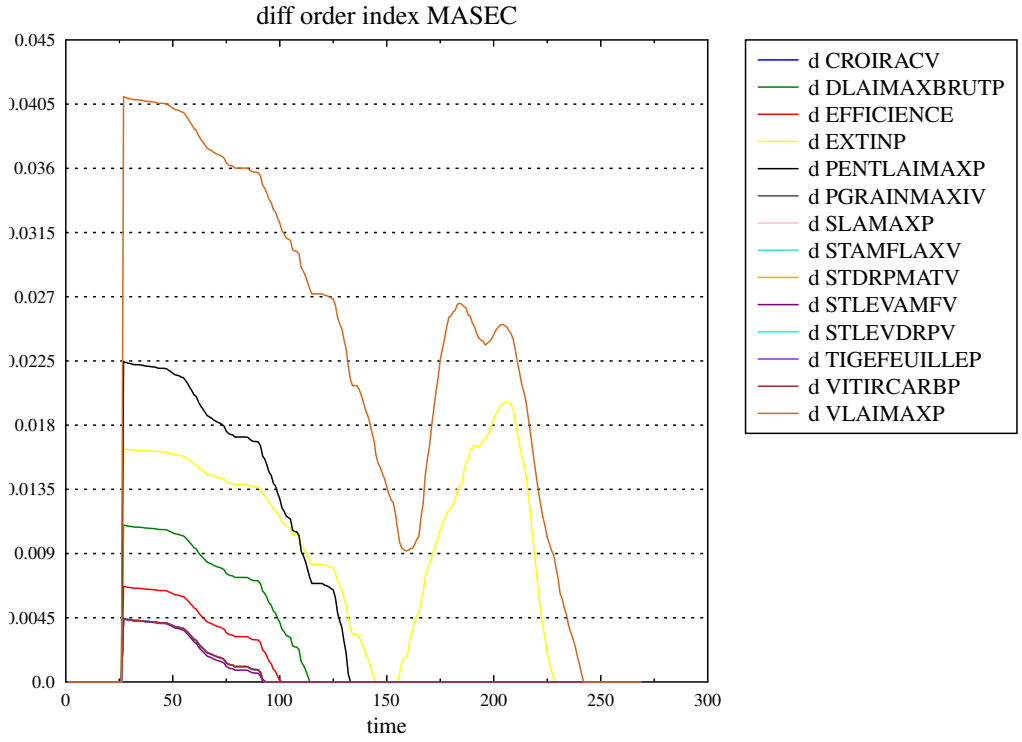


Figure 3.29 – différences entre ordre totaux et premier ordre pour la variable ”MASEC”

Discussion Il apparait qu'il y a deux phases que l'on peut distinguer. Une première où 5 paramètres interagissent et une seconde où seuls VLAIMAXP et EXTINP vont avoir un fort impact sur la non-linéarité de la sortie.

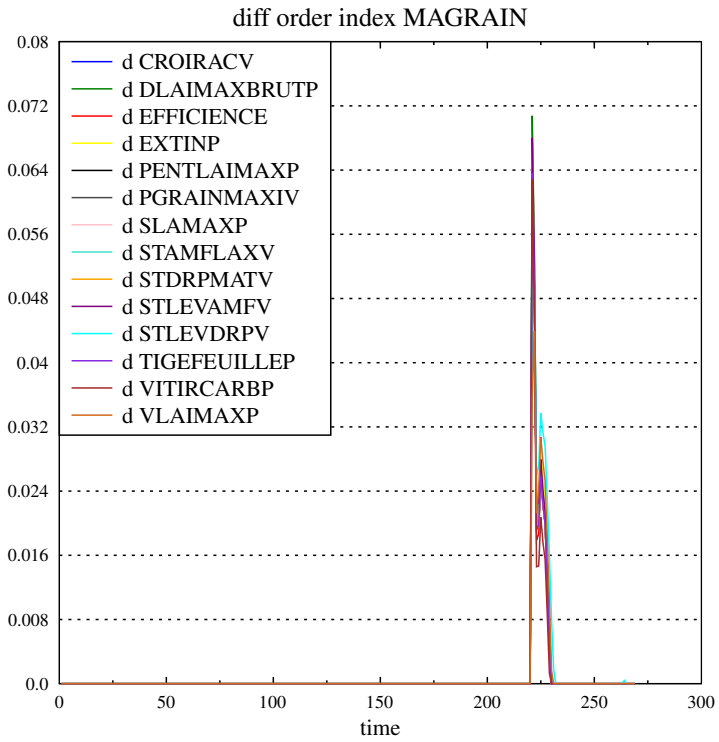


Figure 3.30 – différences entre ordre totaux et premier ordre pour la variable "MAGRAIN"

Discussion On voit qu'au moment de l'apparition de l'épi, le changement de régime d'allocation dans la plante se traduit par une interaction impliquant la plupart des paramètres.

3.2.3.4 Analyse de sensibilité pour l'estimation paramètrique

Nous allons à nouveau reconduire une analyse de sensibilité pour l'estimation paramètrique pour le modèle STICS en prenant comme intervalles de variations :

paramètre	S_i	S_{ti}
VLAIMAXP	0.464262	0.479582
STAMFLAXV	0.431873	0.456539
DLAIMAXBRUTP	0.0598993	0.0675172
STLEVDRPV	0.00796458	0.0124984
EFFICIENCE	0.00414454	0.00389019
PENTLAIMAXP	0.00143799	0.00364104
VITIRCARBP	0.00134827	0.00347699
EXTINP	0.00131374	0.00406082
STLEVAMFV	0.00130522	0.0132263
STDRPMATV	0.000426107	0.00292081
TIGEFEUILLEP	0	0.00246641
SLAMAXP	0	0.00246641
PGRAINMAXIV	0	0.00246641
CROIRACV	0	0.00246641

Table 3.14 – données sobol-gls sur stics 2013

Les résultats de l'analyse sont représentés dans la figure suivante :

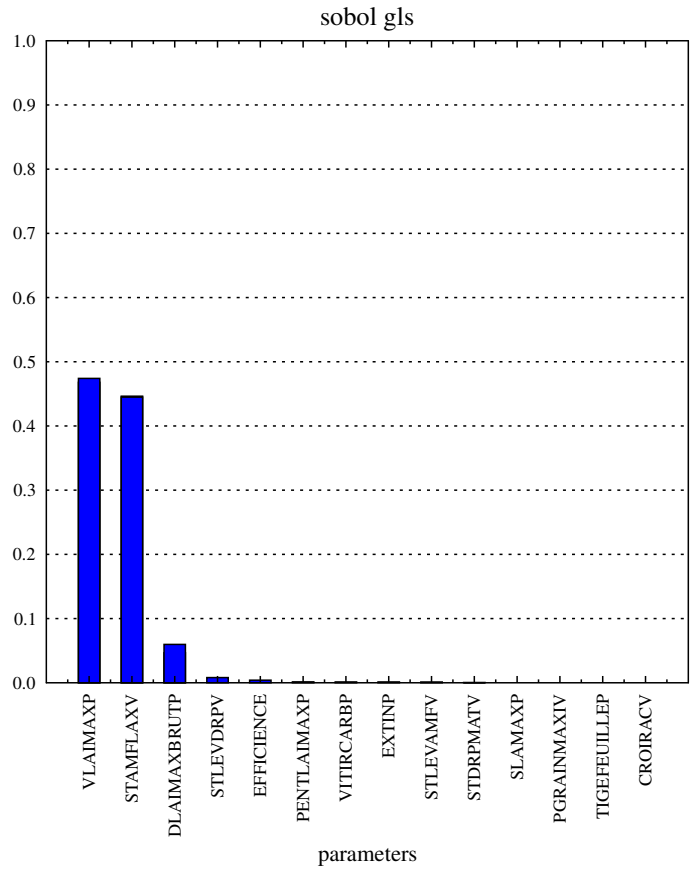


Figure 3.31 – visualisation sobol-gls stics 2013

Nous voyons qu'il y a clairement une domination des paramètres VLAIMAXP et STAMFLAXV ainsi qu'une lègére participation de DLAIMAXBRUTP.

3.2.3.5 Validation de l'estimation sur données simulées

Dans cette partie nous nous assurerons de la validité de l'algorithme AITKEN (moindres carrés généralisés) qui sera utilisé pour l'estimation du modèle STICS, qui est un modèle déterministe et donc pour lequel les méthodes bayesiennes ne sont pas applicables directement.

Nous simulerons des données à partir d'un point p_0 puis nous donnerons un point initial p_1 à l'algorithme d'estimation tel que p_1 soit plausible d'un point de vue agronomique. En sortie de l'estimation nous récupèrerons le point estimé ainsi que son écart-type et la matrice de covariance associée, calculé par l'inverse de la matrice d'information de Fisher. (Cournède et al. 2011)

paramètre	p_0	p_1
VLAIMAXP	2.2	1.9
STAMFLAXV	340	230
DLAIMAXBRUTP	4.6e-4	4.4e-4

Table 3.15 – point p_0, p_1

Nous conduirons l'estimation paramètrique avec un estimateur de Aitken à 2 étapes. L'algorithme de minimisation associé est Gauss-Newton. Nous configurons l'algorithme de la façon suivante.

```
--target "y_sim.csv"
--initial-parameters "p1.txt"
--selected-parameters '["DLAIMAXBRUTP","VLAIMAXP","STAMFLAXV"]'
```

Code 3.11 – configuration d'AITKEN pour l'estimation des paramètres de STICS

Nous obtenons comme point estimé :

paramètre	valeur estimée	valeur réelle
VLAIMAXP	2.208	2.2
STAMFLAXV	341	340
DLAIMAXBRUTP	4.65e-4	4.6e-4

Table 3.16 – point $p_{\hat{\theta}}$

Les simulations à partir du point estimé sont représentées dans les figures suivantes :

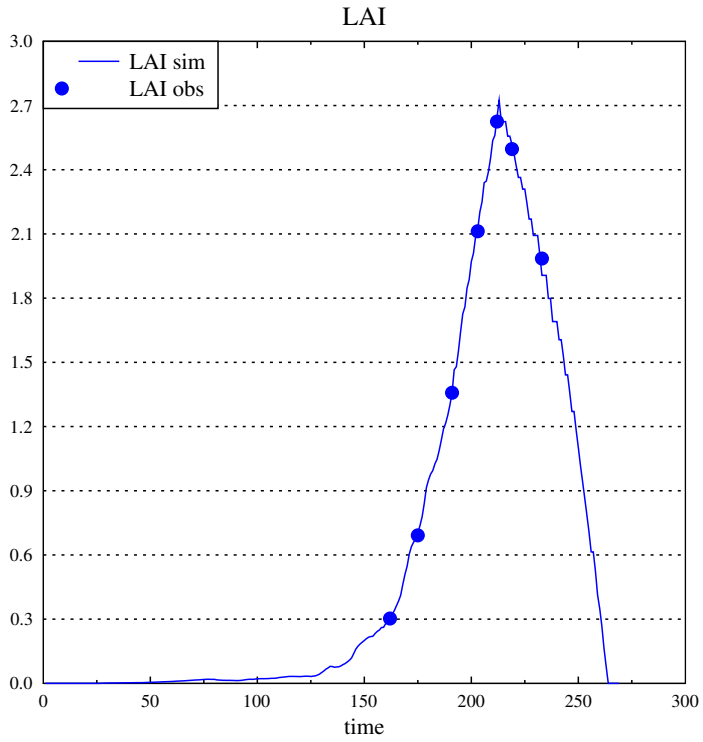


Figure 3.32 – simulation avec les paramètres estimés versus les données simulées pour "LAI"

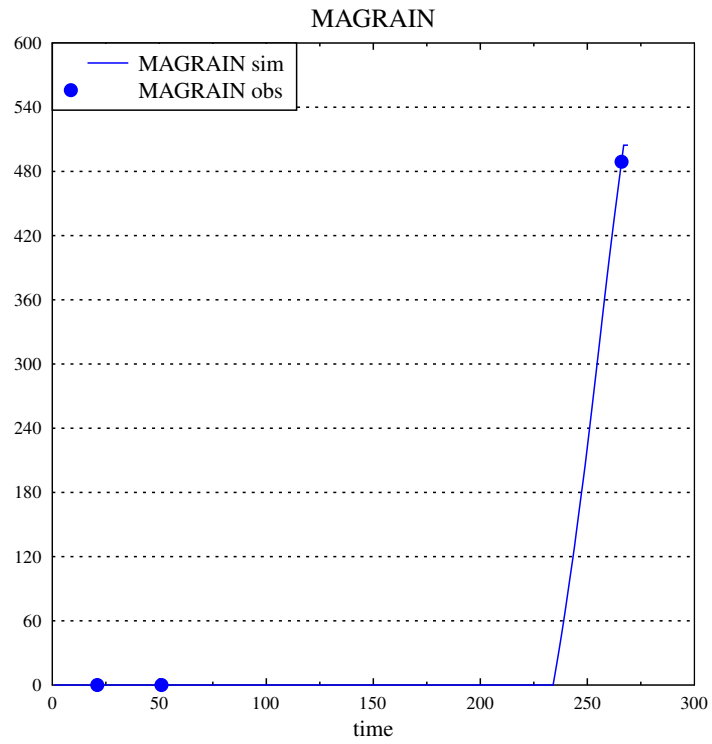


Figure 3.33 – simulation avec les paramètres estimés versus les données simulées pour ”MAGRAIN”

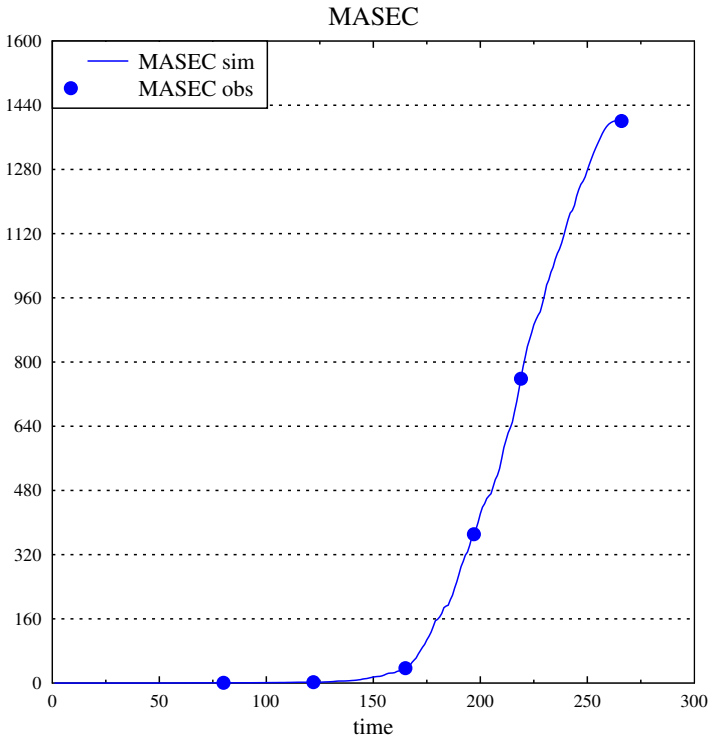


Figure 3.34 – simulation avec les paramètres estimés versus les données simulées pour ”MASEC”

Discussion Nous avons maintenant une vue générale sur le modèle et avons confirmation que sur au moins des données simulées il n’y a pas de problème à l’estimation. Nous allons maintenant estimer les valeurs des paramètres du modèle sur données réelles.

3.2.3.6 Estimation paramètrique sur un jeu de données d'apprentissage

Nous utiliserons les données de l'année 2013 présentées dans le tableau 3.12.

L'analyse de sensibilité nous a donné comme paramètres à étudier dans l'ordre :

- VLAIMAXP
- STAMFLAXV
- DLAIMAXBRUTP
- STLEVDRPV
- EFFICIENCE
- PENTLAIMAXP
- VITIRCARBP
- EXTINP

Nous nous restreignons à ces paramètres et nous allons étudier la qualité des estimations à partir des critères de types AIC, AICc, et BIC. Nous opterons pour un schéma d'estimation faisant progressivement entrer chaque paramètre l'un après l'autre. Ainsi, nous étudierons seulement l'estimation de VLAIMAXP dans un premier temps, puis VLAIMAXP et STAMFLAXV, etc .

Les résultats obtenus sont présentés en table 3.17 et sur la figure 3.35.

nombre de paramètres estimés	log-likelihood	AIC	AICc	BIC
1	81.4565	164.913	165.221	165.621
2	73.7394	151.479	152.479	152.895
3	**52.3947**	**110.789**	**112.971**	**112.914**
4	55.3323	118.665	122.665	121.497
5	51.1288	112.258	118.924	115.798
6	51.1053	114.211	124.711	118.459
7	45.5674	105.135	121.135	110.091
8	45.8463	107.693	131.693	113.357

Table 3.17 – valeurs des critéres de sélection de STICS sur les données 2013

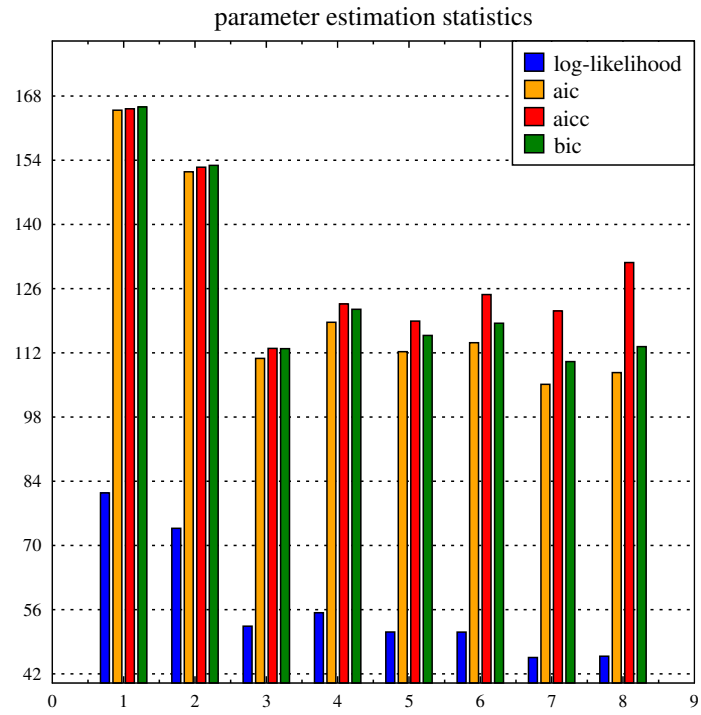


Figure 3.35 – visualisation des critères de sélection pour STICS sur données 2013

On peut s'apercevoir que l'AICc est minimal lorsque l'on estime l'ensemble avec uniquement 3 paramètres que sont VLAIMAXP, STAMFLAXV et DLAIMAXBRUTP.

La solution à 7 paramètres aurait également été pertinence (AIC et BIC minimaux). Cela dit, dans un objectif de robustesse, il est préférable de privilégier l'approche avec le moins de paramètres de calibration.

