ISIMA
INSTITUT D'INFORMATIQUE
Université Clermont Auvergne

INRAE

Rapport de stage – ISIMA 2éme année
Filière : Modélisation Mathématique et Science de Données

Amélioration d'un outil de calibration et son utilisation

Présenté par : Aafaf ARHARAS

Responsables entreprise : M. Loïc MANCEAU
M. Pierre MARTRE

Avril 2021 – Aout 2021

Responsables ISIMA : M. gilles LEBORGNE

Date de soutenance : 02/09/2021

Remerciements :

Je tiens avant tout à remercier mes responsables de stage Monsieur Loïc MANCEAU et Monsieur Pierre MARTRE d'avoir accepté de m'accompagner durant ce stage ainsi que pour leur aide, leur disponibilité et leur implication dans mon travail.

Je remercie aussi Monsieur Gilles LEBORGNE mon tuteur académique de l'ISIMA pour sa disponibilité.

Je remercie également toutes les personnes de l'INRAE qui m'ont aidé de près ou de loin dans la réalisation de ce travail, Notamment Monsieur Samuel Buis qui m'a apporté une grande aide et des explications précieuses pour l'avancement du projet.

Je tiens finalement à remercier profondément tout le cadre professoral de l'ISIMA pour la formation de qualité qu'ils m'assurent. Je souhaite que le travail réalisé soit à la hauteur de leurs espérances ainsi qu'aux attentes de mes encadrants et le jury.

Glossaire :

Heuristique : une méthode de calcul qui fournit rapidement une solution réalisable qui n'est pas nécessairement optimale, pour un problème d'optimisation difficile.

Simplexe : une généralisation du triangle à une dimension quelconque. Plus exactement, un simplexe est l'enveloppe convexe d'un ensemble de (n+1) points utilisé pour former un repère affine dans un espace affine de dimension n.

Temps thermique : une mesure empirique utilisée pour calculer l'accumulation de chaleur qui sert à estimer la durée d'un développement biologique.

Trichome : Le trichome concerne toutes les parties d'une plante : végétatives (feuilles, tiges, racines) et reproductives (sépales, pétales, étamines, graines et fruits)

Fonctions non régulières : elles comprennent les fonctions non différenciables et discontinues.

Méthodes MCMC : sont une classe de méthodes d'échantillonnage à partir de distributions de probabilité. Ces méthodes de Monte-Carlo se basent sur le parcours de chaînes de Markov qui ont pour lois stationnaires les distributions à échantillonner.

Langage R : un langage de programmation destiné aux statistiques et à la science des données soutenu par la R Foundation for Statistical Computing. Il fait partie de la liste des paquets GNU et il est écrit en C (langage), Fortran et R.

RStudio : un environnement de développement gratuit, libre et multiplateforme pour R.

Markdown : une syntaxe simplifiée pour mettre en forme des documents contenant à la fois du texte, des instructions R et le résultat fourni par R.

Package R : une collection de fonctions R. Le répertoire où les packages sont stockés est appelé Library. R est livré avec un ensemble standard de packages.

Modèle de culture : une représentation ou une simplification du fonctionnement réel du système sol-plante en interaction avec le climat et les opérations techniques.

Un wrapper : un programme dont la fonction principale est d'appeler une autre fonction.

Table des illustrations :

Liste des abréviations :

INRAE : l'Institut national de recherche pour l'agriculture, l'alimentation et l'environnement

LEPSE : Laboratoire d'écophysiologie des plantes sous Stress Environnementaux

MAGE : Modélisation et Analyse de l'interaction Génotype Environnement

SQ : SiriusQuality

LAI : Leaf Area Index (indice de surface foliaire)

GAI : Green Area Index (indice foliaire vert)

P : Phyllochrone

PLDAE : Emergence date (date d'émergence)

ADAT : Anthesis date (date de floraison)

HDATE : Harvest date (date de récolte)

Résumé :

Les modèles de simulation peuvent être utilisés pour analyser le fonctionnement des plantes dans un contexte de changements climatiques, ainsi que la variabilité génétique faces à ces changements. Pour mener ce type de recherches, L'INRAE a développé le modèle de culture **SiriusQuality.**

Les prédictions de ce modèle dépendent d'un jeu de paramètres dont il faut connaitre la valeur a priori. Les valeurs de ces paramètres peuvent être soit mesurées soit calibrées en fonction d'observations. C'est ainsi que L'INRAE (UMR EMMAH, Avignon) a développé un outil **CroptimizR** dédié à l'estimation des paramètres et à l'analyse de sensibilité de modèles de culture. Il est basé sur une approche statistique Bayesienne et fréquentiste et il permet d'implémenter différentes méthodes (SIMPLEX, DREAMz…).

Le stage a porté sur l'amélioration de cet outil et son utilisation avec SiriusQuality. Pour atteindre cet objectif, J'ai travaillé sur l'optimisation du code existant pour en faire un package R facile d'utilisation. Enfin, une étude pratique a été réalisé pour tester le fonctionnement du package.

Mots-clés : modèle de simulation, estimation de paramètres, package R, SiriusQuality, CroptimizR, statistique bayésienne, statistique fréqentiste.

Abstract :

Models of simulation can be used to analyze how plants develop themselves in the context of climate change, as well as the genetic variability in response to these changes. To carry out this type of research, INRAE has developed the SiriusQuality culture model.

The predictions of this model depend on a set of parameters whose an a priori value must be known. The values of these parameters can be measured or calibrated according to observations. Thus, INRAE (UMR EMMAH, Avignon) has developed a CroptimizR tool dedicated to parameter estimation and sensitivity analysis of crop models. It is based on a bayesian and frequentist statistical approach and allows different methods to be implemented (SIMPLEX, DREAMz, etc.).

The internship has as objective to improve this tool and using it with SiriusQuality. To achieve this goal, I worked on optimizing the existing code to make it an easy-to-use R package. Finally, a practical study was carried out to test the functionality of the package.

Keywords: simulation model, parameter estimation, R package, SiriusQuality, CroptimizR, bayesian statistics, frequentist statistics.

Table des Matières :

Introduction :

Dans le cadre de la deuxième année du cycle ingénieur à l'école ISIMA et afin d'acquérir des compétences dans le milieu professionnel, j'ai effectué mon stage de cinq mois au sein de l'équipe MAGE de l'Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement (INRAE) de Montpellier.

L'objectif de ce stage a été l'amélioration d'un outil de calibration et son utilisation avec SiriusQuality qui est un modèle de culture développé par l'INRAE. J'ai dû concevoir un package R qui permet l'estimation des paramètres de ce modèle en se basant sur des approches statistiques. Enfin, il m'a fallu réaliser une étude de cas pratique qui sert comme test de fonctionnement du package.

Ce rapport se présente comme il suit : La première partie est consacrée à une introduction générale, à la présentation de l'entreprise et d'une manière générale comment s'est déroulé le stage. Et aussi, je présenterai le contexte, l'objectif et la problématique du sujet, ainsi que le planning mis en place pour réaliser l'étude. La deuxième partie concerne le travail réalisé. Elle est divisée en différentes sections : l'amélioration du code wrapper, la création du package SQoptimizR, l'étude de cas et enfin les tests effectués. Finalement, La dernière partie qui comprend les discussions des résultats ainsi que les perspectives du travail.

Partie 1 : EXPOSITION DU CONTEXTE INITIAL

1.1- Présentation du cadre :

1.1.1- L'INRAE :

INRAE (Institut national de recherche pour l'agriculture, l'alimentation et l'environnement) a été fondé le 1er janvier 2020, issu de la fusion entre l'INRA (Institut national de la recherche agronomique, fondé en 1946) et IRSTEA (institut national de recherche en sciences et technologies pour l'environnement et l'agriculture).

C'est le premier organisme spécialisé dans les domaines suivants : agriculture, alimentation et environnement. INRAE contribue à relever les défis relatifs à ces domaines en proposant par la recherche, l'innovation et l'appui aux politiques publiques de nouvelles orientations pour accompagner l'émergence des systèmes agricoles et alimentaires durables. INRAE met en œuvre une recherche finalisée, associant science fondamentale et appliquée, approches disciplinaires et interdisciplinaires grâce à la richesse de ses collectifs de recherche: 18 centres de recherche au cœur de dynamiques régionales et 14 départements scientifiques, 20 centres de recherche et plus de 150 implantations dans lesquels travaillent 1837 chercheurs, 2590 ingénieurs, 4061 techniciens et administratifs, 2000 doctorants, 1800 chercheurs et étudiants étrangers accueillis chaque année dans les laboratoires.

L'institut dispose d'infrastructures de recherche et d'unités expérimentales uniques en Europe. INRAE adopte résolument les démarches de science ouverte et participative associant toujours plus des citoyens. Il collabore avec les meilleurs équipes Européens et mondiales grâce à son réseau international. Le budget est de 882 millions d'euros, dont 77% financé par le ministère de la Recherche et 20% par d'autres instances publiques. L'objectif global sous-jacent de toute recherche à INRAE est de nourrir le monde de manière saine et durable.

1.1.2- LEPSE :

Plus précisément mon stage a eu lieu à l'UMR LEPSE (Laboratoire d'écophysiologie des plantes sous Stress Environnementaux, https://www6.montpellier.inrae.fr/lepse). C'est une unité mixte de recherche (INRAE-SupAgro Montpellier) dépendant du Département Agroécosystèmes et Biologie et Amélioration des Plantes d'INRAE. L'unité se situe au sein de l'IBIP (l'institut de Biologie Intégrative des Plantes), là où d'autres unités existent aussi comme le B&PMP (l'unité de Biochimie et Physiologie Moléculaire des Plantes)

Le LEPSE voit le jour en 1993 au sein de cet institut, dans le but de produire et améliorer les méthodologies visant l'amélioration de la productivité des plantes en tenant compte des contraintes environnementales. Dirigé par Pierre Martre, directeur de recherches INRAE, le laboratoire mène des travaux d'accompagnement de l'émergence de nouvelles méthodes d'agriculture, avec des critères d'économie et respect de l'environnement en tenant compte du changement climatique. En passant par l'identification des caractéristiques des plantes favorables à la production en condition de déficit hydrique, la stratégie vise la mise en

conditions des espèces végétales (sécheresse, températures élevées…) afin de bien saisir le comportement et aussi les performances des modes de cultures actuelles.

Organigramme du LEPSE :

L'UMR LEPSE regroupe actuellement 13 chercheurs dont 2 enseignants-chercheurs, 13 ingénieurs, techniciens, et administratifs et accueille autant de CDD (post-doctorats, doctorants, stagiaires...). Le laboratoire est composé de 3 équipes scientifiques autour de 3 grands axes majeurs de la recherche :

- L'équipe MAGE : « Modélisation et Analyse de l'interaction Génotype Environnement »
- L'équipe ETAP : « Efficience de transpiration et adaptation des plantes aux climats secs »
- L'équipe M3P-DEV :« Méthodes & Développement sur les Plateformes de Phénotypage des Plantes »

A ces équipes scientifiques, s'ajoutent 2 équipes techniques :

- M3P –EXP : « Appui expérimental sur les Plateformes de Phénotypage des Plantes »
- L'équipe IE2M :« Infrastructure Equipements, Méthodes & Mesures»La direction de l'unité est assistée d'un comité de direction (CoDir) et de services administratifs et informatiques.

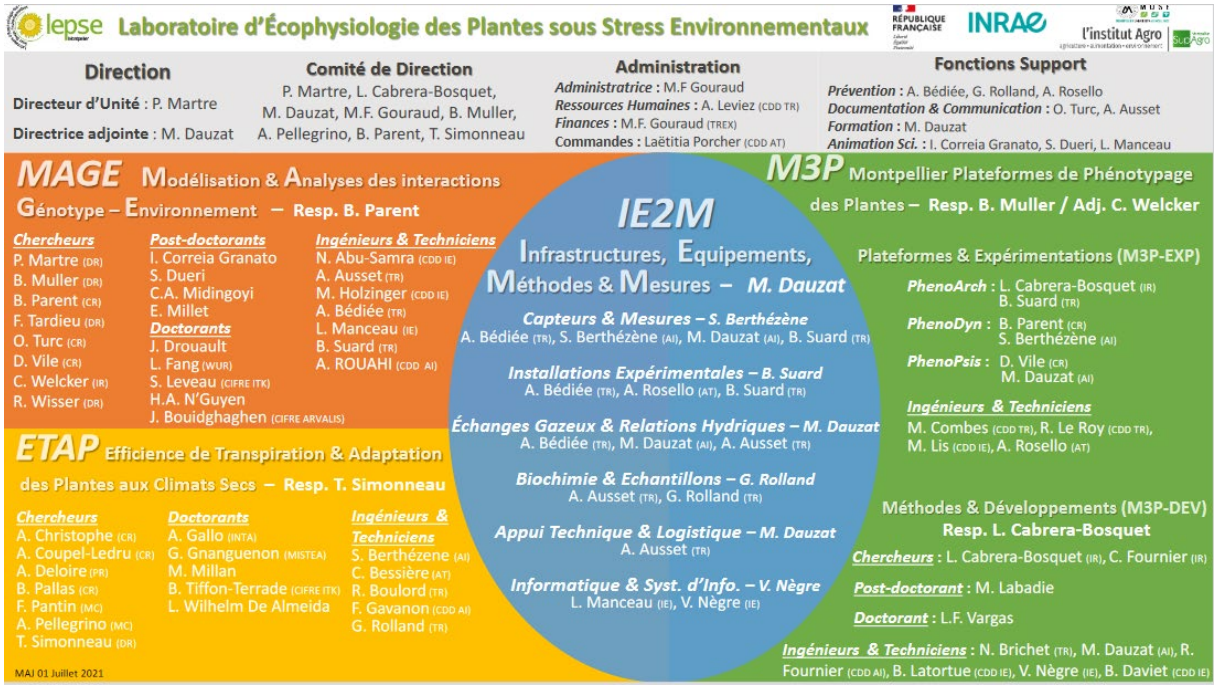


Figure 1 : Organigramme du LEPSE

1.1.3- L'équipe MAGE :

J'ai pu mener mon stage au sein de L'équipe MAGE (Modélisation et Analyse de l'interaction Génotype Environnement). Elle adopte une approche innovante faisant appel à la génétique quantitative, la dissection écophysiologique, le phénotypage à haut débit et la modélisation des cultures.

Cette équipe travaille sur les sujets suivants :

1)- L'analyse des profils temporels par phénotypage à haut débit des vitesses d'expansion (soies de maïs, feuilles, parties aériennes de plantes entières,…) et développement des modèles écophysiologiques à l'échelle du processus.

2)- L'analyse de la variabilité génétique de ces processus

3)- Développement de modèles de cultures (SiriusQuality pour le blé et le maïs), notamment pour analyser les conséquences de la variabilité génétique sur la production végétale dans divers scénarios.

1.2- La modélisation agronomique :

Dans le domaine de l'agronomie, les chercheurs font recours de plus en plus à la modélisation afin de bien comprendre le comportement des cultures et leur interaction avec les différentes composantes de leur environnement, comme le climat ou l'être humain... Et ce dans le but de sécuriser la production agricole qui en effet la base de l'alimentation mondiale. Ces modèles peuvent aussi être utilisés par les agriculteurs eux-mêmes pour développer ou bien adapter leur décision.

Un modèle de culture est composé d'un ensemble d'équations décrivant le comportement du système sol-plante et aussi ses interactions avec le climat et les pratiques agricoles.

De nos jours, la modélisation devient primordiale dans le domaine. Avec le développement des méthodes informatiques, ces outils sont devenus incontournables permettant ainsi de comprendre le fonctionnement des cultures et de développer de nouvelles solutions bien adaptées à la production.

1.3- Le sujet :

1.3.1- SiriusQuality :

SiriusQuality (SQ) est un modèle de culture développé par l'INRA. Il est basé sur des processus écophysiologiques qui simule la phénologie, la mise en place de la surface foliaire et les rendements ainsi que les flux d'eau, d'azote et de carbone dans le continuum sol-plante-atmosphère en réponse aux facteurs de l'environnement et de l'itinéraire technique.

SQ a été développé et calibré pour plusieurs variétés de blé d'hiver et de printemps et de blé dur. La généricité des formalismes permet de simuler la croissance de la plupart des

espèces de céréales à paille, dont l'orge, l'avoine, le riz, et le triticale. Récemment une version maïs du modèle a été développée.

SQ fournit une interface graphique conviviale qui propose plusieurs outils pour la configuration de simulations complexes et pour la calibration (étalonnage du modèle) et l'optimisation (recherche d'idiotypes culturaux) des paramètres ou l'analyse de sensibilité et d'incertitude.

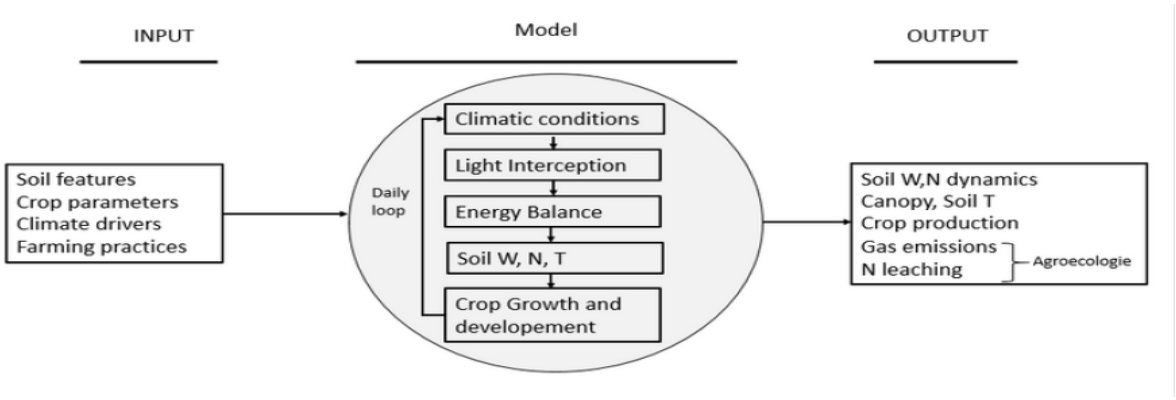


Figure 2 : Schéma de fonctionnement de SiriusQuality

SiriusQuality a comme entrées :

- **Soil features :** les paramètres qui décrivent la nature du sol (par exemple : le contenu initial en eau et en azote dans chaque couche de sol)
- **Crop parameters :** les paramètres variétaux et non variétaux,
- **Climate drivers :** les paramètres du climat : la température minimum et maximum de l'air, la vitesse du vent, la quantité de pluie, ...
- **Farming practices :** les paramètres décrivant les pratiques agricoles.

