cuspy: Guide d'Utilisation et Développement

Version 1.0.2

J. PRATS-RODRÍGUEZ **P.-A. DANIS**

janv. 26, 2023

Contenu :

CHAPITRE 1

Présentation

1.1 Qu'est-ce que c'est le paquet `cuspy` ?

Le paquet `cuspy` (Calibration, Incertitude et analyse de Sensitivité en PYthon) est un paquet en Python 3 qui proportionne des instruments pour l'estimation de paramètres, et pour les analyses de sensitivité et d'incertitude des modèles.

The package is based on the use of the Python package `pyemu` (White et al. 2016) and the PEST++ suite of software tools (White et al. 2019). PEST and PEST++ (Doherty et al. 2018) are software suites designed for and used mainly by the hydrological community. Ils implémentent plusieurs analyses d'incertitudes linéaires et non linéaires, méthodes de calibration et analyses de sensitivité globale. Le paquet `pyemu` présente une interface Python pour utiliser et configurer les outils de PEST++ et pour accéder et traiter leurs résultats.

Cependant, le paquet `cuspy` n'offre pas toutes les fonctionnalités de PEST++ et de `pyemu` (par exemple, les paramètres liés ne sont pas supportés). En fait, le paquet `cuspy` a été désigné avec l'objective de faciliter l'application des fonctionnalités de `pyemu` et PEST++ aux modèles de température de l'eau et hydrologiques implémentés dans les paquets `okplm`, `glmtools` et `tributary`. De toute façon, le design du paquet est suffisamment général pour être assez indépendant du modèle. Les principaux besoins sont qu'il soit possible de faire tourner le modèle depuis la ligne de commandes et que la communication avec l'utilisateur soit par le moyen de fichiers texte.

1.2 Auteurs

— Jordi Prats-Rodríguez (jprats@segula.es)
— Pierre-Alain Danis (pierre-alain.danis@ofb.gouv.fr)

1.3 Références

— Doherty, J. ; White, J. ; Welter, D. (2018) *PEST & PEST++. An Overview.* Watermark Numerical Computing. 48 p.
— White, J.T. ; Fienen, M.N. ; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling & Software*, 85, 217-228, doi : 10.1016/j.envsoft.2016.08.017.
— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16.* PEST++ Development Team.

CHAPITRE 2

Un peu de théorie

De façon à aider l'utilisateur à mieux comprendre l'utilisation et fonctionnalités de ce paquet, nous faisons une petite introduction à la calibration et l'analyse d'incertitudes.

2.1 Simulations et modèles

2.1.1 Définitions

On peut considérer les modèles comme des approximations de la réalité basées sur des relations mathématiques et sur des inférences dérivées des données. Un modèle reçoit des données de forçage (données d'entrée) et les transforme en données de sortie (simulation) par l'application d'une série de procédures mathématiques. Le retour d'un modèle peut être de plusieurs types, définis dans la table suivante :

Tableau 2.1 – Définitions des types de retour d'un modèle[1]

Terme	Définition
Simulation	Reproduction quantitative du comportement d'un système, données quelques entrées mais sans faire référence à aucun retour observé.
Prédiction	Reproduction quantitative du comportement d'un système à l'avance, mais avec des observations des entrées, variables d'état, et sorties jusqu'au départ de la prédiction.
Simulation rétrospective	Application de la technique de prédiction à données dans le passé comme si les entrées et sorties n'était disponibles que jusqu'à un certain point dans le temps.
Projection	Simulation du comportement future d'un système depuis des suppositions a priori sur les données d'entrée futures.

En fonction d'approximations de la réalité, les modèles sont souvent utiliser pour donner support à des décisions de gestion et pour comprendre mieux le comportement des systèmes qu'ils essayent de reproduire. À travers la comparaison des résultats de simulation aux observations il est possible d'identifier des cas où le comportement du modèle est pire. Par l'analyse de ces cas, et la récollection de données complémentaires, les modèles peuvent être utilisés aussi pour améliorer notre compréhension du système modélisé.

1. Basé sur Beven & Young (2013).

Note : Les modèles sont des instruments utiles pour analyser le comportement d'un système et pour en incrémenter notre compréhension.

2.1.2 Le procès de modélisation

Le procès de modélisation peut être sous-divisé en 10 étapes (Jakeman et al., 2006) :

1. Définition du but du modèle
2. Spécification du contexte de modélisation : portée et ressources
3. Conceptualisation du système, spécification de données et des connaissances a priori
4. Sélection des caractéristiques du modèle
5. Détermination de la méthode pour identifier la structure du modèle et les valeurs des paramètres
6. Sélection des critères de performance et algorithmes
7. Identification de la structure du modèle et des valeurs des paramètres
8. Vérification conditionnelle, inclus les tests de diagnose
9. Quantification de l'incertitude
10. Évaluation du modèle

De ces étapes, nous sommes intéressés ici aux dernières. Cependant, quelques-unes des techniques présentées dans ce document peuvent apporter information utile pour d'autres étapes du procès aussi. De plus, le modèle calibré peut être d'utilité pour donner support aux étapes initiales du procès de modélisation (p. ex., quelles mesures additionnelles il faudrait prendre) en itérations postérieures.

À la fin du procès, on espère d'atteindre un modèle qui ait les caractéristiques suivantes (Beck 1987) :
— Les paramètres du modèle ne devraient varier pas dans le temps. Les valeurs des paramètres qui varient dans le temps peuvent indiquer une structure du modèle imparfaite ou l'existence de procès non pris en considération.
— La variance des résidus et la covariance des paramètres sont petites, ce qui indique que les paramètres du modèle ont été bien identifiés.
— Les résidus sont petits et ne sont pas attribuables à aucun mécanisme causal non aléatoire.

2.1.3 Évaluation de la performance du modèle

Si les modèles doivent être utilisés comme des outils de gestion, pour apporter de l'information sur laquelle baser des décisions, il est complètement naturel d'évaluer leur capacité de simuler le comportement du système. La comparaison des simulations aux observations ou résultats espérés est connue comme évaluation du modèle et peut prendre plusieurs formes.

Tableau 2.2 – Définitions sur l'évaluation de modèles[page 5, 2]

Terme	Définition
Vérification	Seulement pour tester si le code informatique du modèle reproduis de façon correcte les équations qu'il est censé de résoudre. Il s'agit généralement seulement d'une vérification conditionnelle qui dépends de la portée des tests réalisés.
Validation	Évaluation conditionnelle que le code informatique d'un modèle reproduis quelque comportement observé du système étudié avec une qualité acceptable.
Falsification	L'appréciation que le code informatique d'un modèle ne reproduis pas quelque comportement observé du système étudié avec une qualité acceptable.
Apte à l'utilisation prévue	Évaluation conditionnelle que le code informatique d'un modèle est approprié pour son utilisation dans un certain type particulier d'application.

Bennett et al. (2013) propose a five-step procedure to assess the performance of environmental models :

1. Réanalyse de l'objet, échelle et portée du modèle. Il faut s'assurer d'avoir une idée claire de l'objectif du modèle : quelles sont les variables d'intérêt principales et comment elles seront utilisées. Ça inclue décider quelle est la qualité désirée des simulations, soit, qu'est-ce qui constitue une bonne performance dans le type particulier de scénario de modélisation.
2. Caractérisation des données à utiliser pour la calibration et la réalisation de tests. Il faut vérifier la qualité des données (présence de valeurs aberrantes, inhomogénéités, etc.) et leur extension (est-ce qu'elles sont suffisantes ?). Sélection de sous-ensembles de données pour la calibration et pour la validation.
3. Analyses visuelles et d'autre type pour identifier des mauvais comportements du modèle et pour caractériser la performance globale. Il faut utiliser figures, graphiques, animations et d'autres méthodes visuelles pour acquérir une vision d'ensemble de la performance du modèle.
4. Sélection de critères basiques de performance. Il faut choisir des métriques, telles que le RMSE et r^2, pour mesurer la performance du modèle. Bennett et al. (2013) provide several methods for measuring quantitative model performance. La méthode choisie doit prendre en considération l'objectif de la modélisation et les données disponibles. Pour une évaluation compréhensive des modèles il est conseillé d'utiliser systèmes de métriques (ou fonctions à objectif multiple) qui mesurent différents aspects de la performance du modèle.
5. Utilisation de méthodes plus puisées pour résoudre problèmes de modélisation. Il faut analyser l'évaluation de la performance du modèle et voire s'il est nécessaire de faire quelque amélioration dans le procès, retournant aux étapes antérieures s'il le faut.

Une mauvaise performance du modèle peut indiquer :

— les données disponibles sont insuffisantes pour calibrer le modèle de façon appropriée, et il faudrait obtenir plus de données ;
— il y a des problèmes avec les données d'entrée (il pourrait y avoir des biais non détectés ou d'autres problèmes) ;
— l'utilisation d'une autre méthode de calibration pourrait être préférable (la méthode pourrait avoir atteint un maximum local, au lieu d'un maximum global) ;
— la structure du modèle doit être améliorée.