Nous choisirons donc cette solution. Les résultats de simulation avec ce paramètre donne comme représentation graphique :

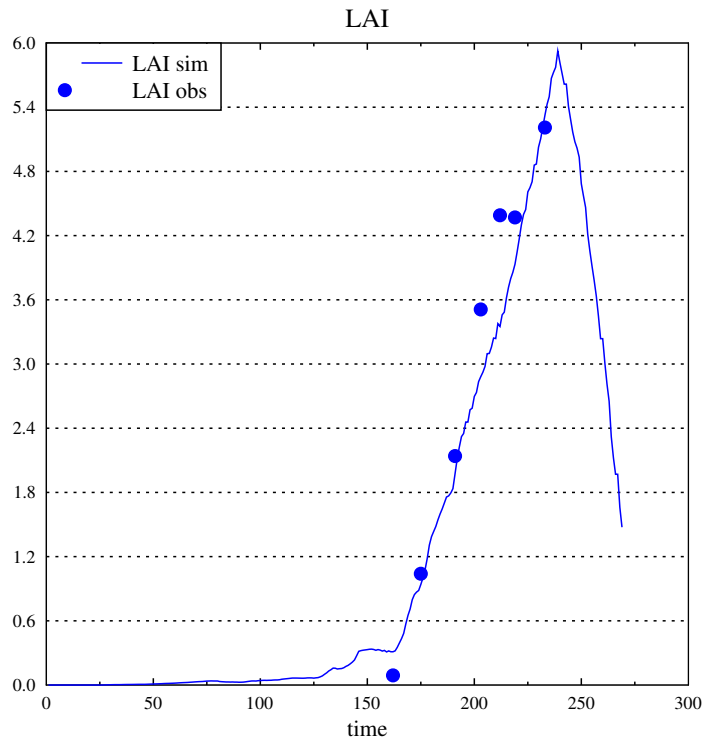


Figure 3.36 – simulation avec les paramètres estimés versus les données simulées pour ”LAI”

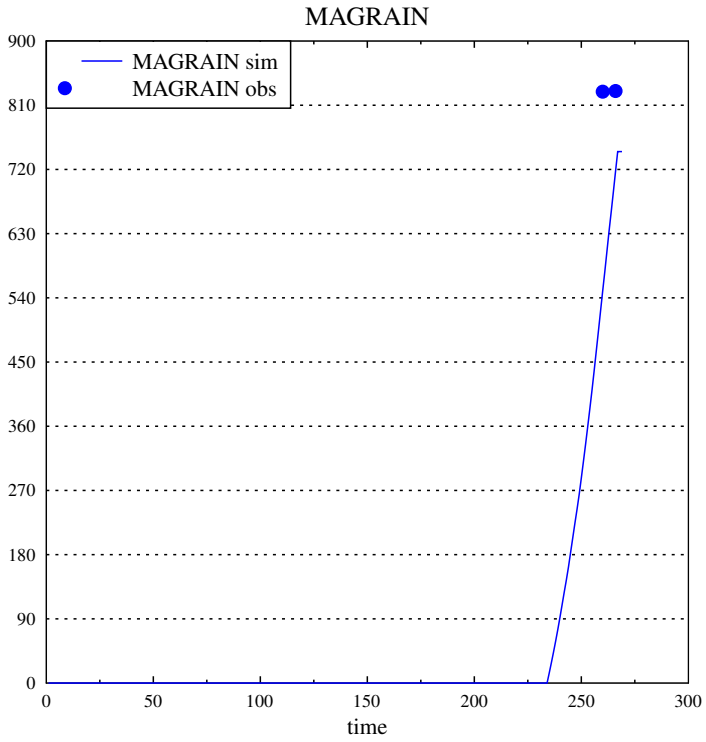


Figure 3.37 – simulation avec les paramètres estimés versus les données simulées pour ”MAGRAIN”

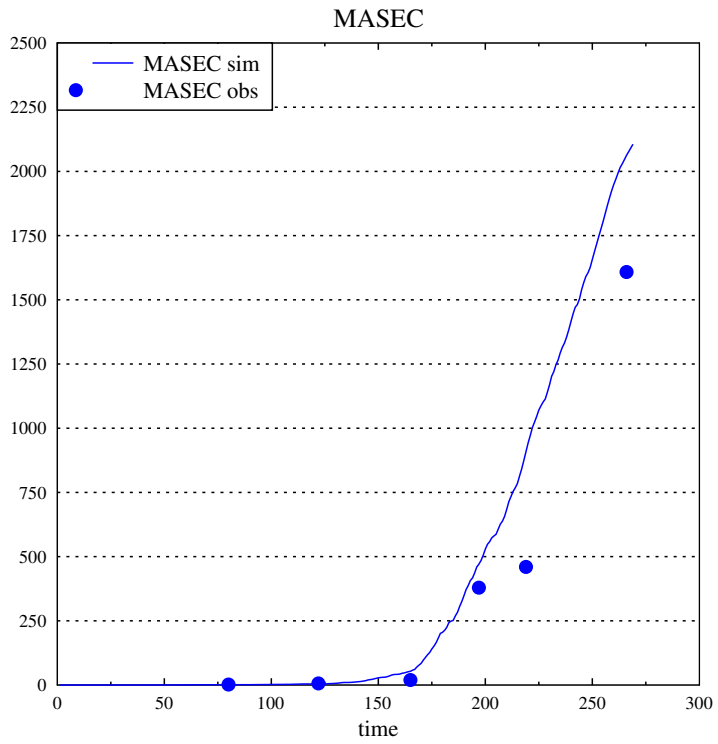


Figure 3.38 – simulation avec les paramètres estimés versus les données simulées pour ”MASEC”

On peut constater qu'il y a une surestimation de la masse sèche.

3.2.3.7 Caractérisation des incertitudes sur les paramètres et la prévision

Nous renvoyons le lecteur aux différentes caractérisations à la page 105.

paramètre	valeur estimée	IC^-	IC^+
DLAIMAXBRUTP	0.000671	-0.000154	0.00150
STAMFLAXV [4]	674	633	715
VLAIMAXP	2.14	1.88	2.41

Table 3.18 – intervalle de confiance à 95% sur les paramètres

Intervalle de confiance à 95% sur les paramètres

Intervalle de prévision à 95% Regardons maintenant l'incertitude autour du point estimé en regardant un échantillonnage qui utilise la matrice de covariance sur les paramètres obtenue par l'inverse de la matrice de Fisher.

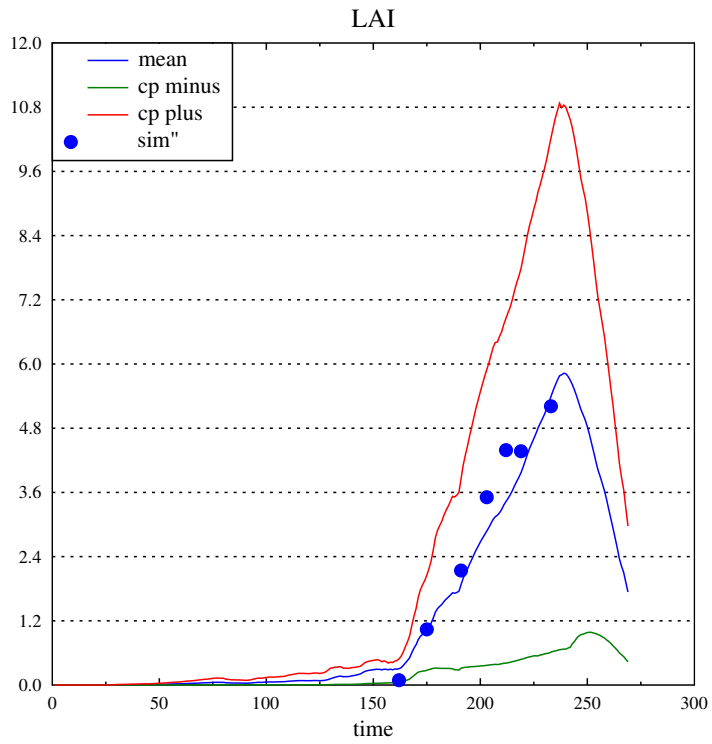


Figure 3.39 – analyse d'incertitudes pour la variable "LAI" avec échantillonnage univarié

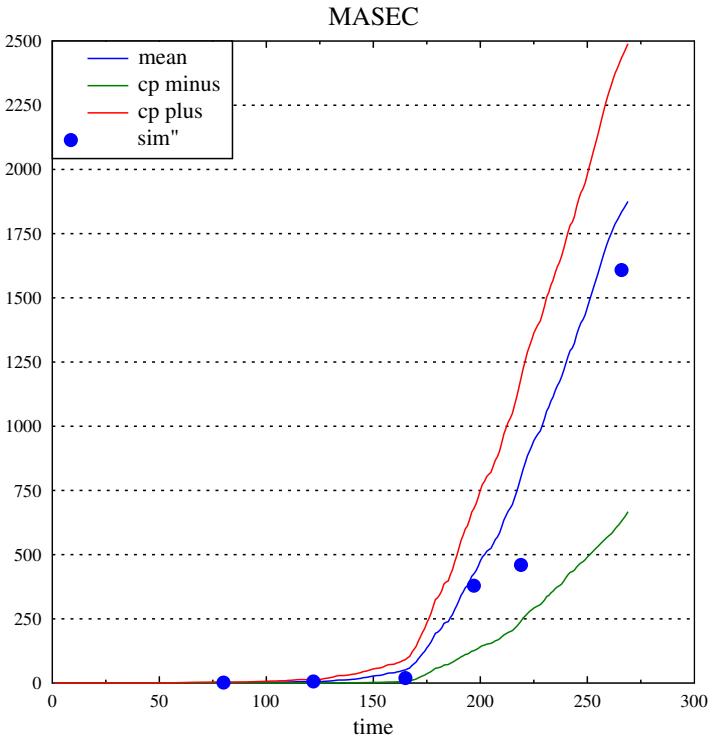


Figure 3.40 – analyse d'incertitudes pour la variable "MASEC" avec échantillonnage univarié

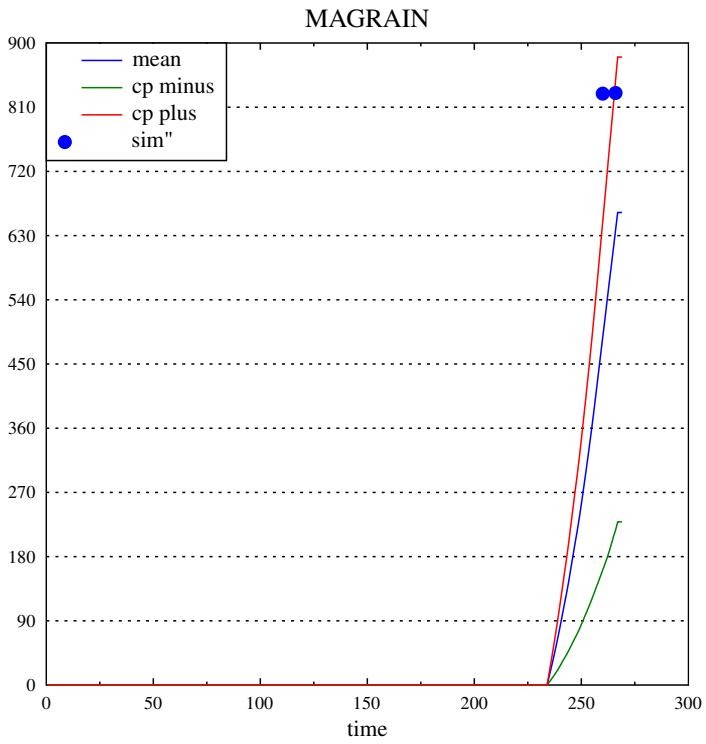


Figure 3.41 – analyse d'incertitudes pour la variable "MAGRAIN" avec échantillonnage univarié

3.2.3.8 Prévision sur un jeu de données de validation

Nous testons le modèle calibré sur le jeu de données de l'année 2013 présentée dans le tableau 3.12.
En ce qui concerne les critères de prévision, nous étudierons le RMSEP (racine carré de l'erreur quadratique de la moyenne de prédiction) et EF (efficience de modélisation) sur les données présentées dans la table 3.11 (page 107), qui correspondent à l'année 2012.
Nous obtenons comme valeurs :

variable observée	RMSEP	EF
MAFV	47.8952	0.947155
MAGRAIN	20.9654	0.994186
MASEC	144.357	0.923262

Table 3.19 – critères de prévision

Nous avons un estimateur relativement fiable sur la masse du grain, et également correct en ce qui concerne la masse des feuilles vertes et la masse sèche.

3.2.4 Analyse de sensibilité pour LIGNUM avec simulation sous GROIMP

Dans cette section, nous montrons uniquement une étude d'analyse de sensibilité appliquées au modèle LIGNUM (Perttunen et al. 2002) afin de montrer la faisabilité d'externaliser les simulations.

Cette étude a été menée en autonomie par un autre membre de l'équipe à partir des outils développés grâce aux éléments présentés précédemment (Streit et al. 2016).

Ce modèle, qui est un modèle structure-fonction, a été codé sous la platforme GroIMP (Kniemeyer et Kurth 2008) avec le langage dédié à cette plateforme nommé XL (Kniemeyer 2008). Ce langage permet une expression plus aisée des éléments propres aux modèles structure-fonction et possèdent aussi des briques logicielles dédiées à l'interception lumineuse.

En plus du calcul de la structure et de la biomasse, le modèle calcule l'absorption et la transmission de la lumière comme s'il le faisait pour un seul arbre (Perttunen, Sievänen et Nikinmaa 1998) mais rajoute la contrainte d'être entouré par une forêt homogène autour de l'arbre considéré (Sievänen et al. 2008). Ainsi l'arbre généré est en compétition avec d'autres pour ce qui est de l'absorption lumineuse.

3.2.4.1 Analyse de sensibilité générale

L'analyse de sensibilité porte uniquement sur l'impact de 5 paramètres sur la variation de biomasse.
Ces paramètres sont :

- l'angle de branchage noté "pba"
- la courbure annuelle notée "pdown"
- la longeur des épines notée "pnl"
- le ratio entre le rayon et la longeur des épines noté "plr"
- l'angle des épines noté "pna"

On trouvera ci-dessous dans la table 3.20 les valeurs des échantillonnages à partir de lois uniformes.

paramètre	intervalle
pba	$[40.5, 49.5]$
pdown	$[1.8, 2.2]$
pnl	$[0.016, 0.059]$
plr	$[0.008, 0.012]$
pna	$[25, 45]$

Table 3.20 – échantillonnage des paramètres de LIGNUM

Les résultats de l'analyse de sensibilité montrent que le modèle devient de plus en plus non-linéaire avec le temps en fonction de la biomasse et des paramètres sélectionnés (figure 3.42). On peut notamment voir que la contribution du paramètre concernant la longueur des épines pour l'échantillonnage considéré s'accroit avec le temps (figure 3.43)

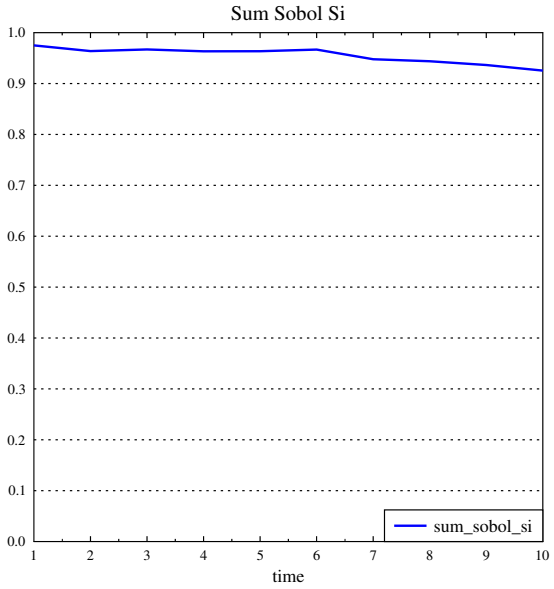


Figure 3.42 – accumulation des paramètres de premier ordre par rapport à la biomasse totale

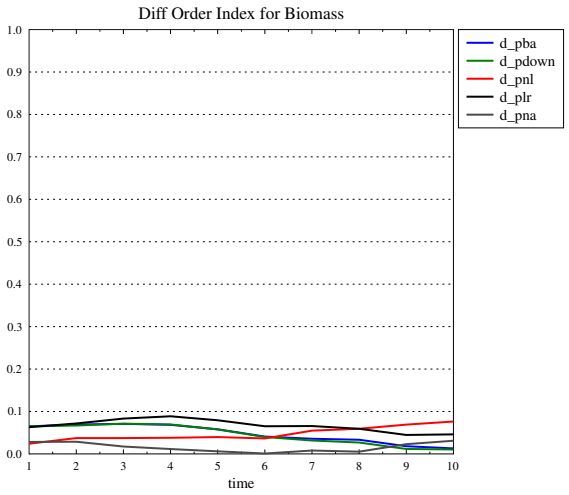


Figure 3.43 – différences entre l'ordre total et le premier ordre pour la biomasse totale

3.2.4.2 Bilan

Nous avons présenté les résultats d'une analyse de sensibilité sur le modèle Lignum, dont le simulateur est codé dans la plateforme GroIMP, et qui est donc appelé dans Pygmalion comme simulateur externe. Cette étude se poursuit par des travaux de calibration du modèle, et démontre qu'une personne conduisant une analyse avec son simulateur propre est parfaitement à même de discuter et d'échanger avec les autres modélisateurs et concepteurs d'algorithmes grâce à la transversalité du langage.

3.3 Discussion générale

Ces trois études ont mené à, d'une part, la mise au point d'un estimateur dans le cadre de la betterave avec LNAS et dans le cadre du blé pour STICS, et d'autre part, la démonstration de la faisabilité technique d'une étude dans le cas d'un simulateur indépendant à tout le système avec un modèle de croissance du pin basé sur LIGNUM et écrit sous la plateforme GroIMP.

Il faut bien prendre conscience que ce l'on cherche à montrer ici c'est l'utilisation du noyau et du langage défini dans les précédents chapitres.

Ainsi il est important de comprendre que ces études ont été menées grâce à un simple fichier où nous avons entré les différentes configurations. Nous avons affiné les calculs en les relançant plusieurs fois et en cherchant les bons paramètres en fonction des sorties des différents algorithmes. La syntaxe et sémantique communes permettent de passer d'un algorithme à l'autre facilement pour peu que l'on connaisse le domaine d'application et les types de résultats produits. Nous avons ainsi drastiquement réduit le temps d'apprentissage et la faisabilité de phases de validation et vérification des modèles implémentés. Un modèle qui ne passerait pas par ces phases ne devrait même pas être considéré comme un candidat comme modèle en production (utilisation par un expérimentateur, utilisation par un organisme externe, ...)

Nous allons voir quelques éléments factuels autour du développement de cette plateforme et de son utilisation avant de conclure sur ce chapitre.

3.3.1 Coût en développement

En terme de développement, les plateformes représentent un temps de total de 785 jours d'activité de développement. Un jour étant considéré comme actif s'il y a eu au moins un commit dans le système de version. Le rythme est de 4 commits par jour actif. Dans la première version, il y a eu au total 140 000

lignes de code C++ tout confondus et 25 000 lignes de code pour la seconde version. Pour la description des différentes architectures on peut se reporter à la section 2.2.1. Le système étant codé en C++ nous avons utilisé différents compilateurs afin de s'assurer de sa portabilité à savoir le compilateur GNU GCC sous Linux, le compilateur Clang basé sur LLVM sous Mac OS X, le compilateur de Visual Studio sous Windows 7 et enfin le compilateur ICPC d'Intel pour le cluster sous CentOS6. L'ensemble est intégré dans une chaîne d'intégration continue avec production automatique de documentation Doxygen et passage de tests unitaires et d'intégration sur chaque commit.

3.3.2 Utilisation sur un cluster

En ce qui concerne le passage à l'échelle, le travail effectué a permis de former environ 30 personnes autour des notions développées.

En regardant les temps calculs de l'équipe qui utilise cette base de code, nous pouvons visualiser sur le graphique 3.44 la consommation totale en termes de temps calcul avec une projection sur l'année 2016 qui correspondant à une simple multiplication par 3 des 4 premiers mois effectués. On peut voir qu'il y a depuis l'introduction de la plateforme, une augmentation du temps cpu total par année avec un nombre d'utilisateurs variant autour de 10 mais avec une augmentation du nombre d'années de temps calcul par utilisateur. Cela pourrait traduire soit une meilleure efficience de l'utilisation générale du système sinon une amélioration dans les choix techniques dédiés aux algorithmes.