Le modèle est basé sur des processus qui permettent de simuler la production agricole. En effet, l'évaluation de la surface foliaire permet de calculer l'interception de la lumière par le couvert et ainsi la température du couvert et celle du sol (composant Energy Balance). Celles-ci agissent directement sur le rythme de croissance de la plante (Phénologie) et sur sa transpiration. La lumière interceptée est aussi convertie en matière sèche (poids de la plante ou des grains) grâce au phénomène de photosynthèse. La transpiration de la plante impacte la dynamique de l'eau et de l'azote transporté par l'eau dans le sol et finalement la disponibilité des réserves.

En sorti du modèle l'utilisateur peut choisir d'exporter un grand nombre de variables dont le rendement (ou poids du grain à maturité) qui est important pour la sécurité alimentaire, la surface foliaire ou LAI (Leaf Area Index) ou encore l'azote et les gaz à effet de serre perdus par le sol qui sont importants pour évaluer l'impact de la culture sur l'environnement.

Les fichiers d'entrée et de sortie de SiriusQuality sont enregistré dans 5 dossiers : "1-Project", "2-WeatherData", "3-Output", "5-Optimization", "6-Observation"

- **« 1-Project »** : il contient les répertoires et les principaux fichiers d'entrée.

- **« 2-WheatherData »** : il contient des fichiers ASCII pour les paramètres décrivant le climat (la vitesse du vent, la quantité de pluie, la température minimale et maximale de l'air, …)

- **« 3-Output »** : il sert à stocker Les fichiers de sortie (résultats de simulation)

- **« 6-Observation »** : il contient les fichiers d'observation obligatoires pour l'optimisation.

- **« 5-Optimization »** : il sert à enregistrer les résultats de la procédure d'optimisation (les plots, les résultats...)

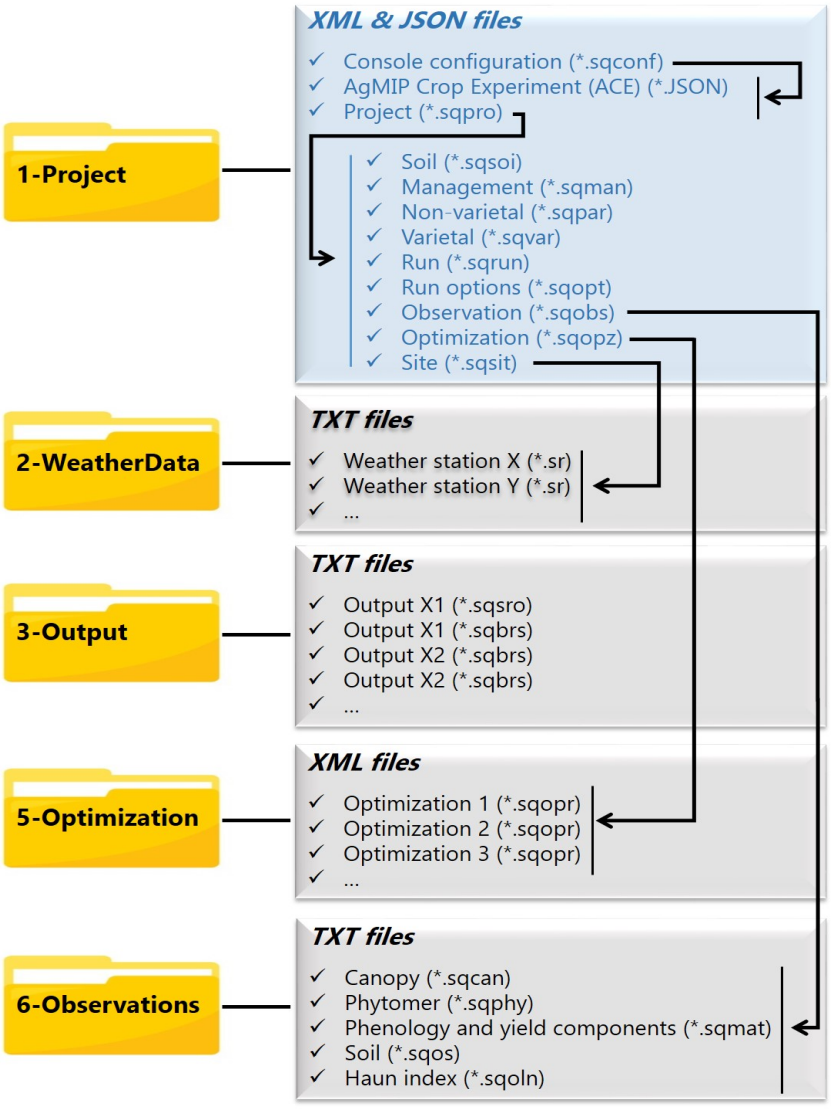


Figure 3 : Schéma des fichiers d'entrée et de sortie de SQ

Il faut mentionner que SQ est également exécutable en mode console (ligne de commande).

Voici un exemple :

SiriusQuality-Console.exe -simoverride true true C:/SQ_Release/1-Project/Project.sqpro --Var [Yecora Rojo] AreaPL 30 EarGR 100 [Variety2] par3 val3… --NonVar [Spring_Wheat] AMNLFNO 7 –Soil [Maricopa] CtoN 10 –Site [MARA-92-93] MinSowingDate 1992-11-01 --Man [901-Dry-High_N-Ambient_CO2] SoilWaterDeficit 170

Après le chemin du fichier de projet *(C:/SQ_Release/1-Project/Project.sqpro)* on trouve le type de paramètre à modifier *(Varietal, Non-Varietal, Soil, Site ou Management)* précédé de deux tirets *(« -- »)* et l'élément *(item)* sur lequel doit avoir lieu la modification (p. ex., un nom de variété) écrit entre crochets. Le nom du paramètre à surcharger et sa valeur sont écrits après le nom de l'élément *(par3 val3).*

SiriusQuality fournit également la possibilité d'optimiser les paramètres du modèle. Le réglage de l'optimisation (par ex. l'algorithme de la méthode ou la fonction fitness) se fait via l'onglet suivant :

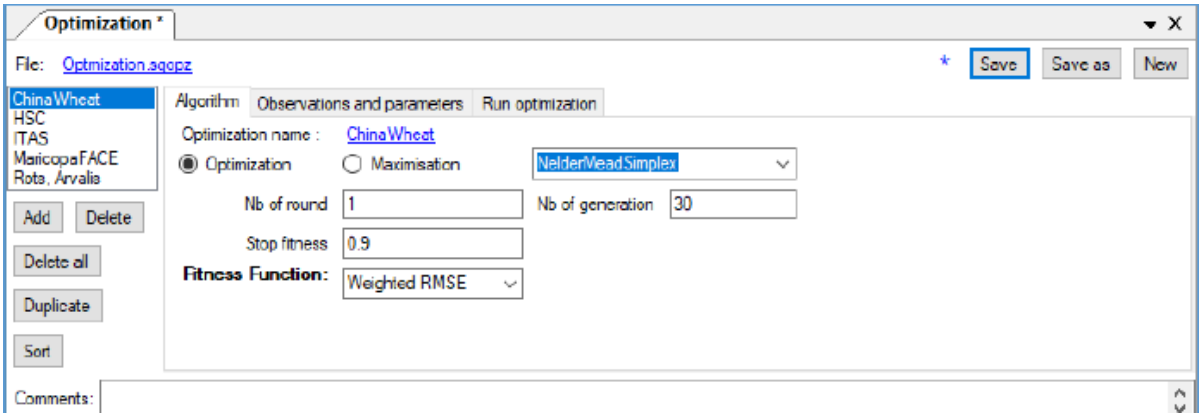


Figure 4 : l'onglet Optimization de SiriusQuality

L'optimisation repose sur les données observées qui sont chargées dans l'onglet :

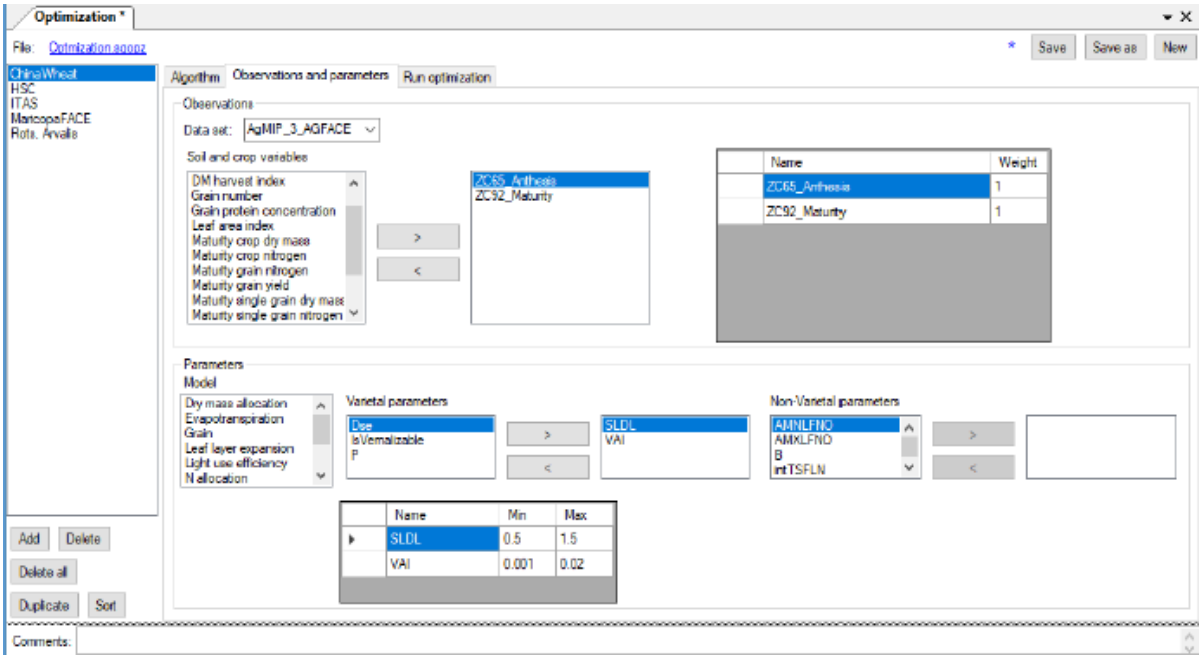


Figure 5 : le sous onglet 'Observations and parameters' de l'onglet Optimization

Il s'agit donc de faire un grand nombre de simulations avec des valeurs de différents paramètres et de ne retenir que celles pour lesquelles les observations sont le mieux reproduites. Or, l'optimisation fournit par SiriusQuality n'est pas performante. Entre autres problème, elle est très lente et utilise seulement des méthodes d'optimisation fréquentistes inadaptées qui aboutissent souvent à un minimum local qui ne reflète pas la réalité. C'est ainsi qu'on a besoin de CroptimizR.

1.3.2- CroptimizR :

C'est un package Développé en R par l'unité EMMAH une unité mixte de recherche (UMR) entre l'Université d'Avignon et l'INRAE. Il est dédié à l'analyse probabiliste de l'incertitude, à l'analyse de sensibilité et à l'estimation des paramètres pour les modèles de cultures.
Pour le moment, seules les fonctions d'estimation des paramètres ont été développées.

```
estim_param(obs_list, crit_function = crit_log_cwss, model_function,
  model_options = NULL, optim_method = "nloptr.simplex", optim_options,
  param_info, transform_obs = NULL, transform_sim = NULL,
  satisfy_par_const = NULL, var_names = NULL)
```

Figure 6 : Prototype de la fonction estim_param

Estim_param est la principale fonction qui permet l'estimation des paramètres à l'aide de différentes méthodes en se basant sur :

- Une approche fréquentiste :

Elle repose sur la loi des observations et consiste à trouver la distribution la plus probable au vue des données.

✓ **Nelder-Mead simplex :**

La méthode de Nelder-Mead, Appelée parfois Méthode du simplexe. C'est un algorithme d'optimisation non linéaire qui a été publiée par John Nelder et Roger Mead en 1965. Il s'agit d'une méthode numérique fréquentiste heuristique* qui cherche à minimiser une fonction continue dans un espace à plusieurs dimensions.

Voici l'organigramme de l'algorithme :

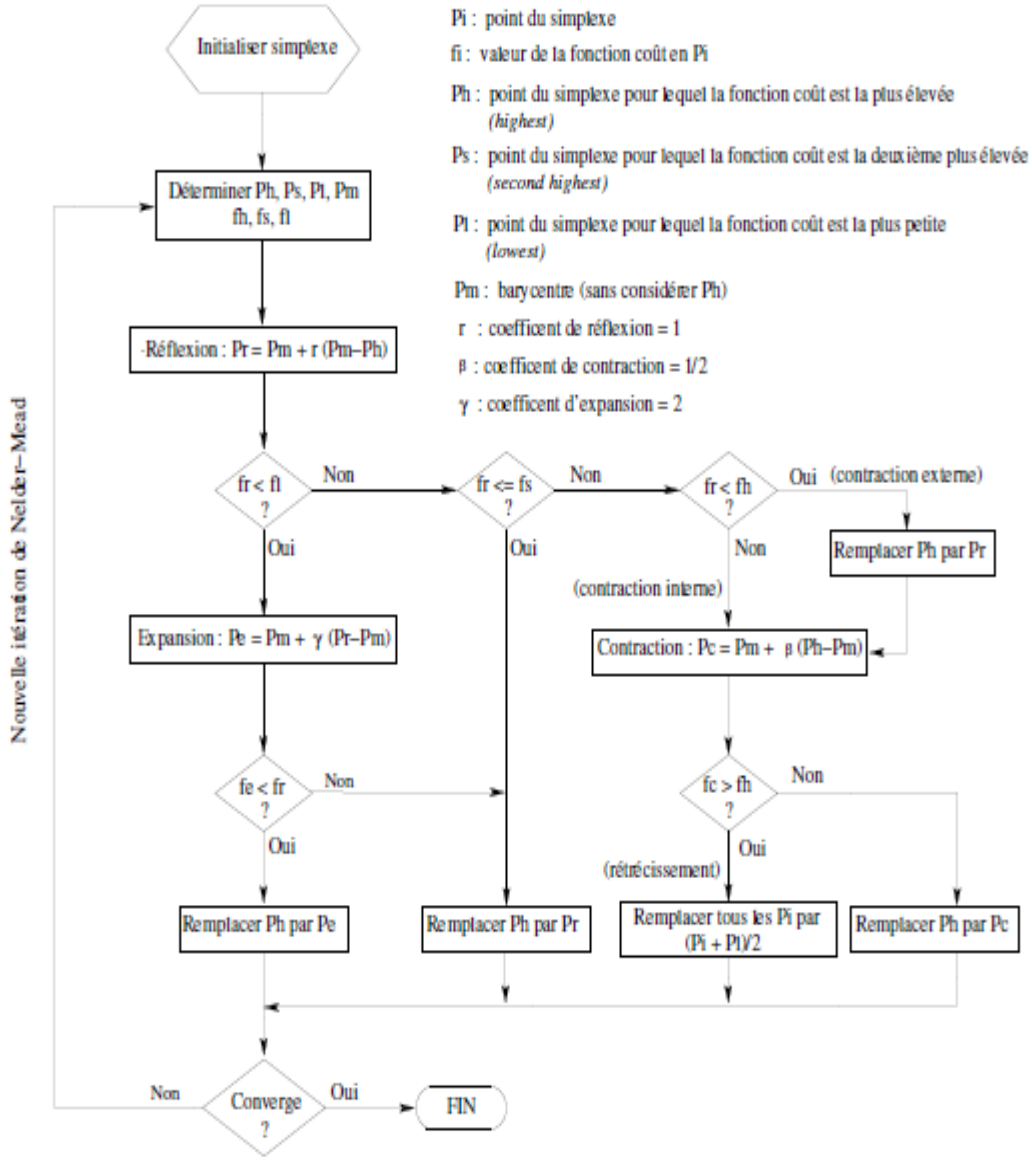


Figure 7 : L'organigramme de l'algorithme de Nelder-Mead simplex

Les modèles de culture présentent souvent des discontinuités qui rendent difficile l'utilisation de méthodes de minimisation par gradient (Gauss_Newton, Levenberg-Marquardt). CroptimizR utilise l'algorithme de Nelder-Mead Simplex qui est adapté aux fonctions non régulières* car la recherche de l'optimum n'y est pas basée sur le calcul du gradient de la fonction. Ainsi cette méthode d'optimisation fréquentiste est bien plus efficace que celle d'origine dans Sirius.

- Une approche bayésienne :

Elle se base sur le théorème de Bayes, elle combine l'information apportée par les données avec les connaissances a priori dans le but d'obtenir une information a posteriori. Cette approche fait intervenir trois concepts qui se recoupent :

- ✓ Un priori : une information obtenue d'une expérience précédente.
- ✓ Des preuves : les données de l'expérience actuelle.
- ✓ Un posteriori : l'information actualisée obtenue à partir de l'a priori et des preuves. C'est ce que l'on obtient par l'analyse bayésienne.

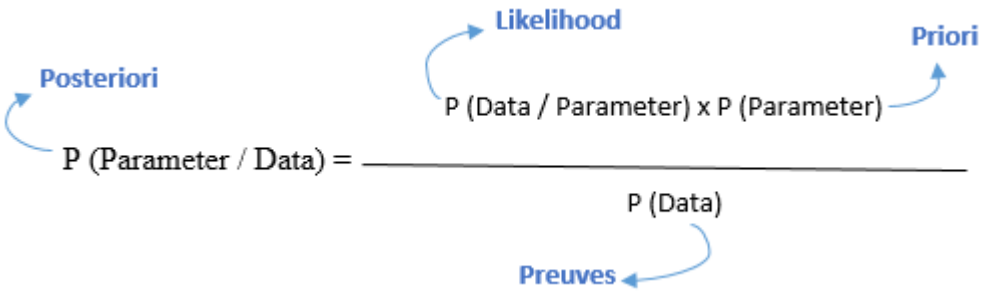


Figure 8 : Théorème de Bayes

- ✓ **DREAM-zs :**

C'est une méthode MCMC * (Monte-Carlo par chaînes de Markov) multi-chaîne qui est reconnue comme étant efficace pour des distributions cibles complexes, de grande dimension et multimodales. Elle permet de résoudre les problèmes d'estimation des paramètres postérieurs discrets, non continus et combinatoires.