2.1.4 Une note d'avertissement

Bien que les modèles soit des outils inestimable pour la recherche scientifique et la gestion, ils ne sont libres de défauts et il faut que le modélisateur soit conscient de leur limitations. La validation et vérification absolues des modèles numériques est impossible pour des raisons épistémiques (Oreskes et al. 1994). Un modèle ne peut pas reproduire exactement le comportement d'un système naturel à cause de nos connaissances limitées du système. En fait, un modèle est une représentation incomplète de la réalité qui utilise des approximations et des suppositions. De plus, les mesures capturent l'état du système de façon limitée : elles sont susceptibles d'erreurs de mesure et il peut y avoir de limitations à la résolution spatiale ou temporelle.

Ainsi, la validation et vérification sont seulement conditionnelles (Boschetti et al. 2011). Elles sont conditionnelles aux rangs des valeurs des paramètres et des données d'entrée utilisés : imperfections du modèle ou des données pourrait être détectées dans un temps futur, quand le modèle a été testé en d'autres situations ou avec plus de données.

Le retour du modèle dépends aussi des suppositions implicites ou explicites faites dans sa spécification et de comment on a implémenté le comportement du système. En fait, le modèle ne sera pas capable de reproduire un comportement que n'a pas été spécifié dans son code. Aussi, les données de calibration conditionnent les valeurs des paramètres. En conséquence, le modèle aura une meilleur performance quand le comportement du système est similaire au comportement imbu dans les données de calibration.

Note : Une simulation ne peut pas être meilleur que ce que lui permettent les données et la spécification du modèle.

De façon à se protéger des imperfections du modèle et des données il est une pratique exemplaire d'estimer et communiquer l'incertitude.

2. Modifié depuis Beven & Young (2013, Table 3).

2.2 Identifiabilité

Un concept important dans le procès de modélisation est celui d'identifiabilité. L'analyse de l'identifiabilité des paramètres étudie si c'est possible de déterminer un vecteur de valeurs des paramètres unique qui optimise la performance du modèle ou s'il y a multiples valeurs des paramètres qui donnent une performance du modèle comparable. Le résultat d'un manque d'identifiabilité est l'absence d'une combinaison optimale des valeurs des paramètres unique (le modèle donne des simulations similairement bonnes avec différentes combinaisons des valeurs des paramètres) et l'existence de grandes variances de l'erreur et de la covariance des paramètres. Bonnes introductions au concept d'identifiabilité sont données par Beck (1987) et Guillaume et al. (2019).

> « L'analyse d'identifiabilité des paramètres étudie si c'est théoriquement possible d'estimer valeurs des paramètres uniques depuis les données, d'après les quantités mesurées, les conditions présentes dans les données de forçage, la structure du modèle (et fonction objectif), et les caractéristiques des erreurs dans le modèle et les observations. Dit autrement, elle s'occupe du problème de si des données du *type* correcte sont disponibles pour estimer les valeurs des paramètres souhaitées. » Guillaume et al. (2019, p. 418)

Ainsi, l'analyse d'identifiabilité a relation avec le plan expérimental et les programmes de surveillance, de façon que le mesures enregistrées assurent une identifiabilité optimale. Si la structure du modèle est appropriée au système modélisé, l'acquisition de plus de données devrait contribuer à réduire l'incertitude des paramètres. Cependant, ça ne s'avère que si la mesure est du type correcte. Ceci est lié aussi au concept de valeur des données.

Dans un système linéaire d'équations, les valeurs des paramètres sont identifiables quand le nombre d'inconnues est moins que le nombre d'observations. Quand le nombre d'inconnues est majeur que le nombre d'observations, le système est sous-déterminé et le problème n'est pas bien posé (est il y a infinies possibles solutions).

En modèles environnementaux complexes non linéaires, par contre, la règle précédente n'est pas applicable et il faut utiliser des méthodes plus sophistiquées. En ce type de modèles on trouve souvent un manque d'identifiabilité, puisque la complexité du modèle est d'habitude plus importante que l'information disponible sur le comportement du système. Mais le fait d'avoir autant (ou plus) d'observations que paramètres n'assure pas qu'ils soient identifiables. Par exemple, un modèle peut inclure des descriptions de comportements qui ne sont pas représentés dans les données de calibration, et en conséquence les observations n'apportent pas information pour déterminer les valeurs de leurs paramètres. Dans d'autres cas, il peut être difficile de distinguer plusieurs procès qui se superposent depuis les données disponibles. De plus, le modèle peut même simuler variables d'état pour lesquels il n0y a pas des observations disponibles.

2.2.1 Sources de la manque d'identifiabilité

On peut trouver plusieurs types de manque d'identifiabilité :

- Manque d'identifiabilité structurale ou a priori. Il est dû à la structure du modèle (conceptualisation du système, équations, fonction d'objectifs, modèle de la structure de l'erreur). Ce type de manque d'identifiabilité peut être analysé indépendamment de la disponibilité des données. Quand il y a manque d'identifiabilité, il y a « un manque d'information sur un ou plusieurs paramètres du modèle et produit une variation non borné des paramètres même en absence de bruit. » La résolution du problème du manque d'identifiabilité structurale peut comporter l'acquisition de données sur d'autres variables du modèle ou l'utilisation d'une différente structure du modèle.
- Manque d'identifiabilité pratique ou a posteriori : il correspond au manque d'identifiabilité dues à autres causes que la structure du modèle, principalement données de forçage et erreurs du modèle et des observations.
  - Manque d'identifiabilité dû aux données de forçage. Les modèles complexes peuvent inclure formulations qui décrivent plusieurs types de comportements. Si un de ces comportements n'est pas représenté par les données de calibration, il n'y a pas d'information pour estimer la valeur des paramètres spécifiquement liés à ce comportement. Pour résoudre ce problème il faudrait collecter des données sous différentes conditions expérimentales ou environnementales.
  - Manque d'identifiabilité due aux erreurs du modèle et des observations. La présence d'erreurs structurales et bruit/erreur dans les mesures cause une incertitude des paramètres qui peut affecter l'identifiabilité des paramètres. Pour améliorer la situation il peut être utile d'analyser l'influence de l'erreur de mesure sur l'estimation des paramètres.

Note : Le manque d'identifiabilité structurale implique un manque d'identifiabilité pratique. Mais l'identifiabilité structurale n'implique pas l'identifiabilité pratique.

2.2.2 Méthodes pour analyser l'identifiabilité

L'identifiabilité structurale peut être évaluée par l'analyse des équations du modèle. Par exemple, dans les équations suivantes les valeurs des paramètres ne sont pas identifiables :

$$Y = (\alpha + \beta)X$$
$$Y = \alpha\beta X$$

Infinies combinaisons de valeurs pour α et β donnent le même résultat.

L'identifiabilité peut être évaluée aussi avec des outils d'analyse de sensitivité. Si la réponse d'un modèle n'est pas sensible à un paramètre, le paramètre n'est pas identifiable. Cependant, un modèle peut être sensible à un certain paramètre et il peut être comme même impossible d'identifier la valeur du paramètre.

L'identifiabilité peut être analysée aussi par la visualisation de la surface de réponse du modèle avec des données réelles ou synthétiques. L'existence de surfaces plates indique manque d'identifiabilité, alors que différents pics avec valeurs pareilles de la fonction d'objectifs indiquent identifiabilité locale. Pour de modèles simples, il est possible de visualiser la surface de réponse avec de graphiques en 2D ou 3D. Pour des modèles plus complexes, il est possible d'utiliser de graphiques de points, qui montrent la variation de la variable de réponse pour un ou deux paramètres à la fois que varient tous les autres paramètres.

Finalement, quelques indices ont été proposés pour évaluer l'identifiabilité. C'est le cas de la mesure d'identifiabilité proposé par Doherty & Hunt (2009) basée sur l'analyse de la variance de l'erreur.

2.3 Calibration

Calibration, estimation de paramètres et couplage d'historique sont trois synonymes pour le procès d'identification de la valeur optimale des paramètres d'un modèle. L'optimalité est définie ici comme la minimisation d'une fonction d'objectifs, qui représente quelque sorte de distance entre valeurs simulées et observées. D'habitude le modélisateur essaye d'obtenir un optimum global, et après analyse l'incertitude des paramètres et prédictive par rapport à cet optimum. L'objectif est de réussir à réduire l'incertitude prédictive et des paramètres après le procès de calibration.

Optimum global : meilleure solution au large de l'entier espace des paramètres.

Optimum local : meilleure solution dans le voisinage immédiat dans l'espace des paramètres.

Quand les conditions sont optimales, c'est-à-dire, il y a un optimum global clair et il n'y a pas d'optimums locaux, un algorithme automatique de descente de pente peut être utilisé.

Cependant, l'identification d'un optimum peut être empêchée par l'utilisation de paramètres de seuil, par la corrélation entre paramètres, par l'autocorrélation et heteroscedasticité dans les résidus et par les paramètres auquel le modèle n'est pas sensible. Ces effets peuvent causer des minimums locaux, des vallées et plateaux dans la surface de réponse des paramètres. Dans de tels cas, les méthodes globales de calibration (telle que l'Évolution Différentielle) peuvent être préférables. Mais il y a des cas où même ces méthodes peuvent avoir des problèmes à identifier un ensemble de valeurs des paramètres unique. Au lieu d'un optimum il y a plusieurs optimums avec performance pareille. Ceci est connu comme equifinalité (Beven & Freer, 2001).