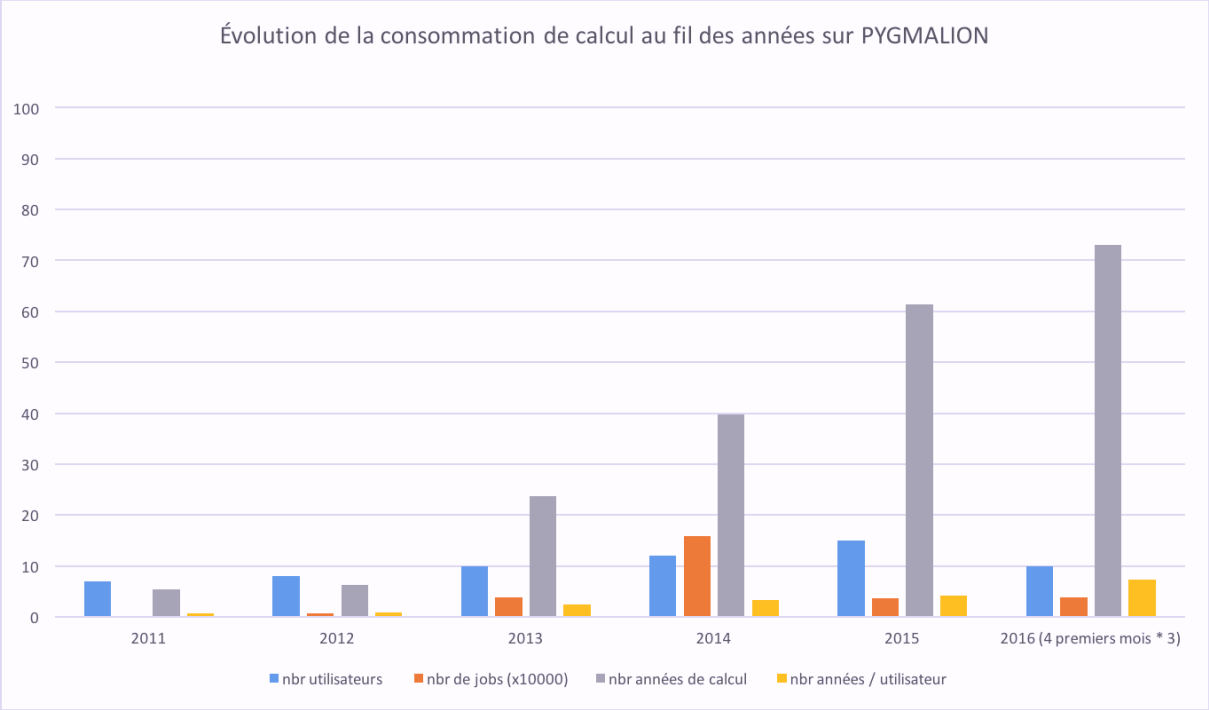


Figure 3.44 – évolution de la consommation calcul au fil du temps

En enlevant les utilisateurs avec le moins de travaux effectués sur le cluster de CentraleSupélec, nous passons de 24 personnes à 18 et nous pouvons créer une lecture des profils "types" de la plateforme en rapportant le temps moyen par calcul sur un coeur.

On peut voir, sur la figure 3.45, quelques types de profils qui se dégagent très peu de travaux mais temps cpu long, généralement des gens qui analysent un modèle, beaucoup de travaux mais temps cpu court, généralement de la conception d'algorithmes sur des modèles "jouets", et les profils atypiques comme très peu de jobs et très peu de temps cpu, généralement des personnes qui sont restés peu de temps, ou beaucoup de jobs et beaucoup de temps, on retrouve ici plutôt aussi des concepteurs d'algorithmes.

Pour information, le profil de l'auteur est le profil numéro 4.

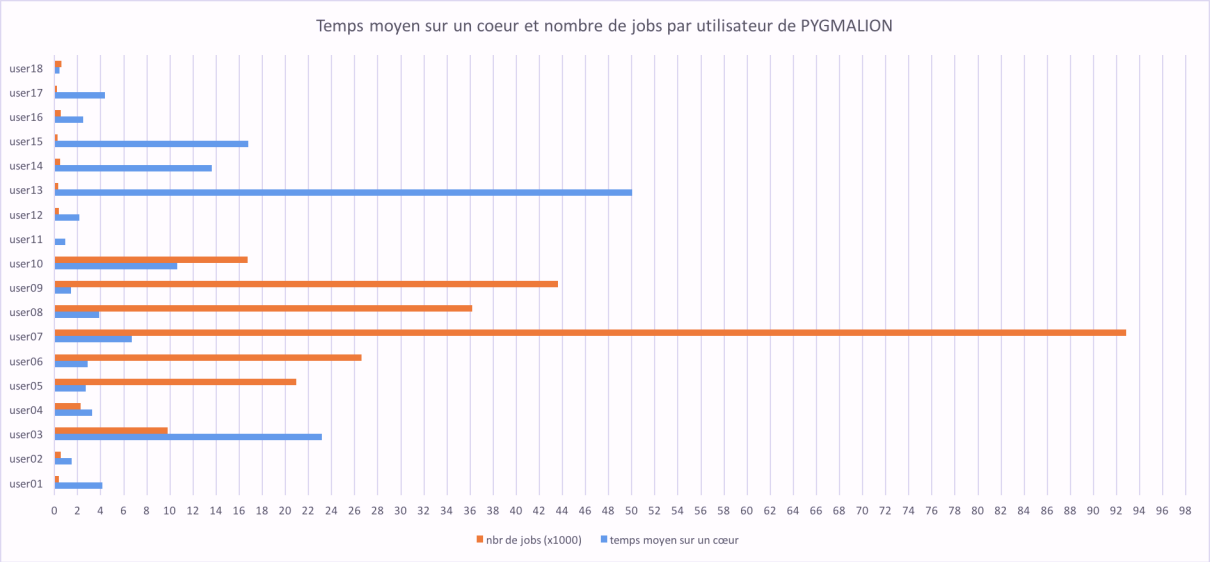


Figure 3.45 – profils utilisateurs en fonction du nombre de jobs et temps moyen par job sur un coeur

3.4 Conclusion

Nous avons vu l'utilisation d'une plateforme développée par le truchement du langage créé en amont. Ce langage est dédié à la chaîne d'analyse des modèles, mais peut aussi s'appliquer dans le cadre de la création d'un modèle. Il possède une version implémenté en tant que langage dédié embarqué en C++ et une version autonome avec son propre compilateur mais restreint au cadre de modélisation des systèmes dynamiques stochastiques discrets en temps.

Cette utilisation a permis de fédérer l'ensemble des personnes autour de la création d'un environnement commum permettant de réaliser l'ensemble de la chaîne d'étude des modèles et pouvant aussi être appliqué à n'importe quel simulateur extérieur à la plateforme en restreignant le cadre formel.

La normalisation du langage, en n'introduisant que très peu de concepts extérieurs au modèle mathématique d'origine, permet une communication efficace entre les modélisateurs et les algorithmes. Cette plateforme a été utilisée par environ 30 personnes au fil des années.

L'utilisation du cube de simulation comme cadre conceptuel a permis de développer les algorithmes autour des notions de calcul parallèle, permettant ainsi d'accumuler entre 2011 et 2015 plus de 135 années de temps calcul si l'ensemble était calculé sur une machine mono-coeur.

Aujourd'hui, il existe des outils permettant de traiter une partie ou une autre de la chaîne, l'ensemble de la communauté y gagnerait à utiliser le formalisme développé comme base conceptuelle et langage d'échange afin de permettre, à minima, la comparaison des expériences faites. On pourrait imaginer une plateforme centrale lançant indifférement tel ou tel outil sur la base de ce langage.

Chapitre 4

Perspectives et conclusion

La conclusion revient sur les diverses contributions scientifiques en termes de logiciels et de publications ainsi que sur les perspectives de développement du système informatique décrit.

4.1 Contributions

Cette thèse a donné lieu à plusieurs publications et développements logiciels.

4.1.1 Apports de nos travaux

Si nous reprenons la formulation de notre problématique, nous pouvons apporter quelques éléments de réponse :

"Quels sont les éléments de vocabulaire, la sémantique et l'arbre de syntaxe nécessaires à la création d'un langage dédié pour la simulation et l'analyse quantitative des modèles ?" À partir de la représentation d'un système dynamique à espace d'états, nous avons construit un vocabulaire associé à des types et des signatures permettant la création à la fois d'un langage dédié embarqué pour la création de certains modèles et des algorithmes d'analyse, mais aussi d'un langage dédié permettant de créer plus facilement des modèles dynamiques discrets stochastiques. Nous verrons dans les perspectives comment faire évoluer ce langage en essayant notamment d'aller vers d'autres domaines spécifiques.

"Comment coupler N modèles à M algorithmes afin d'éviter de redévelopper chaque algorithme pour chaque modèle, ce qui est généralement la norme dans les développements actuels ?" Nous avons proposé de modifier nos algorithmes afin de fonctionner avec tout type de simulateur externe. Toutefois, cette généralisation est à l'heure actuelle peu performante du fait des coûts en entrées-sorties. Il faudrait voir à créer un format adapté basé sur une représentation binaire pour stocker l'ensemble des simulations. Nous verrons cela dans la prochaine section. Il nous faudrait aussi voir comment réinsérer le tout dans des flux de programmation plus classiques du domaine comme MATLAB ou R.

"Comment gérer l'aléatoire dans le cadre des modèles et des algorithmes ?" Nous avons étudié le champ des générateurs aléatoires et avons implémenté une solution pragmatique qui ne représente pas l'état de l'art mais permet néanmoins d'être confiant dans les résultats produits. Toutefois, il serait bon de revoir si des bibliothéques ou des logiciels candidats ont émergé depuis dans la communauté et que nous serions à même d'intégrer dans notre simulateur.

"Comment mettre en place un flux de travail permettant de sélectionner le bon modèle vis à vis d'une problématique tout en garantissant son domaine de validité ?" Nous avons montré des études mettant en jeu différents modèles et l'ensemble de la chaîne des bonnes pratiques de modélisation en discutant du choix des algorithmes implémentés. Ce flux utilise le langage créé en amont afin d'interfacer l'ensemble des algorithmes. Nous verrons par la suite qu'il serait bien de s'inspirer de ce travail afin de proposer un standard pour échanger les différentes expériences virtuelles menées entre les différents logiciels d'analyse et d'estimation sur un même modèle.

"Comment réduire la courbe d'apprentissage et l'opérationnalité d'un nouveau modélisateur ?" Nous avons présenté ce langage et ces outils à des personnes n'ayant pas fait parti du processus de conception et développement qui ont réussi à prendre en main l'ensemble sans réelle indication si ce n'est les concepts relatifs au domaine de l'analyse de sensibilité ou à l'estimation paramétrique. Une fois les concepts compris, les études ont pu être menées sans trop d'intervention. Cela laisse à penser que l'étude formelle du domaine et la construction de l'ensemble permet de réduire la complexité des études. Toutefois, pour l'instant il faut avoir certaines connaissances en termes de manipulation des calculateurs. Nous montrerons quelque perspectives permettant de réduire encore plus la complexité et d'aller au délà en tentant de fédérer une communauté.

"Comment exploiter les architectures modernes de calcul afin de pouvoir gérer la montée en charge et ainsi renforcer la performance et par extension la robustesse de la chaîne de travail ?" Pour l'instant, nous exploitons les architectures à mémoire partagée. Cela s'est montré suffisant pour l'ensemble de nos études. Toutefois, il apparait clair que nous devons mettre au point un simulateur permettant de monter en charge sur des architectures à mémoire distribuée voire massivement parallèle.
Pour résumer :

> Nos travaux ont porté sur la formalisation de la chaîne des bonnes pratiques de modélisation en démarrant de la représentation d'état de systèmes dynamiques discrets stochastiques. L'originalité de notre démarche tient dans la cohérence générale du système qui lui est conféré par la mise en place d'un langage développé sur la base de la modélisation mathématique adaptée à notre cadre de modélisation mécaniste. Au lieu de regarder vers la composition des systèmes sous-jacents du modèle, nous avons regardé la composition des systèmes au-dessus du système considéré qui permettent ensemble de conduire une bonne étude de modélisation permettant de passer du statut de modèle à celui de modèle vérifié et validé. Cette formalisation a permis le développement d'une plateforme cohérente et pouvant exploiter certaines capacités parallèles des calculateurs. Il permet à la fois la conception du système interne mais aussi l'ensemble des communications entre les différents programmes. Nous sommes en train de travailler sur le passage à l'échelle de l'ensemble avant de voir comment l'utiliser pour connecter d'autres éléments qui ont été initialement non-prévues pour la plateforme. Nous pensons que ces éléments de langage vont permettre de définir un format standard d'échange concernant la validation et vérification des modèles. Nous en reparlerons plus loin lors des perspectives et de la notion de passeport de modélisation.

4.1.2 Logiciels

Il y a eu un important effort de développement fourni à différents niveaux. Si l'on pense l'ensemble de la plateforme au moyen d'une architecture trois-tiers alors nous avons des logiciels pour chaque tiers à savoir le tiers données, le tiers applications, le tiers présentation.

nom du projet	but	tiers
stochasticity	étude du comportement de plusieurs générateurs	application
quantitative-agronomy-ontology	rédaction d'une ontologie avec Protégé	données
data-manager	utilitaire en ligne de commande pour interragir avec une base de données fichier (csv, json)	données
pygmalion A	bibliothèque C++ utilisant des modèles templatisés	application
pygmalion B	bibliothèque C++ utilisant des modèles templatisés ou des bibliothèques dynamiques ou le langage dédié	application
kaffee	prototype de client-serveur avec pygmalion B en tant que serveur	application
cinammon	outil en ligne de commande interfaçant data-manager et pygmalion-b qui a pour but d'être le client de kaffee	présentation
coconut	web-gui de cinnamon avec gestion de comptes - orienté applications métiers de la modélisation	présentation
gaia	web-gui de cinnamon orienté applications métiers de l'agronomie	présentation

4.1.3 Publications

Nous faisons la distinction entre les publications en tant qu'auteur principal et les publications en tant qu'auteur associé.

4.1.3.1 En tant qu'auteur principal

— Towards an EDSL to enhance good modelling practice for non-linear stochastic discrete dynamical models Application to plant growth models (Bayol, Chen et Cournède 2013) décrit les bases du modèle informatique, de l'EDSL, du cube de simulation et introduit les fonctionnalités des bonnes pratiques de modélisation.
— Promoting Good Modeling Practice with a Domain-Specific Language and Statistical Algorithms designed for Parallel Computing (WIP) (Bayol et al. 2015) introduit le DSL et montre les performances d'un algorithme de type CPF.

4.1.3.2 En tant qu'auteur associé

— Development and Evaluation of Plant Growth Models : Methodology and Implementation in the PYGMALION platform (Cournède et al. 2013) est une revue générale de certaines méthodes et de la méthodologie au sein de la plateforme.
— Sensitivity Analysis for Plant Models with Correlated Parameters : Application to the Characterization of Sunflower Genotypes (Wu et al. 2013) montre l'application des méthodes d'analyse de sensibilité implémentées dans la plateforme sur des modèles de plantes dans le cadre de la caractérisation des génotypes du tournesol.
— Filtrage par noyaux de convolution itératif (Chen et al. 2012) présente la méthode du Convolution Particle Filter existant dans la plateforme.
— A Comparison of a Generic MCMC-based Algorithm for Bayesian Estimation in C++, R and Julia. Application to Plant Growth Modeling (Viaud et al. 2015) montre les différences de performances

existantes entre un code MCMC tournant sur le modèle LNAS et implémenté soit en C++, en Julia ou en R.

— Impact of geometrical traits on light interception in conifers : analysis using an FSPM for Scots pine (Streit et al. 2016). Les méthodes d'analyse et de calibration sont celles de PyGMAlion qui utilise le simulator GroIMP en tant que simulateur externe.

4.2 Perspectives

D'une manière générale :

— d'un point de vue scientifique, les outils développés devront être étendus à des algorithmes de type contrôle optimal et une extension à d'autres domaines d'études comme l'économie, la finance, la médecine ou encore les transports. Il sera d'ailleurs intéressant de voir le pouvoir prédictif des modèles ainsi établis en comparaison avec les algorithmes d'apprentissage automatique.

— d'un point de vue ingénierie, les outils développés devront gagner à la fois sur le terrain de la performance en activant pour certains algorithmes les architectures distribués comme MPI (Forum 1994) ainsi que sur le terrain du stockage en mettant au point des lectures de format standard comme le HDF5 (The HDF Group 1997)

Nous trouverons ci-après des perspectives d'évolutions plus thématiques.

4.2.1 Du point de vue du noyau de modélisation et de simulation

Au niveau de la modélisation et de la simulation, nous avons vu en amont que nous avons restreint notre étude à un seul type de formalisme et que nous n'avions pas intégré le multi-échelle. De plus, même si nous avons regardé la gestion des bruits et de la stochasticité, nous pouvons surement faire mieux, notamment pour le passage aux architectures distribuées.

Dans la suite, nous proposons donc d'améliorer le support des générateurs aléatoires pour les architectures à mémoire distribuée, de recadrer notre modèle informatique par rapport à DEVS (Zeigler 1976) et notamment STDEVS (Castro, Kofman et Wainer 2008), de renforcer la robustesse des concepts autour des modèles et simulateur par la programmation conceptuelle, et enfin de faire en sorte que le simulateur soit de facto un simulateur pour les architectures parallèles à mémoire distribuée a minima.

4.2.1.1 Vers une meilleure gestion de la stochasticité

Le long de cette thèse, nous avons choisi un modèle de gestion des flux paralléles en prenant un même type de générateur mais dont les états initiaux respectifs sont tous différents. Il serait intéressant de revoir ce point en intégrant une bibliothèque dédiée comme SPRNG (Mascagni et Srinivasan 2000) par exemple, ou en recréant en interne un découpage de flux sur la base d'un générateur comme MRG32k3a (L'ecuyer 1999). Pour l'instant, il semble que la communauté des générateurs aléatoire propose un ensemble de bonnes pratiques mais qu'il n'y a pas d'outil majeur qui se distingue (L'Ecuyer, Oreshkin et Simard 2014).

De plus, l'initialisation de ces générateurs pouvant être problématique il faudrait pourvoir l'ensemble de mots-clés dédiés à la stochasticité afin de renforcer la reproducibilité des résultats et de générer quelque soit l'expérience menée une première visualisation des problémes courants que peuvent rencontrer les générateurs.

Dans le cadre de la reproducibilité des travaux, il faudrait ajouter des éléments concernant la stochasticité au langage existant. On pourrait spécifier les choses selon le code suivant :

```
./bin/src --model ./model_list/liblv.so
          --initial-state x0.dat
          --control u.dat
          --observation-function '[("x",[50,60,70,80,90,100]),("y", [15,25,35,45,55,65])]'
          --initial-parameters p0.dat
          --univariate-rule-list '[("a","normal",[0,0.05]),("b","normal",[0,0.3])]'
          --nbr-samples 10000
          --generator-type "mt19937" # or "mtdc", "mrk32k3a"
          --generator-strategy-type "on\_pseudo\_random(5)" #on_clock() #single_genarator(3)
          --output-directory ./
```

Code 4.1 – ajout des commandes generator-type et generator-strategy-type pour tous les algorithmes

4.2.1.2 Quelle place pour notre modèle informatique vis à vis de l'existant ?

Un point important est que nous avons développé notre modèle informatique afin de pouvoir mener nos études sur les algorithmes du flux des bonnes pratiques de modélisation. Si nous avons fait des recherches autour de DEVS (Zeigler 1976), nous n'avons cependant pas trouvé la formulation qui nous permettrait de passer de l'un vers l'autre actuellement. C'est un point qui serait intéressant de regarder car il apparait que des notions se recoupent entre notre modélisation, celle de STDEVS (Castro, Kofman et Wainer 2008) et celle de DTSS (Zeigler, Kim et Praehofer 2000).

4.2.1.3 Support du calcul distribué et du calcul massivement parallèle

Comme nous l'avons décrit dans la section 2.3.3, nous avons introduit le calcul en mémoire partagée au sein de la plateforme autant que cela était possible. En fonction des besoins scientifiques et afin d'explorer plus largement ou plus rapidement les espaces, nous pourrions introduire le calcul distribué au moyen de MPI (Forum 1994) ou bien au moyen du calcul massivement parallèle avec CUDA (Nickolls et al. 2008), ou OpenCL (Stone, Gohara et Shi 2010).

4.2.2 Du point de vue du langage dédié à la modélisation

Le langage dédié développé porte sur le domaine des systèmes dynamiques discrets. La même démarche pourrait être faite afin de proposer des langages pour les systèmes dynamiques à base d'équations différentielles ordinaires, de dérivées partielles, ou encore tout autre type de formalisme.

4.2.2.1 Vers un simulateur-interpréteur orienté DEVS avec support du FMI ?

Dans la section précédente, nous avons avancé l'hypothèse que notre cadre informatique peut être recadré par rapport à celui de DEVS.

Toutefois, à notre connaissance, les outils existants autour de ce formalisme sont soit construits sous la forme de bibliothéques à utiliser avec un langage de programmation général sinon avec un paradigme de programmation visuelle (Quesnel, Duboz et Ramat 2009 ; Nutaro 2010 ; Bergero et Kofman 2011 ; Nutaro 2013). Nous pensons que cette complexité d'écriture des systèmes nuit au développement des modèles et notamment des modèles avec plusieurs formalismes et multi-échelles. Il serait intéressant de voir si nous pouvons créer un simulateur qui contient à la fois les capacités de DEVS pour la gestion de la simulation et les capacités de compilation à la volée de LLVM (Lattner et Adve 2004) pour convertir des modèles écrits à un haut niveau de description.

Par exemple, peut-on coupler un de nos modèles discrets stochastiques et un modèle Modelica par le biais d'un tel simulateur ?

Pour l'instant l'ensemble des éléments mis en œuvre ne permet pas facilement de résoudre ce problème.