Dans CroptimzR, Les paramètres estimés sont traités comme des variables aléatoires et on cherche à déterminer leur distribution de probabilité conjointe, appelée distribution postérieure en se basant sur le théorème de Bayes. Ainsi, l'a priori est la loi de distribution des paramètres que l'on veut estimer telle que l'on peut la définir avant de considérer les observations, i.e. avant l'estimation de ces paramètres. C'est souvent une loi uniforme car on a en général que peu d'informations sur l'incertitude des paramètres avant leur estimation. Dans CroptimizR on ne considère pour l'instant que des lois uniformes pour les a priori, qui sont définies à partir des bornes données dans param_info (lb et ub). Ce qui est appelé preuve (Evidence) est la loi de probabilité des observations. La loi a posteriori, c'est ce que l'on cherche à obtenir, i.e. la loi de probabilité des paramètres quand on prend en compte les observations.

Il faut noter que CroptimizR est générique. En effet, toutes les fonctionnalités peuvent être utilisées sur n'importe quel modèle de culture pour lequel un wrapper R peut être implémenté.

1.3.3- SQ_wrapper :

Pour lier le modèle SiriusQuality et le package CroptimizR il faut passer en argument d'estim_param :

- ✓ **Model_function** : la fonction wrapper du modèle de culture utilisé.

Pour Sirius Quality on utilise la fonction **SQ_wrapper**, voici un schéma pour comprendre le fonctionnement du wrapper :

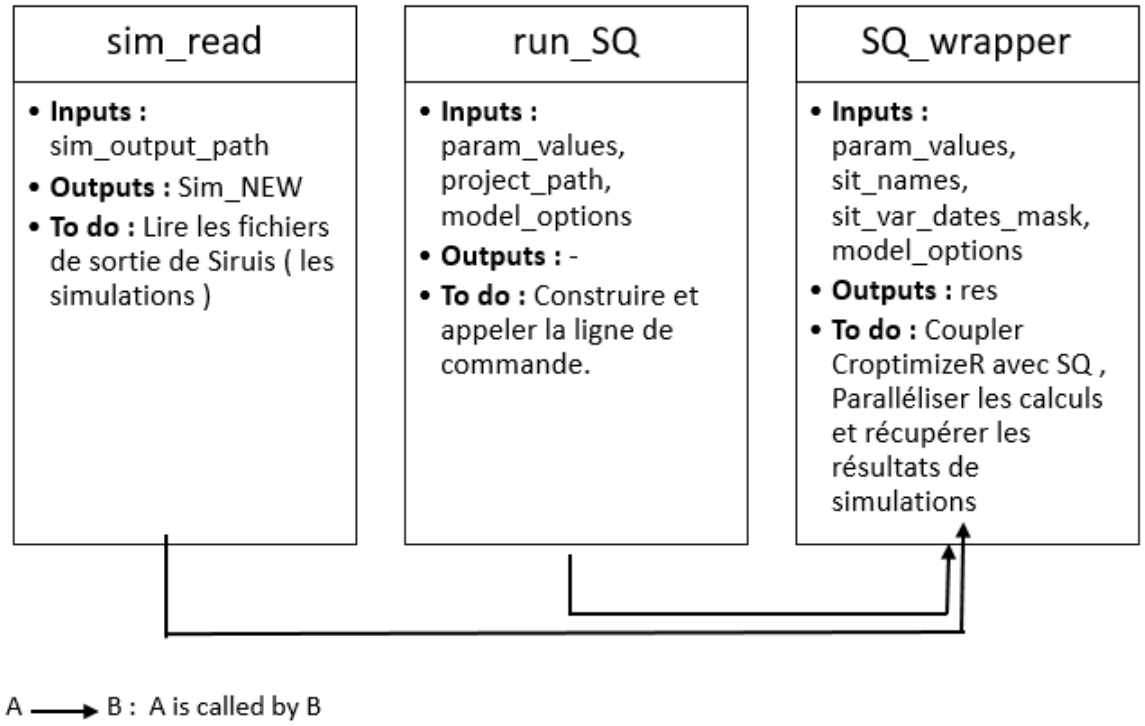


Figure 9 : Schéma d'appel des fonctions du SQ_wrapper

La fonction **SQ_wrapper** contient une boucle qui permet la parallélisation du calcul selon les situations. Ainsi, pour chaque situation la fonction **run_SQ** est appelée pour exécuter la ligne de commande pour les valeurs de param_values ce qui permet de générer les simulations via le modèle SiriusQuality. Ensuite, la fonction **sim_read** est appelée pour récupérer ces simulations. Finalement, les résultats de la boucle sont rassemblés dans le variable res sortie de SQ_wrapper. Il faut donc passer en arguments :

- ✓ **Sim_output_path :** le chemin vers les fichiers de sortie de SiruisQuality.
- ✓ **Project_path :** le chemin vers un projet (ensemble de fichiers des paramètres Management, Soil, Paramètres variétaux, non-variétaux, ..)

- ✓ **Param_values :** Les valeurs des paramètres à estimer qui sont définit dans les fichiers d'entrée de SQ.
- ✓ **Sit_names :** Les noms des situations à estimer.
- ✓ **Sit_var_dates_mask :** Variable pour filtrer les situations à traiter.
- ✓ **Model_options :** Liste des options pour le wrapper et elle doit contenir :

- time_display : booléan (TRUE/FALSE) pour l'affichage du temps de calcul

- cores_nb : Nombre de cœurs

- home.wd : le répertoire du R script

- batch_run : le répertoire qui contient les fichiers d'entrée

- Param_list : un data frame qui contient les informations sur les paramètres à estimer (nom et type)

- Variety : Nom de la variété à simuler (une seule variété pour le moment ["ANVERGUR"])

1.3.4- Objectifs et Problématiques :

Ce stage porte sur l'amélioration de CroptimizR et son utilisation avec SiriusQuality sur un cas pratique. Il s'agissait donc d'améliorer la fonction **SQ_wrapper** tout en optimisant le code existant pour le rendre plus générique et facile d'utilisation. Par ailleurs, Il faut en faire **un package R.** Ensuite, Il faut étudier un cas pratique en faisant tourner une optimisation sur un jeu de données (nommé Breed Wheat). Dans la deuxième partie, Il aurait dû être fait une étude de performance des différentes méthodes de parallélisation pour avoir un code plus rapide. Mais par manque de temps cette étude fera uniquement l'objet d'une partie perspectives dans la conclusion.

Le sujet amène différents problématiques, Quelle sous-fonction à modifier pour améliorer le wrapper ? Quelles sont les paramètres à garder et ceux à modifier ? Quelles sont les fonctions à avoir dans le package final ? Quelles sont les différentes méthodes de parallélisation sous R ?

1.3.5- Organisation du stage :

Après avoir pris en main le sujet de mon stage, il était indispensable de bien planifier le travail pour organiser le déroulement des étapes du projet dans le temps. J'ai opté pour le diagramme de GANTT, étant donné qu'il permet de visualiser simplement toutes les tâches planifiées d'un projet.

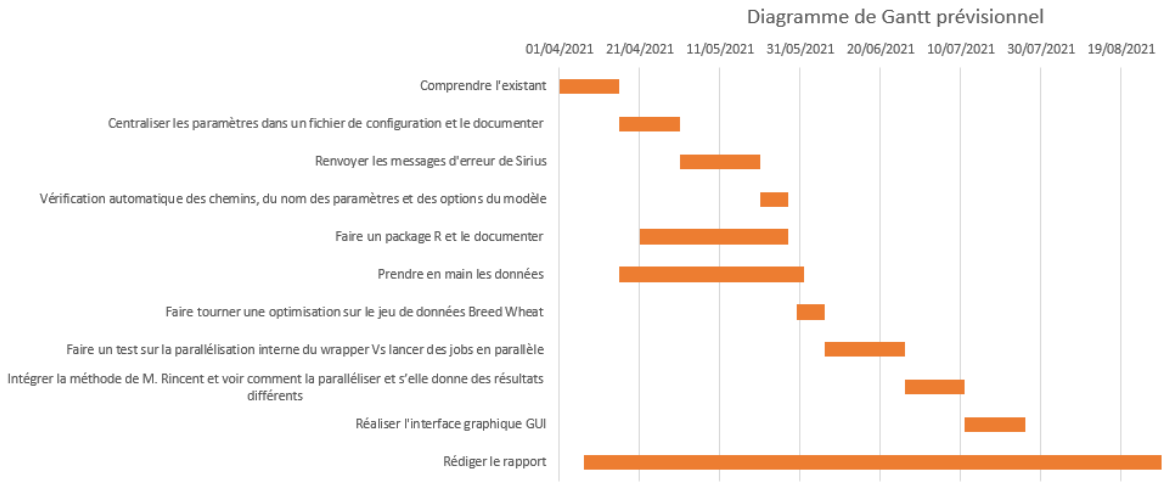


Figure 10 : Diagramme de Gantt prévisionnel

Dans un premier temps, pour bien se lancer dans le travail et puisque c'est la première fois que je découvert le langage R, Une mise à niveau en langage R été requise. De plus, les sous-tâches n'étaient pas bien définis au début puisque l'optimisation d'un code généralement se base sur le principe d'essai : On essaye de trouver la fonction qui améliore le plus notre code. Ainsi, par manque du temps certaines tâches sont annulées tandis que certaines sont ajoutées. Le nouveau diagramme de Gantt est le suivant :

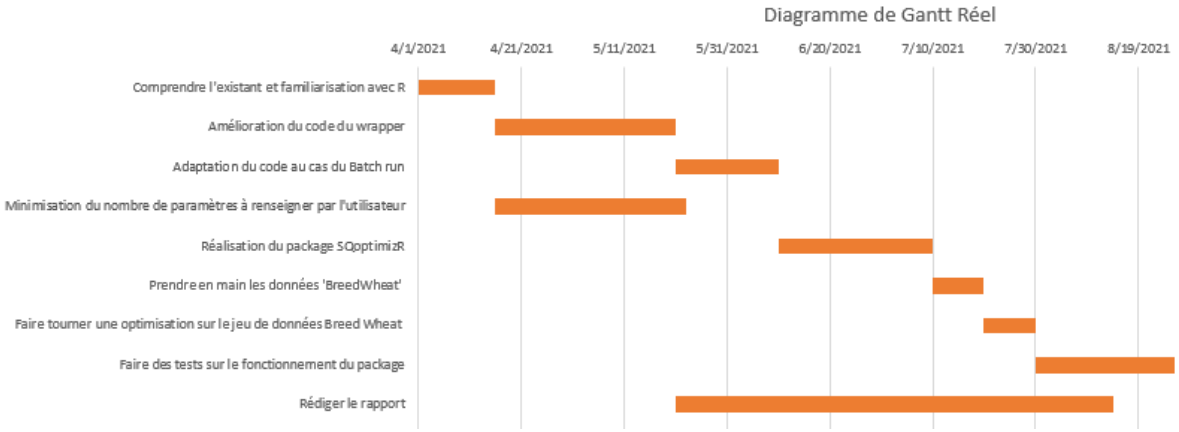


Figure 11 : Diagramme de Gantt réel

Partie 2 : MATERIEL ET METHODES

2.1- Amélioration du SQ_wrapper :

2.1.1- Code existant :

Il s'agit de trois scripts initiales :

Load_obs_mean.R : le script qui permet de lire le fichier des observations. C'est un fichier csv qui contient les observations de la variable LAI (leaf area index) pour 6 situations. Le LAI est une grandeur sans dimension qui exprime la surface foliaire d'une plante par unité de surface de sol. Il est déterminé par le calcul de l'intégralité des surfaces des feuilles des plantes sur la surface de sol que couvre ces plantes.

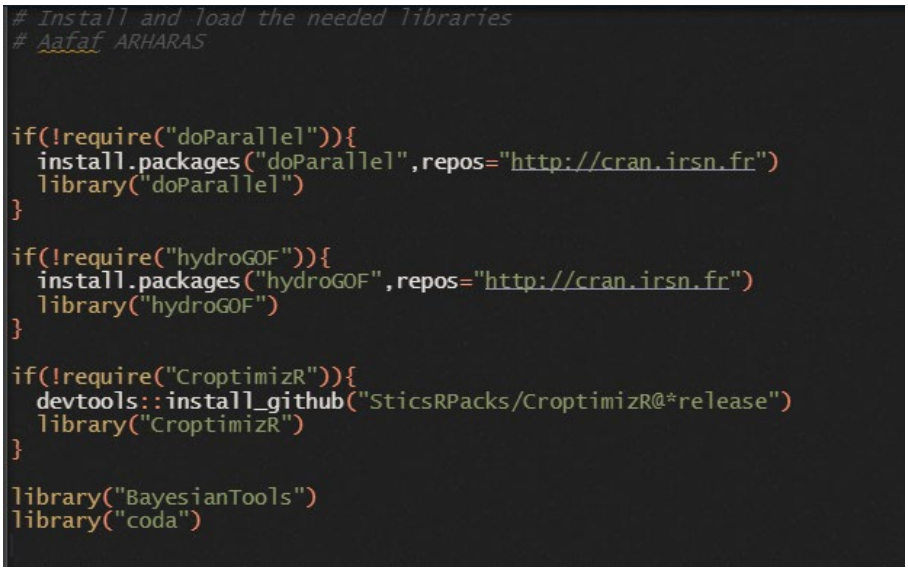
1	Treat;Year;Date;mean_LAI;Soil
2	T1;2019;03/30/19;0.205181411;SOERE_T1_2019
3	T2;2019;03/30/19;0.20503822;SOERE_T2_2019
4	T3;2019;03/30/19;0.20579336;SOERE_T3_2019
5	T4;2019;03/30/19;0.20411705;SOERE_T4_2019
6	T5;2019;03/30/19;0.202906388;SOERE_T5_2019
7	T6;2019;03/30/19;0.202601152;SOERE_T6_2019
8	T1;2019;04/21/19;0.327453372;SOERE_T1_2019
9	T2;2019;04/21/19;0.327065508;SOERE_T2_2019
10	T3;2019;04/21/19;0.328040754;SOERE_T3_2019
11	T4;2019;04/21/19;0.329642715;SOERE_T4_2019
12	T5;2019;04/21/19;0.324093934;SOERE_T5_2019
13	T6;2019;04/21/19;0.325167466;SOERE_T6_2019
14	T1;2019;06/28/19;2.590614657;SOERE_T1_2019

Figure 12 : Extrait du fichier observations Mons

SQ_model_Parallel.R : C'est le script qui contient les fonctions SQ_wrapper, run_SQ et sim_read (voir 1.3.3)

SQ_final_mean.Rmd : C'est le markdown* qui permet d'installer et charger les bibliothèques utilisés, définir les paramètres nécessaires et finalement faire tourner une optimisation.

Ainsi, La première étape à faire a été d'organiser les scripts qui existent et de mieux les séparer. D'abord J'ai mis les instructions qui permettent l'installation des bibliothèques dont on a besoin dans un script **Libraries.R**



```
# Install and load the needed libraries
# Aafaf ARHARAS


if(!require("doParallel")){
  install.packages("doParallel",repos="http://cran.irsn.fr")
  library("doParallel")
}

if(!require("hydroGOF")){
  install.packages("hydroGOF",repos="http://cran.irsn.fr")
  library("hydroGOF")
}

if(!require("CroptimizR")){
  devtools::install_github("SticsRPacks/CroptimizR@*release")
  library("CroptimizR")
}

library("BayesianTools")
library("coda")
```

Figure 13 : le script Libraries.R

Ensuite, J'ai centralisé les paramètres à renseigner par l'utilisateur dans un script **Parameters.R.** De plus, j'ai rassemblé tous les fonctions que je viens de créer et qui servent à améliorer le wrapper dans un script **functions.R**

2.1.2- Amélioration de Sim_read :

Sim_read est la fonction qui permet de lire les fichiers de sortie de SiriusQuality. Ils sont enregistrés dans le répertoire **3-Output**, pour chaque situation on trouve deux fichiers :

- Un fichier des simulations journalières (Daily) avec l'extension «.sqsro »
- Un fichier des simulations saisonnières (Summary) avec l'extension « .sqbrs »

	A	B	C	D	E	F	
1	SiriusQuality2.5 output file	8/3/2021	14:06:41		build:3.1.0.0		
2	Project	C:\Users\arharasa\Documents\Stage_Aafaf\Breedwheat\1-Project\Breedwheat.sqpro					
3	Management	Breedwheat Biotech panel trials for model calibration (T1.3)_2021-07-15_Management.sqman					
4	Non-varietal parameters	AgMIP_4_GLOBAL - Copie.sqpar					
5	Non-varietal global parameters	?					
6	Run options	v1.5-RunOption.sqopt					
7	Site	Breedwheat Biotech panel trials for model calibration (T1.3)_2021-07-15_Site.sqsit					
8	Soil	Breedwheat Biotech panel trials for model calibration (T1.3)_2021-07-15_Soil.sqsoi					
9	Varietal parameters	AgMIP_4_GLOBAL - Copie.sqvar					
10							
11	Run ID (text)	Treatment name (text)	Crop (text)	Run options (text)	Site name (text)	Soil name (text)	Genoty
12	RUID	TRT_NAME	CRID	RUN_OPT	SITE_NAME	SOIL_NAME	CUL_ID
13	ARVgre2012IR_ALLEZ_Y	ARVgre2012IR	Winter_Wheat	Default	ARV451GRE20112442012274	ARVgre	ALLEZ_
14							
15							

Figure 14 : Exemple du fichier des simulations saisonnières

	A	B	C
31	Date	Green area	
32		index	
33	yyyy-mm-dd	m²/m²	
34	DATE	GAID	
35	10/18/2017	0	
36	10/19/2017	0	
37	10/20/2017	0	
38	10/21/2017	0	
39	10/22/2017	0	
40	10/23/2017	0	
41	10/24/2017	0	
42	10/25/2017	0	
43	10/26/2017	0.01	
44	10/27/2017	0.02	
45	10/28/2017	0.02	
46	10/29/2017	0.02	
47	10/30/2017	0.03	
48	10/31/2017	0.03	
49	11/1/2017	0.04	
50	11/2/2017	0.05	
51	11/3/2017	0.06	
52	11/4/2017	0.08	
53	11/5/2017	0.08	

Figure 15 : Exemple du fichier des simulations journalières

La fonction qu'on avait avant permet de lire les simulations d'une seule variable : « GAID »

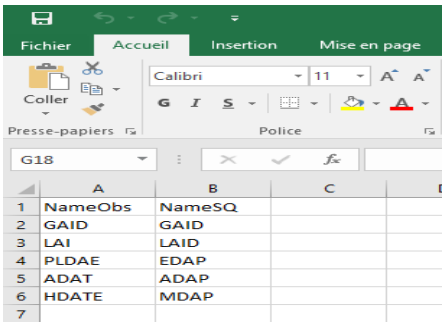
```
Data      <- data.frame(Date = as.Date(Sim_dyn$DATE), GAI = as.numeric(Sim_dyn$GAID))
```

J'ai donc modifié la fonction pour qu'elle puisse lire les simulations de n'importe quelle variable. Pour cela j'ai introduit deux paramètres à définir par l'utilisateur :
dailyVar : un vecteur des noms des variables du fichier journalière à simuler.
sumVar : un vecteur des noms des variables du fichier saisonnier à simuler.