Quelques techniques plus sophistiquées essayent d'affronter le problème de l'equifinalité. Par exemple, les méthodes de Monte Carlo comme GLUE (Generalized Likelihood Uncertainty Estimation) (Beven & Binley, 1992). La méthode GLUE n'est pas implémentée dans ce paquet, mais peut être appliquée facilement par un traitement approprié du retour de la fonction `monte_carlo()`.

Aussi, le Lissage Itératif d'Ensemble, plutôt que chercher un seul ensemble de valeurs des paramètres optimales, en obtient plusieurs. La méthode extrait plusieurs échantillons aléatoires depuis les distributions des paramètres a priori et ajuste leurs valeurs itérativement, de façon que chaque ensemble de valeurs des paramètres peut être considéré comme calibré. L'ensemble des ensembles optimisés peut être considéré comme un échantillon de la distribution a posteriori.

2.4 Validation

La validation ou, plus adroitement, validation conditionnelle consiste en l'évaluation de la performance du modèle sous des conditions différentes de celles de la calibration. L'objectif est de tester si le comportement du modèle est aussi bon sous un groupe différent de circonstances.

Si le comportement simulé du système est cohérent avec le comportement observé, on ne peut pas présumer que le modèle a été absolument validé ; il a été validé seulement pour la série de circonstances inclues dans les jeux de données de calibration et validation.

Si les résultats du modèle sont incohérents avec les observations, il faudrait réviser les données, la structure du modèle et les valeurs des paramètres pour identifier la source du désaccord. Dans un certain sens, plutôt que confirmer la validité du modèle, l'objectif de la validation peut être considéré comme un essai de falsifier le modèle, pour trouver des aires où il a besoin d'amélioration.

2.5 Analyse de sensitivité

Si on varie légèrement les présuppositions du modèle et des données d'entrée, le comportement prédit variera aussi. En quel dégrée varie le comportement simulé si on varie la structure du modèle, les valeurs des paramètres et les données d'entrée dans des rangs raisonnables ? Répondre à cette question est l'objectif de l'analyse de sensitivité.

Dit autrement, l'analyse de sensitivité est l'analyse de comment varie le retour du modèle en fonction des variations des valeurs des paramètres. Cette définition peut être transposée dans la pratique de plusieurs façons :

- En définissant la sensitivité comme la dérivée partielle de la sortie Y par rapport au paramètre X_i : $S_i = \delta Y / \delta X_i$. Ceci est l'usage qu'on trouve dans les analyses de sensitivité locaux (la sensitivité est évaluée sur un point donné), et les indices de sensitivité sont souvent estimés en faisant varier un paramètre à chaque fois. Cependant, dans les modèles non linéaires la sensitivité varie quand on se déplace dans l'espace des paramètres.
- Les coefficients de régression linéaire peuvent être considérés comme des indices de sensitivité pour les modèles linéaires. En particulier, les coefficients de régression standardisés (SRC) indiquent la contribution de la variance de chaque facteur d'entrée sur la variance du retour du modèle. Les données pour la régression linéaire peut être obtenue à partir de simulations de Monte Carlo. Du moment où les SRCs sont calculés sur un rang de valeurs ils peuvent être considérés comme des indices de sensitivité globaux. De plus, les modèles de régression linéaire peuvent être utilisés pour évaluer le dégrée de linéarité d'un modèle. Si le coefficient de détermination du modèle est élevé (par exemple, $R^2 \geq 0.7$), le modèle peut être considéré linéaire et les SRCs peuvent être utilisés pour l'analyse de sensitivité (Hall et al. 2009).
- Si le système est non linéaire ($R^2 < 0.7$), des méthodes (globales) plus sophistiquées seront nécessaires. Good introductions to global sensitivity analysis are given by Saltelli et al. (2004) and Saltelli et al. (2008). On donne ci-dessous quelques options :
  - Méthode de Morris. Il s'agit d'une méthode de « variation une à la fois » qu'analyse la sensitivité à différents points le long du rang de valeurs des paramètres pour obtenir un mesure de sensitivité moyenne. Elle a un bas coût computationnel et est donc spécialement adaptée aux modèles avec des longs temps de calcul ou quand il y a un grand nombre de paramètres incertaines. La méthode de Morris est implémentée dans ce paquet.
  - Méthodes de décomposition de la variance. La variance du retour du modèle est décomposé en différentes composantes, chaque une attribuable à un paramètre (ou combinaison de paramètres). On a implémenté la méthode de Sobol dans ce paquet. Elle a un coût computationnel plus élevé que la méthode de Morris.

— Le filtrage de Monte Carlo (ou aussi méthode de Hornberger-Spear-Young ou Analyse de Sensitivité Régionalisée). On utilise un seuil de performance pour séparer entre des simulations « acceptables » et « inacceptables » (et ensembles de paramètres). S'il y a des différences entre la distribution des valeurs d'un paramètre pour les simulations acceptables et inacceptables, le modèle est sensible à ce paramètre. Ils ont un coût computationnel très élevé (de l'ordre de milliers d'exécutions de Monte Carlo).

Note : Les résultats de l'analyse de sensitivité dépend du rang de valeurs des paramètres exploré pendant l'analyse, c'est-à-dire, de l'incertitude des paramètres individuels.

L'analyse de sensitivité peut être utilisé pour répondre aux questions suivantes (Hall et al. : 2009) :

— Quels sont les facteurs qu'influence le plus le retour du modèle ?
— Il y a des facteurs qu'il faudrait investiguer plus en détail pour augmenter la crédibilité du retour du modèle ?
— Il y a des facteurs qui ont une influence méprisable sur le retour du modèle, de façon qu'ils puissent être écartés dans les analyses subséquentes ?
— Le modèle reproduit de relations connues ?
— Il y a des régions dans l'espace des entrées pour lesquelles la variation du retour du modèle est maximale ?
— Quelles sont les régions optimales dans l'espace des paramètres pour la calibration ?
— Il y a des facteurs ou groupes de facteurs qu'interagissent entre eux ?
— Est-ce que le retour du modèle est robuste face à différences raisonnables de la valeur des facteurs d'entrée ? Ou est-ce que le modèle est très sensible à des petites variations de la valeurs d'un paramètre (ce qui pourrait être indicatif de surcalibration) ?

Saltelli et al. (2019) analysed the main pitfalls in the use of sensitivity analysis and gave recommendations for improving its usage.

2.5.1 Analyse de sensitivité et analyse d'incertitudes

L'analyse de sensitivité et l'analyse d'incertitudes sont des concepts relationnés.

— L'analyse d'incertitudes consiste en la quantification des incertitudes des entrées du modèle et en leur propagation pour obtenir l'incertitude des prédictions du modèle.
— L'analyse de sensitivité, par contre, consiste en l'évaluation de comment les variations des entrées su modèle, individuellement ou en combinaison, influencent la variation des sorties du modèle.

Ainsi, le point central d'intérêt de l'analyse d'incertitudes est la détermination des limites de nos connaissances (et de notre manque de connaissance) sur le système et ses réponses. Alors que l'analyse de sensitivité est plus centré sur la détermination des facteurs qu'affectent les plus le retour du modèle.

2.6 Analyse d'incertitudes

Les modèles environnementaux ne peuvent pas prédire le futur. Ce qu'ils fournissent sont des possibles comportements réalistes qui s'approximent au comportement du système réel. L'analyse d'incertitudes s'occupe de la crédibilité qu'on peut attribuer à ces prévisions et des rangs dans lesquels une quantité prédite peut varier.

L'incertitude des simulations d'un modèle a trois causes principales :

— Incertitude structurale, due à l'incertitude dans la structure du modèle. Comme notre connaissance du comportement du système est incomplète, notre implémentation des procès du système ne sera parfaite non plus.
— Incertitude paramétrique, due à l'incertitude sur les valeurs des paramètres du modèle. Les données disponibles et les mesures de l'état du système sont limitées et elles sont une représentation partielle du système étudié. Ainsi, les valeurs estimées des paramètres basées sur ces données sont imprécises.
— Erreur de la mesure et variabilité naturelle. En conséquence, il y a incertitude sur les conditions de frontière du modèle (état initial et données de forçage) et état du système. La variabilité naturelle inclue différentes composantes, telles que l'hétérogénéité spatiale, la variabilité environnementale et la variabilité génétique.

Idéalement, après un certain temps, on peut réduire l'incertitude des simulations par l'amélioration de notre connaissance du système (en adaptant la structure du modèle d'après les données disponibles), par la réduction de l'erreur de mesure (avec de données plus précises) et par l'identification plus précise des valeurs des paramètres (avec plus de données et données plus précises). Cependant, un certain dégrée d'incertitude reste toujours.