Il faudra surement s'inspirer des travaux autour de DEVS (Zeigler et al. 1995) ou (Le Chevalier et Jaeger 2010) afin d'apporter une solution. De plus un standard semble émerger dans la communauté de la modélisation et simulation pour faciliter l'échange de modèles ainsi que la co-simulation : le Functional Mock-up Interface (Blochwitz et al. 2011). Il semble intéressant de voir comment générer, directement depuis le langage de modélisation, des paquets compatibles avec ce standard.

4.2.2.2 Extension du langage dédié pour les types primitifs

Sans rentrer dans la complexité de la précédente suggestion, le langage dédié pourrait être étendu en supportant :

— les vecteurs et matrices comme types primitifs,
— les instructions conditionnelles,
— la vérification des unités au parsing afin d'éviter de lancer des modèles incohérents.

Une extension intéressante serait un support a minima des graphes pour des modèles comme Greenlab (De Reffye et Hu 2003), on pourrait s'inspirer des travaux autour de GroIMP (Smolenova, Kurth et Cournède 2012).

4.2.2.3 Ajout d'autre formalismes, notamment la modélisation guidée par les données

Une autre évolution consiste à permettre d'abstraire certains processus par le biais de l'apprentissage automatique au sein d'un modèle mécaniste. C'est ce qui est proposé dans le papier (Fan et al. 2015). Nous proposons de rajouter à l'ensemble du vocabulaire existant une prise en compte de mots-clés tels que `neural_network` qui permettrait de décrire au sein d'un modèle une prise en charge d'une base d'apprentissage et des variables d'intérêt. Nous présentons dans le code 4.2 une illustration de ce propos.

```
dynamical_model: crop_model
state:
  ...

control:
  ...

parameters:
  ...

noises:
  ...

neural_network climate:
   training_dataset: ["data_1.csv","data_2.csv", "data_3.csv"]
   interest_variable: ["temperature"]
   parameters: ["a", "b"]

equations:
   module_call(first_module)
   module_call(climate)
   module_call(another_module)
```

Code 4.2 – code source DSL avec support réseaux de neurones

4.2.2.4 Génération de code à partir du langage dédié

Une autre activité intéressante est la génération de code à partir du langage dédié. On peut traduire d'un langage source (le DSL) à un langage cible (C++, Python, Java, autre DSL, ...) facilement étant donné que l'on possède l'arbre de syntaxe.

On peut ainsi imaginer des outils du type : dsl2another_dsl, dsl2cpp, dsl2python, ...

À l'inverse on peut ainsi imaginer un code "parsant" le code C++ d'un modèle de la plateforme pour ensuite générer directement du code CUDA (Nickolls et al. 2008) ou OpenCL (Stone, Gohara et Shi 2010) en exploitant les possibilités de certains compilateurs comme Clang (*Clang*) qui permet d'accéder à l'arbre de syntaxe C++.

4.2.3 Du point de vue des bonnes pratiques de modélisation

4.2.3.1 Ajout de nouveaux algorithmes et fonctionnalités

Le développement de la plateforme a nécessité de nombreuses itérations et a vu le développement de nombreux algorithmes. Aujourd'hui un effort est fait afin de réinjecter l'ensemble sous une forme unique. Nous présentons dans le tableau 4.1 l'état actuel des implémentations.

Figure 4.1 – état actuel des implémentations. en vert (implémenté dans la dernière version) ; en orange (implémenté dans une version précédente) ; en rouge (pas encore implémenté ou implémenté en dehors du système)

4.2.3.2 Langage dédié du flux de travail

Nous avons pour l'instant créé un langage dédié autour des modèles dynamiques discrets déterministes ou stochastiques. Nous pourrions aussi fournir une extension à ce langage afin d'abstraire les algorithmes, leur configuration ainsi que leur exécution sur un environnment de calcul dédié afin de faciliter la manipulation de l'ensemble par la communauté mais aussi de pouvoir facilement stocker et sauvegarder les expériences qui ont été menées.

4.2.3.3 Standardisation, passeport, domaine de validité et cadre expérimental

Les études introduites en chapitre 3.2 sont une version simplifiée d'une vue plus complexe quant à la validation des modèles et à leur visualisation en tant que solution pour un problème donné.

Nous proposons de mettre au point un standard afin d'unifier et d'automatiser les analyses de modèles. Ce standard pourrait être connu sous la notion de "passeport de modèle".

Ce "passeport" pourrait être un simple fichier contenant le nom et la version du modèle avec en premier lieu des analyse ne nécessitant pas de données particulières et permettant de comprendre les paramètres importants et son fonctionnement général. C'est le cas des analyses de sensibilité par exemple. Puis dans un second temps nous retrouverions les caractéristiques des données expérimentales qui ont été utilisées afin de faire de l'estimation paramétrique et de la caractérisation des incertitudes.

Le but est qu'un échange de modèle ne consiste pas seulement à un échange d'équations mais d'une implémentation accompagnée d'un "passeport" permettant une première lecture rapide de son comportement et des domaines de validité associés. Cela s'inspire des travaux autour des notions de cadre expérimental (Zeigler, Kim et Praehofer 2000 ; Cheon 2007 ; Foures, Albert et Nketsa 2013). Toutefois, le cadre expérimental n'est qu'une partie de ce "passeport" car nous souhaitons mettre une emphase sur la notion de "domaine de validité" qui correspond aux variations acceptables par le jeu de paramètres du modèle. Par conséquent, il ne peut pas être confondu avec le cadre expérimental qui peut être interchangeable quel que soit le modèle ou système étudié (Zeigler, Kim et Praehofer 2000).

On peut aussi s'inspirer du standard SBML (Hucka et al. 2010) en essayant de formuler la chose au moyen du langage XML afin de le prévoir comme une extension à ce format. En effet, ce dernier permet l'export du modèle mais pas du cadre expérimental ou du domaine de validité.

4.2.4 Du point de vue de la communauté

D'une manière générale, la démarche utilisée et le langage créé nous permettraient de contribuer à la reproducibilité des modèles mécanistes et expériences associées sur différentes plateformes et environnement.

4.2.4.1 Création d'une ontologie de l'agronomie quantitative

Dans les chapitres précédents, nous avons construit à partir de la représentation d'états un vocabulaire associé à une sémantique. Ces éléments ont été implémentés dans un langage dédié embarqué. Ce langage embarqué exploite la syntaxe du C++, son langage hôte. Nous avons aussi présenté le flux de travail général autour d'un modèle et quels sont les algorithmes principaux que nous exploitons afin de développer et valider un modèle.

Ces algorithmes ont été standardisés dans leurs entrées et sorties afin d'exploiter le langage créé et ils sont manipulables ensemble afin de créer des benchmarks automatiques vis à vis d'un problème donné afin de se faire une idée plus rapide et précise du comportement du modèle vis à vis de données expérimentales.

Nous proposons aussi la constitution d'une ontologie pour l'ensemble de la communauté dont les buts seraient de :

- descendre le niveau de complexité d'écriture d'un modèle de système dynamique discret stochastique par le biais d'un langage dédié complétement autonome et ne nécessitant pas de compilation, favorisant ainsi l'implémentation de nouveaux modèles et le retour d'expériences sur leur évaluation, par exemple sur le test de nouvelles hypothèses sur les processus de fonctionnement.
- représenter la connaissance du domaine de la modélisation des systèmes agronomiques à travers une ontologie encodée sous format OWL (W3C 2009)
- augmenter le périmètre de cette connaissance grâce à des annotations sémantiques guidées par une ontologie dans le code source des modèles mathématiques permettant de lier les concepts des

modèles aux concepts des systèmes par des experts, i.e. un modélisateur et un agronome doivent pouvoir communiquer plus facilement quant à des questions du type ”Qu’est ce que représente ce paramètre ?” ou ”Comment a été prise en compte la sénéscence dans cette structure de modèle ?”.

Dans cette section nous ferons une description des connaissances liées au domaine de l’agronomie quantitative, puis nous montrerons quelques requêtes possibles au sein de l’ontologie définie.

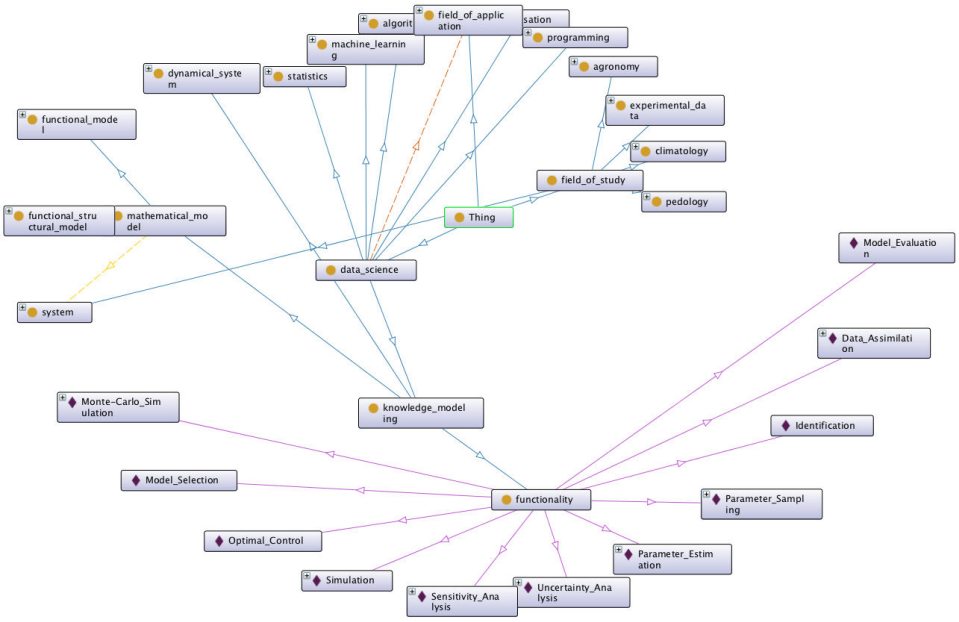


Figure 4.2 – visualisation d’une partie de l’ontologie

Par l’essor des technologies issues de l’apprentissage automatique et l’émergence de certains modèles de cultures ou de plantes, le secteur de l’agronomie se retrouve à jongler avec des notions telles que réseaux de neurones, machines à vecteurs de support, système dynamique, inférence bayésienne, contrôle optimal ou modèle de Markov caché en plus de ses propres termes comme photosynthèse, stress hydrique ou évapotranspiration. De plus, la complexité des systèmes biologiques étudiés amène souvent à dépasser le cadre du système lui-même car il est en interaction avec son environnement. Par exemple, en plus des termes agronomiques à proprement parler nous trouverons des termes issus de la pédologie ou encore de la climatologie. Ainsi nous définirons un premier élément dans notre ontologie que sont les **champs d’études** de l’agronomie quantitative.

De plus nous avons deux grands types d’analyses quantitatives qui peuvent être conduites : soit une analyse par les données uniquement, c’est le cadre de l’apprentissage automatique par exemple, soit une analyse par les modèles mécanistes calibrés avec les jeux de données. À partir de ces outils, les agronomes peuvent réfléchir autour de notions telles que la prévision de rendement, l’amélioration des parcours techniques et plus particulièrement l’optimisation des pratiques culturales et de la sélection variétale. Nous définissons ainsi un deuxième élement dans notre ontologie que sont les **champs d’application** de l’agronomie quantitative.

Dans le cadre de l’agronomie quantitative par les modèles, la tâche consiste à calibrer des modèles en sachant qu’il y a plusieurs phases d’analyse et de refactorisation. Ces phases utilisent un ensemble d’algorithmes permettant d’effectuer des boucles de rétroaction sur la structure du modèle afin de l’améliorer. Si les deux premiers éléments de notre ontologie sont relatifs aux entrées et sorties de l’activité de quantification, à savoir les champs d’études (entrées) et les champs d’applications (sorties), le troisième élément porte sur la quantification en elle-même et par conséquent les algorithmes. Ainsi nous ajoutons un troisième élement qui est la **science des données** et plus particulièrement les approches par apprentissage automatique ou par modélisation mécaniste.

À partir de ces trois notions, nous pouvons classer un certain nombre de concepts avant de les relier entre eux afin de créer un graphe des connaissances.

La figure 4.2 permet de prendre une première mesure de la complexité définie en affichant certains sous-niveaux de la hiérarchie générale. Notamment, nous avons mis en avant les sous-niveaux du champ d'étude et les sous-niveaux de la modélisation mécaniste (dénommée "knowledge-based modeling").

Exemples de requêtes Les requêtes soumises se doivent de répondre à des questions types d'utilisateurs qui peuvent être *"Quels sont les paramètres du modèle A qui ont à voir avec la production de masse sèche ?"*, ou encore *"Quels algorithmes existent pour faire une analyse de sensibilité ?"* Par le biais de l'ontologie et d'un langage de requête comme SPARQL (Pérez, Arenas et Gutierrez 2009) nous pouvons formuler ce type de requête. Des exemples de requêtes sont donnés dans le code 4.3

```
#quels sont les modèles qui modélisent un champ
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?model
WHERE { ?model qa:represents qa:crop }

#quels sont les modèles qui modélisent une plante
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?model
WHERE { ?model qa:represents qa:plant }

#quels sont les algorithmes disponibles pour une estimation paramétrique
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?algorithm
       WHERE { qa:Parameter_Estimation qa:uses ?algorithm}

#quels sont les paramètres de la photosynthèse dans le modèle LNAS
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?parameters
       WHERE { qa:lnas qa:has_parameters ?parameters,
              ?parameters qa:used_in qa:photosynthesis
            }

#quels sont les paramètres de la photosynthèse dans le modèle Greenlab
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?parameters
       WHERE { qa:greenlab qa:has_parameters ?parameters,
              ?parameters qa:used_in qa:photosynthesis
            }

#quel est le nom du module dans LNAS qui représente la photosynthèse
PREFIX qa: <http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl#>
SELECT ?module
       WHERE { qa:lnas qa:has_module ?module,
              ?module qa:represents qa:photosynthesis
            }
```

Code 4.3 – exemples de requêtes SPARQL.

Il faudra sûrement pourvoir ce type d'outil d'une interface graphique utilisateur afin de formuler plus facilement les requêtes et ainsi explorer toute la base de connaissances.

Nous avons donc une base de connaissances décrivant l'état actuel des connaissances. Toutefois, à

chaque mise à jour d'un modèle ou à la création d'un nouveau, nous voulons ajouter cette nouvelle connaissance. Ce qui nous amène aux questions suivantes : comment augmenter la taille de la base de connaissances, ainsi que garantir sa mise à jour vis à vis des nouvelles connaissances ? Nous proposons dans la partie qui suit une génération automatique à partir du langage dédié décrit précédemment.

Génération automatique d'ontologies à partir d'un langage dédié Dans cette section nous nous efforcerons de montrer comment mettre en place le passage de l'ontologie de l'agronomie quantitative vers l'ontologie d'un modèle. Nous proposons le couplage entre les différentes ontologies par l'utilisation d'un langage de description dédié aux systèmes dynamiques discrets stochastiques. Il s'agit d'intégrer les deux éléments (langage dédié, ontologie) ensemble pour permettre à l'utilisateur d'affiner sa pensée en termes plus précis dans le domaine concerné.

Nous avons mis en place un langage dédié mélangeant à la fois une syntaxe proche de Modelica (Elmqvist et Ab 1997) et à la fois des éléments de langages propres aux langages dédiés probabilistes (par exemple le langage BLOG (Getoor et Taskar 2007). Nous souhaitons lui ajouter des capacités d'annotation pour l'exploitation d'une ontologie.

Voici une présentation rapide du langage :

```
1  ontology: http://digiplante.mas.ecp.fr/ontology/quantitative_agronomy.owl qa
2  dynamical_model: lnas
3    ...
4
5    module thermal_time qa:represents qa:thermal_time
6      x[n+1].thermal_time = x[n].thermal_time + max(0.0, u[n].Temp - p.T_b)
7
8    module deterministic_production:
9      x[n+1].deterministic_produced_biomass = u[n].PARISH * p.rue * (1 - exp(-p.k_B *
   ↪   x[n].dry_green_leaf_mass / p.e))
10
11   module stochastic_production:
12     x[n+1].stochastic_produced_biomass = max(0.0, x[n+1].deterministic_produced_biomass * (1 +
   ↪   e.sigma_production()))
13
14   module production qa:represents qa:dry_biomass_production:
15     module_call(deterministic_production)
16     module_call(stochastic_production)
17
18     ...
```

Figure 4.3 – exemple du langage dédié avec annotations destinées à compléter l'ontologie.

Nous pouvons remarquer dans le code 4.3 l'ajout d'une instruction `"ontology"` à la ligne 1 et d'une instruction `"qa:represents"` sur la définition des modules relatifs au temps thermique et à la production de biomasse. Cette dernière instruction permet de renseigner quels sont les éléments qui rentrent en jeu dans la composition du module. En effet, ayant accès à l'arbre de syntaxe au moyen du "parser", nous pouvons par conséquent renseigner les paramètres, les variables d'états et externes qui rentrent en compte dans la composition du processus agronomique relatif à la production de biomasse sèche (dénoté ici `"qa:dry_biomass_production"`).

De fait, en initialisant une ontologie de l'agronomique quantitative en renseignant différents éléments relatifs aux champs d'études et d'applications, nous pouvons au moyen de ce langage augmenter la base de connaissance relatives à ce champ en injectant directement dans l'ontologie les modèles nouvellement créés et permettre leur exploration non plus par la dimension purement mathématique mais par une analyse systèmique des éléments composants le modèle.

Afin d'être compatible avec les autres modèles et simulateurs, il faudrait mettre au point un logiciel de parsing avec des instructions précises pour définir des modules au sein de code C++ ou Java par exemple et ainsi permettre à tout modélisateur, quelque soit sa plateforme, de contribuer à cette ontologie.

4.2.4.2 Plateformes de modélisation et modélisation continue

Avec l'exploitation d'une plateforme centrale, on peut extrapoler les comportements des modélisateurs face à une situation et ainsi soit se concentrer sur la performance de certains algorithmes au regard de leur forte charge sur les ressources de calcul, ou bien améliorer la formation afin que les modélisateurs utilisent certains algorithmes ou certaines parties de façon plus optimale.

Nous proposons dans la figure 4.4 une visualisation d'un tel type de plateforme qui consiste à présenter le vocabulaire 2.1 sous la forme d'une interface graphique et enregistrant en base de données l'ensemble des configurations utilisées pour lancer les algorithmes ainsi que les ressources nécessaires pour les faire fonctionner. (prototype "Coconut")

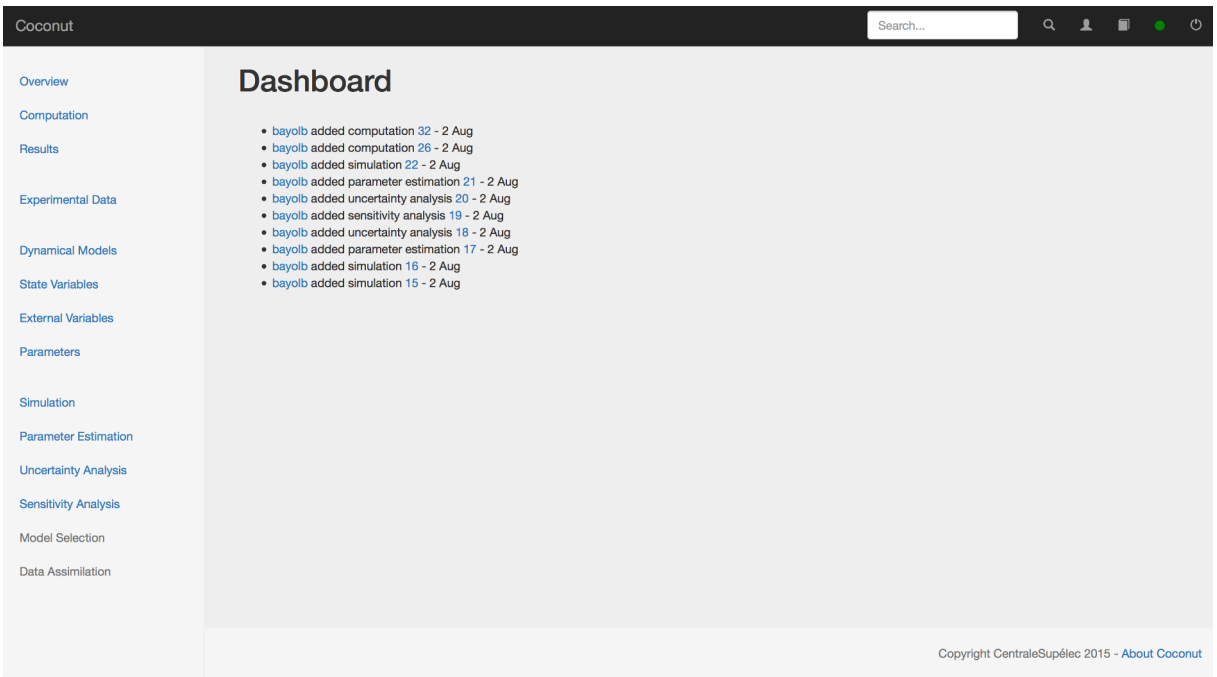


Figure 4.4 – visualisation d'une plateforme logicielle d'aide à la modélisation mécaniste

On retrouve sur la gauche de cette figure une colonne dédiée aux activités de modélisation et plus particulièrement aux fonctionnalités de type "analyse de sensibilité" ou "estimation paramétrique". Au centre nous avons les résultats ou la paramétrisation des algorithmes. En haut nous avons les accès au profil du modélisateur et à l'aide générale. Tous les calculs sont envoyés sur une machine distante.