Ces deux paramètres sont utilisés pour construire :

- ✓ Var_names : le vecteur des noms (utilisés dans les fichiers des observations) des variables à simuler, ce qui est fourni par l'utilisateur. Or, les codes attribués aux noms des variables sur lesquels est basée la lecture des données dans les fichiers des observations sont différents de ceux utilisés dans les fichiers de simulations. C'est ainsi qu'on a besoin d'un autre paramètre Var_simule.

- ✓ Var_simule : le vecteur des noms (utilisés dans les fichiers de sortie de SQ) des variables à simuler. Il est construit à partir d'un fichier Excel qui fait le lien entre les noms des fichiers d'observations et ceux de simulations.



	A	B	C
1	NameObs	NameSQ	
2	GAID	GAID	
3	LAI	LAID	
4	PLDAE	EDAP	
5	ADAT	ADAP	
6	HDATE	MDAP	
7			

Figure 16 : Extrait du fichier « correspondance entre les codes observation et simulation »

Var_names et Var_simule sont des éléments de la liste model_options. Ainsi la fonction devient :

sim_read
Input : sim_output_path
Output : Sim_NEW
lire les simulations pour la variable GAID

→

sim_read
Input : sim_output_path
Output : Sim_NEW
lire les simulations pour la variable GAID

2.1.3- Amélioration de run_SQ :

Run_SQ est la fonction qui permet de construire et appeler la ligne de commande. La fonction qu'on avait avant permet de construire la ligne de commande pour optimiser les paramètres de sol uniquement :

```
Param_name_soil = Param_list$name[Param_list$type == "Soil"]
Param_val_soil  = param_values[Param_list$type == "Soil"]
N_soil = length(Param_name_soil)

soil_name<-paste("[",Soil,"]",sep="")
console_text = ""
console_text = paste(console_text,"--Soil",soil_name,sep = " ")

for (n in 1:N_soil){console_text = paste(console_text, Param_name_soil[n], Param_val_soil[n], sep = " ")}
```

Figure 17 : Extrait de la fonction run_SQ du code initial

Il faut noter que la ligne de commande doit être appelée pour chaque **situation**. Une situation représente la combinaison de plusieurs **items :** le site de culture (**Site**), l'espèce à cultiver **(NonVar)**, la variété à cultiver **(Var),** le sol à utiliser **(Soil)** et un item pour l'ensemble de pratiques agricoles **(Man)** qui sont toutes nécessaires pour simuler une saison de culture. Il faut mentionner aussi que plusieurs situations peuvent être regroupées en un **RUN** dans le cas où l'on veut simuler plusieurs saisons de culture ensemble.

Ainsi, notre objectif était d'avoir une fonction run_SQ qui est générique c'est-à-dire une fonction qui appelle la ligne de commande pour un RUN donné et qui change potentiellement les valeurs de paramètre pour les items suivant le choix de l'utilisateur. Pour cela on a besoin de la fonction **Set_runs_list** qui permet de construire à partir des fichiers d'entrée de SQ la liste des situations et des items de différentes variables associées aux runs choisis par l'utilisateur via les paramètres **runs2simulate** et **runs2notsimulate.** Les deux dernières variables sont rentrées par l'utilisateur alors que les autres variables sont lue dans les fichiers d'entrée Sirius.

Set_runs_list
Input : ▪ runs2simulate, ▪ runs2notsimulate, ▪ project_path
Output : runs_list

Exemple :

```
> print(runs_list)
[[1]]
     name
"RUN_all"

[[2]]
                situation   Soil       Var       NonVar          Man                          Site
1      ARVgre2012IR_APACHE ARVgre    APACHE Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
2       ARVgre2012IR_NOGAL ARVgre     NOGAL Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
3     ARVgre2012IR_CELLULE ARVgre   CELLULE Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
4     ARVgre2012IR_RUBISKO ARVgre   RUBISKO Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
5     ARVgre2012IR_CH_NARA ARVgre   CH_NARA Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
6     ARVgre2012IR_ALLEZ_Y ARVgre   ALLEZ_Y Winter_Wheat ARVgre2012IR      ARV451GRE20112442012274
7      ARVgre2012RF_APACHE ARVgre    APACHE Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
8       ARVgre2012RF_NOGAL ARVgre     NOGAL Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
9     ARVgre2012RF_RUBISKO ARVgre   RUBISKO Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
10    ARVgre2012RF_CELLULE ARVgre   CELLULE Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
11    ARVgre2012RF_CH_NARA ARVgre   CH_NARA Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
12    ARVgre2012RF_ALLEZ_Y ARVgre   ALLEZ_Y Winter_Wheat ARVgre2012RF      ARV451GRE20112442012274
13     ARVgre2014IR_APACHE ARVgre    APACHE Winter_Wheat ARVgre2014IR      ARV451GRE20132442014273
```

Figure 18 : Exemple de la liste 'runs_list' pour le run 'RUN_all'

On remarque qu'il y a pas mal d'items qui se répètent, pour enlever les doublons et avoir des listes prêtes à utiliser par run_SQ. J'ai créé la fonction **Set_console_data** :

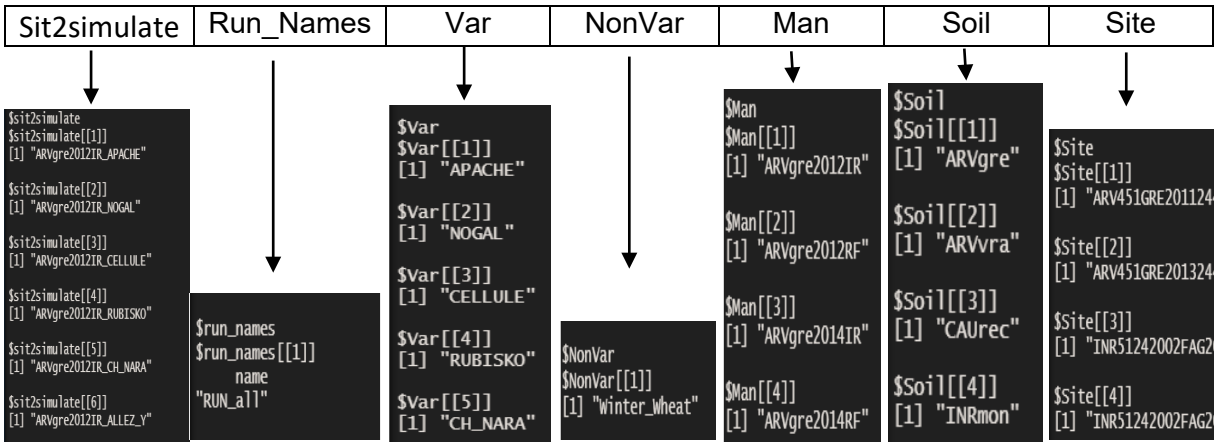
Set_console_data
Input : ▪ runs_list
Output : console_data

Cette fonction retourne **console_data** une liste qui contient :

- Sit2simulate : liste des situations à simuler

- Run_Names : liste des runs à simuler
- Var : liste des items (variétés) pour les paramètres variétaux
- NonVar : liste des items pour les paramètres non variétaux
- Man : liste des items pour les paramètres de management
- Soil : liste des items pour les paramètres de Soil
- Site : liste des items pour les paramètres de Site

Exemple :



Après avoir les données nécessaires, les options --Run et –iRun permettent de préciser le run et la situation à simuler. En effet, il fallait avant séparer un fichier d'entrée SQ (project file) en autant de fichiers individuels qu'il y avait de situations. Cette option nous permet maintenant de travailler avec un seul fichier contenant toutes les runs à simuler et leurs situations associées.

```
#Ajouter la situation :
console_text = paste(console_text,"--iRun",sep = " ")
console_text = paste(console_text,situation,sep = " ")
```

```
console_text = paste(console_text,"--Run",sep = " ")
for(run in console_data$run_names){
  console_text = paste(console_text,run,sep = " ")
}
```

De plus, l'option –Outpath permet de préciser le chemin vers le répertoire dans lequel on enregistre les fichiers de sortie pour une situation donnée.

```
# Ajouter le chemin de sortie
console_text = paste(console_text,"--OutPath",sep = " ")
console_text = paste(console_text,sim_output_path,sep = " ")
```

Ainsi, avec toutes ces modifications on a pu avoir une fonction générique pour l'appel de la ligne de commande. Dans la ligne de commande tous les items à modifier sont présent, mais le changement de la valeur de paramètre pour l'optimisation n'aura lieu que lorsqu'un item est présent dans une situation donnée.

```
[1] "-simoverride true true C:/Users/arharasa/Documents/Stage_Aafaf/Breedwheat/1-Project/Breedwheat.sqpro  --iRun
 SYNlev2018LI_APACHE --Run RUN_all --Var [APACHE] Dse 20 P 50 Dgf 100 [NOGAL] Dse 20 P 50 Dgf 100 [CELLULE] Dse 20
 P 50 Dgf 100 [RUBISKO] Dse 20 P 50 Dgf 100 [CH_NARA] Dse 20 P 50 Dgf 100 [ALLEZ_Y] Dse 20 P 50 Dgf 100 [WIWA] Dse
 20 P 50 Dgf 100 [ANTEQUERA] Dse 20 P 50 Dgf 100 --NonVar [Winter_Wheat] Pincr 1 Pdecr 0.1 --OutPath C:/Users/arha
rasa/Documents/Stage_Aafaf/Breedwheat/3-Output/SYNlev2018LI_APACHE"
```

Figure 19 : Exemple de la ligne de commande construite par run_SQ

Run_SQ
Input : `param_values, project_path, model_options`
Output : -
construire la ligne de commande qui permet l'optimisation des paramètres du Sol pour une seule situation

→

Run_SQ
Input : `param_values, model_options, sim_output_path, situation`
Output : -
construire la ligne de commande qui permet l'optimisation de tous les paramètres pour n'importe quelle situation

2.1.4- Amélioration de Load_obs :

Load_obs est la fonction qui permet de lire les fichiers des observations et qui doit retourner une liste nommée (noms = noms de situations) de data.frame contenant une colonne nommée Date avec les dates (Date ou format POSIXct) des différentes observations et une colonne par variable observée avec les valeurs mesurées ou NA, si la variable n'est pas observée à la date donnée. La fonction qu'on avait avant permet de lire les observations pour la variable LAI (leaf area index) uniquement :

```
Data_dyn        <- obs_dyn[obs_dyn$Soil == name_sit,c('Date', 'mean_LAI')]
Data            <- data.frame(Date  = as.Date(Data_dyn$Date,format="%m/%d/%y"),
                              GAI = Data_dyn$mean_LAI)
```

J'ai donc modifié la fonction pour qu'elle puisse lire les observations de toutes les variables rentrées par l'utilisateur en argument :

- **dailyVar :** vecteur des noms des variables dans le fichier des observations journalières
- **sumVar :** vecteur des noms des variables dans le fichier des observations saisonnières. Ces variables peuvent être des dates (par exemple : date d'émergence, date de floraison...), pour les optimiser on les gère en jour julien comptés depuis le jour de semis.

Ainsi, on a besoin de lire les dates de semis « Planting date » à partir d'un troisième fichier dont on met le chemin en argument.

experiment_ID	treatment_ID	planting_distribution	plot_area	plot_harvest_area	plot_row_number	row_spacing	plot_separation	planting_depth	plot_layout	planting_date	plant_pop_at_planting	average_emergence_date
text	text	code	m2	m2	number	cm	cm	mm	text	date	number/m2	date
EID	TREAT_ID	PLDS	PLTA	PLTHA	PLTRno	PLRS	PLTSP	PLDP	PLLAY	PDATE	PLPOP	APLDAE
ARVvra2012		R	24	10	11			30	Augmented	2011-10-22	375	2011-11-02
ARVgre2012		R		11.2	11	17.5		30	Augmented	2011-11-17	350	2011-12-06
CAUrec2012		R	7.84	6.75				30	Augmented	2011-10-28	318	
INRmon2012		R	9.1	5.2	6	20		30	Augmented	2011-10-20	225	2011-11-03
R2NIou2012		R	10.7	8.4				30	Augmented	2011-10-17	250	
BAYvra2013		R	24	10				30	Augmented	2012-10-23	375	2012-11-10
INRmon2013		R	9.1	5.2	6	20		30	Augmented	2012-10-24	225	
LIMver2013		R		10				30	Augmented	2012-10-29	250	2012-11-20
ARVgre2014		R	24	11.2	11	17.5		30	Augmented	2013-11-07	330	11/13/2013
SYNlev2018		R		7.68	8	16		30		2017-10-19	275	
ARVgre2018		R	23.1	11.88	11	17.5		30	Split-plot	2017-10-29	300	2017-11-23

Fill in **TREAT-ID** column only for experiments for which planting event differs

Figure 20 : Extrait du fichier « Planting_events »

La fonction devient donc :

Load_obs_mean
Input : `obs_dyn_path`
Output : obs_list
Lire les observations journalières de la variable LAI

→

Load_obs
Input : obs_daily_path, obs_sum_path, planting_path, dailyVar, sumVar
Output : -
Lire les observations pour toutes les variables choisis par l'utilisateur (statiques et dynamiques)

2.1.5- Fonctions pour paramètres :

a) - Create_Param_list :

Pour faire de l'optimisation, l'utilisateur doit renseigner des informations sur les paramètres à estimer. Il faut définir :

- **param_info :** Liste d'information sur les paramètres à optimiser. Elle contient :

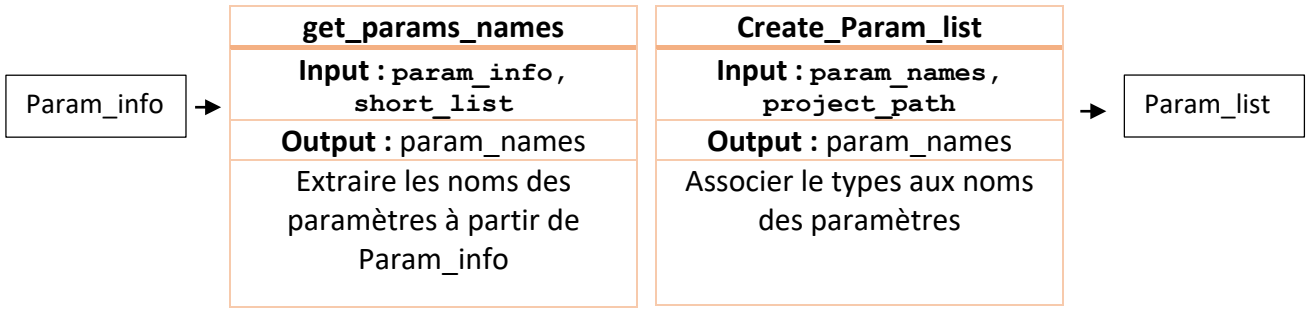
- ub, lb : des vecteurs nommés pour les bornes supérieures et inférieurs des paramètres

- init_values : un data frame contenant les valeurs initiales pour tester les paramètres.

- **Param_list :** un data frame qui contient l'information sur les paramètres à optimiser (le nom et le type)

On s'est rendu compte que l'on pouvait construire Param_list à partir de param_info et des fichiers d'entrée de SiriusQuality. Pour se faire : D'abord, j'ai réutilisé la fonction **get_params_names** du package CroptimizR qui permet d'extraire les noms des paramètres à partir de Param_info.

Ensuite, J'ai créé la fonction **Create_Param_list** qui permet d'associer à chaque paramètre leur type (soit : Var, NonVar, Soil, Man, Site) à partir des fichiers XML d'entrée de SQ.



b) - Set_optim_options :

Estim_param la fonction principale pour optimisation des paramètres dans CroptimizR a comme argument **Optim_options** qui est une liste des options de la méthode d'estimation utilisée, Elle contient :

- Path_results : Le chemin où stocker les résultats (optionnel, par défaut=getwd())
- L'ensemble des options spécifiques en fonction de la méthode utilisée.