Beck (1987) a identifié quatre types de problèmes qui font l'objet de l'analyse d'incertitudes :

— L'analyse de l'incertitude de la structure du modèle (ou sur les relations entre les variables qui caractérisent le comportement du système). Pour répondre cette question on peut utiliser différents modèles ou structures du modèle.
— L'analyse de l'incertitude de la valeurs des paramètres. L'objectif est d'obtenir la distribution des valeurs des paramètres, en base à l'analyse de quelles valeurs des paramètres donnent des simulations cohérentes avec les observations du comportement du système.
— L'analyse de l'incertitude des prédictions faites avec le modèle ou incertitude prédictive. L'incertitude prédictive a priori est l'incertitude des prédictions faites avant d'incorporer l'information de nouvelles mesures. L'incertitude prédictive a posteriori est l'incertitude des prédictions après l'incorporation des nouvelles mesures.
— La planification expérimentale ou programmes de surveillance pour réduire les principales incertitudes. Cette analyse est liée à l'analyse d'identifiabilité et la valeur des données.

2.6.1 Quelle méthode d'incertitude utiliser ?

On trouve deux approches principales à l'analyse d'incertitude prédictive :

— Méthodes de premier ordre. Limitées au cas linéaire, à cause de l'intraitabilité des solutions mathématiques dans les cas non linéaires. Souvent il y a une supposition de fonctions de densité de probabilité normales pour une meilleure traitabilité. Elles utilisent l'algèbre linéaire pour estimer l'incertitude et ont besoin de peu de simulations.
— Méthodes non linéaires. Ces méthodes peuvent prendre en considération toute la complexité des modèles. Quelques exemples :
  — Méthodes de Monte Carlo. Elles sont basées en l'obtention d'un grand nombre de simulations par le biais de la variation des valeurs des paramètres du modèle. Elles requièrent un grand nombre de simulations.
  — Lissage Itératif d'Ensemble. C'est une technique spécialement désignée pour calibrer et quantifier l'incertitude des modèles non linéaires avec grands nombres de paramètres. Le nombre de membres de l'ensemble devrait être supérieur à la dimensionnalité de l'espace de solution. Pour les modèles avec un grand nombre de paramètres, le coût computationnel de l'IES en raison du nombre d'exécutions du modèle est beaucoup plus petit que celui des méthodes de Monte Carlo.

Bien que les modèles environnementaux soient largement non linéaires, les méthodes linéaires peuvent y être appliquées aussi si l'incertitude du modèle est suffisamment petite. Ceci est basé sur le fait que le comportement d'une fonction (continuellement dérivable) non linéaire dans un intervalle autour d'un certain point (la valeur du paramètre) peut être présumé linéaire si l'intervalle est assez petit. On peut considérer qu'un modèle a un comportement approximativement linéaire quand le coefficient de détermination du modèle de régression linéaire est supérieur à 0.7.

Quand on sélectionne une méthode, il faut prendre en considération le coût computationnel (nombre d'exécutions et temps de calcul du modèle) et incertitude des paramètres. Quand l'incertitude des paramètres du modèle est élevée, il est préférable d'utiliser des méthodes non linéaires. En itérations postérieures, quand on a obtenu plus de données et l'incertitude des valeurs des paramètres a diminué assez (et la réponse du modèle peut être considéré linéaire), on peut utiliser des méthodes linéaires moins coûteux.

2.6.2 Valeur des données

La valeur des données ou de l'information est une évaluation de l'utilité des données. La valeur des données peut être relationnée à un analyse de coût-bénéfice, où les mesures additionnelles ont de la valeur si la réduction en incertitude supère l'incrément du coût. Ainsi le concept de la valeur des données est spécialement important pour la planification de réseaux de mesure et de campagnes de collection de données.

La valeur des données mesure le potentiel de réduction de l'incertitude de prédictions relevantes pour la gestion. Il y a deux façons d'évaluer la valeur des données :

— comme une diminution de l'incertitude prédictive due à l'addition de données au jeu de données ;
— comme une augmentation de l'incertitude prédictive due à l'élimination de données au jeu de données ;

Cependant, la situation la plus habituelle est que des nouvelles données soient ajoutées au système.

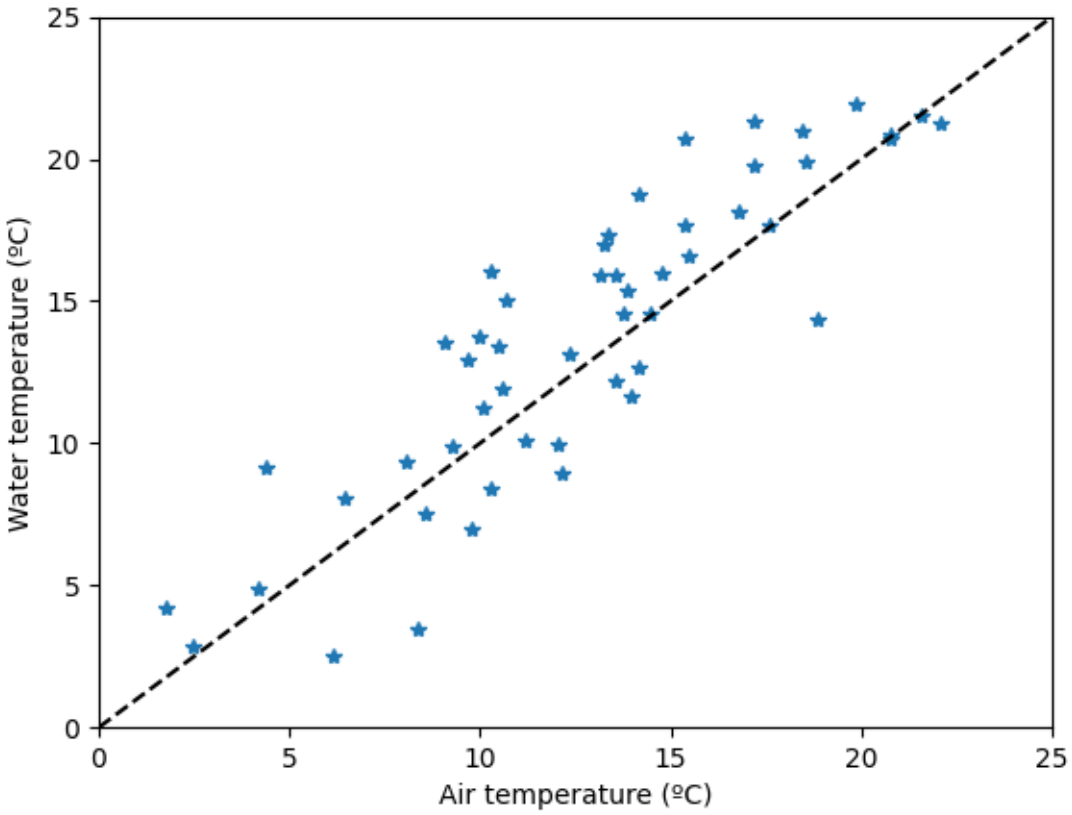
2.7 Un exemple : simulation de la température de la surface de l'eau

Pour démontrer les concepts et techniques implémentées dans ce paquet nous présentons le cas d'étude de la simulation de la température de la surface de l'eau en utilisant un modèle simple qui dépend de la température de l'air.

La relation entre les températures de l'air et de l'eau peut être décrite avec une équation non linéaire. Mohseni et al. (1998) a proposé l'équation sigmoïde suivante :

$$T_w = \mu + \frac{\alpha - \mu}{1 + e^{\gamma(\beta - T_a)}} \tag{2.1}$$

Dans cet exemple, nous appliquons ce modèle au cas de la température de la surface dans la retenue de Naussac. Nous utilisons des mesures par satellite de la température de la surface du jeu de données LakeSST (Prats et al. 2018) et la température de l'air de la réanalyse SAFRAN (Quintana-Seguí et al., 2008), montrées dans la figure suivante. Les données recouvrent la période 1999-2015.



L'objectif de l'exercice est d'estimer la température de l'eau et son incertitude les jours 1er et 15e de chaque mois en 2015.