Cette plateforme peut-être étendue au moyen d'API RESTFUL (Richardson et Ruby 2007) afin de créer d'autres plateformes plus orientées métiers. C'est le cas de la figure 4.5 où nous présentons une plateforme (prototype "Gaia") plus centrée sur les problématiques agronomiques mais s'appuyant sur la première plateforme définie en 4.4. C'est à l'intérieur de ce type de plateforme que nous pouvons mettre en place l'exploitation de l'ontologie définie précédemment afin de réduire le bruit de communication entre les différents acteurs du secteur de la modélisation de la croissance des plantes.

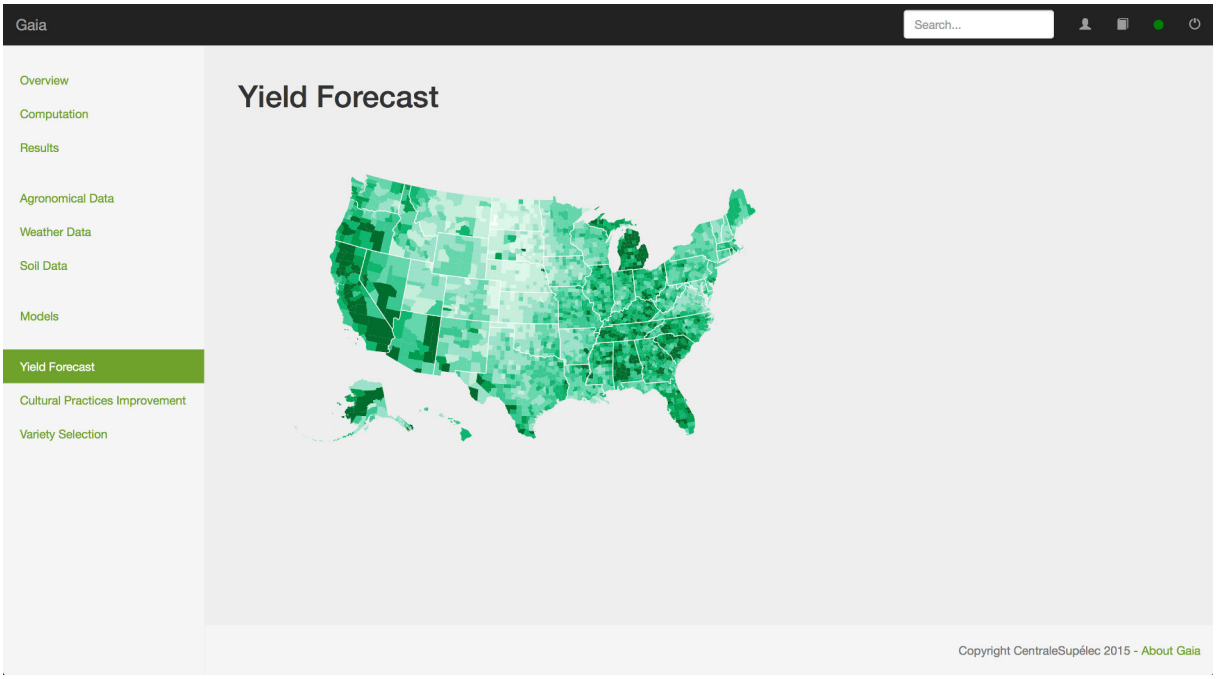


Figure 4.5 – visualisation d'une plateforme logicielle d'aide à la décision basée sur les modèles mécanistes dans le domaine de la biologie végétale (données imaginaires).

De plus, le secteur de l'ingénierie logicielle a fait évoluer les processus ces dernières années grâce à l'évolution des outils accompagnant les équipes de développement, leur permettant de passer de la phase de prototypage à la phase de mise en production plus facilement. On pourra citer notamment le mouvement de l'intégration continue et du déploiement continu qui consiste à tester à chaque nouvelle version écrite (mais pas forcément diffusée) dès que possible et à la diffuser automatiquement auprès des équipes de tests. En reprenant cette idée, nous proposons de combiner l'idée précédente de passeport des modèles afin de tester ceux-ci automatiquement à chaque nouvelle itération du modèle. Ceci doit permettre d'aller vers une modélisation continue afin de voir automatiquement quels sont les impacts de chaque implémentation du modèle. Cette attitude consiste à tester à chaque itération le modèle en fonction d'un jeu de données standardisé vis à vis d'une problématique et ainsi renforcer la confiance en l'implémentation du modèle.

Le code 4.4 donne un aperçu d'un benchmark rédigé sous format JSON (ECMA 2013) et envoyé à un logiciel distant pour effectuer les calculs à chaque fois qu'une nouvelle version du modèle est soumise.

```
{
  "benchmark" : {
    "model" : "lnas",
    "program-list" : [
      {
        "program" : "generalized-least-square",
        "configuration" :
        [
          #we have xxx-id that refers to an entry into a database
          #here we use the parameters that have been saved in the id 45
          "parameters-id" : "45",
          "target-id" : "453"
        ]
      },
      {
        "program" : "sobol",
        "configuration" :
        [
          "parameters-id" : "45",
          "sampling-rule-list" : [("rue", "normal", [0.0, 0.05]),("s_init", "normal", [0.5,
↪  0.1])]
          "observation-function" : [("yield", [10,20,30,40,100,120,160])]
        ]
      },
    ]
  }
}
```

Code 4.4 – fichier de configuration pour benchmark automatique sous JSON

Cette modélisation continue s'intégre au développement d'une nouvelle communauté de modélisateurs en fournissant des automates de diffusion d'informations fiables sur l'évolution des modèles dans le temps. Nous pensons que cela peut permettre une accélération dans la diffusion des modèles complexes.

4.3 Conclusion

Aprés avoir décomposé notre cadre mathématique, à la fois sous l'angle des modèles et celui des algorithmes, nous avons montré quels sont les éléments de syntaxe et de sémantique permettant la création d'outils et d'un langage dédié pour les systèmes dynamiques discrets stochastiques. Ce langage a été utilisé afin de construire une plateforme basé sur les bonnes pratiques de modélisation afin de pouvoir passer plus facilement, rapidement et sûrement de l'état de l'hypothèse à l'état de modèle validé et vérifié. À ce jour une trentaine de personnes ont manipulé directement les concepts relatifs à cette formalisation.

Concrètement, en sortie de cette thèse, nous avons mis au point un cadre conceptuel (cube et langage) permettant de construire la plupart des fonctionnalités mathématiques des bonnes pratiques de modélisation. Nous avons mis en application ce cadre conceptuel pour la création d'outils génériques capables d'analyser à la fois nos modèles dans notre formalisme mais aussi tout simulateur externe. Le langage reste le même dans tous les cas et, parce que construit sur la base du formalisme mathématique, est minimal en terme de vocabulaire et concepts. Ces outils sont capables d'utiliser les ressources multicœurs des ordinateurs récents et un portage vers les architectures distribuées est en cours.

De façon plus abstraite, nous pouvons voir l'ensemble du travail effectué comme la création d'un langage dit "véhiculaire" pour l'ensemble des langages "vernaculaires" qui émergent des modèles, i.e. notre langage permet la communication et l'échange autour des terminologies issues des modèles et algorithmes.

Nous avons abouti au développement d'un écosystème encore jeune mais qui présente de bons atouts par le formalisme développé, sa base de code et sa modularité en termes d'outils.

Les perspectives envisagées permettent de renforcer l'idée que nous ne sommes pas dans une activité favorisant un modèle plutôt qu'un autre mais une activité qui est celle de la modélisation ou le retour d'informations et sa diffusion au sein d'une communauté est primordial. Nous espérons que les travaux effectués pourront donner lieu à la création d'un standard d'échange au sein de la communauté afin non seulement de pouvoir comparer les modèles mais aussi de favoriser la reproducibilité nécessaire à la bonne démarche scientifique en général.

Annexe A

Langages dédiés embarqués en Haskell et Julia

Si nous avons présenté notre noyau en c++, il est à noter que d'autres versions de ce noyau ont été développées, notamment en Haskell et en Julia.

A.1 Haskell

Haskell est un langage fonctionnel pur, par conséquent il refuse dans le principe la mutation d'état et par extension inclut la transparence référentielle (Søndergaard et Sestoft 1990). La volonté d'utiliser ce langage a été de se forcer à travailler dans un cadre bien défini et dont chaque élément peut être expliqué par un noyau mathématique fiable. En effet tout élément dans Haskell dérive de la théorie des catégories et plus particulièrement du système F. Ainsi toute structure de données ou fonction possède une version "formalisable".
Un code a été fait avec Haskell afin de confronter les éléments mis en jeu vis à vis d'une approche purement fonctionnelle. Le code est très succinct mais permet de bien comprendre les éléments mis en jeu dans l'ensemble du système.

A.2 Extrait de code Haskell

Dans les deux extraits de code suivants il faut regarder l'écriture des structures de données de la plateforme et ce que cela implique pour la fonction de simulation. La version présentée ici est déterministe.

```
{
  "benchmark" : {
    "model" : "lnas",
    "program-list" : [
      {
        "program" : "generalized-least-square",
        "configuration" :
        [
          #we have xxx-id that refers to an entry into a database
          #here we use the parameters that have been saved in the id 45
          "parameters-id" : "45",
          "target-id" : "453"
        ]
      },
      {
        "program" : "sobol",
        "configuration" :
        [
          "parameters-id" : "45",
          "sampling-rule-list" : [("rue", "normal", [0.0, 0.05]),("s_init", "normal", [0.5,
↪  0.1])]
          "observation-function" : [("yield", [10,20,30,40,100,120,160])]
        ]
      },
    ]
  }
}
```

Code A.1 – code source Haskell des types pygmalion

Maintenant que nous avons défini les types, nous pouvons créer la fonction de simulation en exploitant un paradigme de programmation fonctionnelle :

```
{
  "benchmark" : {
    "model" : "lnas",
    "program-list" : [
      {
        "program" : "generalized-least-square",
        "configuration" :
        [
          #we have xxx-id that refers to an entry into a database
          #here we use the parameters that have been saved in the id 45
          "parameters-id" : "45",
          "target-id" : "453"
        ]
      },
      {
        "program" : "sobol",
        "configuration" :
        [
          "parameters-id" : "45",
          "sampling-rule-list" : [("rue", "normal", [0.0, 0.05]),("s_init", "normal", [0.5,
↪  0.1])]
          "observation-function" : [("yield", [10,20,30,40,100,120,160])]
        ]
      },
    ]
  }
}
```

Code A.2 – code source Haskell de la fonction simulate déterministe

A.3 Julia

Julia (julialang.org) est un langage de programmation orienté calcul scientifique développé avec C et Scheme au dessus de la bibliothèque LLVM. Il utilise notamment le mécanisme de compilation à la volée afin de générer le code et l'optimiser à l'exécution du programme.

Il a l'avantage d'être rapide à l'exécution même s'il possède une phase d'initialisation plutôt longue lors de l'exécution d'un programme étant donné qu'il doit recompiler à la volée ses propres fonctions.

Le noyau et un certain nombre d'algorithmes ont été redéveloppés autour des notions présentées ci-dessus (Viaud et al. 2015).

Annexe B

Big Crush pour Mersenne Twister

Cette annexe montre les résultats issus de différents tests ”Big Crush” conduits sur le générateur Mersenne Twister en le découpant en différentes sous-suites. Les tests sont les mêmes mais sur des flux différents. Lorsque l'on utilise un flux pour les différents bruits au sein d'une même simulation on va trouver des incohérences dans les sous-suites extraites comme le montre les 3 tests suivants. En effet $lfsr\_e1$, $lfsr\_e5$, et $lfsr\_e7$ qui sont respectivement les sous-suites 1,5 et 7 sur 8 ont montré des irrégularités sur les sorties par TestU01. Le fait de ne pas retrouver le même comportement entre la suite d'un générateur MT199337 et les sous-suites extraites (irrégularité dans les tests, non-conformité au caractère d'échec standard du Mersonne Twister) nous a fait préférer une approche avec un bruit par générateur.

B.1 Test 1

```
========= Summary results of BigCrush =========

 Version:          TestU01 1.2.3
 Generator:        lfsr_e1
 Number of statistics:  160
 Total CPU time:   20:20:59.11
 The following tests gave p-values outside [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

       Test                          p-value
 ----------------------------------------------
 61  WeightDistrib, r = 28           0.9995
 ----------------------------------------------
 All other tests were passed
```

B.2 Test 2

```
========= Summary results of BigCrush =========

 Version:          TestU01 1.2.3
 Generator:        lfsr_e5
 Number of statistics:  160
 Total CPU time:   23:05:41.30
```

```
 The following tests gave p-values outside [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

       Test                          p-value
 ------------------------------------------------
 56  SampleCorr, k = 2               0.9996
 ------------------------------------------------
 All other tests were passed
```

B.3 Test 3

```
========= Summary results of BigCrush =========

 Version:          TestU01 1.2.3
 Generator:        lfsr_e7
 Number of statistics:  160
 Total CPU time:   20:14:39.08
 The following tests gave p-values outside [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

       Test                          p-value
 ------------------------------------------------
 32  CouponCollector, r = 20         0.9996
 74  RandomWalk1 C (L=50, r=0)       0.9997
 ------------------------------------------------
 All other tests were passed
```

Annexe C

Examples de programmes pour les études

C.1 Programme pour LNAS et SRC

```
./bin/src --model ./model_list/liblnas.so \
          --control "./u.txt" \
          --initial-parameters "./p.txt" \
          --sampling-rule-list '[("rue","uniform",[3,4]), \
            ("e","uniform",[50,70]), \
            ("s_init","uniform",[0.7,1.0]), \
            ("s_end","uniform",[0.0,0.3]), \
            ("mu_alloc","uniform",[400,800]), \
            ("s_alloc","uniform",[200,2000]), \
            ("mu_sen","uniform",[2000,3000]), \
            ("s_sen","uniform",[500,6000])]' \
          --observation-function '[("yield",[1:1:160]), \
            ("dry_green_leaf_mass",[1:1:160])]' \
          --nbr-samples 10000 \
          --output-directory "./"

#postprocessing
./bin/csv2obs --input-filename src-yield --output-filename src-yield
./bin/obs2gle --input-filename src-yield --output-dir ./

./bin/csv2obs --input-filename src-dry_green_leaf_mass --output-filename
↪  src-dry_green_leaf_mass
./bin/obs2gle --input-filename src-dry_green_leaf_mass --output-dir ./

#call to gle graphics
gle -vb 0 -landscape *.gle
```

C.2 Programme pour LNAS et SOBOL

```
./bin/sobol --model ./model_list/liblnas.so \
            --control "./u.txt" \
            --parameters "./p.txt" \
            --sampling-rule-list '[("rue","uniform",[3,4]), \
              ("e","uniform",[50,70]), \
              ("s_init","uniform",[0.5,0.8]), \
              ("s_end","uniform",[0.0,0.3]), \
              ("mu_alloc","uniform",[400,800]), \
              ("s_alloc","uniform",[200,2000]), \
              ("mu_sen","uniform",[2000,3000]), \
              ("s_sen","uniform",[500,6000])]' \
            --observation-function '[("dry_green_leaf_mass",[1:1:160]),("yield",[1:1:160])]' \
            --nbr-samples 10000 \
            --group-list [] \
            --output-directory "./"

#postprocessing
./bin/csv2obs --input-filename sobol-yield --output-filename sobol-yield.obs
./bin/obs2gle --input-filename sobol-yield.obs --output-dir ./

./bin/csv2obs --input-filename sobol-dry_green_leaf_mass --output-filename
↪  sobol-dry_green_leaf_mass.obs
./bin/obs2gle --input-filename sobol-dry_green_leaf_mass.obs --output-dir ./

#gle graphics
gle -vb 0 -landscape *.gle
```

C.3 Programme pour LNAS et SOBOL-GLS

```
./bin/sobol-gls --model ./model_list/liblnas.so \
                --control "./u.txt" \
                --parameters "./p_det.txt" \
                --sampling-rule-list '[("rue","uniform",[3,4]), \
                  ("e","uniform",[50,70]), \
                  ("s_init","uniform",[0.5,0.8]), \
                  ("s_end","uniform",[0.0,0.3]), \
                  ("mu_alloc","uniform",[400,800]), \
                  ("s_alloc","uniform",[200,2000]), \
                  ("mu_sen","uniform",[2000,3000]), \
                  ("s_sen","uniform",[500,6000])]' \
                --estimation-data "./ref2.txt" \
                --nbr-samples 10000 \
                --group-list [] \
                --output-directory "./"

#call gle graphics
gle -vb 0 -landscape sobol_gls.gle
```

C.4 Programme pour LNAS et AITKEN

```
#scheme of parameters to be analyzed (could be done automatically after sobol-gls)
parameters[0]='["s_init"]'
parameters[1]='["s_init","s_end"]'
parameters[2]='["s_init","s_end","rue"]'
parameters[3]='["s_init","s_end","rue","e"]'
parameters[4]='["s_init","s_end","rue","e","mu_alloc"]'

#main loop
for i in {0..4}
  do
    mkdir -p $i
    echo "${parameters[$i]}"
    ./bin/aitken --model ./model_list/liblnas.so \
                 --initial-parameters "./p.txt" \
                 --control "./u.txt" \
                 --target "./ref2.txt" \
                 --optimization-function "gauss-newton" \
                 --weight-matrix "variance-per-observation-model" \
                 --selected-parameters "${parameters[$i]}" \
                 --output-directory "./$i/"

    ./bin/simulate --model ./model_list/liblnas.so \
                   --parameters "./$i/p_chap.txt" \
                   --control "./u.txt" \
                   --observation-function
 ↪  '[("dry_green_leaf_mass",[1:1:160]),("yield",[1:1:160])]' \
                   --output-filename "./$i/y_sim.csv"

   #postprocessing for generating some graphics
   ./bin/csv2obs --input-filename "./$i/y_sim.csv" --output-filename "./$i/y_sim.obs"
   ./bin/csv2obs --input-filename "./ref2.txt" --output-filename "./$i/ref.obs"
   ./bin/simvsexp2gle --input-filename-exp "./$i/ref.obs" --input-filename-sim "$i/y_sim.obs"
 ↪  --output-dir "$i/"
cd $i
#call to gle graphics
gle -vb 0 -landscape *.gle
cd ..
```

C.5 Programme pour LNAS, RMSEP et EF

```
./bin/prediction --model ./model_list/liblnas.so \
                 --control "./u.txt" \
                 --parameters "./p_chap.txt" \
                 --target "./y_exp_2008.csv" \
                 --output-directory "./"
```

Annexe D

Format des entrées-sorties

D.1 Fichier DAT

```
density=1.19e+1
rue=3.56e+0
e=6.00e+1
k_b=7.00e-1
thermal_time_sen=6.44e+2
s_init=6.25e-1
s_end=1.50e-1
mu_alloc=5.50e+2
s_alloc=3.00e+2
mu_sen=2.40e+3
s_sen=4.52e+3
weight_seed=3.00e-3
temperature_base=0.00e+0
thermal_time_init=2.42e+2
mean_production=0.00e+0
std_dev_production=2.00e-2
mean_allocation=0.00e+0
std_dev_allocation=2.00e-2
mean_dry_green_leaf_mass=0.00e+0
std_dev_dry_green_leaf_mass=1.50e-1
mean_yield=0.00e+0
std_dev_yield=1.50e-1
```