Pour initialiser Optim_options, J'ai créé la fonction **Set_optim_options** qui permet à l'utilisateur de définir des options qui correspondent à la méthode choisie en argument **'optim_method' :**

❖ **Optim_method = "simplex"**

L'utilisateur peut fournir certaines conditions de terminaison spécifiant quand l'algorithme s'arrête. Il peut donc renseigner les options de contrôle suivants :

- **maxeval :** Nombre maximal d'évaluations de la fonction critère. La simulation se termine lorsque le nombre d'évaluations de fonctions atteint maxeval. Il est utile si vous voulez vous assurer que l'algorithme vous donne une réponse dans un délai raisonnable, même si elle n'est pas absolument optimale `[maxeval=500]`
- **xtol_rel :** Critère de tolérance entre deux itérations (seuil pour la différence relative des valeurs des paramètres entre les deux itérations précédentes) `[xtol_rel=1e-4]`
- **ranseed :** graine pour les nombres aléatoires pour que chaque exécution donne les mêmes résultats. Si vous voulez la randomisation, ne la définissez pas. `[ranseed=NULL]`

Puisqu'il s'agit d'une méthode d'optimisation locale (simplex). La fonction « wrap_nloptr » dans CroptimizR propose de répéter automatiquement la minimisation à partir de différentes valeurs de paramètres de départ pour minimiser le risque de converger vers un minimum local. Ainsi, l'utilisateur peut spécifier le nombre de répétitions également :

- **nb_rep :** Nombre de répétitions de la minimisation (chaque fois en commençant par différentes valeurs initiales pour les paramètres estimés) `[nb_rep=5]`

❖ Optim_method = "dreamzs"

La fonction DREAMzs interfacé à partir du package BayesianTools a comme argument **settings** une liste des options pour contrôler la performance de l'algorithme. L'utilisateur permet donc via la fonction optim_options de modifier n'importe quel paramètre dans la liste settings.

```
DREAMzs(bayesianSetup, settings = list(iterations = 10000, nCR = 3, gamma
  = NULL, eps = 0, e = 0.05, pCRupdate = FALSE, updateInterval = 10, burnin
  = 0, thin = 1, adaptation = 0.2, parallel = NULL, Z = NULL,
  ZupdateFrequency = 10, pSnooker = 0.1, DEpairs = 2, consoleUpdates = 10,
  startValue = NULL, currentChain = 1, message = FALSE))
```

Figure 21 : prototype de la fonction DREAMzs

- ✓ Settings : Liste des paramètres qui contient :
  - Iterations : Nombre d'évaluations du modèle
  - nCR : paramètre déterminant le nombre de propositions croisées. Si nCR = 1 tous les paramètres sont mis à jour conjointement.
  - updateInterval : l'intervalle pour le pCR (probabilités croisées)
  - gamma : Paramètre de Kurtosis pour l'inférence bayesienne.
  - Eps : Terme d'ergodicité
  - pCRupdate : Mise à jour pour probabilités de croisement
  - burnin : Nombre d'itérations traitées comme des itérations d'insertion. Ces itérations ne sont pas enregistrées dans la chaîne.
  - Thin : Détermine l'intervalle dans lequel les valeurs sont enregistrées.
  - Adaptation : Nombre ou pourcentage d'échantillons utilisés pour l'adaptation dans DREAM
  - DEpairs : Nombre de paires utilisées pour générer la proposition
  - ZupdateFrequency : fréquence pour mettre à jour la matrice Z
  - pSnooker : probabilité de mise à jour de snooker
  - Z : matrice de départ pour Z
  - startValue : soit une matrice contenant les valeurs de début ou un entier pour définir le nombre de chaînes qui sont exécutées
  - consoleUpdates : Intervalle dans lequel la progression de l'échantillonnage est imprimée sur la console
  - message : booléen qui détermine si la progression de l'échantillonneur doit être imprimée

c) - Set_model_options :

Estim_param a comme argument aussi model_options qui présente la liste des options nécessaires pour le wrapper. Pour SiriusQuality, la fonction **set_model_options** permet de construire cette liste qui doit contenir :

- cores_nb : nombre de cœurs sur lesquels on simule
- time_display : flag (TRUE/FALSE) pour l'affichage du temps de calcul
- projects_path : chemin vers le fichier projet (cas du Batch run)
- output_path : chemin vers les fichiers de sortie
- console_path : chemin vers le console
- print_console : un booléen pour afficher la progression de la simulation sur le console.
- saves_daily_output : un booléen pour enregistrer les résultats journalières
- Var_simule : le vecteur des noms (utilisés dans les fichiers de sortie de SQ) des variables à simuler.
- Var_names : le vecteur des noms (utilisés dans les fichiers des observations) des variables à simuler, ce qui est fourni par l'utilisateur.
- Param_list : un data frame qui contient les informations sur les paramètres à estimer (nom et type)
- console_data : Liste des items nécessaires pour la construction de la ligne de commande

2.2- Création du Package 'SQoptimizR' :

« Trois fois le même code : écrire une fonction. Trois fonctions qui se complètent : écrire un package. »

Eric Marcon , *R: Créer un package*

Il faut donc créer un package qui permet de rassembler toutes ces fonctions et qui sert à optimiser les paramètres pour le modèle de culture SiriusQuality. La création d'un package R se fait en trois étapes :

2.2.1- Création du projet Rstudio :

Il faut d'abord installer les packages suivants :

- **Devtools** : C'est un package qui fournit des outils qui simplifie la création d'un package en R
- **roxygen2** : C'est un package R qui sert à faire la documentation.

Ensuite, j'ai créé un projet que l'on nomme **« SQoptimizR »** qui sera le nom de notre package. Un nouveau dossier a été donc créé, Il contient un fichier nommé DESCRIPTION, un fichier nommé NAMESPACE plus les répertoires /R/et /man/.

- Le fichier DESCRIPTION qui permet de décrire ce que fait le package

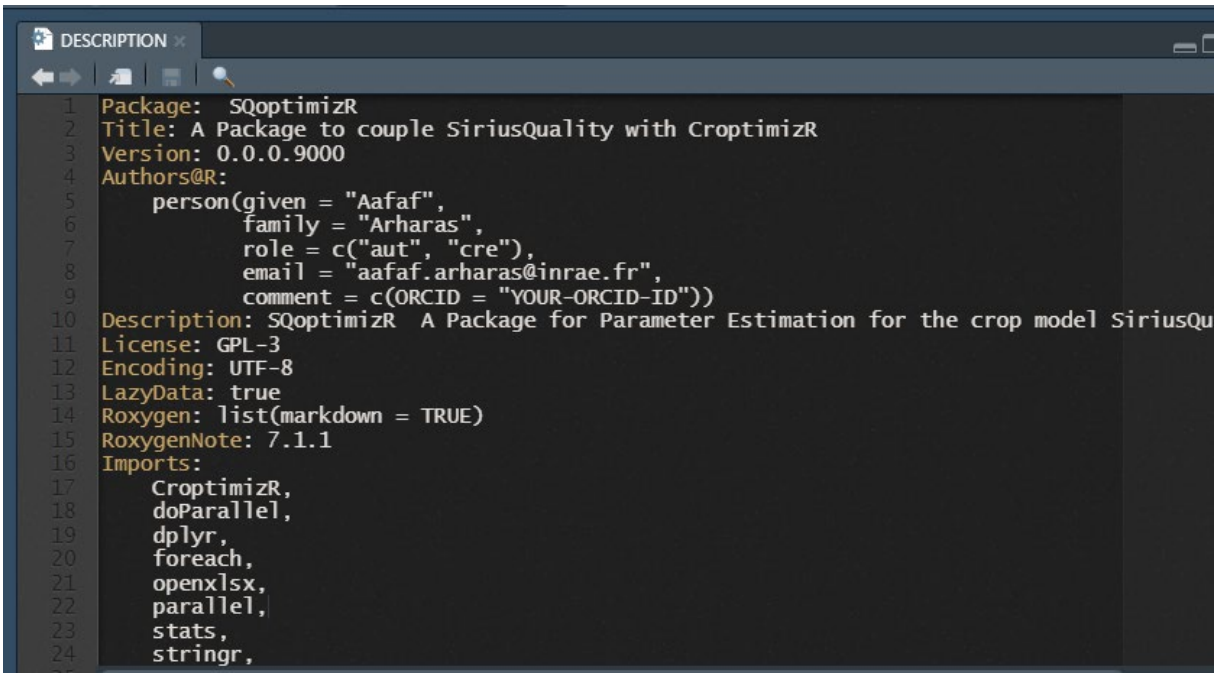


Figure 22 : fichier DESCRIPTION du package SQoptimizR

- Le fichier NAMESPACE qui permet d'importer des fonctions d'autres packages

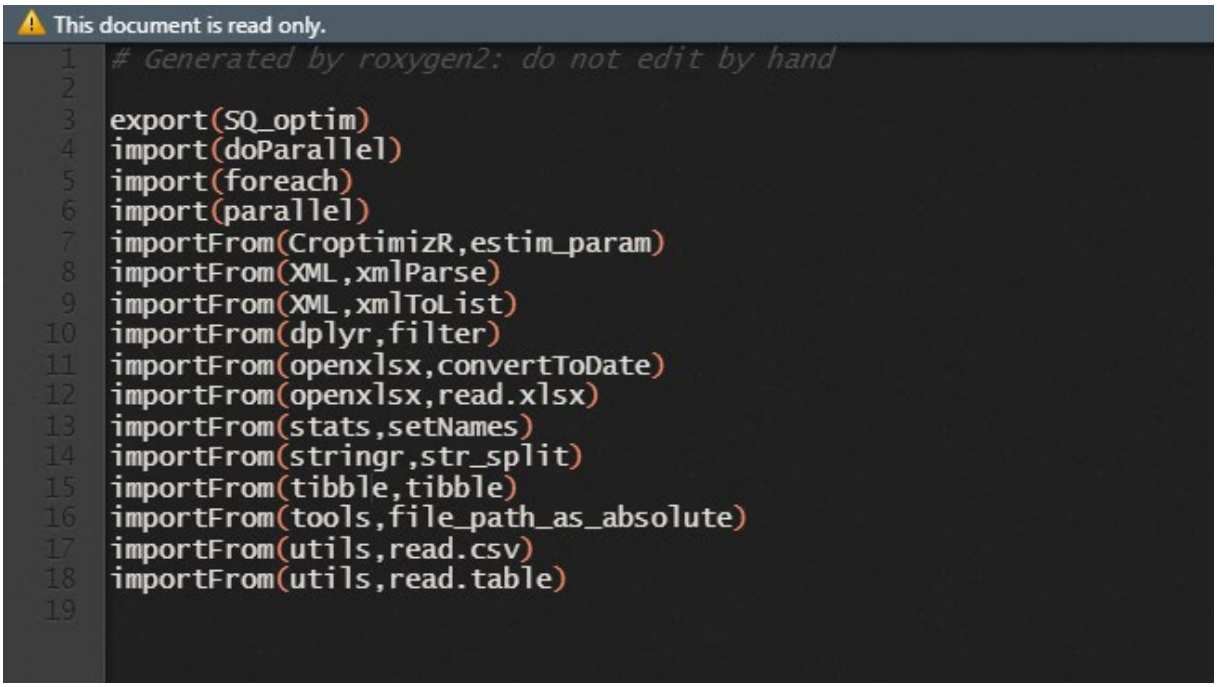


Figure 23 : fichier NAMESPACE du package SQoptimizR

- Le répertoire /R/ qui contient le code source des programmes.
- Le répertoire /man/ qui contient les fichiers d'aide

2.2.2- Création des fonctions :

Après avoir toutes les fonctions nécessaires, il faut maintenant les ranger consciencieusement dans le dossier /R/ de l'arborescence. D'abord j'ai réalisé un schéma qui décrit l'appel des fonctions dans notre package :

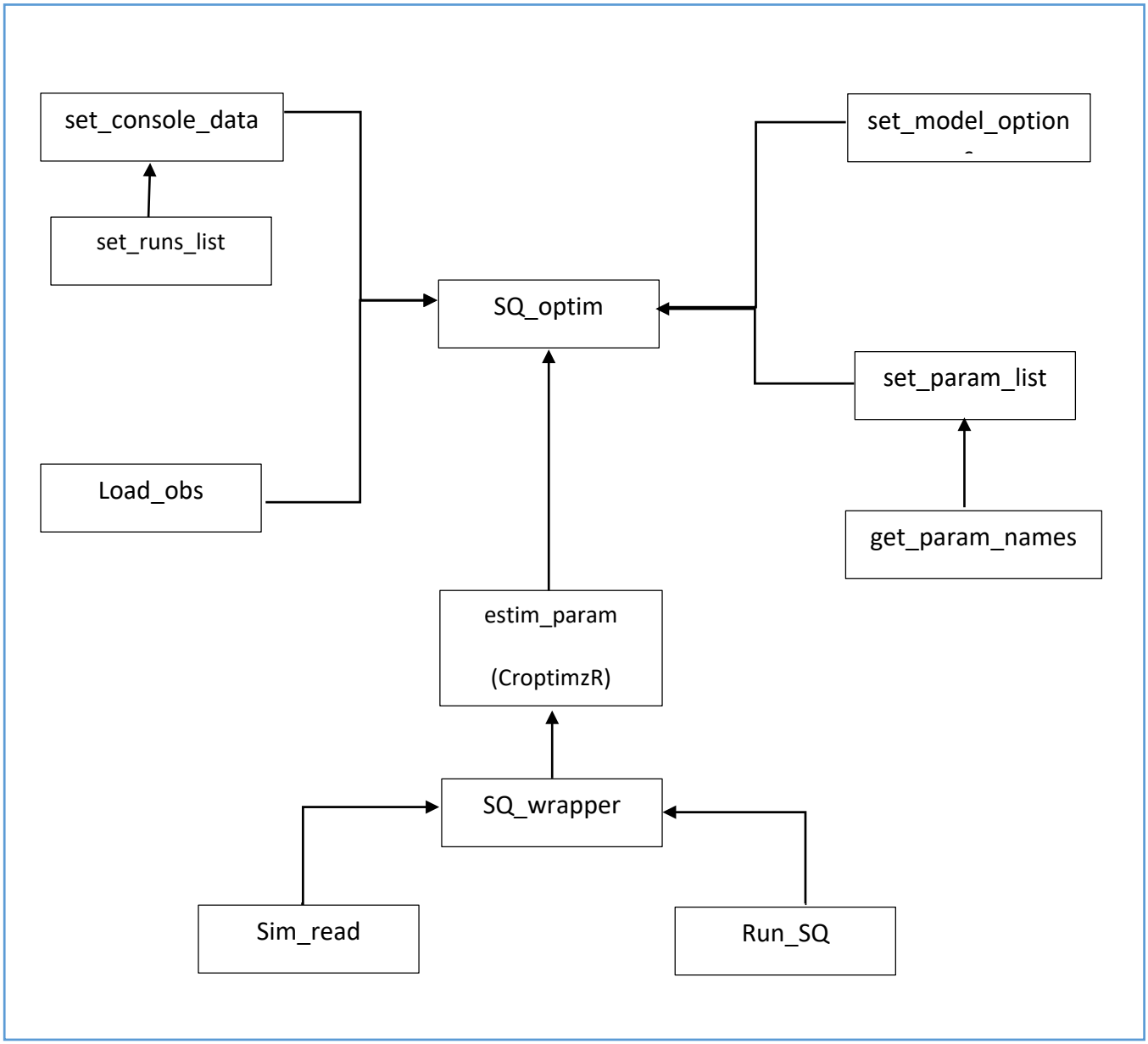


Figure 24 : Schéma des fonctions du package SQoptimizR

Outre les fonctions que j'ai décrit dans la paragraphe '2.1- Amélioration du SQ_wrapper ', J'ai ajouté la fonction **SQ_optim** qui représente la fonction principale pour l'estimation des paramètres de SiriusQuality. Elle appelle la fonction **Load_obs** pour charger les observations, les différentes fonctions pour l'initialisation des paramètres nécessaires pour le wrapper et finalement la fonction **estim_param** du package CroptimizR, qui elle-même appelle le **SQ wrapper.**

Ainsi, il nous reste à définir les niveaux d'accessibilité des fonctions. Le fichier NAMESPACE sert à définir la visibilité de nos fonctions et des fonctions des autres packages, On peut choisir que certaines fonctions soient accessibles à l'utilisateur, que d'autres soient réservées au programmeur et à son programme. Les fonctions accessibles sont appelées publiques, les réservées sont dites privées. Le statut public ou privé se définit dans le fichier NAMESPACE. Les fonctions publiques doivent être déclarées dans **export**. Les fonctions privées doivent simplement ne pas être déclarées.

Pour notre package, on veut que « SQ_optim » soit la seule fonction accessible par l'utilisateur et la seule fonction visible également. Ainsi, lorsque l'utilisateur demande un affichage général par **ls(),** la fonction SQ_optim est affichée :

```
> ls("package:SQoptimizR")
[1] "SQ_optim"
```

2.2.3- Documentation :

Le programme tourne, l'arborescence est faite, les fichiers DESCRIPTION et NAMESPACE sont modifiés. Nous touchons au but, il ne reste plus que de documenter le package.

L'option "`Insert Roxygen Skeleton`" dans Rstudio permet de générer les fichiers d'aides au format .Rd grâce à {roxygen2}. Les fichiers doivent être enregistrés dans le répertoire /man/. Ça permet ensuite à l'utilisateur qui se sert de notre package de faire afficher les différents fichiers d'aide grâce à la commande help.

Voici un exemple :

Si on tape : `> help(load_obs)`

La fenêtre suivante sera ouverte :

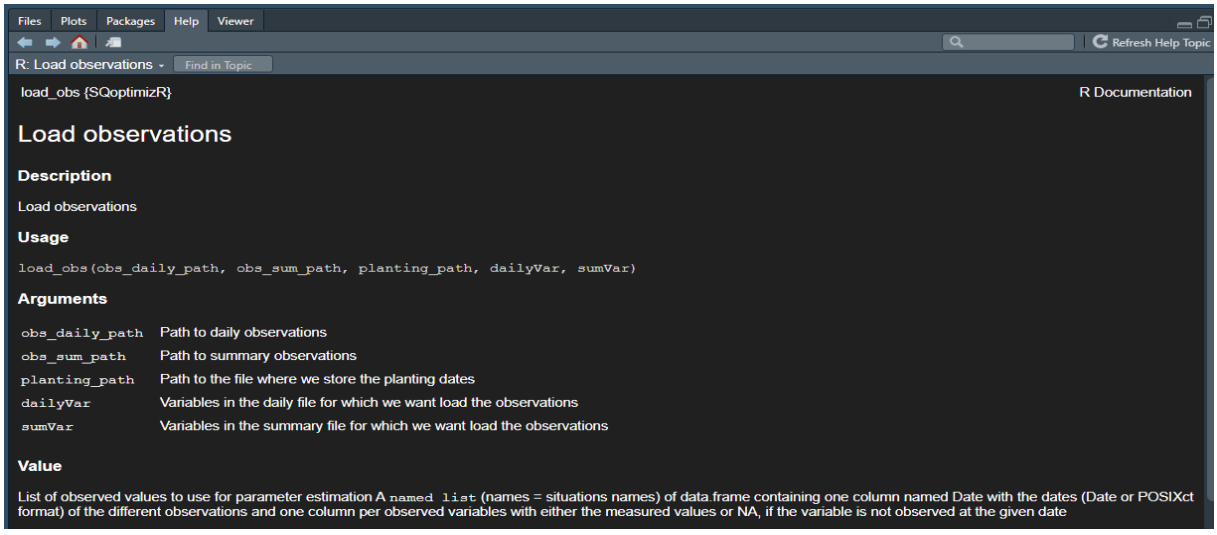
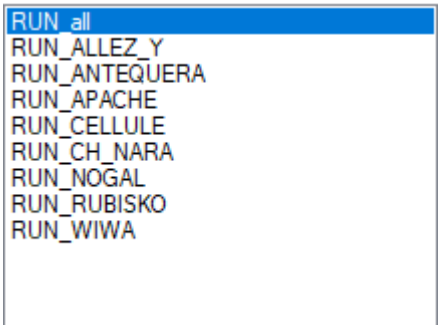


Figure 25 : Documentation de la fonction load_obs

2.3- Etude de cas :

Après avoir créé notre package, Il faut le tester sur des cas pratiques. On va faire tourner différentes optimisations sur un jeu de données nommé 'Breed-Wheat'. Il s'agit d'un seul run nommé 'RUN_ALL' qui rassemble 122 situations différentes pour 8 variétés de blé de type Winter wheat : APACHE, ALLE_Y, ANTEQUERA, CELLULE, CH_NARA, NOGAL, RUBISKO et WIWA. Ainsi on doit diviser le run principal en autant de runs qu'il y avait de variétés, on a choisi la variété **APACHE** pour faire notre étude.