2.7.1 Incertitude a priori

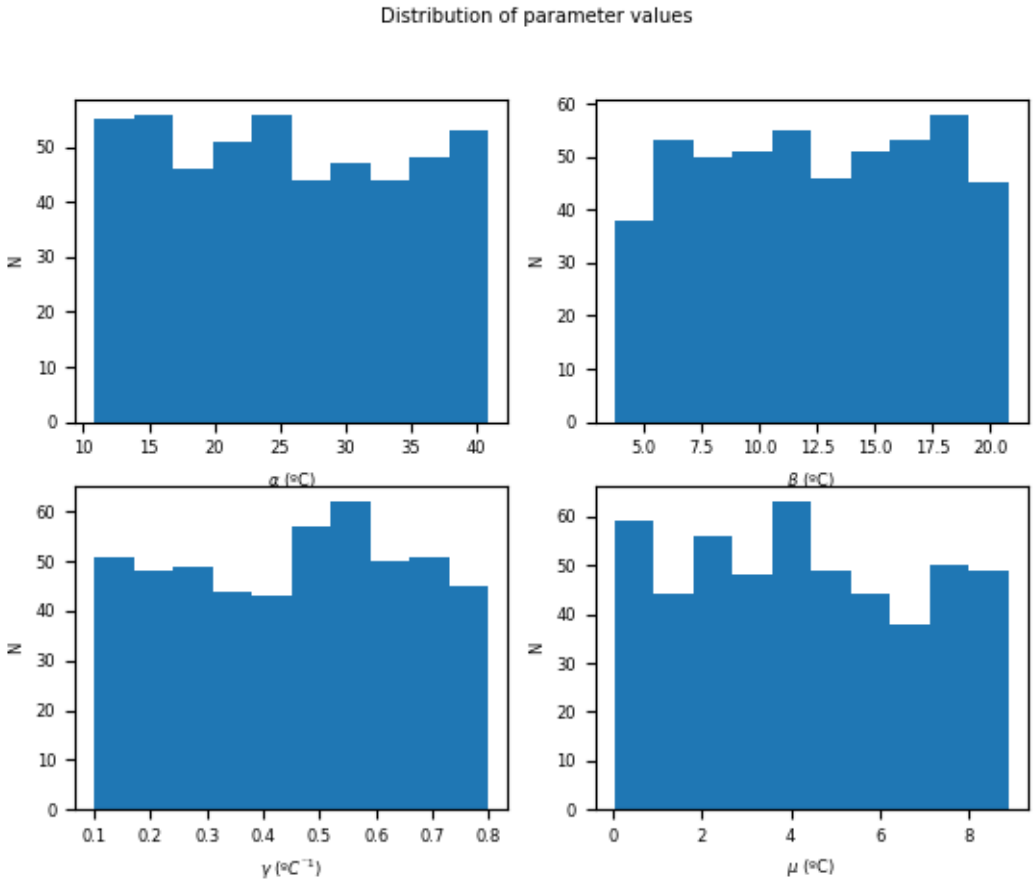
Supposez que nous n'avons pas de données sur la température de la surface dans le réservoir. Et supposez que nous avons besoin de l'estimer. Quelle confidence peut-on attacher à nous estimations ? On peut répondre cette question avec des simulations de Monte Carlo.

Même si on n'a pas de données de terrain, on peut encore faire une estimation en base aux connaissances scientifiques antérieures. On sait que la température de l'eau est très corrélée à la température de l'air et qu'elle peut être estimée avec assez de précision avec l'Éq. (2.1). Aussi, beaucoup d'études ont calibré ce modèle dans plusieurs cas, et en conséquence on peut utiliser l'avis d'expert pour avoir une estimation de la valeur espérée et du rang de variation des paramètres du modèle. Les valeurs de la table suivante sont dérivées depuis les résultats obtenus par Mohseni et al. (1998).

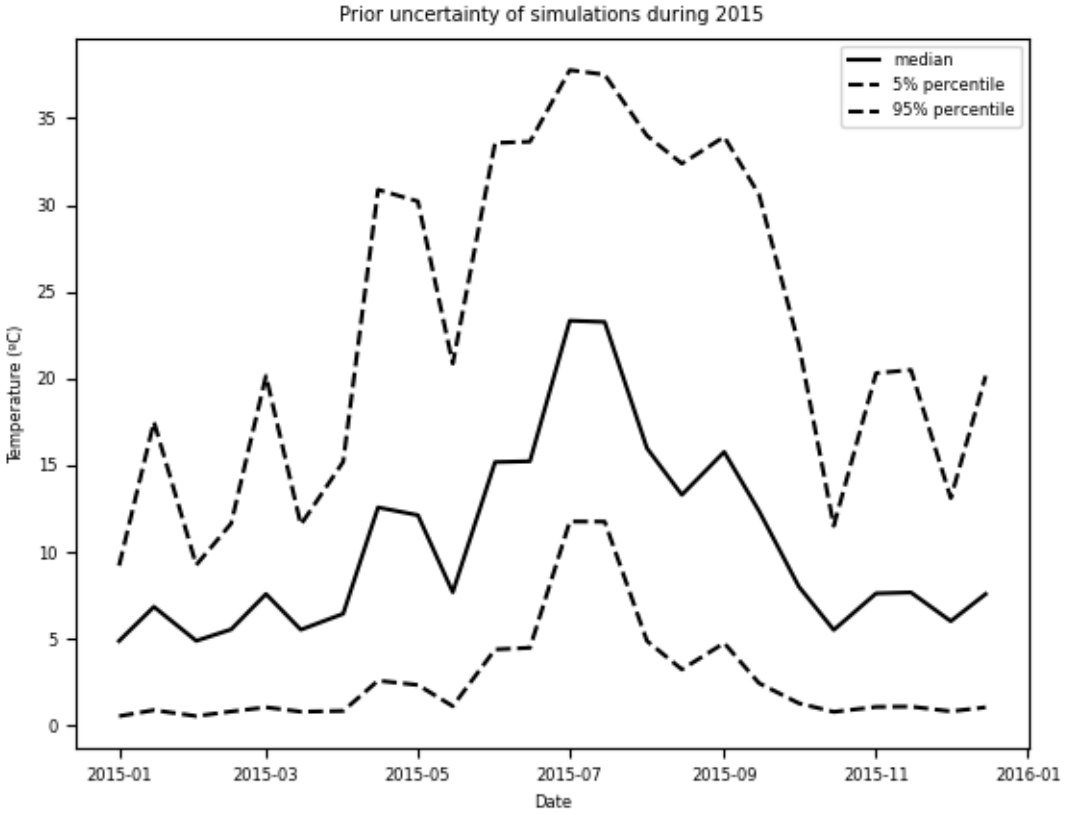
Tableau 2.3 – Une première estimation des paramètres du modèle (basée sur l'avis d'expert)

Paramètre.	Moyenne	Min.	Max.
α (ºC)	26.2	10.8	40.9
β (ºC)	13.3	3.7	20.8
γ (ºC^{-1})	0.18	0.10	0.80
μ (ºC)	0.8	0.0	8.9

On peut utiliser ces valeurs pour extraire plusieurs échantillons aléatoires (p. ex., 500 échantillons) des valeurs des paramètres et réaliser une simulation pour chaque jeu de valeurs des paramètres. Dans ce cas, on présume qu'on n'a pas d'information additionnelle. Quand il n'y a pas d'information sur la forme de la distribution des valeurs des paramètres, on assume souvent qu'elles suivent une distribution uniforme, qui peut être considérée comme une distribution non informative.



Les valeurs simulées pour les différents jeux de paramètres proportionnent l'incertitude a priori des estimations de la température en base aux connaissances antérieurement existantes. On peut voire que, comme c'était attendu, l'incertitude est assez large en ce moment.



2.7.2 Analyse de sensitivité

Pour réduire l'incertitude de la simulation, nous avons besoin de calibrer le modèle et choisir de valeurs appropriées pour les valeurs des paramètres. Cependant, avant de passer à la calibration il est d'utilité de faire une analyse de sensitivité pour investiguer quels sont les paramètres avec une majeure influence sur les résultats de simulation. Vu la large incertitude à ce point-ci, il faudrait utiliser une méthode d'analyse de sensitivité globale. Nous utiliserons ici la méthode de Sobol.

La méthode de Sobol est une méthode de Monte Carlo de décomposition de la variance. L'objectif de la méthode est d'évaluer la proportion de la variance des résultats qui est due à chaque paramètre. Cette méthode peut révéler l'existence de non-linéarité et d'interactions entre paramètres. Quand on utilise cette méthode d'habitude on calcule deux indexes : l'index de sensitivité de premier ordre (S_i) et l'index des effets totaux (S_{T_i}).

Tableau 2.4 – Sensitivité des paramètres du modèle (méthode de Sobol, 400 échantillons)

Paramètre.	S_i	S_{T_i}
α	0.29	0.52
β	0.37	0.55
γ	0.03	0.08
μ	0.01	0.03

L'index de sensitivité de premier ordre S_i mesure la proportion de la variance du retour du modèle qu'est expliquée par chaque paramètre individuellement (sans interactions). L'index S_i varie théoriquement entre 0 (non sensible) et 1