D.2 Fichier DAT version CSV

```
density,1.19e+1
rue,3.56e+0
e,6.00e+1
k_b,7.00e-1
thermal_time_sen,6.44e+2
s_init,6.25e-1
s_end,1.50e-1
mu_alloc,5.50e+2
s_alloc,3.00e+2
```

```
mu_sen,2.40e+3
s_sen,4.52e+3
weight_seed,3.00e-3
temperature_base,0.00e+0
thermal_time_init,2.42e+2
mean_production,0.00e+0
std_dev_production,2.00e-2
mean_allocation,0.00e+0
std_dev_allocation,2.00e-2
mean_dry_green_leaf_mass,0.00e+0
std_dev_dry_green_leaf_mass,1.50e-1
mean_yield,0.00e+0
std_dev_yield,1.50e-1
```

D.3 Fichier OBS

```
context: lnas_obs
------------
------------
observation_model: dry_green_leaf_mass
t dry_green_leaf_mass
------------
54 85.25325752
68 372.9242426
76 447.6030787
83 440.8144718
90 620.4440797
98 523.7814327
104 541.3559076
110 620.2181499
118 627.5066991
125 757.6166454
132 760.4626555
139 598.3444817
145 670.674038
160 628.3938506
------------
------------
observation_model: dry_senescent_leaf_mass
------------
t dry_senescent_leaf_mass
------------
54 0
68 4.87910448
76 9.80904528
83 52.4205216
90 52.4205216
98 81.29909808
104 111.519517
110 146.213532
```

```
118 182.774304
125 216.4811486
132 255.2680786
139 300.5183146
145 300.5183146
160 397.7918395
------------
------------
observation_model: yield
------------
t yield
------------
54 23.12759999
68 199.7962467
76 302.3477421
83 409.1974238
90 709.229525
98 768.099376
104 863.8542682
110 1232.475372
118 1498.826875
125 1770.169614
132 1878.170841
139 1913.743693
145 2118.387183
160 2274.717358
```

D.4 Fichier OBS version CSV

```
0,dry_green_leaf_mass,54,dry_green_leaf_mass,85.2533
14,yield,54,yield,23.1276
1,dry_green_leaf_mass,68,dry_green_leaf_mass,372.924
15,yield,68,yield,199.796
2,dry_green_leaf_mass,76,dry_green_leaf_mass,447.603
16,yield,76,yield,302.348
3,dry_green_leaf_mass,83,dry_green_leaf_mass,440.814
17,yield,83,yield,409.197
4,dry_green_leaf_mass,90,dry_green_leaf_mass,620.444
18,yield,90,yield,709.23
5,dry_green_leaf_mass,98,dry_green_leaf_mass,523.781
19,yield,98,yield,768.099
6,dry_green_leaf_mass,104,dry_green_leaf_mass,541.356
20,yield,104,yield,863.854
7,dry_green_leaf_mass,110,dry_green_leaf_mass,620.218
21,yield,110,yield,1232.48
8,dry_green_leaf_mass,118,dry_green_leaf_mass,627.507
22,yield,118,yield,1498.83
9,dry_green_leaf_mass,125,dry_green_leaf_mass,757.617
23,yield,125,yield,1770.17
10,dry_green_leaf_mass,132,dry_green_leaf_mass,760.463
```

```
24,yield,132,yield,1878.17
11,dry_green_leaf_mass,139,dry_green_leaf_mass,598.344
25,yield,139,yield,1913.74
12,dry_green_leaf_mass,145,dry_green_leaf_mass,670.674
26,yield,145,yield,2118.39
13,dry_green_leaf_mass,160,dry_green_leaf_mass,628.394
27,yield,160,yield,2274.72
```

Annexe E

C++11

Nous avons choisi de présenter une version du noyau en c++ dans le standard 11. Nous introduirons ici certains éléments syntaxiques de ce langage afin de faciliter la lecture des codes exemples.

- Passage par valeur, pointeur et référence. Ici nous utiliserons préférentiellement le passage par référence noté &.
- Inférence de type. La version 11 de la norme c++ introduit l'inférence de type automatique au moyen du mot-clé auto ainsi auto y = f(x) déduira automatiquement le type de y selon la signature de la fonction f.
- static est un mot-clé permettant de dire entre autre qu'une fonction membre ne peut accéder qu'à des éléments membres static eux aussi. Je l'ai introduit afin de prévenir un bug consistant à stocker des éléments dans des membres privés de la mémoire et qui par conséquent permettait de créer une fuite mémoire dans les modèles qui exploitent l'aspect orienté objet du langage.
- l'initialisation de toute variable peut se faire au moyen de l'opérateur {} afin de donner une syntaxe unique à l'ensemble du langage. Ainsi std ::vector v = {1,2,3,4} créera un vecteur de taille 4 avec les valeurs 1,2,3,4 pour v[0],v[1],v[2] et v[3].

Annexe F

Modèle Lotka-Volterra

Le modèle Lotka-Volterra est un modèle mettant en jeu des proies, x, et des prédateurs, y.
Les paramètres représentent :

- le taux de reproduction d'une proie, $alpha$;
- le taux de mortalité d'un prédateur par proie, $beta$;
- le taux de mortalité des prédateurs, $gamma$;
- le taux de reproduction des prédateurs par proie, $delta$;

$$\begin{cases} \frac{dx}{dt} &= \alpha \times x - \beta \times x \times y \\ \frac{dy}{dt} &= -\gamma \times y + \delta \times x \times y \end{cases} \tag{F.1}$$

Annexe G

SBML du modèle Lotka-Volterra

Ce fichier SBML a été généré par le projet Sputnik, une bibliothèque de réseaux de Pétri stochastiques écrite en Python. On remarquera que la nomenclature est fortement inspirée des réactions chimiques étant donné que le projet SBML a émergé dans cette communauté.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by Sputnik version 0.0.1a on 2012-03-16 14:08 with libSBML version 5.3.0. -->
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1">
  <model>
    <listOfCompartments>
      <compartment id="Cell" spatialDimensions="3"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="X1" compartment="Cell" initialAmount="100"/>
      <species id="X2" compartment="Cell" initialAmount="20"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="preyBirth">
        <listOfReactants>
          <speciesReference species="X1" stoichiometry="1"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="X1" stoichiometry="2"/>
        </listOfProducts>
        <kineticLaw>
          <listOfLocalParameters>
            <localParameter id="rate0" value="1"/>
          </listOfLocalParameters>
        </kineticLaw>
      </reaction>
      <reaction id="predation">
        <listOfReactants>
          <speciesReference species="X1" stoichiometry="1"/>
          <speciesReference species="X2" stoichiometry="1"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="X2" stoichiometry="2"/>
        </listOfProducts>
        <kineticLaw>
```