2.3.1- Optimisation de la phénologie :

Etudier la phénologie revient à étudier l'apparition d'événements périodiques (par exemple : l'émergence, la floraison, l'apparition des feuilles, la croissance et la maturation des épis…) déterminée par les variations saisonnières du climat. Il s'agit d'un phénomène essentiel pour la compréhension de la croissance des plantes. C'est notamment un outil de suivi de l'adaptation des végétaux aux changements climatiques.

Afin d'étudier la phénologie on simule les variables suivantes :

Emergence_date (PLDAE) : il présente la date d'émergence, c'est-à-dire la date de production des couches superficielles de la plante, n'ayant pas valeur de tige ou de feuille et s'apparentant plutôt aux constituants du trichome*.

anthesis_date (ADAT) : C'est le début de la période de floraison pendant laquelle une fleur est complètement ouverte et fonctionnelle.

Harvest_date (HDATE) : la date de récolte

Ces variables sont dans le fichier des observations saisonnières, l'utilisateur doit renseigner alors les codes utilisées pour ces variables :

```
dailyVar = NULL
sumVar = c('PLDAE','ADAT','HDATE')
```

On va donc optimiser les paramètres qui influence le plus la phénologie : Des, P, et Dgf qui sont des variables variétales et Pincr et Pdecr qui sont des variables non variétales :

- **Des :** le temps thermique* à l'émergence
- **P :** Le phyllochrone qui est l'intervalle de temps qui s'écoule entre l'émergence séquentielle de deux feuilles successives sur la tige principale d'une plante.
- **Dgf :** la durée de remplissage du grain (de l'anthèse à la maturité physiologique)
- **Pincr :** facteur qui augmente le phyllochrone pour un nombre de feuilles donné sur la tige principale
- **Pdecr :** facteur qui diminue le phyllochrone pour un autre nombre de feuilles donné sur la tige principale

Le phyllochrone est une fonction bilinéaire. Pour les premières feuilles le phyllochrone est plus petit que sa valeur nominale (P), il vaut P* Pdecr. Puis viens une phase où le phyllochrone vaut P. Finalement, pour les dernières feuilles le phyllochrone augmente et vaut P*Pincr.

Ainsi, param_info doit être sous la forme suivante :

```
param_info  <- list(lb=c(Dse=20,  P=50,  Dgf=100, Pincr=1, Pdecr=0.1),
                    ub=c(Dse=200, P=200, Dgf=1000, Pincr=2, Pdecr=1),
                    init_values = data.frame(Dse = c(90),  P = c(98),  Dgf = c(450), Pincr=c(1.25), Pdecr=c(0.4)))
```

a. Estimation des paramètres par Dreamzs :

On fixe les options suivantes pour la méthode Dreamzs :

```
path_results    <-paste0("C:/Users/arharasa/Documents/Stage_Aafaf/Breedwheat/5-Optimization/RUN_APACHE") # path


optim_method    <- "BayesianTools.dreamzs"   # Name of the parameter estimation method to use (simplex, Dreamz)
crit_function   <- likelihood_log_ciidn      # Function implementing the criterion to optimize
iterations      <-  600                      # Total number of iterations
startValue      <-  3                        # Number of markov chains #22
ranseed         <-  1234                     # seed for random numbers
```

Les résultats imprimés en sortie sur la console R sont les suivants :

```
# # # # # # # # # # # # # # # # # # # # # # # # # # #
## MCMC chain summary ##
# # # # # # # # # # # # # # # # # # # # # # # # # # #

# MCMC sampler:  DREAMzs
# Nr. Chains:  3
# Iterations per chain:  161
# Rejection rate:  0.993
# Effective sample size:  229
# Runtime:  1681.75  sec.

# Parameters
      psf     MAP    2.5%  median   97.5%
Dse    NA 171.097 143.427 160.797 171.097
P      NA  71.582  52.686  73.198 193.971
Dgf    NA 190.054 190.054 311.154 661.277
Pincr  NA   1.516   1.056   1.516   1.916
Pdecr  NA   0.748   0.208   0.748   0.928

## DIC:  24556.86
## Convergence
 Gelman Rubin multivariate psrf:

## Correlations
          Dse      P    Dgf  Pincr  Pdecr
Dse    1.000  0.077 -0.806 -0.191  0.449
P      0.077  1.000 -0.285 -0.670  0.757
Dgf   -0.806 -0.285  1.000  0.423 -0.689
Pincr -0.191 -0.670  0.423  1.000 -0.637
Pdecr  0.449  0.757 -0.689 -0.637  1.000
```

Figure 26 : Résultats de l'optimisation de la phénologie par la méthode Dreamzs

- **Rejection rate** est le taux de rejet des valeurs proposées. Selon (Vrugt, 2016), une valeur comprise entre 0,7 et 1 indique généralement une bonne performance d'une méthode de simulation MCMC.
- **Effective sample** de l'échantillon doit être ici le nombre de valeurs différentes dans la chaîne pour le vecteur de paramètre dans l'échantillon postérieur.

Dans la section **« Parameters »** sont données des statistiques sur l'échantillon postérieur. **MAP** signifie Maximum A Posteriori. Ce sont les valeurs des paramètres de l'échantillon postérieur qui conduisent à la valeur maximale de la densité postérieure.

```
$MAP
       Dse          P        Dgf       Pincr       Pdecr
171.0974393  71.5822821 190.0538779   1.5161023   0.7481186
```

La valeur de sortie de estim_param contient :

- **`statistics`, `quantiles` et `MAP` :** statistiques sur l'échantillon postérieur

- `post_sample` : un échantillon de la distribution postérieure (à l'exclusion de la phase burnin),
- `out` : la liste retournée par le paquet BayesianTools.

Elle est stockée avec des graphiques complémentaires dans le dossier optim_options$path_results. Parmi ces graphiques :

- **gelmanDiagPlots.pdf** trace l'évolution du diagnostic Gelman. Gelman-Rubin mesure s'il y a une différence significative entre la variance au sein de plusieurs chaînes et la variance entre plusieurs chaînes par une valeur appelée « facteurs de réduction d'échelle ». GelmanDiag vous donne les facteurs de réduction d'échelle pour chaque paramètre. Selon (Vrugt, 2016), l'algorithme est considéré comme ayant convergé vers la distribution postérieure lorsque toutes les valeurs de ce diagnostic sont inférieures à 1,2. Dans notre cas, nous obtenons :

--- Median
--- 97,5 %

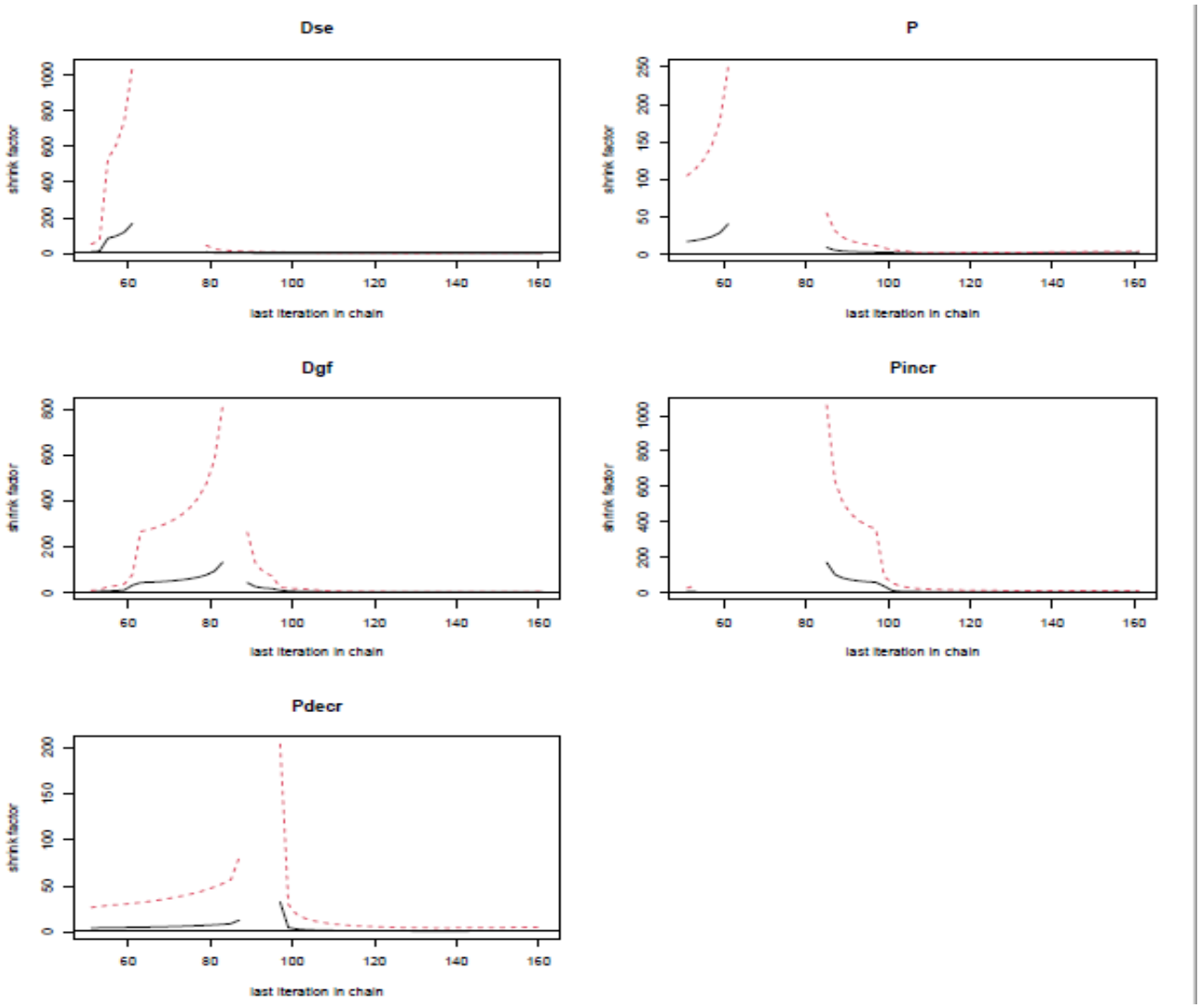


Figure 27 : Tracés de diagnostic Gelman pour l'optimisation de la phénologie

- **marginalPlots.pdf** montre l'estimation des densités antérieures (bleues) et postérieures (roses) :

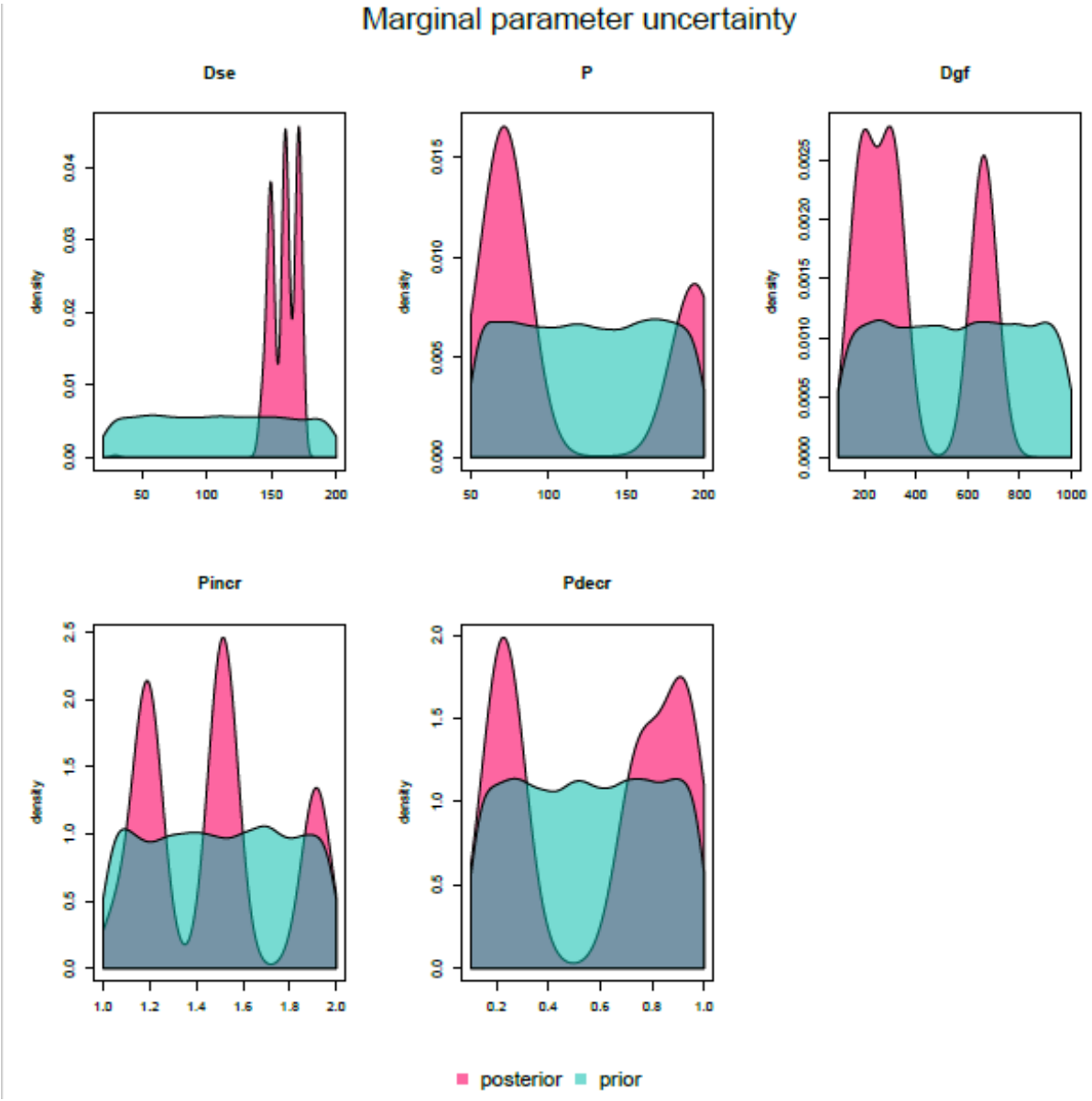


Figure 28 : Tracés marginales pour l'optimisation de la phénologie

- **correlationPlots.pdf** donne des informations sur la corrélation entre les paramètres :

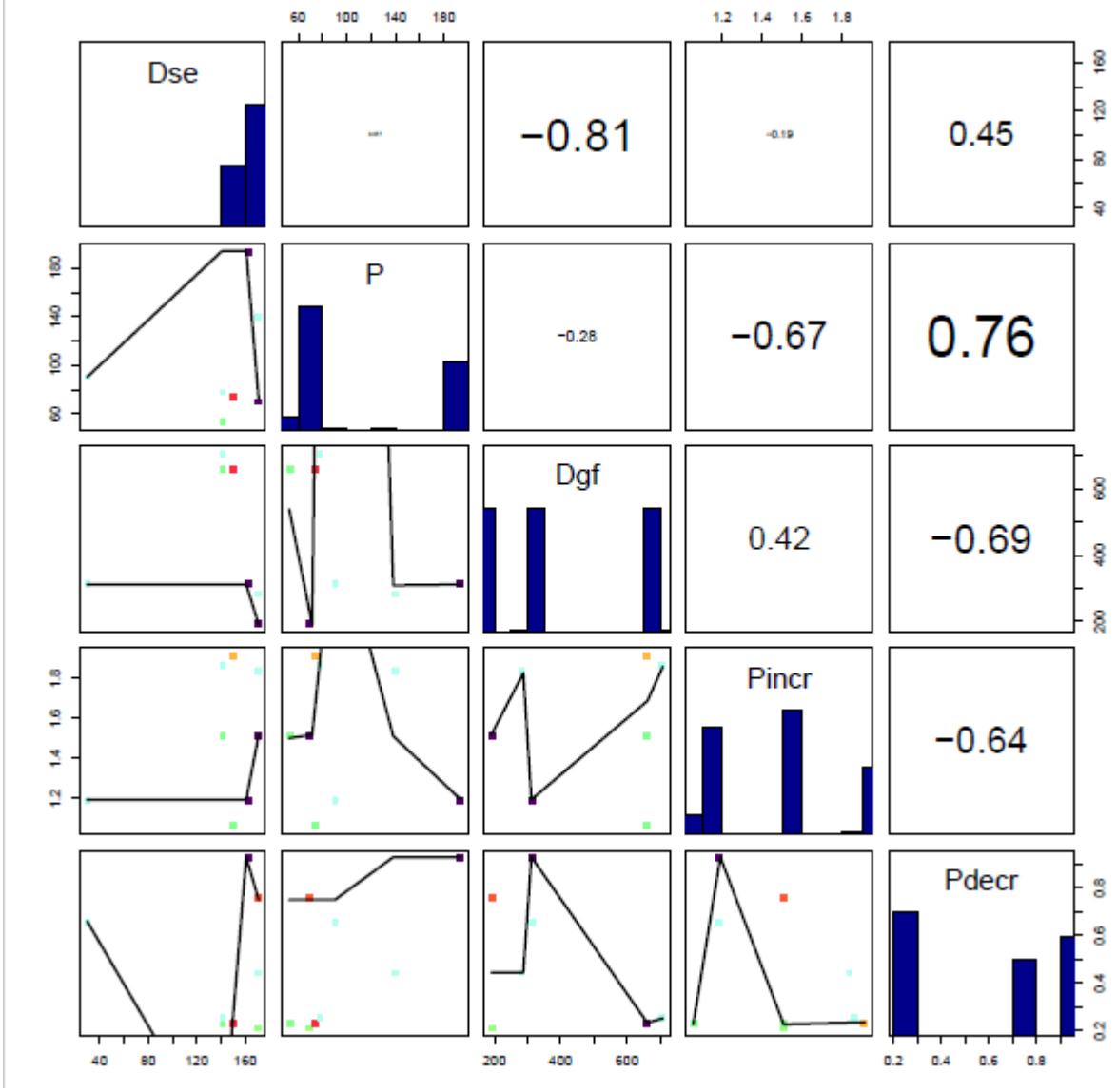


Figure 29 : Tracés de corrélation pour l'optimisation de la phénologie

b. Estimation des paramètres par Simplex :

optim_options doit contenir les options de la méthode d'estimation des paramètres. Nous avons défini ici quelques options importantes pour la méthode simplex du paquet nloptr. Il est conseillé de régler le nombre de **répétitions** sur au moins 5, tandis que 7 est une valeur raisonnable. **maxeval** doit être utilisé pour arrêter la minimisation uniquement si les résultats doivent être produits dans une durée donnée, sinon il doit être réglé à une valeur élevée de sorte que la minimisation s'arrête lorsque le critère basé sur **xtol_rel** est satisfait.

```
optim_method    <-"nloptr.simplex"
crit_function   <- crit_log_cwss
nb_rep          <-  7                 # Number of repetitions of the minimization (each time star
maxeval         <-  1000              # Maximum number of evaluations of the minimized criteria
xtol_rel        <-  1e-04            # Tolerance criterion between two iterations(threshold for t
```