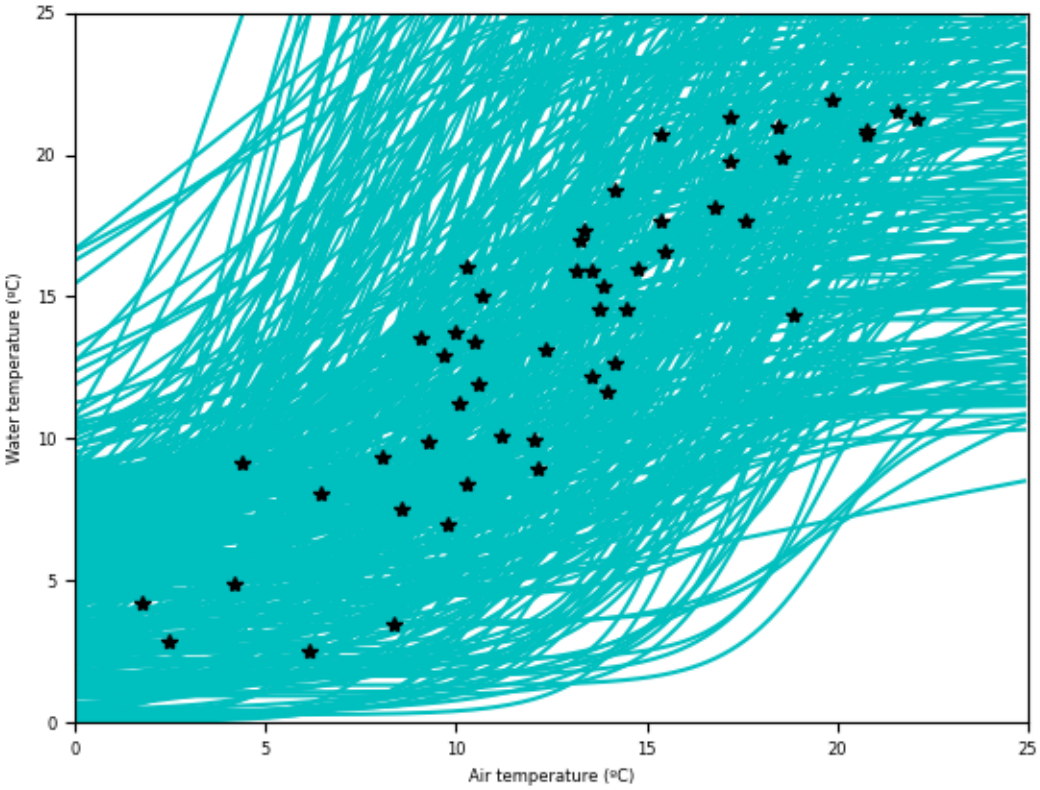
(sensitivité maximale). La somme de tous les S_i est égale à 1 pour des modèles additifs (l'effet de chaque paramètre est indépendant), et elle est moins de 1 pour des modèles non-additifs (il y a des interactions entre les paramètres). Dans notre cas, la somme de tous les S_i est 0.7. Ainsi, la variation des paramètres individuels explique un 70 % de la variance des sorties et on peut considérer que le modèle a un comportement largement linéaire, au malgré de l'espace des paramètres assez large qu'on a exploré ici.

Note : Quand S_i est calculé avec la méthode de Sobol, il peut prendre des valeurs négatives à cause d'imprécisions numériques quand S_i est proche de zéro et la taille de l'échantillon est petite.

L'index des effets totaux S_{T_i} mesure la contribution à la variance des sorties du modèle d'un paramètre en incluant les interactions avec d'autres paramètres. L'index S_{T_i} est égal à S_i si'l n'y a pas d'interactions entre le paramètre X_i et d'autres paramètres, et plus grand que S_i s'il y a des interactions avec d'autres paramètres. Si $S_{T_i} = 0$, le paramètre X_i n'a aucune influence. La somme de tous les S_{T_i} est habituellement supérieure à 1 ; c'est 1 seulement si le modèle est parfaitement additif (pas d'interactions entre paramètres).

Dans notre exemple, les paramètres α et β sont les paramètres les plus influents : la fixation de la valeur d'un entre eux réduirait la variance des sorties de plus d'un 50 %. Le paramètre γ est beaucoup moins influent. Et le paramètre μ n'a pas d'influence puisque son index d'effets totaux est presque zéro. De plus, comme la somme de S_i est moins de 1, la somme de S_{T_i} est plus de 1, et $S_{T_i} > S_i$ pour les paramètres α, β et γ, il y a des interactions entre ces paramètres.

La figure suivante montre les 400 réalisations aléatoires utilisées pour calculer les indexes de Sobol surimposées aux mesures. La figure montre visuellement qu'il y a une grande incertitude dans la valeur des paramètres α et β, qui déterminent, respectivement, l'asymptote supérieure et le point d'inflexion point de la courbe logistique.

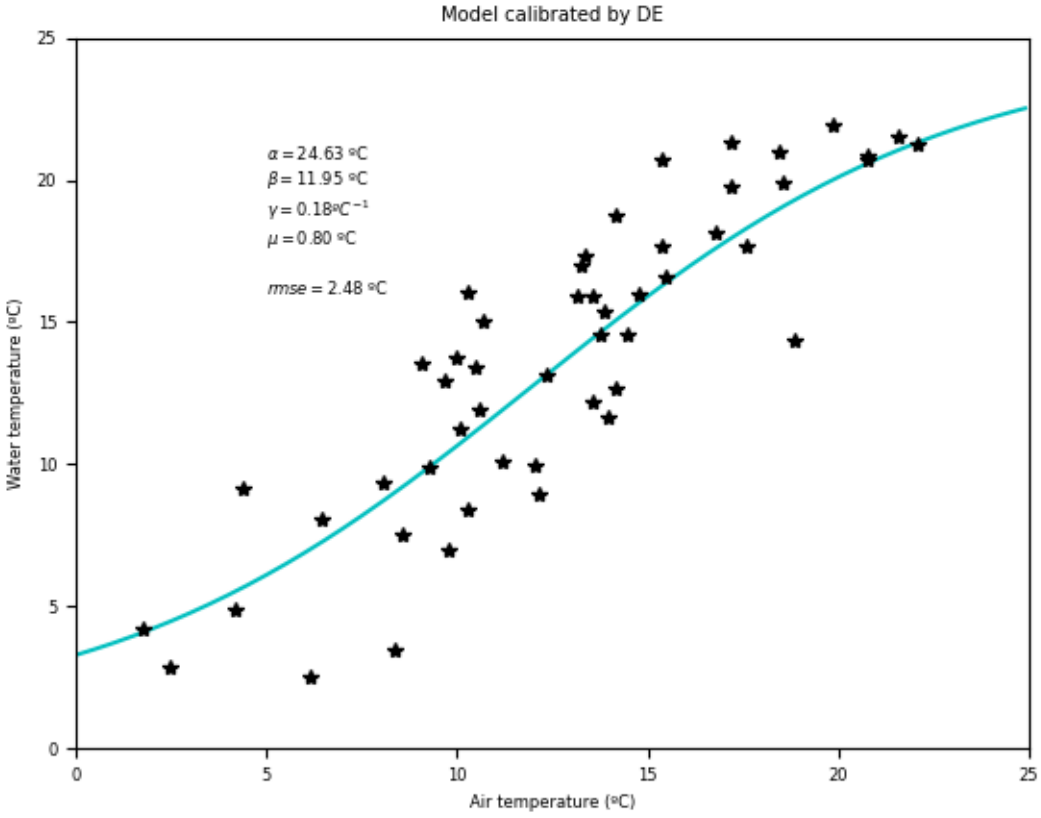


2.7.3 Calibration

Dans la section antérieure on a vu que les paramètres γ et μ n'ont presque pas d'influence sur les résultats de la simulation. Ceci signifie que plusieurs combinaisons des valeurs des paramètres produisent des simulations de qualité similaire, ce qui fait augmenter l'incertitude des valeurs des paramètres. Pour réduire cette incertitude, on peut fixer la valeur des paramètres γ et μ sur les valeurs moyennes, et calibrer la valeur des paramètres α et β.

Le procès susdit de fixer la valeur de certains paramètres est un exemple de régularisation (manuelle), une opération par laquelle on réduit la complexité d'un modèle de façon que le reste de paramètres puisse être estimé de façon univoque (i.e., avec une basse incertitude). D'autres méthodes de régularisation plus sophistiqués incluent la régularisation de Tikhonov et la régularisation par sous-espaces.

Comme il y a une grande incertitude par rapport à la valeur des paramètres, nous utiliserons la méthode d'Évolution Différentielle (DE) pour faire une première calibration. La DE est une méthode de calibration globale qui explore l'entier espace des paramètres, plutôt qu'un rang local de valeurs des paramètres. Ainsi cette méthode proportionne un optimum global.



Après l'application d'une méthode de calibration globale il est conseillé de raffiner la calibration avec une méthode locale. Ainsi, nous répétons la calibration avec l'algorithme GLM, mais maintenant nous ne fixons plus les paramètres γ et μ. On peut voire qu'il y a juste une petite amélioration du RMSE.