```
        <listOfLocalParameters>
          <localParameter id="rate1" value="0.005"/>
        </listOfLocalParameters>
      </kineticLaw>
    </reaction>
    <reaction id="predatorDeath">
      <listOfReactants>
        <speciesReference species="X2" stoichiometry="1"/>
      </listOfReactants>
      <kineticLaw>
        <listOfLocalParameters>
          <localParameter id="rate2" value="0.6"/>
        </listOfLocalParameters>
      </kineticLaw>
    </reaction>
  </listOfReactions>
</model>
</sbml>
```

Annexe H

Représentation graphiques des bonnes pratiques de modélisation

H.1 D'après (Scholten et al. 2000)

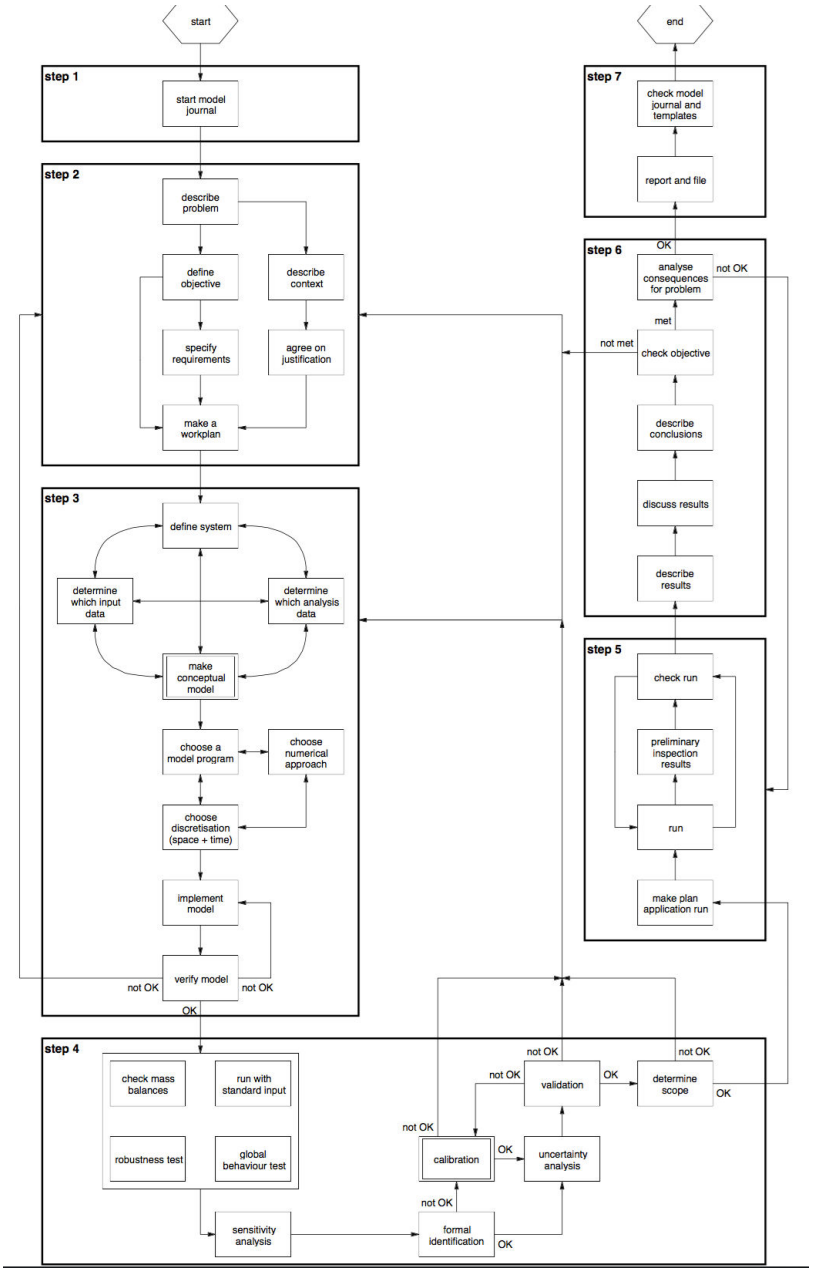


Figure H.1

H.2 D'après (Pavé 2006)

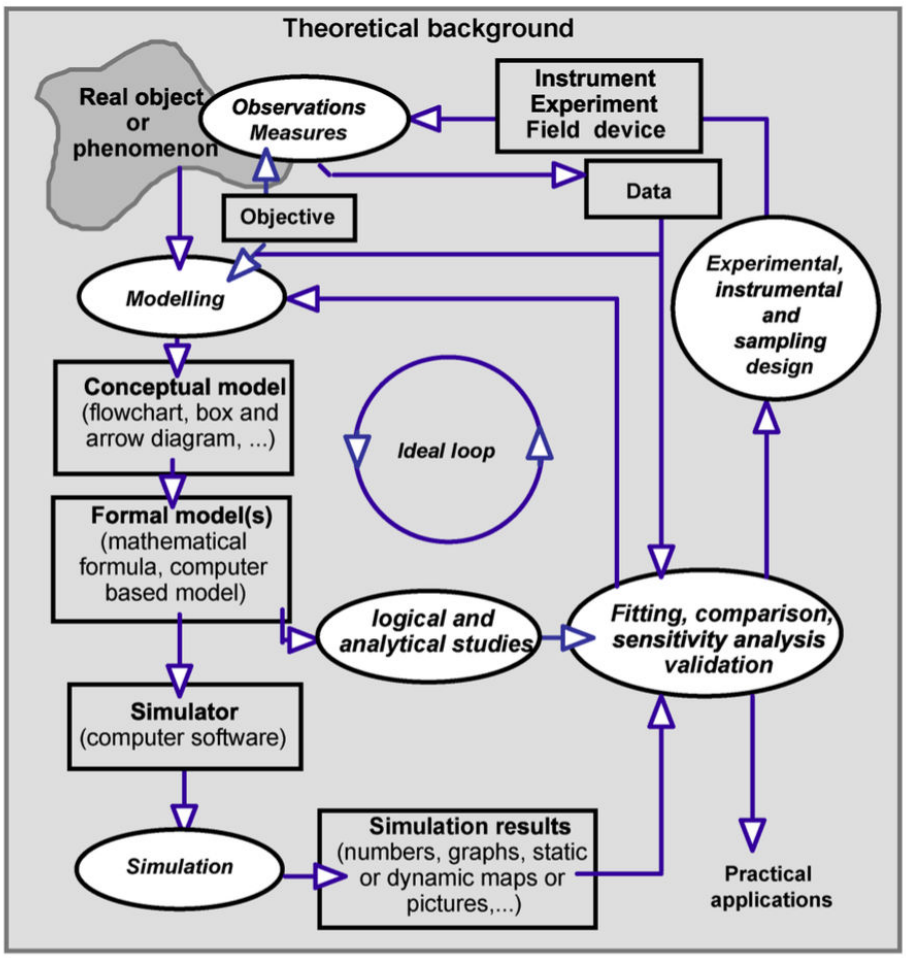


Figure H.2

Liste des figures

Liste des tables

Liste des équations

Liste des codes

Références

Akaike, H. (1973). “Information theory and an extension of the maximum likelihood principle”. In : *Second International Symposium on Information Theory*. Sous la dir. de B. N. Petrov et F. Csaki. Budapest : Akadémiai Kiado, p. 267–281 (cf. p. 89).

Åkesson, Johan, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl et Hubertus Tummescheit (2010). “Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for Solving Large-Scale Dynamic Optimization Problems”. In : *Computers and Chemical Engineering* 34.11, p. 1737–1749 (cf. p. 32).

AMS (1968). *Some Mathematical Problems in Biology*. Sous la dir. d’American Mathematical Society. Lectures on mathematics in the life sciences. American Mathematical Society. isbn : 9780821897058. url : https://books.google.fr/books?id=A_ktBhUqB7QC (cf. p. 46).

Aycock, John (2003). “A Brief History of Just-in-time”. In : *ACM Comput. Surv.* 35.2, p. 97–113. issn : 0360-0300. doi : 10.1145/857076.857077. url : http://doi.acm.org/10.1145/857076.857077 (cf. p. 39).

Baey, C., A. Didier, S. Lemaire, F. Maupas et P.-H. Cournède (2013a). “Modelling the interindividual variability of organogenesis in sugar beet populations using a hierarchical segmented model”. In : *Ecological Modelling* 263, p. 56–63 (cf. p. 91).

Baey, Charlotte (2014). “Modélisation de la variabilité inter-individuelle dans les modèles de croissance de plantes et sélection de modèles pour la prévision”. Thèse de doctorat dirigée par Cournède, Paul-Henry Mathématiques appliquées Châtenay-Malabry, Ecole centrale de Paris 2014. Thèse de doct. url : http://www.theses.fr/2014ECAP0024 (cf. p. 108).

Baey, Charlotte, Anne Didier, Li Song, Sébastien Lemaire, Fabienne Maupas et Paul-Henry Cournède (2012). “Evaluation of the Predictive Capacity of Five Plant Growth Models for Sugar Beet”. In : *Plant Growth Modeling, Simulation, Visualization and Applications - PMA12*. Sous la dir. d’Yan Guo Mengzhen Kang Yves Dumont. Shanghai, China : IEEE, p. 30–37. url : https://hal.archives-ouvertes.fr/hal-00776389 (cf. p. 35, 80).

Baey, Charlotte, Anne Didier, Sébastien Lemaire, Fabienne Maupas et Paul-Henry Cournède (2013b). “Parametrization of five classical plant growth models applied to sugar beet and comparison of their predictive capacity on root yield and total biomass”. In : *Ecological Modelling* (cf. p. 89, 90).

Barros, Fernando J. (1998). “Abstract Simulators for the DSDE Formalism”. In : *Proceedings of the 30th Conference on Winter Simulation*. WSC ’98. Washington, D.C., USA : IEEE Computer Society Press, p. 407–412. isbn : 0-7803-5134-7. url : http://dl.acm.org/citation.cfm?id=293172.293257 (cf. p. 17).

Baudin, Michaël, Anne Dutfoy, Bertrand Iooss et Anne-Laure Popelin (2015). “Title : Open TURNS : An industrial software for uncertainty quantification in simulation”. url : https://hal.archives-ouvertes.fr/hal-01107849 (cf. p. 32).

Bayol, Benoit, Yuting Chen et Paul-Henry Cournède (2013). “Towards an EDSL to enhance good modelling practice for non-linear stochastic discrete dynamical models Application to plant growth models”. In : *SIMULTECH 2013*. Iceland, p. 132–138. doi : 10.5220/0004481101320138. url : https://hal-ecp.archives-ouvertes.fr/hal-01001669 (cf. p. 41, 51, 128).

Bayol, Benoît, Yuting Chen, Charlotte Baey, Gautier Viaud et Paul-Henry Cournède (2015). “WIP - Promoting Good Modeling Practice with a Domain-specific Language and Statistical Algorithms Designed for Parallel Computing”. In : *Proceedings of the Symposium on Theory of Modeling & Simulation : DEVS Integrative M&S Symposium*. DEVS ’15. Alexandria, Virginia : Society for Computer Simulation International, p. 143–148. isbn : 978-1-5108-0105-9. url : http://dl.acm.org/citation.cfm?id=2872965.2872985 (cf. p. 41, 70, 128).

Bergero, Federico et Ernesto Kofman (2011). “PowerDEVS : A Tool for Hybrid System Modeling and Real-time Simulation”. In : *Simulation* 87.1-2, p. 113–132. issn : 0037-5497. doi : 10.1177/0037549710368029. url : http://dx.doi.org/10.1177/0037549710368029 (cf. p. 130).

Bergez, J.-E. et al. (2013). “An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems”. English. In : *Environmental Modelling and Software* 39.Complete, p. 39–49. doi : 10.1016/j.envsoft.2012.03.011 (cf. p. 31).

Bessonov, N., F. Crauste et V. Volpert (2011). “Modelling of Plant Growth with Apical or Basal Meristem”. In : *Mathematical Modelling of Natural Phenomena* 6 (02), p. 107–132 (cf. p. 35).

Beucher, Ottmar (2006). *MATLAB und Simulink (Scientific Computing)*. Pearson Studium. isbn : 3827372062 (cf. p. 71).

Blochwitz, Torsten et al. (2011). “The functional mockup interface for tool independent exchange of simulation models”. In : *Proceedings of the 8th International Modelica Conference*. Linköping University Press, p. 105–114 (cf. p. 131).

Blom, Henk AP, John Lygeros, M Everdij, S Loizou et K Kyriakopoulos (2006). *Stochastic hybrid systems : theory and safety critical applications*. T. 337. Springer Heidelberg (cf. p. 13).

Borgonovo, E. (2007). “A new uncertainty importance measure”. In : *Reliability Engineering & System Safety* 92.6, p. 771 –784. issn : 0951-8320. doi : http://dx.doi.org/10.1016/j.ress.2006.04.015. url : http://www.sciencedirect.com/science/article/pii/S0951832006000883 (cf. p. 83).

Bray, Tim, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler et François Yergeau (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. World Wide Web Consortium, Recommendation REC-xml-20081126 (cf. p. 21).

Brisson, N. (2008). *Conceptual Basis, Formalisations and Parameterization of the STICS Crop Model*. Collection Update sciences & technologies. Editions Quae. isbn : 9782759201693. url : https://books.google.fr/books?id=YwfZFcWgAAMC (cf. p. 27, 35).

Brisson, N. et al. (1998). “STICS : a generic model for the simulation of crops and their water and nitrogen balances. I. Theory, and parameterization applied to wheat and corn”. In : *Agronomie* 18, p. 311–346 (cf. p. 90, 108).

Brun, F., D. Wallach, D. Makowski et J.W. Jones (2006). *Working with Dynamic Crop Models : Evaluation, Analysis, Parameterization, and Applications*. Elsevier Science. isbn : 9780080461939. url : http://books.google.fr/books?id=nG7DEXen9QAC (cf. p. 25, 80, 164).

Brun, Rene et Fons Rademakers (1996). “ROOT - An Object Oriented Data Analysis Framework”. In : *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. Meth. in Phys. Res. A 389 (1997) 81-86. See also http ://root.cern.ch/.* (Cf. p. 32).

Campbell, Stephen L, Jean-Philippe Chancelier et Ramine Nikoukhah (2006a). *Modeling and Simulation in SCILAB*. Springer (cf. p. 18, 31).

Campbell, Stephen L., Jean-Philippe Chancelier et Ramine Nikoukhah (2006b). *Modeling and Simulation in Scilab, Scicos*. eng. Springer (cf. p. 71).

Campillo, F. et V. Rossi (2009). “Convolution Particle Filter for Parameter Estimation in General State-Space Models”. In : *IEEE Transactions in Aerospace and Electronics.* 45.3, p. 1063–1072 (cf. p. 85).

Casadebaig, Pierre, Lydie Guilioni, Jérémie Lecoeur, Angélique Christophe, Luc Champolivier et Philippe Debaeke (2011). “SUNFLO, a model to simulate genotype-specific performance of the sunflower crop in contrasting environments”. In : *Agricultural and Forest Meteorology* 151.2, p. 163 –178. issn : 0168-1923. doi : http://dx.doi.org/10.1016/j.agrformet.2010.09.012. url : http://www.sciencedirect.com/science/article/pii/S0168192310002583 (cf. p. 27).

Castro, Rodrigo, Ernesto Kofman et Gabriel Wainer (2008). “A formal framework for stochastic DEVS modeling and simulation”. In : *Proceedings of the 2008 Spring simulation multiconference*. Society for Computer Simulation International, p. 421–428 (cf. p. 17, 129, 130).

Cellier, F.E. (1991). *Continuous System Modeling*. Continuous System Modeling. Springer. isbn : 9780387975023. url : https://books.google.fr/books?id=c8pODAtyEUAC (cf. p. 10, 11, 17).

Chapelle, Dominique, Marc Fragu, Vivien Mallet et Philippe Moireau (2013). “Fundamental principles of data assimilation underlying the Verdandi library : applications to biophysical model personalization within euHeart”. In : *Medical & biological engineering & computing* 51, p. 1221–1233. doi : 10.1007/s11517-012-0969-6. url : https://hal.inria.fr/hal-00760887 (cf. p. 32).

Chen, Yuting (2014). “Inférence bayésienne dans les modèles de croissance de plantes pour la prévision et la caractérisation des incertitudes”. Thèse de doctorat dirigée par Cournède, Paul-Henry Mathématiques appliquées Châtenay-Malabry, Ecole centrale de Paris 2014. Thèse de doct. url : http://www.theses.fr/2014ECAP0040 (cf. p. 36, 58, 86, 87).

Chen, Yuting et Paul-Henry Cournède (2014). “Data assimilation to reduce uncertainty of crop model prediction with convolution particle filtering”. In : *Ecological Modelling*, in press. doi : 10.1016/j.ecolmodel.2014.01.030. url : https://hal.archives-ouvertes.fr/hal-00997728 (cf. p. 58, 85, 101, 107).

Chen, Yuting, Benoît Bayol, Cédric Loi, Samis Trevezas et Paul-Henry Cournède (2012). “Filtrage par noyaux de convolution itératif”. In : *Les 44 Journées de Statistique*. Bruxelles, Belgium, CHEN. url : https://hal.archives-ouvertes.fr/hal-00776397 (cf. p. 70, 87, 128).

Cheney, James, Sam Lindley et Philip Wadler (2013). “A Practical Theory of Language-integrated Query”. In : *SIGPLAN Not.* 48.9, p. 403–416. issn : 0362-1340. doi : 10.1145/2544174.2500586. url : http://doi.acm.org/10.1145/2544174.2500586 (cf. p. 49).

Cheon, S. (2007). “Experimental frame structuring for automated model construction : application to simulated weather generation.” Thèse de doct. University of Arizona (cf. p. 18, 133).

Chow, AC-H (1996). “Parallel DEVS : a parallel, hierarchical, modular modeling formalism and its distributed simulator”. In : *Transactions of the Society for Computer Simulation International* 13.2, p. 55–67 (cf. p. 17).

Coddington, Paul D. (1997). “Random Number Generators for Parallel Computers”. In : *Northeast Parallel Architecture Center* (cf. p. 59, 70).

Costes, Evelyne, Colin Smith, Michael Renton, Yann Guédon, Premyslaw Prusinkiewicz et Christophe Godin (2008). “MAppleT : simulation of apple tree development using mixed stochastic and biomechanical models”. In : *Functional Plant Biology* 35.10, p. 936–950. url : https://hal.inria.fr/hal-00831809 (cf. p. 27).

Cournède, P.-H., M.Z. Kang, A. Mathieu, J.-F. Barczi, H.P. Yan, B.G. Hu et P. de Reffye (2006). “Structural factorization of plants to compute their functional and architectural growth”. In : *Simulation* 82.7, p. 427–438 (cf. p. 27, 31).

Cournède, P.-H., A. Mathieu, F. Houllier, D. Barthélémy et P. de Reffye (2008). “Computing competition for light in the GreenLab model of plant growth : a contribution to the study of the effects of density on resource acquisition and architectural development”. In : *Annals of Botany* 101.8 (cf. p. 35).

Cournède, P.-H., V. Letort, A. Mathieu, M.-Z. Kang, S. Lemaire, S. Trevezas, F. Houllier et P. de Reffye (2011). “Some Parameter Estimation Issues in Functional-Structural Plant Modelling”. In : *Mathematical Modelling of Natural Phenomena* 6.2, p. 133–159 (cf. p. 31, 113).

Cournède, P.-H., Y. Chen, Q. Wu, C. Baey et B. Bayol (2013). “Development and Evaluation of Plant Growth Models : Methodology and Implementation in the PYGMALION platform”. In : *Mathematical Modelling of Natural Phenomena* 8 (cf. p. 92).

Cournède, Paul-Henry, Yuting Chen, Qiongli Wu, Charlotte Baey et Benoît Bayol (2013). “Development and Evaluation of Plant Growth Models : Methodology and Implementation in the PYGMALION platform”. In : *Mathematical Modelling of Natural Phenomena* 8.4, p. 112–130. url : https://hal-ecp.archives-ouvertes.fr/hal-00860902 (cf. p. 27, 36, 78, 90, 128).

De Reffye, Philippe et B.G. Hu (2003). “Relevant Qualitative and Quantitative choices for building an efficient dynamic plant growth model : Greenlab Case”. In : *International Symposium on Plant Growth Modeling, Simulation, Visualization and their Applications - PMA’03*. Sous la dir. de B.G. Hu et Jaeger M. Plant Growth Modeling and Applications. Beijing / China : Springer and Tsinghua University Press, p. 87–107. url : https://hal.inria.fr/inria-00126213 (cf. p. 27, 92, 131).

Deursen, Arie van, Paul Klint et Joost Visser (2000). “Domain-specific Languages : An Annotated Bibliography”. In : *SIGPLAN Not.* 35.6, p. 26–36. issn : 0362-1340. doi : 10.1145/352029.352035. url : http://doi.acm.org/10.1145/352029.352035 (cf. p. 18, 19).

ECMA (2013). *The JSON Data Interchange Format*. Rapp. tech. Standard ECMA-404 1st Edition / October 2013. url : http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (cf. p. 138).

Eldred, Michael S, Anthony A Giunta, Bart G van Bloemen Waanders, Steven F Wojtkiewicz, William E Hart et Mario P Alleva (2007). *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis : Version 4.1 reference manual*. Citeseer (cf. p. 32).

Elmqvist, Hilding et Dynasim Ab (1997). “Modelica - The Next Generation Modeling Language An International Design Effort”. In : *In Proceedings of First World Congress of System Simulation*, p. 1–3 (cf. p. 18, 31, 40, 136).

Evers, Jochem B, Jan Vos, Christian Fournier, Bruno Andrieu, Michael Chelle et Paul C Struik (2005). “Towards a generic architectural model of tillering in Gramineae, as exemplified by spring wheat (Triticum aestivum)”. In : *New Phytologist* 166.3, p. 801–812 (cf. p. 27).

Faivre, R., B. Iooss, S. Mahévas, D. Makowski et H. Monod (2013). *Analyse de sensibilité et exploration de modèles : application aux sciences de la nature et de l’environnement*. Collection Savoir-faire. Éd. Quae. isbn : 9782759219063. url : https://books.google.fr/books?id=6JZP7Uy4fD4C (cf. p. 14, 80, 88).

Fan, Xing-Rong, Meng-Zhen Kang, Ep Heuvelink, Philippe de Reffye et Bao-Gang Hu (2015). “A knowledge-and-data-driven modeling approach for simulating plant growth : A case study on tomato growth”. In : *Ecological Modelling* 312, p. 363 –373. issn : 0304-3800. doi : http://dx.doi.org/10.1016/j.ecolmodel.2015.06.006. url : http://www.sciencedirect.com/science/article/pii/S0304380015002550 (cf. p. 23, 131).

Flynn, M. J. (1972). “Some Computer Organizations and Their Effectiveness”. In : *IEEE Transactions on Computers* C-21.9, p. 948–960. issn : 0018-9340. doi : 10.1109/TC.1972.5009071 (cf. p. 58).

Forum, Message P (1994). *MPI : A Message-Passing Interface Standard*. Rapp. tech. Knoxville, TN, USA (cf. p. 58, 129, 130).

Foures, Damien, Vincent Albert et Alexandre Nketsa (2013). “Simulation validation using the compatibility between Simulation Model and Experimental Frame”. In : *SummerSim’13 (45th Summer Simulation Multi-conference 2013)*. 7 pages, work in progress. Toronto, Canada, p. 75. url : https://hal.archives-ouvertes.fr/hal-00868406 (cf. p. 18, 133).

Fournier, C. et B. Andrieu (1999). “ADEL-Maize : An L-system based model for the integration of growth process from the organ to the canopy. Application to regulation of morphogenesis by light availability”. In : *Agronomy* 19, p. 313–327 (cf. p. 35).

Fritzson, Peter et Peter Bunus (2002). “Modelica – A General Object-Oriented Language for Continuous and Discrete-Event System Modeling”. In : *IN PROCEEDINGS OF THE 35TH ANNUAL SIMULATION SYMPOSIUM*, p. 14–18 (cf. p. 71).

Gabriel, Edgar et al. (2004). “Open MPI : Goals, Concept, and Design of a Next Generation MPI Implementation”. In : *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, p. 97–104 (cf. p. 19).

Gaines, Brian R (1979). “General systems research : quo vadis”. In : (cf. p. 10).

Gamma, Erich, Richard Helm, Ralph Johnson et John M. Vlissides (1995). *Design patterns : elements of reusable object-oriented software*. Pearson Education India (cf. p. 38).

Gennari, John H., Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy et Samson W. Tu (2003). “The Evolution of ProtÉGé : An Environment for Knowledge-based Systems Development”. In : *Int. J. Hum.-Comput. Stud.* 58.1, p. 89–123. issn : 1071-5819. doi : 10.1016/S1071-5819(02)00127-1. url : http://dx.doi.org/10.1016/S1071-5819(02)00127-1 (cf. p. 20).

Getoor, L. et B. Taskar (2007). *Introduction to Statistical Relational Learning*. Adaptive computation and machine learning. MIT Press. isbn : 9780262072885. url : https://books.google.fr/books?id=lSkIewOw2WoC (cf. p. 71, 136).

Glimm, Birte, Ian Horrocks, Boris Motik, Giorgos Stoilos et Zhe Wang (2014). “HermiT : An OWL 2 Reasoner”. English. In : *Journal of Automated Reasoning* 53.3, p. 245–269. issn : 0168-7433. doi : 10.1007/s10817-014-9305-1. url : http://dx.doi.org/10.1007/s10817-014-9305-1 (cf. p. 20).

Goonatilake, Suran et Sukhdev Khebbal (1994). *Intelligent hybrid systems*. John Wiley & Sons, Inc. (cf. p. 13).

Gropp, William, Ewing Lusk et Anthony Skjellum (1999). *Using MPI (2Nd Ed.) : Portable Parallel Programming with the Message-passing Interface*. Cambridge, MA, USA : MIT Press. isbn : 0-262-57132-3 (cf. p. 19).

Gruber, Thomas R. (1995). “Toward Principles for the Design of Ontologies Used for Knowledge Sharing”. In : *Int. J. Hum.-Comput. Stud.* 43.5-6, p. 907–928. issn : 1071-5819. doi : 10.1006/ijhc.1995.1081. url : http://dx.doi.org/10.1006/ijhc.1995.1081 (cf. p. 20).

Hastie, Trevor, Robert Tibshirani et Jerome Friedman (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA : Springer New York Inc. (cf. p. 13, 14).

Herdman, J. A., W. P. Gaudin, O. Perks, D. A. Beckingsale, A. C. Mallinson et S. A. Jarvis (2014). “Achieving Portability and Performance Through OpenACC”. In : *Proceedings of the First Workshop on Accelerator Programming Using Directives*. WACCPD '14. New Orleans, Louisiana : IEEE Press, p. 19–26. isbn : 978-1-4799-7023-0. doi : 10.1109/WACCPD.2014.10. url : http://dx.doi.org/10.1109/WACCPD.2014.10 (cf. p. 59).

Herlihy, Maurice et Nir Shavit (2008). *The Art of Multiprocessor Programming*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc. isbn : 0123705916, 9780123705914 (cf. p. 19).

Hill, David (2014). “Simulations stochastiques et calcul à haute performance : la « parallélisation » des générateurs de nombres pseudo-aléatoires”. In : sous la dir. de Frank Varenne et Marc Silberstein Editeurs. T. 2 (cf. p. 59).

Hucka, M., F. Bergmann, S. Hoops, S. Keating, S. Sahle, L. Schaff J. Smith et D. Wilkinson (2010). *The Systems Biology Markup Language (SBML) : Language Specification for Level 3 Version 1 Core* (cf. p. 21, 133).

Hucka, Michael et al. (2003). “The systems biology markup language (SBML) : a medium for representation and exchange of biochemical network models”. In : *Bioinformatics* 19.4, p. 524–531 (cf. p. 21).

Hughes, John (1989). “Why functional programming matters”. In : *The computer journal* 32.2, p. 98–107 (cf. p. 39).

Hurvich, Clifford M et Chih-Ling Tsai (1989). “Regression and time series model selection in small samples”. In : *Biometrika* 76.2, p. 297–307 (cf. p. 89).

Illinois, University of. *Clang* (cf. p. 132).

Ingram, GD, IT Cameron et KM Hangos (2004). “Classification and analysis of integrating frameworks in multiscale modelling”. In : *Chemical engineering science* 59.11, p. 2171–2187 (cf. p. 13).

Iooss, Bertrand (2011). “Revue sur l’analyse de sensibilité globale de modèles numériques”. In : *Journal de la Société Française de Statistique* 152.1, p. 1–23. url : https://hal.archives-ouvertes.fr/hal-00503179 (cf. p. 82).

Iooss, Bertrand et Paul Lemaître (2015). “A review on global sensitivity analysis methods”. In : *Uncertainty Management in Simulation-Optimization of Complex Systems*. Springer, p. 101–122 (cf. p. 82, 83).

ISO (2003). *ISO/IEC 14882 :2003 : Programming languages : C++*. url : http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38110 (cf. p. 37).

– (2011). *ISO/IEC 14882 :2011, Programming languages – C++*. First. 11 West 42nd Street, New York, New York 1O036 : American National Standards Institute (ANSI). url : http://www.open-std.org/jtc1/sc22/wg21/ (cf. p. 37).

Jones, C.A., J.R. Kiniry et P.T. Dyke (1986). *CERES-Maize : a simulation model of maize growth and development*. Texas A&M University Press. isbn : 9780890962695. url : https://books.google.fr/books?id=SCkzAAAAMAAJ (cf. p. 27).

Julier, S., J. Uhlmann et H. Durrant-Whyte (2000). “A New Method for the Non-Linear Transformation of Means and Covariances in Filters and Estimators.” In : *IEEE Transaction on Automatic Control* 45.3, p. 477–482 (cf. p. 85, 88).

Kalman, R. E. (1960). “A New Approach to Linear Filtering And Prediction Problems”. In : *ASME Journal of Basic Engineering* (cf. p. 85).