La fonction estim_param retourne une liste qui est également stockée dans le fichier optim_results.Rdata du dossier optim_options$path_results. Cette liste contient des informations différentes selon la méthode utilisée. Pour la méthode Nelder-Mead simplex, elle comprend entre autres :

- Les valeurs finales estimées pour les paramètres
- Les valeurs initiales et finales des paramètres estimés pour chaque répétition de la minimisation
- La valeur minimale du critère pour toutes les répétitions et pour toutes les répétitions

On trouve aussi la variable **nlo** qui est une liste retournée par la fonction nloptr qui contient des informations détaillées sur les résultats de la minimisation pour chaque répétition.

nlo est une liste de taille égale au nombre de répétitions. Chaque élément stocke de nombreuses informations sur la minimisation correspondante, y compris le nombre d'itérations effectuées et un message indiquant pourquoi la minimisation a cessé.

```
Call:
nloptr::nloptr(x0 = as.numeric(init_values[irep, ]), eval_f = main_crit,
    lb = bounds$lb, ub = bounds$ub, opts = list(algorithm = "NLOPT_LN_NELDERMEAD",
        xtol_rel = xtol_rel, maxeval = maxeval, ranseed = ranseed),
    crit_options = crit_options)


Minimization using NLopt version 2.4.2

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because xtol_rel or xtol_abs
(above) was reached. )

Number of Iterations....: 156
Termination conditions:  xtol_rel: 1e-04        maxeval: 1000
Number of inequality constraints:  0
Number of equality constraints:    0
Optimal value of objective function:  148.842416500171
Optimal value of controls: 159.451 53.71763 825.2296 1.043628 0.1
```

Figure 30 : Résultats de l'optimisation de la phénologie par la méthode Simplex

On trouve également dans le dossier optim_options$path_results les graphiques des valeurs estimées par rapport aux valeurs initiales de différents paramètres. Les nombres représentent le nombre de répétitions de la minimisation. Le nombre en rouge, 1 dans ce cas, est la minimisation qui conduit à la valeur minimale du critère parmi toutes les répétitions. Dans ce cas, les minimisations convergent vers des valeurs différentes pour les paramètres, ce qui indique la présence de minima locaux. Les processus rentrant dans le calcul de la phénologie sont non-linéaire. L'optimisation des paramètres pour ce processus essentiel n'est donc pas

facile et il n'est pas étonnant de trouver des minimas locaux. La méthode DREAMz est plus adaptée à ce genre de calibration.

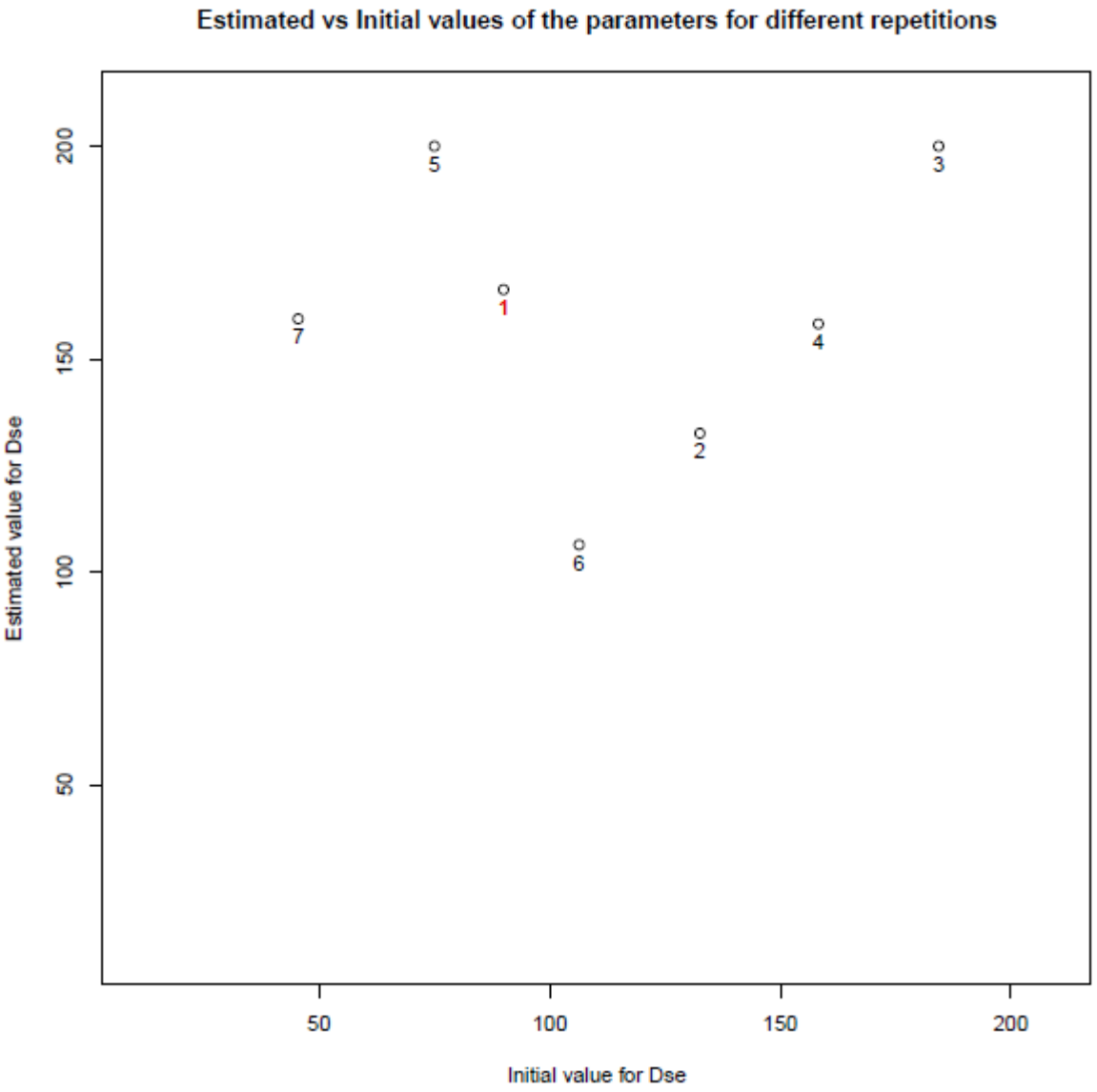


Figure 31 : Tracé des valeurs estimés Vs les valeurs initiales

2.3.2- Optimisation du GAI :

Le GAI (Green Area Index) est une grandeur sans dimension qui exprime la surface des feuilles et des tiges vertes par unité de surface de sol sur lequel la culture pousse.

On optimise les variables qui influence le plus le GAI :

- AreaPL : surface maximale potentielle des feuilles produites après l'initiation florale (ou début de la floraison)
- AreaSL : surface potentielle des feuilles produites avant le début de la floraison
- AreaSS : surface potentielle de la gaine des feuilles produites avant le début de la floraison

- NLL : nombre de feuilles produites après l'initiation florale

```
param_info  <- list(lb=c(AreaPL=10, NLL=2, AreaSL=0, AreaSS=0),
                    ub=c(AreaPL=70, NLL=10, AreaSL=5, AreaSS=5),
                    init_values = data.frame(AreaPL = c(35.7), NLL = c(6.5), AreaSL=c(2.5), AreaSS=c(1.83)))
```

Les plots de Gelman montrent que l'algorithme a bien convergé vers la distribution postérieure. Les plots de corrélation, les plots marginaux et de densité sont dans l'annexe.

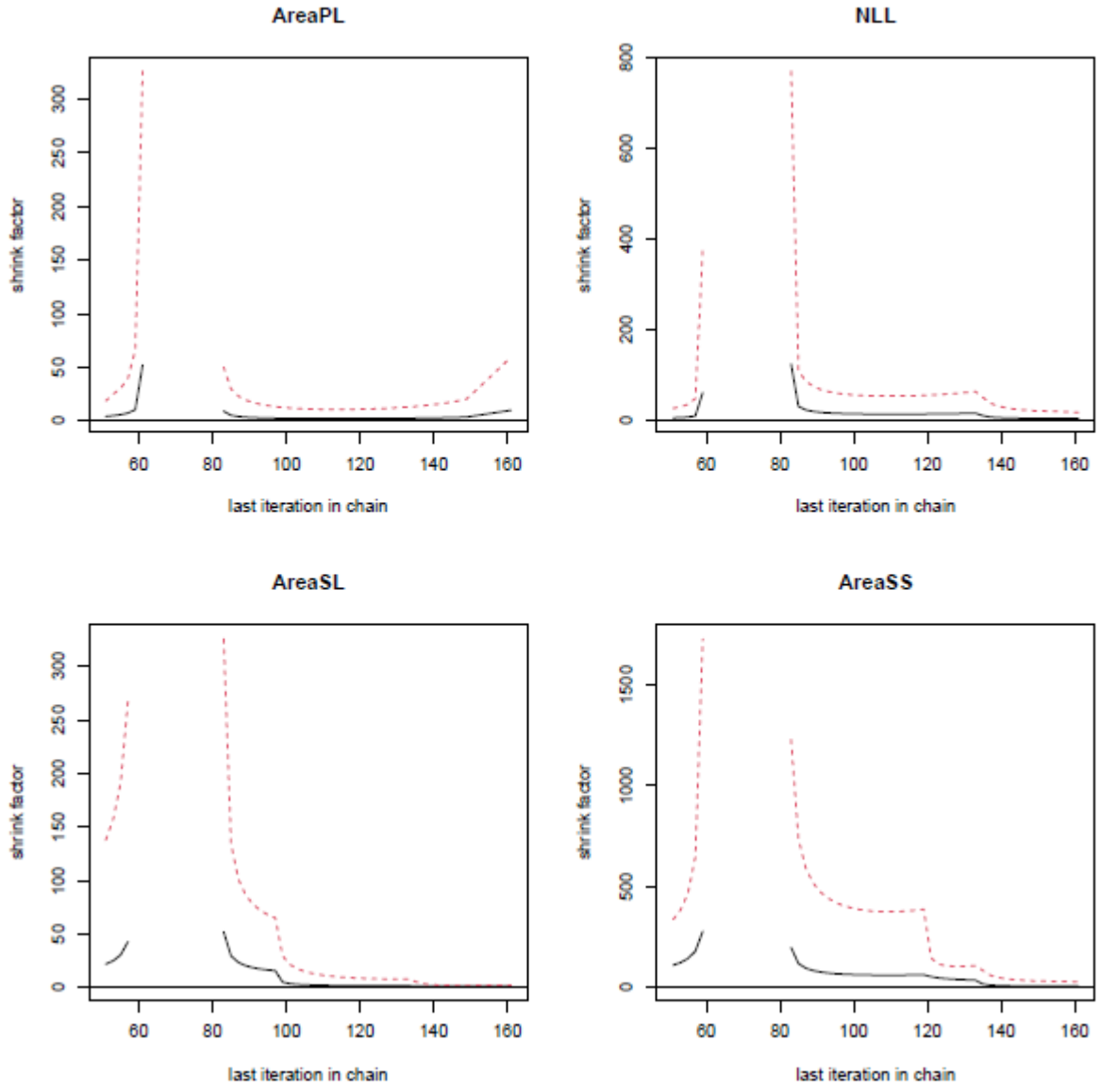


Figure 32 : Tracés de diagnostic Gelman pour l'optimisation du GAI

Les valeurs optimales des paramètres sont enregistrées dans la variable MAP :

```
$MAP
    AreaPL        NLL     AreaSL     AreaSS
19.3076945  8.7668679  2.7904521  0.8330959
```

2.4- Tests :

Pour tester la validité de notre package, il y a deux moyens :

2.4.1- Test wrapper :

La fonction **test_wrapper** est une fonction du package CroptimizR permet d'effectuer certains tests sur les wrappers des modèles de cultures.

```
test_wrapper(
  model_function,
  model_options,
  param_values,
  sit_names,
  var_names = NULL
)
```

Figure 33 : Prototype de la fonction test_wrapper

Elle a comme entrées :

- **model_function:** la fonction wrapper du modèle de culture à utiliser.
- **model_options:** la liste des options pour le wrapper.
- **param_values:** un vecteur nommé qui contient les noms et les valeurs d'au moins deux différentes paramètres de modèle (e.g. param_values <- c(P1=10, P2=50))
- **sit_names:** un vecteur de situations pour lesquels les résultats sont testés.
- **var_names (optional):** un vecteur des noms des variables pour lesquels les résultats sont testés.

La fonction permet de vérifier :

- Le format des résultats retournés
- Si les résultats sont différents lorsque différents sous-ensembles de param_values sont utilisés
- Si les résultats sont identiques lorsque les mêmes sous-ensembles de param_values sont utilisés.

La fonction exécute le wrapper du modèle donné consécutivement avec différents sous-ensembles de param_values.

```
param_values_1 <- param_values[1]
param_values_2 <- param_values
```

Or, la fonction run_SQ appelée par le wrapper exige que param_values ait la même taille que Param_list. Ainsi, j'ai modifié la fonction test_wrapper pour qu'elle exécute le wrapper consécutivement avec différents vecteurs 'param_values'. Je modifierais ensuite le

wrapper pendant les deux dernières semaines de mon stage pour que param_values et Param_list ait la même taille et que le wrapper soit conforme au test.

```
param_values_1= c(Dse=20,  P=50,  Dgf=100, Pincr=1, Pdecr=0.1)
param_values_2= c(Dse=200, P=200, Dgf=1000, Pincr=2, Pdecr=1)
```

Finalement, l'exécution du test_wrapper retourne :

```
Test the wrapper returns outputs in expected format ...
... OK

Test the wrapper gives identical results when running with same inputs ...
... OK

Test the wrapper gives different results when running with different inputs ...
... OK
```

Figure 34 : Résultats de la fonction test_wrapper

Notre wrapper est bel et bien valide !

2.4.2- Code master Vs SQoptimizR :

Un autre moyen de tester SQoptimizR est de comparer les résultats de l'optimisation du GAI obtenus par SQoptimizR dans la partie "2.3.2- Optimisation de la GAI" avec ceux obtenus par le code master (le code initial sans améliorations). Pour cela j'ai utilisé à la fois pour le package et le master les valeurs de paramètres obtenus pour l'optimisation de la phénologie dans les fichiers d'entrée Sirius.

On obtient avec le code master pour la variété APACHE les valeurs optimales suivantes :

```
$MAP
    AreaPL         NLL      AreaSL      AreaSS
12.9323477  7.1135234  0.7393144  2.1197066
```

On ne peut pas conclure avec ces valeurs, car même si elles ont le bon ordre de grandeur elle s'éloigne de ce qui est attendu pour la variété APACHE qui a déjà était calibré avant mon arrivée et qui sert de référence. Ainsi, à l'aide de l'interface graphique de SiruisQuality on va modifier les valeurs des paramètres pour qu'elles prennent les valeurs optimales afin de générer les simulations avec mon package et les confronter ensuite avec celles générées par le code master.

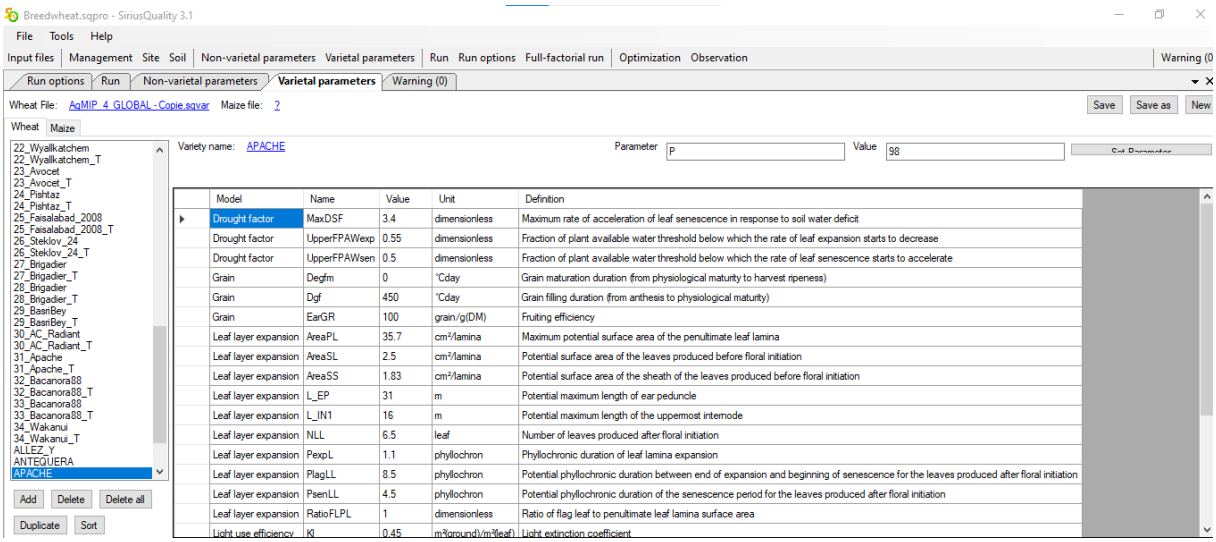


Figure 35 : onglet 'Varietal parameters' de SiriusQuality

J'ai choisi la situation **ARVgre2018LI_APACHE** car elle a plus d'observations :

```
> print(obs_list$ARVgre2018LI_APACHE)
         Date GAID
1  2017-12-07 0.13
2  2018-02-13 0.19
3  2018-02-14 0.14
4  2018-03-22 0.86
5  2018-03-23 0.70
6  2018-03-29 1.40
7  2018-03-30 1.22
8  2018-04-05 2.16
9  2018-04-18 3.88
10 2018-04-19 3.51
11 2018-04-24 4.83
12 2018-04-25 4.09
13 2018-05-09 6.12
14 2018-05-16 4.07
15 2018-05-18   NA
16 2018-06-01 2.39
17 2018-06-07   NA
18 2018-06-27 0.12
19 2018-07-13   NA
```

✓ Comparaison et conclusion :

On remarque que les simulations obtenues par le code master et celles obtenus par notre package SQoptimizR ne sont pas exactement les mêmes mais elles varient dans le même intervalle. Les différences entre les simulations sont dû au caractère aléatoire de la méthode DREAMs.