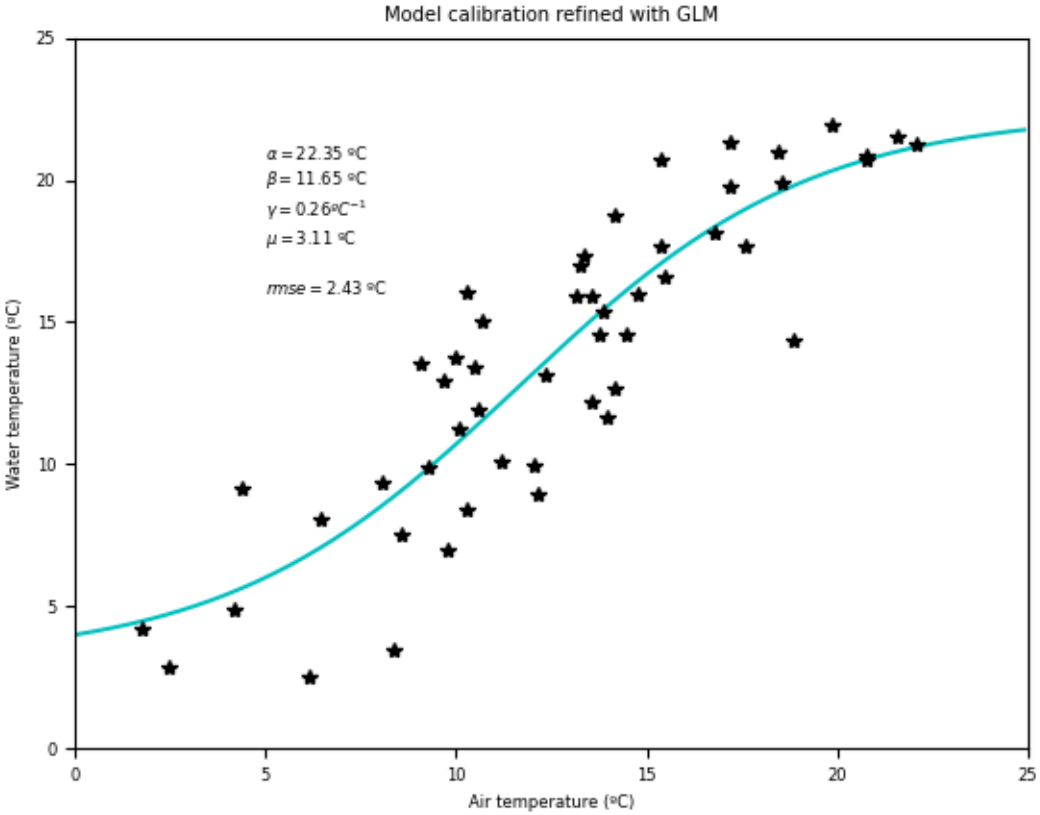


Tableau 2.5 – Valeurs calibrées des paramètres du modèle

Paramètre.	Moyenne	Confidence 2,5 %	Confidence 97,5 %
α (ºC)	22.3	20.2	24.4
β (ºC)	11.7	10.6	12.8
γ (ºC^{-1})	0.26	0.19	0.34
μ (ºC)	3.2	0.6	5.7

2.7.4 Incertitude a posteriori (analyse du complément de Schur)

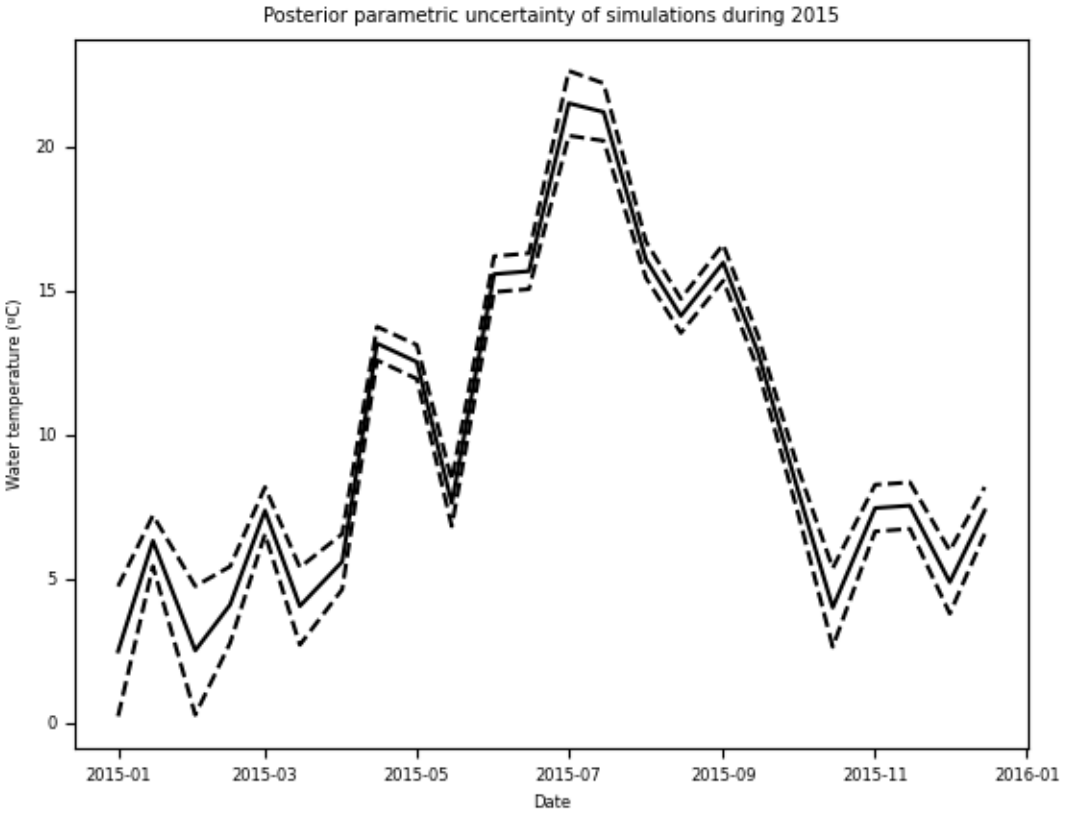
L'analyse du complément de Schur est valide quand le modèle a une réponse linéaire (ou presque linéaire) aux variations de la valeur des paramètres. Nous avons déjà démontré que le modèle a un comportement presque linéaire dans l'espace des paramètres a priori en utilisant la méthode de Sobol de décomposition de la variance. La table suivante montre la réduction d'incertitude des paramètres produite par la calibration d'après l'analyse du complément de Schur.

Tableau 2.6 – Réduction de l'incertitude des paramètres par calibration

Paramètre.	Variance a priori	Variance a posteriori	Réduction dans l'incertitude des par. (%)
α	56.6	1.0	98
β	18.3	0.4	98
γ	0.031	0.002	93
μ	4.95	1.56	69

Après la calibration l'incertitude des paramètres a diminué beaucoup. On peut noter que la réduction dans l'incertitude pour μ est beaucoup moins importante que pour les autres paramètres. En fait, ce paramètre correspond à l'asymptote inférieure de la courbe logistique, mais il y a peu de mesures des basses températures qui puissent apporter information à ce paramètre.

La réduction de l'incertitude des valeurs des paramètres produit une diminution de l'incertitude des simulations. Veuillez noter, pourtant, que dans ce cas l'incertitude comprends seulement l'incertitude paramétrique et ne prend en compte les erreurs de mesure ou les imperfections du modèle.



On voit que l'incertitude n'est pas égale le long de l'année. L'incertitude est plus grande pour les périodes avec des températures élevées et, spécialement, basses.

2.7.5 Valeur des données

La figure suivante montre le pourcentage de réduction de l'incertitude par paramètre et date de la prédiction. On peut voire que les prédictions pour les jours avec de basses température sont influencées principalement par μ et les prédictions pour les jours avec les plus hautes températures sont influencées principalement par α. Ceci est assez logique car α détermine l'asymptote supérieure de la courbe logistique et μ en détermine l'asymptote inférieure.