Kang, Fenni (2013). “Modèles de croissance de plantes et méthodologies adaptées à leur paramétrisation pour l’analyse des phénotypes”. Thèse de doctorat dirigée par Cournède, Paul-Henry Mathématiques appliquées Châtenay-Malabry, Ecole centrale de Paris 2013. Thèse de doct. url : http://www.theses.fr/2013ECAP0035 (cf. p. 30).

Kano, Manabu et Yoshiaki Nakagawa (2008). “Data-based process monitoring, process control, and quality improvement : Recent developments and applications in steel industry”. In : *Computers & Chemical Engineering* 32.1–2. Process Systems Engineering : Contributions on the State-of-the-ArtSelected ex-

tended Papers from {ESCAPE} '16/PSE 2006., p. 12 –24. issn : 0098-1354. doi : http://dx.doi.org/10.1016/j.compchemeng.2007.07.005. url : http://www.sciencedirect.com/science/article/pii/S0098135407001986 (cf. p. 14).

Klipp, Edda, Wolfram Liebermeister, Anselm Helbig, Axel Kowald et Jörg Schaber (2007). "Systems biology standards—the community speaks". In : *Nature biotechnology* 25.4, p. 390–391 (cf. p. 15, 21).

Klöckner, Andreas, Nicolas Pinto, Yunsup Lee, Bryan C. Catanzaro, Paul Ivanov et Ahmed Fasih (2009). "PyCUDA : GPU Run-Time Code Generation for High-Performance Computing". In : *CoRR* abs/0911.3456. url : http://arxiv.org/abs/0911.3456 (cf. p. 18).

Kniemeyer, Ole (2008). "Design and implementation of a graph grammar based language for functional-structural plant modelling." Thèse de doct. (cf. p. 121).

Kniemeyer, Ole et Winfried Kurth (2008). "The Modelling Platform GroIMP and the Programming Language XL". English. In : *Applications of Graph Transformations with Industrial Relevance*. Sous la dir. d'Andy Schürr, Manfred Nagl et Albert Zündorf. T. 5088. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 570–572. isbn : 978-3-540-89019-5. doi : 10.1007/978-3-540-89020-1_39. url : http://dx.doi.org/10.1007/978-3-540-89020-1_39 (cf. p. 31, 121).

Knuth, Donald E. (1997). *The Art of Computer Programming, Volume 2 (3rd Ed.) : Seminumerical Algorithms*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc. isbn : 0-201-89684-2 (cf. p. 61, 62, 67).

Kofman, Ernesto et Sergio Junco (2001). "Quantized-state Systems : A DEVS Approach for Continuous System Simulation". In : *Trans. Soc. Comput. Simul. Int.* 18.3, p. 123–132. issn : 0740-6797. url : http://dl.acm.org/citation.cfm?id=609891.609893 (cf. p. 17).

Korn, G.A. et J.V. Wait (1978). *Digital continuous-system simulation*. Prentice-Hall. isbn : 9780132122740. url : https://books.google.fr/books?id=yN9QAAAAMAAJ (cf. p. 17).

Krasner, Glenn E. et Stephen T. Pope (1988). "A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80". In : *J. Object Oriented Program.* 1.3, p. 26–49. issn : 0896-8438. url : http://dl.acm.org/citation.cfm?id=50757.50759 (cf. p. 48).

Laarhoven, P.J. van et E.H. Aarts (1987). *Simulated Annealing : Theory and Applications*. Ellis Horwood series in mathematics and its applications. Springer. isbn : 9789027725134. url : https://books.google.fr/books?id=-IgUab6Dp_IC (cf. p. 85).

Landin, P. J. (1966). "The Next 700 Programming Languages". In : *Commun. ACM* 9.3, p. 157–166. issn : 0001-0782. doi : 10.1145/365230.365257. url : http://doi.acm.org/10.1145/365230.365257 (cf. p. 71).

Lattner, Chris et Vikram Adve (2004). "LLVM : A Compilation Framework for Lifelong Program Analysis & Transformation". In : *Proceedings of the International Symposium on Code Generation and Optimization : Feedback-directed and Runtime Optimization*. CGO '04. Palo Alto, California : IEEE Computer Society, p. 75–. isbn : 0-7695-2102-9. url : http://dl.acm.org/citation.cfm?id=977395.977673 (cf. p. 20, 58, 70, 72, 130).

Le Chevalier, Vincent et Marc Jaeger (2010). “Bottom-up approach of landscape simulation leading to a generic synchronization.” In : *LandMod 2010 : International Conference on Integrative Landscape Modelling* (cf. p. 40, 131).

Lecoeur, J., R. Poir□Lassus, A. Christophe, B. Pallas, P. Casadebaig, P. Debaeke, F. Vear et L. Guilioni (2011). “Quantifying physiological determinants of genetic variation for yield potential in sunflower. SUNFLO : a model-based analysis”. In : *Functional Plant Biology* 38, p. 246–259 (cf. p. 30).

L’ecuyer, Pierre (1999). “Good parameters and implementations for combined multiple recursive random number generators”. In : *Operations Research* 47.1, p. 159–164 (cf. p. 20, 62, 129).

L’Ecuyer, Pierre et Richard Simard (2007). “TestU01 : A C Library for Empirical Testing of Random Number Generators”. In : *ACM Trans. Math. Softw.* 33.4. issn : 0098-3500. doi : 10.1145/1268776.1268777. url : http://doi.acm.org/10.1145/1268776.1268777 (cf. p. 60, 64, 67).

L’ecuyer, Pierre, Richard Simard, E Jack Chen et W David Kelton (2002). “An object-oriented random-number package with many long streams and substreams”. In : *Operations research* 50.6, p. 1073–1075 (cf. p. 62).

Lehmer, D. H. (1951). “Mathematical Methods in Large Scale Computing Units”. In : *Annals Comp. Laboratory Harvard University* 26, p. 141–146 (cf. p. 60).

Leijen, Daan et Erik Meijer (1999). “Domain specific embedded compilers”. In : *ACM Sigplan Notices*. T. 35. 1. ACM, p. 109–122 (cf. p. 18, 19).

Lemaire, S., F. Maupas, P.-H. Cournède et P. de Reffye (2008). “A morphogenetic crop model for sugar-beet (*Beta vulgaris L.*).” In : *International Symposium on Crop Modeling and Decision Support : ISCMDS 2008, April 19-22, 2008, Nanjing, China* (cf. p. 91).

Letort, V. (2008). “Multi-scale analysis of source-sink relationships in plant growth models for parameter identification. Case of the GreenLab model.” Thèse de doct. Ecole Centrale Paris (cf. p. 36).

Letort, Véronique, Paul Mahe, Paul-Henry Cournède, Philippe De Reffye et Brigitte Courtois (2008). “Quantitative genetics and functional-structural plant growth models : Simulation of quantitative trait loci detection for model parameters and application to potential yield optimization”. English. In : *Annals of Botany* 101.8. A-08-24 A-08-24, p. 1243–1254. doi : 10.1093/aob/mcm197. url : http://hal-sde.archives-ouvertes.fr/halsde-00288538 (cf. p. 29).

Lotka, Alfred J. (1925). *Elements of physical biology*. Sous la dir. de Williams et Wilkins Company (cf. p. 46).

L’Ecuyer, Pierre, Boris Oreshkin et Richard Simard (2014). “Random numbers for parallel computers : requirements and methods”. In : *to appear* (cf. p. 20, 59, 64, 66, 67, 70, 129).

Mailhol, J.C., A. Olufayo et P. Ruelle (1997). “Sorghum and sunflower evapotranspiration and yield from simulated leaf area index”. In : *Agri. Water Manag.* 35, p. 167–182 (cf. p. 27).

Mansinghka, Vikash K., Daniel Selsam et Yura N. Perov (2014). “Venture : a higher-order probabilistic programming platform with programmable inference”. In : *CoRR* abs/1404.0099. url : http://arxiv.org/abs/1404.0099 (cf. p. 71).

Marsaglia, George et al. (2003). “Xorshift rngs”. In : *Journal of Statistical Software* 8.14, p. 1–6 (cf. p. 62).

Mascagni, Michael et Ashok Srinivasan (2000). “Algorithm 806 : SPRNG : A scalable library for pseudo-random number generation”. In : *ACM Transactions on Mathematical Software (TOMS)* 26.3, p. 436–461 (cf. p. 20, 67, 129).

MATLAB (2010). *version 7.10.0 (R2010a)*. Natick, Massachusetts : The MathWorks Inc. (cf. p. 18, 31).

Matsumoto, Makoto et Takuj Nishimura (1998a). “Dynamic creation of pseudorandom number generators”. In : (cf. p. 62, 66).

Matsumoto, Makoto et Takuji Nishimura (1998b). “Mersenne Twister : A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator”. In : *ACM Trans. Model. Comput. Simul.* 8.1, p. 3–30. issn : 1049-3301. doi : 10.1145/272991.272995. url : http://doi.acm.org/10.1145/272991.272995 (cf. p. 60, 62).

Mernik, Marjan, Jan Heering et Anthony M. Sloane (2005). “When and How to Develop Domain-specific Languages”. In : *ACM Comput. Surv.* 37.4, p. 316–344. issn : 0360-0300. doi : 10.1145/1118890.1118892. url : http://doi.acm.org/10.1145/1118890.1118892 (cf. p. 71).

Meyer, Bertrand (1988). *Object-Oriented Software Construction*. 1st. Upper Saddle River, NJ, USA : Prentice-Hall, Inc. isbn : 0136290493 (cf. p. 38).

Meza, J. C., R. A. Oliva, P. D. Hough et P. J. Williams (2007). “OPT++ : An Object-oriented Toolkit for Nonlinear Optimization”. In : *ACM Trans. Math. Softw.* 33.2. issn : 0098-3500. doi : 10.1145/1236463.1236467. url : http://doi.acm.org/10.1145/1236463.1236467 (cf. p. 32).

Monteith, J. L. et C. J. Moss (1977). “Climate and the Efficiency of Crop Production in Britain [and Discussion]”. In : *Philosophical Transactions of the Royal Society of London B : Biological Sciences* 281.980, p. 277–294. issn : 0080-4622. doi : 10.1098/rstb.1977.0140 (cf. p. 25).

Naldi, Giovanni, Lorenzo Pareschi et Giuseppe Toscani (2010). *Mathematical modeling of collective behavior in socio-economic and life sciences*. Springer Science & Business Media (cf. p. 13).

Nickolls, John, Ian Buck, Michael Garland et Kevin Skadron (2008). “Scalable Parallel Programming with CUDA”. In : *Queue* 6.2, p. 40–53. issn : 1542-7730. doi : 10.1145/1365490.1365500. url : http://doi.acm.org/10.1145/1365490.1365500 (cf. p. 19, 130, 132).

Nutaro, James J. (2010). *Building Software for Simulation : Theory and Algorithms, with Applications in C++*. Wiley Publishing. isbn : 0470414693, 9780470414699 (cf. p. 130).

Nutaro, Jim (2013). *adevs (A Discrete Event Simulator) library*. http ://web.ornl.gov/~1qn/adevs/. url : http://web.ornl.gov/\~{}1qn/adevs/ (cf. p. 130).

NVIDIA (2010). *CUDA CURAND Library*. NVIDIA Corporation. Santa Clara, CA, USA (cf. p. 62).

OpenMP Architecture Review Board (2008). *OpenMP Application Program Interface Version 3.0*. url : http://www.openmp.org/mp-documents/spec30.pdf (cf. p. 19, 58, 91).

Panneton, François, Pierre L'ecuyer et Makoto Matsumoto (2006). "Improved long-period generators based on linear recurrences modulo 2". In : *ACM Transactions on Mathematical Software (TOMS)* 32.1, p. 1–16 (cf. p. 62).

Pavé, Alain (2006). "By way of introduction : Modelling living systems, their diversity and their complexity : some methodological and theoretical problems□". In : *Comptes Rendus Biologies* 329.1. Modélisation de systèmes complexes en agronomie et environnementModelling of complex systems in agronomy and environment, p. 3 –12. issn : 1631-0691. doi : http://dx.doi.org/10.1016/j.crvi.2005.09.011. url : http://www.sciencedirect.com/science/article/pii/S1631069105001745 (cf. p. 77, 159).

Pérez, Jorge, Marcelo Arenas et Claudio Gutierrez (2009). "Semantics and Complexity of SPARQL". In : *ACM Trans. Database Syst.* 34.3, 16 :1–16 :45. issn : 0362-5915. doi : 10.1145/1567274.1567278. url : http://doi.acm.org/10.1145/1567274.1567278 (cf. p. 135).

Perttunen, J., R. Sievänen et E. Nikinmaa (1998). "LIGNUM : a model combining the structure and the functioning of trees". In : *Ecological Modelling* 108, p. 189–198 (cf. p. 122).

Perttunen, Jari, Risto Sievänen, Mika Lehtonen et Eero Nikinmaa (2002). "Towards Functional-Structural Analysis of Forest Stands with LIGNUM." In : *ESM*. Sous la dir. de Krzysztof Amborski et Hermann Meuth. SCS Europe, p. 120–122. isbn : 90-77039-07-4. url : http://dblp.uni-trier.de/db/conf/esm/esm2002.html#PerttunenSLN02 (cf. p. 27, 121).

Pitrat, M. et C. Foury (2003). *Histoires de légumes : des origines à l'orée du XXIe siècle*. Institut National de la Recherche Agronomique. isbn : 9782738010667. url : http://books.google.fr/books?id=2M8DmYyUw-QC (cf. p. 29).

Pradal, Christophe, Christian Fournier, Patrick Valduriez et Sarah Cohen-Boulakia (2015). "OpenAlea : Scientific Workflows Combining Data Analysis and Simulation". In : *SSDBM 2015 : 27th International Conference on Scientific and Statistical Database Management*. San Diego, United States. doi : 10.1145/2791347.2791365. url : https://hal.archives-ouvertes.fr/hal-01166298 (cf. p. 31).

Prusinkiewicz, P. (2004). "Modeling plant growth and development". In : *Current opinion in plant biology* 7.1, p. 79–84 (cf. p. 35).

Quesnel, Gauthier, Raphaël Duboz et Éric Ramat (2009). "The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems". In : *Simulation Modelling Practice and Theory* 17, p. 641–653 (cf. p. 18, 31, 130).

R Core Team (2013). *R : A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria. url : http://www.R-project.org/ (cf. p. 31).

Rand_Corporation (1955). *A Million Random Digits with 100,000 Normal Deviates*. Rand. isbn : 9780833030474. url : https://books.google.fr/books?id=XvwX1fxryIgC (cf. p. 60).

Reffye, P. de, E. Heuvelink, D. Barthélémy et P.-H. Cournède (2008). "Plant Growth Models". In : *Ecological Models. Vol. 4 of Encyclopedia of Ecology (5 volumes)*. Sous la dir. de S.E. Jorgensen et B. Fath. Elsevier, Oxford, p. 2824–2837 (cf. p. 35).

Reuillon, Romain (2008). "Simulations stochastiques en environnements distribués. Application aux grilles de calcul". Thèse de doct. Université Blaise Pascal-Clermont-Ferrand II (cf. p. 59).

Reuillon, Romain, Mathieu Leclaire et Sebastien Rey-Coyrehourcq (2013). “OpenMOLE, a Workflow Engine Specifically Tailored for the Distributed Exploration of Simulation Models”. In : *Future Gener. Comput. Syst.* 29.8, p. 1981–1990. issn : 0167-739X. doi : 10.1016/j.future.2013.05.003. url : http://dx.doi.org/10.1016/j.future.2013.05.003 (cf. p. 32).

Reuillon, Romain, Mamadou K. Traoré, Jonathan Passerat-Palmbach et David R. C. Hill (2012). “Parallel stochastic simulations with rigorous distribution of pseudo-random numbers with DistMe : Application to life science simulations”. In : *Concurrency and Computation : Practice and Experience* 24.7, p. 723–738. doi : 10.1002/cpe.1883. url : http://dx.doi.org/10.1002/cpe.1883 (cf. p. 20).

Richardson, Leonard et Sam Ruby (2007). *Restful Web Services*. First. O’Reilly. isbn : 9780596529260 (cf. p. 137).

Saito, Mutsuo (2010). “A Variant of Mersenne Twister Suitable for Graphic Processors”. In : *CoRR* abs/1005.4973. url : http://arxiv.org/abs/1005.4973 (cf. p. 62).

Scholten, H., S. Groot, F. C. Van Geer et J. J. Noort (2000). “Good Modelling Practice in water management”. In : *in Proceedings HydroInformatics2000. International Association for Hydraulic Research, Cedar Rapids* (cf. p. 15, 32, 77, 158).

Schwarz, Gideon et al. (1978). “Estimating the dimension of a model”. In : *The annals of statistics* 6.2, p. 461–464 (cf. p. 90).

Shi, Yuhui et Russell Eberhart (1998). “A modified particle swarm optimizer”. In : *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, p. 69–73 (cf. p. 85).

Sievänen, R., E. Nikinmaa, P. Nygren, H. Ozier-Lafontaine, J. Perttunen et H. Hakula (2000). “Components of a functional-structural tree model”. In : *Annals of Forest Sciences* 57, p. 399–412 (cf. p. 27).

Sievänen, R., J. Perttunen, E. Nikinmaa et P. Kaitaniemi (2008). “Toward extension of a single tree functional-structural model of Scots pine to stand level : effect of the canopy of randomly distributed, identical trees on development of tree structure”. In : *Functional Plant Biology* 35.10, p. 964–975 (cf. p. 122).

Smolenova, Katarina, Winfried Kurth et Paul-Henry Cournède (2012). “Parallel Graph Grammars with Instantiation Rules Allow Efficient Structural Factorization of Virtual Vegetation”. In : *Electronic Communications of the EASST*, p. 114–128. url : https://hal-ecp.archives-ouvertes.fr/hal-00872370 (cf. p. 27, 131).

Soetaert, KER et Thomas Petzoldt (2010). “Inverse modelling, sensitivity and monte carlo analysis in R using package FME”. In : *Journal of Statistical Software* 33 (cf. p. 32).

Spiegelhalter, D. J., N. G. Best, B. P. Carlin et A. Van der Linde (1998). *Bayesian deviance, the effective number of parameters, and the comparison of arbitrarily complex models*. Rapp. tech. (cf. p. 90).

Spinellis, Diomidis (2001). *Notable design patterns for domain-specific languages* (cf. p. 71).

Stone, John E., David Gohara et Guochun Shi (2010). “OpenCL : A Parallel Programming Standard for Heterogeneous Computing Systems”. In : *IEEE Des. Test* 12.3, p. 66–73. issn : 0740-7475. doi : 10.1109/MCSE.2010.69. url : http://dx.doi.org/10.1109/MCSE.2010.69 (cf. p. 19, 130, 132).

Straume, Martin et Michael L Johnson (2010). “Analysis of residuals : criteria for determining goodness-of-fit”. In : sous la dir. d’Essential Numerical Computer Methods. Academic Press, p. 37 (cf. p. 90).

Streit, Katarina, Michael Henke, Benoît Bayol, Paul-Henry Cournède et Winfried Kurth (2016). “Impact of geometrical traits on light interception in conifers : analysis using an FSPM for Scots pine”. In : *FSPMA* (cf. p. 121, 129).

Søndergaard, Harald et Peter Sestoft (1990). “Referential transparency, definiteness and unfoldability”. English. In : *Acta Informatica* 27.6, p. 505–517. issn : 0001-5903. doi : 10.1007/BF00277387. url : http://dx.doi.org/10.1007/BF00277387 (cf. p. 141).

Tardieu, F. (2003). “Virtual plants : modelling as a tool for the genomics of tolerance to water deficit”. In : *Trends in Plant Science* 8.1, p. 9–14 (cf. p. 36).

The HDF Group (1997). *Hierarchical Data Format, version 5*. http ://www.hdfgroup.org/HDF5/ (cf. p. 129).

Trevezas, S. et P.-H. Cournède (2013). “A Sequential Monte Carlo Approach for MLE in a Plant Growth Model”. In : *Journal of Agricultural, Biological, and Environmental Statistics* 18.2, p. 250–270. doi : 10.1007/s13253-013-0134-1. url : http://dx.doi.org/10.1007/s13253-013-0134-1 (cf. p. 86).

Trevezas, Samis et Paul-Henry Cournède (2013). “A Sequential Monte Carlo Approach for MLE in a Plant Growth Model”. English. In : *Journal of Agricultural, Biological, and Environmental Statistics* 18.2, p. 250–270. issn : 1085-7117. doi : 10.1007/s13253-013-0134-1. url : http://dx.doi.org/10.1007/s13253-013-0134-1 (cf. p. 36).

Uhlmann, J.K. (1995). *Dynamic Map Building and Localization : New Theoretical Foundations*. University of Oxford. url : https://books.google.fr/books?id=eJb4MgEACAAJ (cf. p. 88).

Viaud, Gautier, Yuting Chen, Benoît Bayol et Paul-Henry Cournède (2015). “A Comparison of a Generic MCMC-Based Algorithm for Bayesian Estimation in C++, R and Julia. Application to Plant Growth Modeling”. In : *ANSS*. Alexandria, United States. url : https://hal.archives-ouvertes.fr/hal-01250999 (cf. p. 31, 86, 128, 143).

Volterra, Vito (1926). “Fluctuations in the Abundance of a Species considered Mathematically”. In : *Nature* (cf. p. 46).

– (1931). *Leçons sur la théorie mathématique de la lutte pour la vie*. Sous la dir. de Gauthier-Villars (cf. p. 46).

Von Neumann, John (1951). “Various Techniques Used in Connection With Random Digits”. In : (cf. p. 60).

Vos, J, Jochem B Evers, GH Buck-Sorlin, Bruno Andrieu, M Chelle et Pieter HB De Visser (2010). “Functional–structural plant modelling : a new versatile tool in crop science”. In : *Journal of experimental Botany* 61.8, p. 2101–2115 (cf. p. 27).

W3C (2009). *OWL 2 Web Ontology Language : Document Overview*. Available at http://www.w3.org/TR/owl2-overview/. W3C Recommendation (cf. p. 133).

Wächter, Andreas et Lorenz T Biegler (2006). “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In : *Mathematical programming* 106.1, p. 25–57 (cf. p. 32).

Walter, E. et L. Pronzato (1994). *Identification de modèles paramétriques : à partir de données expérimentales*. MASC modélisation, analyse, simulation, commande. Masson. isbn : 9782225844072. url : https://books.google.fr/books?id=JPiTAAAACAAJ (cf. p. 10, 11, 85, 90).

Wiberg, D. (1983). “Dynamic system identification : Experiment design and data analysis”. In : *Automatic Control, IEEE Transactions on* 28.10, p. 999–1000. issn : 0018-9286. doi : 10.1109/TAC.1983.1103156 (cf. p. 85).

Wu, Q. (2012). “Sensitivity Analysis for Functional Structural Plant Modelling”. Thèse de doct. École Centrale Des Arts et Manufactures(Ecole Centrale Paris) (cf. p. 82, 83).

Wu, Qiongli et Paul-Henry Cournède (2014). “A comprehensive methodology of global sensitivity analysis for complex mechanistic models with an application to plant growth”. In : *Ecological Complexity* 20, p. 219 –232. issn : 1476-945X. doi : http://dx.doi.org/10.1016/j.ecocom.2013.12.005. url : http://www.sciencedirect.com/science/article/pii/S1476945X13001062 (cf. p. 83).

Wu, Qiongli, Benoît Bayol, Fenni Kang, Jérémie Lecoeur et Paul-Henry Cournède (2013). “Sensitivity Analysis for Plant Models with Correlated Parameters : Application to the Characterization of Sun flower Genotypes”. In : *7th International Conference on Sensitivity Analysis of Model Output*. Nice, France, p. ... url : https://hal.archives-ouvertes.fr/hal-00826104 (cf. p. 82, 128).

Zeigler, Bernard P. (1976). *Theory of Modeling and Simulation*. John Wiley (cf. p. 10, 129, 130).

Zeigler, Bernard P., Tag Gon Kim et Herbert Praehofer (2000). *Theory of Modeling and Simulation*. 2nd. Orlando, FL, USA : Academic Press, Inc. isbn : 0127784551 (cf. p. 17, 130, 133).

Zeigler, BernardP., HaeSang Song, TagGon Kim et Herbert Praehofer (1995). “DEVS framework for modelling, simulation, analysis, and design of hybrid systems”. English. In : *Hybrid Systems II*. Sous la dir. de Panos Antsaklis, Wolf Kohn, Anil Nerode et Shankar Sastry. T. 999. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 529–551. isbn : 978-3-540-60472-3. doi : 10.1007/3-540-60472-3_27. url : http://dx.doi.org/10.1007/3-540-60472-3_27 (cf. p. 17, 31, 40, 131).

Index