	A	B	C	D
1	Date	GAID_Master	GAID_SQoptimizR	
2	1/19/2018	0	0	
3	1/20/2018	0	0	
4	1/21/2018	0	0	
5	1/22/2018	0	0	
6	1/23/2018	0	0.01	
7	1/24/2018	0	0.01	
8	1/25/2018	0	0.02	
9	1/26/2018	0.01	0.02	
10	1/27/2018	0.01	0.02	
11	1/28/2018	0.01	0.03	
12	1/29/2018	0.01	0.03	
13	1/30/2018	0.01	0.04	
14	1/31/2018	0.01	0.04	
15	2/1/2018	0.01	0.05	
16	2/2/2018	0.01	0.05	
17	2/3/2018	0.01	0.05	
18	2/4/2018	0.02	0.06	
19	2/5/2018	0.02	0.06	
20	2/6/2018	0.02	0.06	
21	2/7/2018	0.02	0.07	
22	2/8/2018	0.02	0.07	

Figure 36 : Simulations master Vs simulations package pour GAI

Partie 3 : DISCUSSIONS ET PERSPECTIVES

3.1- Résultats et discussions :

Notre package reproduit bien les résultats du code initial, il ne reste qu'à d'évaluer la capacité du package à estimer les paramètres du modèle qui permettent de produire les meilleures simulations. L'évaluation consiste donc à confronter les résultats d'une simulation aux différentes observations disponibles.

On traite le cas de la phénologie. D'abord, j'ai modifié via l'interface graphique de SiruisQuality les valeurs des paramètres : Des, P, Dgf, Pincr et Pdecr

```
$MAP
       Dse         P       Dgf      Pincr      Pdecr
171.0974393 71.5822821 190.0538779 1.5161023 0.7481186
```

Ensuite, j'ai fait tourner une simulation des variables PLDAE, ADAT et HDATE pour les différentes situations du run RUN_APACHE et finalement j'ai mis les résultats dans un fichier excel pour la comparaison :

Les variables PLDAE, ADAT et HDATE sont le nombre de jours depuis le semis d'émergence, d'anthèse et de récolte.

	A	B	C	D	E	F	G
1	situation	PLDAE_obs	PLDAE_sim	ADAT_obs	ADAT_sim	HDATE_obs	HDATE_sim
2	ARVgre2012IR_APACHE	18	61	189	146	243	166
3	ARVgre2012RF_APACHE	18	61	189	146	243	166
4	ARVgre2014IR_APACHE	5	47	190	154	247	178
5	ARVgre2014RF_APACHE	5	47	189	153	237	175
6	ARVgre2018HI_APACHE	24	56	199	160	256	177
7	ARVgre2018LI_APACHE	24	56	198	160	256	176
8	ARVvra2012HN_APACHE	10	101	NA	186	283	207
9	ARVvra2012LN_APACHE	10		NA		283	
10	BAYvra2013HN_APACHE	17		NA		292	
11	BAYvra2013LN_APACHE	17		NA		292	
12	CAUrec2012HN_APACHE	NA		NA		NA	
13	CAUrec2012LN_APACHE	NA		NA		NA	
14	INRmon2012HN_APACHE	13		222		283	
15	INRmon2012LN_APACHE	13		221		279	
16	INRmon2013HN_APACHE	NA		231		NA	

Figure 37 : Simulations de la phénologie Vs observations

Il y a une différence de 40 jours ou plus entre les simulations et les observations. Les simulations sont assez éloignées de celle des observations. Ceci est dû aux valeurs optimales trouvées qui s'éloignent de ce qui est attendu pour la variété APACHE. Voici quelques justifications possibles :

- Le manque des observations pour certaines situations.
- Les erreurs de saisie des observations.
- Un mauvais choix des paramètres optimaux de la méthode Dreamzs.

- Une mauvaise construction de la liste des observations. En effet, la fonction estim_param du package 'CroptimizR' exige la colonne 'Date' dans la liste des observations ce qui est plus adapté aux variables journalières et non aux variables saisonniers comme la date d'émergence, d'anthèse et de récolte.
- Une mauvaise stratégie de calibration pour la phénologie. Il aurait été plus judicieux de faire 3 calibrations séparées : une pour l'émergence, une pour l'anthèse et une pour la maturité.

A noter que si les observations de la phénologie sont mal reproduites, il est difficile de calibrer le GAI. Donc on ne peut pas comparer directement les observations avec les simulations. Malgré tous, si le code master et le code que j'ai développé sont basés sur les mêmes paramètres de phénologie on peut quand même savoir s'ils donnent les mêmes résultats.

3.2- Perspective du travail :

Ce rapport étant rédigé avant l'échéance du stage. Il me reste donc quelques améliorations à faire d'ici la fin :

- Traiter le cas où param_values n'a pas la même taille que Param_list afin de faire passer le test_wrapper sans modifications.
- Lire les dates de semis 'Planting_date' à partir des fichiers d'entrée de SQ 'Management' afin de construire la liste des observations.
- Vérifier que dans le package la liste des observations est bien construite pour laisser un code opérationnel après mon départ

En dehors de l'optimisation du code et des algorithmes, une autre façon d'obtenir un code performant et d'améliorer le wrapper est de tirer profit des architectures parallèles des ordinateurs modernes. Il s'agit alors de paralléliser son code afin de faire des opérations simultanées sur des parties distinctes d'une même simulation, en utilisant différent cœurs de calcul. On ne réduit pas le temps de calcul total nécessaire, mais l'ensemble des opérations s'exécute plus rapidement puisqu'elles sont faites en parallèle.

Il existe un nombre non négligeable d'algorithmes qui sont d'un "parallélisme embarrassant", c'est-à-dire dont les calculs peuvent se décomposer en plusieurs sous-calculs indépendants. Les opérations nécessaires pour l'établissement d'un code parallèle sont les suivantes :

1. Démarrer un processus "travailleurs" (i.e. cœurs de calcul) et les initialiser
2. Envoyer les fonctions et données nécessaires pour chaque tâche aux travailleurs
3. Séparer les taches en un nombre m opérations d'envergure similaire et les envoyer aux travailleurs
4. Attendre que tous les travailleurs aient terminer leurs calculs et obtenir leurs résultats
5. Rassembler les résultats des différents travailleurs
6. Arrêter les processus travailleurs

Dans le wrapper, la parallélisation du calcul se fait selon les situations, l'appel de la ligne de commande et la lecture de simulations sont exécutés dans une boucle dont les opérations

sont indépendantes d'une itération à l'autre. Ainsi, on utilise le package R nommé « parallel » qui permet de démarrer et d'arrêter un "cluster" de plusieurs processus travailleur (étape 1). En plus du package parallel, on utilise le package doParallel qui permet de gérer les processus travailleurs et la communication (étapes 1) et l'articulation avec le package foreach qui permet lui de gérer le dialogue avec les travailleurs (envois, réception et rassemblement des résultats - étapes 2, 3, 4 et 5).

```
sim_batch <- foreach(Ind_situation = 1:N,
                     .export = c('run_SQ',
                                 'sim_read')) %dopar% {
```

Ainsi, Il serait donc intéressant d'essayer d'autres méthodes qui permettent la parallélisation sur R ou de voir la possibilité de faire la parallélisation par variété. Deux approches sont possibles : étudier les packages R qui permettent la parallélisation ou découper les processus en jobs que l'on peut envoyer sur un serveur de calcul.

Finalement, le package SQoptimizR peut se compléter avec le développement d'une interface graphique à l'aide du package R Shiny.

Conclusion :

Le travail effectué durant ce stage avait pour objectif l'amélioration d'un wrapper pour lier le package 'CroptimizR' et le modèle de culture SiruisQualty. Les principaux objectifs fixés au début ont été atteints. Plusieurs améliorations sont portées sur le code initial à savoir : L'utilisateur peut choisir les variables à simuler (des variables journalières ou saisonniers) aussi bien que le run (l'ensemble de situations) à simuler, La ligne de commande est construite pour tout type de paramètres. Par conséquent, l'utilisateur peut choisir différents paramètres à optimiser (Var, NonVar, Man, Site, Soil). De plus, le wrapper est capable de traiter le cas de Batch_run où un seul fichier d'entrée de SiruisQuality est utilisé. Ensuite, on a réalisé le package R qui permet l'estimation des paramètres du modèle de culture SiruisQualty avec le moins possible d'intervention de la part de l'utilisateur. Finalement, des tests sont été effectués pour valider le wrapper aussi bien qu'une étude de calibration de phénologie et de GAI.

Ce travail s'est montré très enrichissant personnellement. En effet, ce stage a été une bonne opportunité pour se familiariser avec le milieu de la recherche, les codes informatiques de calcul scientifique et les modèles de simulation. Il m'a permet également d'apprendre à développer dans le langage R qui est largement utilisé par les biologistes et les statisticiens.

Des améliorations peuvent être apporter au niveau de la structure informatique ainsi que de la rapidité. En effet, les architectures parallèles des ordinateurs modernes peuvent nous servir à réduire le temps de calcul total nécessaire ce qui est important dans le cas où on travaille sur plusieurs variétés. De plus, Il semblerait intéressant de réaliser une interface graphique du package SQoptimizR.

Bibliographie :

[1] MAGE : Modélisation et Analyse de l'interaction Génotype Environnement https://www6.montpellier.inrae.fr/lepse/Organisation/Equipe-MAGE

[2] SiriusQuality https://www6.montpellier.inrae.fr/lepse/Personnel/Chercheurs/Pierre-Martre/SiriusQuality

[3] Hubert Wassner. avril 9, 2021. Méthode fréquentiste ou bayésienne, https://www.abtasty.com/fr/blog/statistiques-bayesiennes-ab-testing/

[4] Christophe Genolini. Construire un Package Construire un PackageClassic. https://cran.r-project.org/doc/contrib/Genolini-ConstruireUnPackage.pdf

[5] Simplexe https://fr.wikipedia.org/wiki/Simplexe

[6] Méthode de Monte-Carlo par chaînes de Markov https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Monte-Carlo_par_cha%C3%AEnes_de_Markov

[7] Degré jour de croissance https://fr.wikipedia.org/wiki/Degr%C3%A9_jour_de_croissance

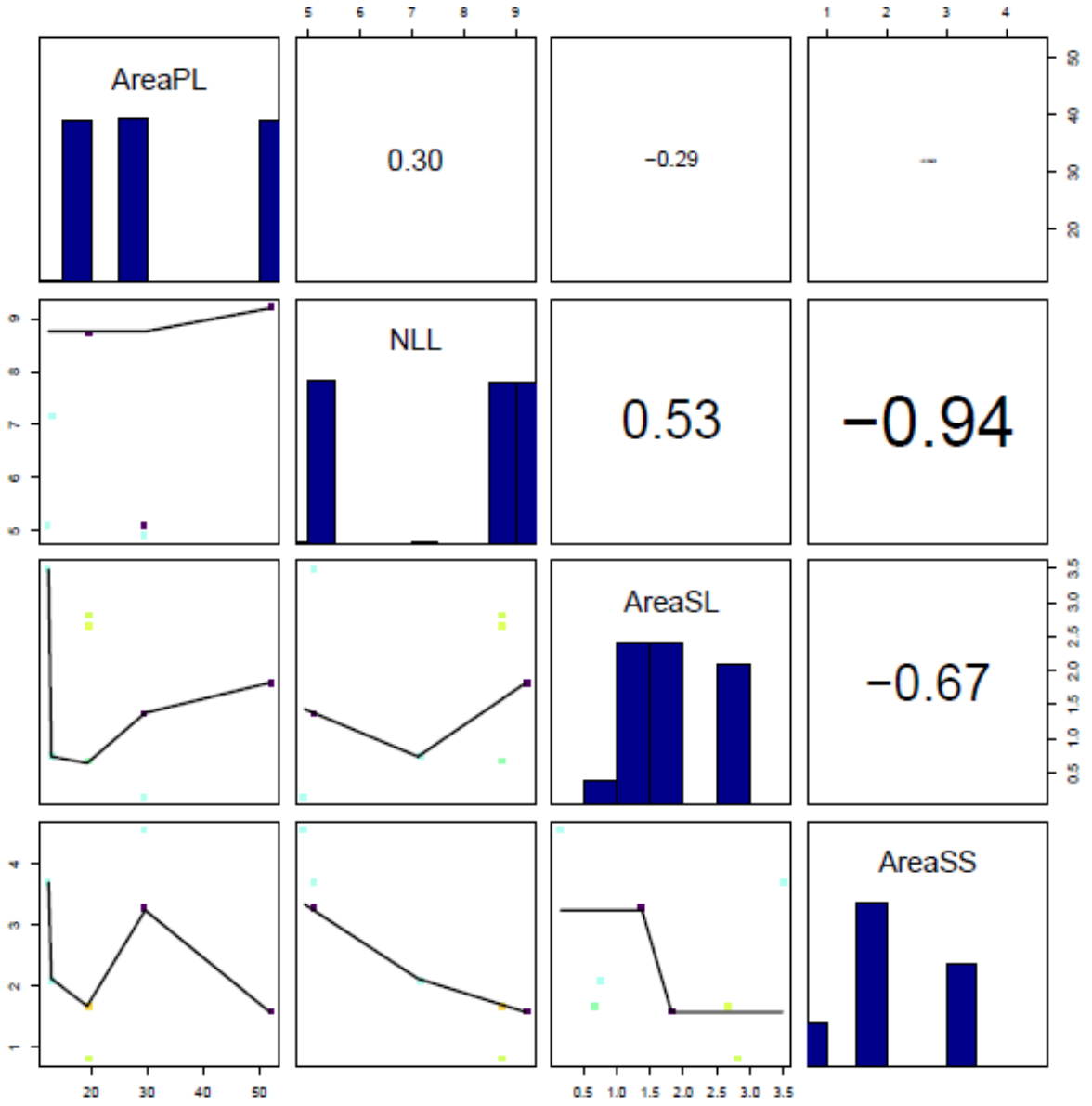
[8] Phyllochrone https://fr.wikipedia.org/wiki/Phyllochrone

[9] Parallélisation du code R https://r-dev-perf.borishejblum.science/parallelisation-du-code-r.html

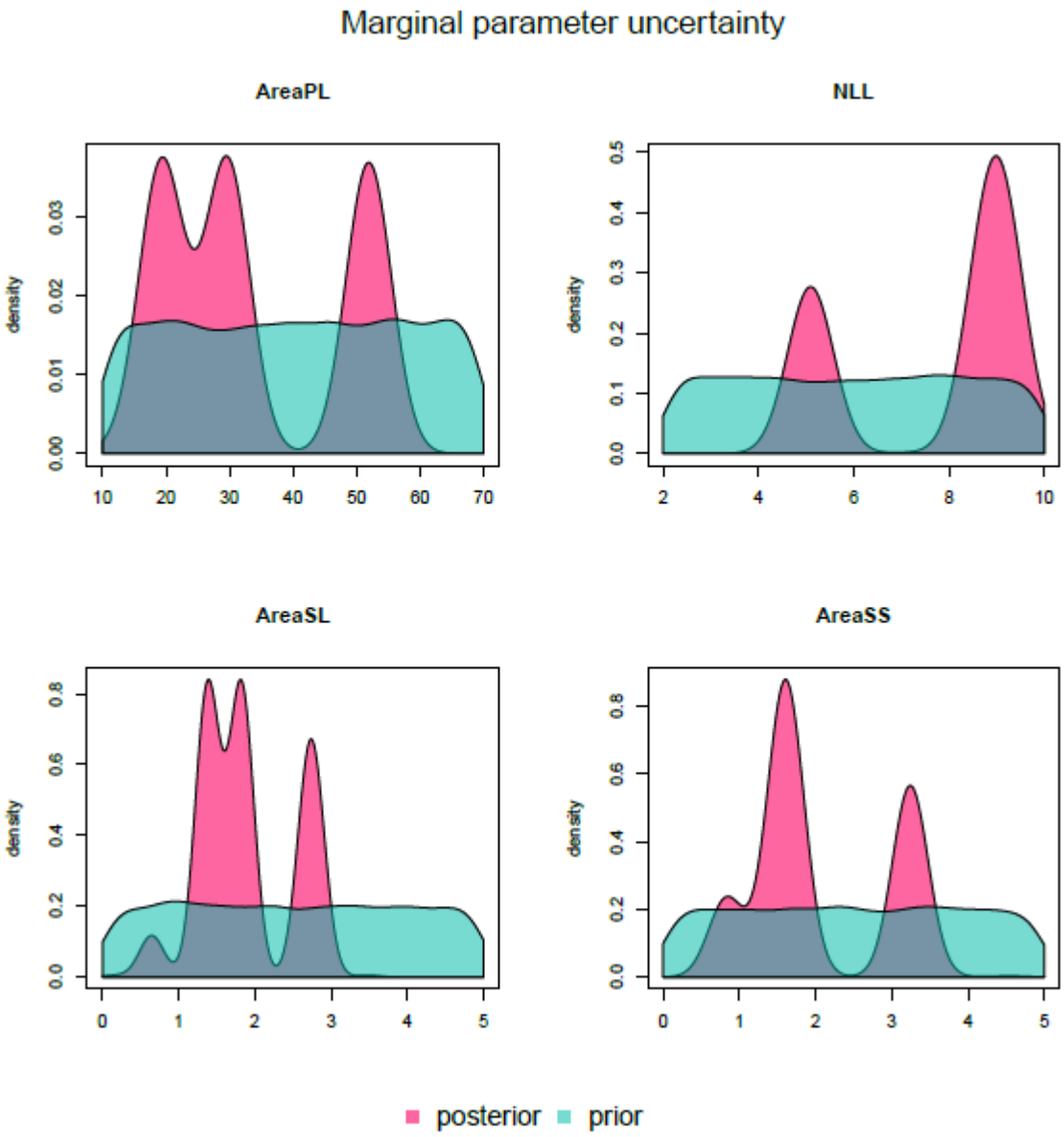
[10] Jasper A. Vrugt. (2015). Markov chain Monte Carlo simulation using the DREAM softwarepackage: Theory, concepts, and MATLAB implementation. Journal of Environmental Modelling & Software.

https://www.researchgate.net/publication/284136871_Markov_chain_Monte_Carlo_simulation_using_the_DREAM_software_package_Theory_concepts_and_MATLAB_implementation

Annexe :



Tracés de corrélation pour l'optimisation de GAI



Tracés marginales pour l'optimisation de GAI