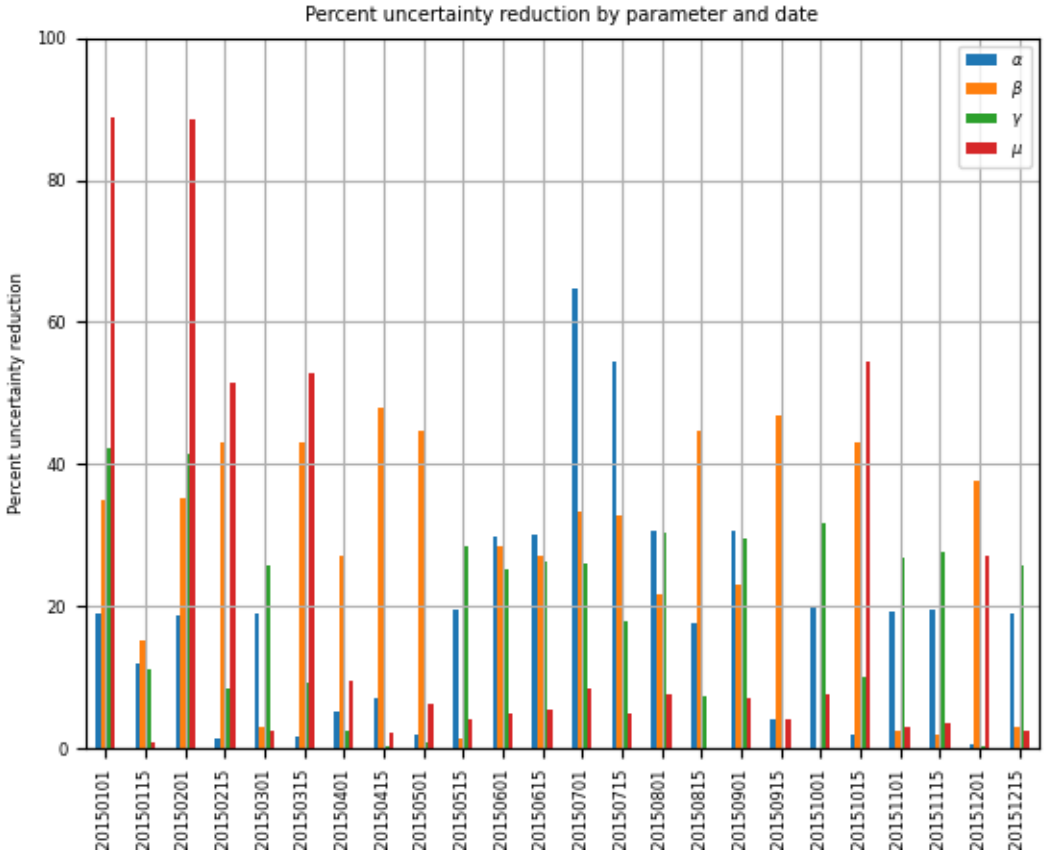
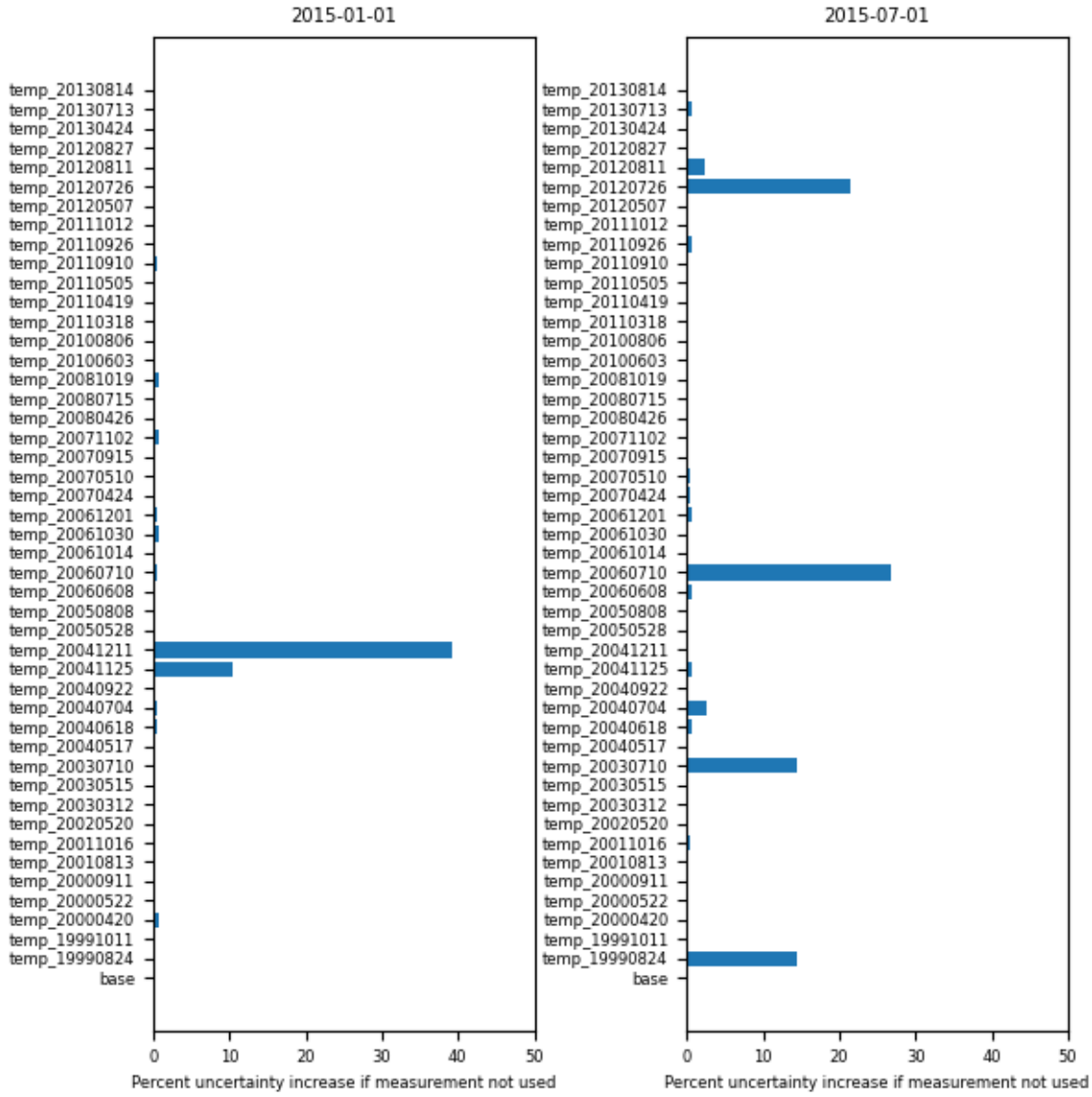


Fig. 2.1 – Cette figure montre le pourcentage de réduction de l'incertitude de l'estimation des paramètres qu'on obtiendrait par l'acquisition d'une mesure dans les jours donnés.

Si on voulait réduire l'incertitude des prédictions en été et en hiver, quels mesures additionnels devrions-nous prendre ? On peut investiguer cette question avec le concept de valeur des données. La figure suivante montre la valeur des données comme le pourcentage d'incrément de l'incertitude quand on élimine une observation pour deux prévisions, celles du 1 janvier 2015 et du 1 juillet 2015. On voit que les données qui ont plus de valeur pour réduire l'incertitude des prédictions sont celles qui correspondent à la même période de l'année. On devrait donc concentrer ses efforts de mesure en été et hiver.



2.7.6 Erreur prédictive a posteriori (analyse de la variance de l'erreur)

L'analyse de la variance de l'erreur est une méthode linéaire alternative pour l'analyse d'incertitudes qui décompose la variance de l'erreur prédictive en trois composantes :

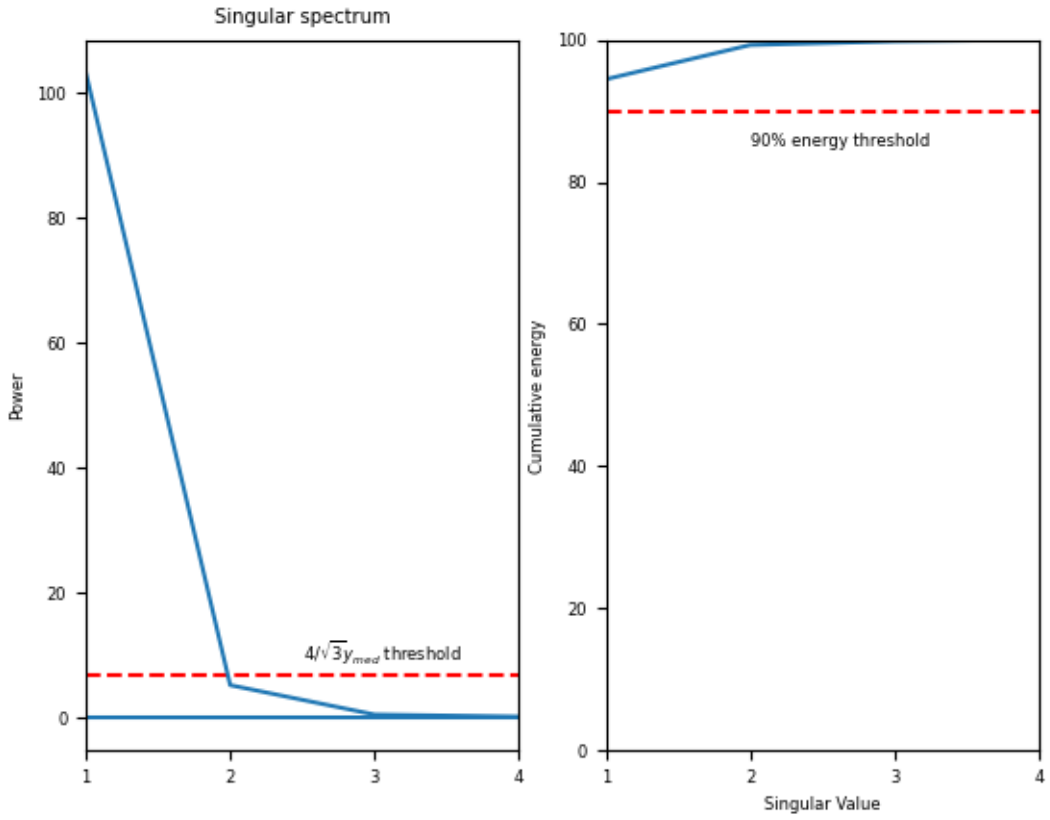
- Erreur potentielle de prédiction due à l'incertitude des paramètres.
- Erreur potentielle de prédiction due à l'écart entre modèle et mesures (incluant l'erreur des mesures et l'erreur structurelle).
- Erreur potentielle de prédiction due à la calibration seulement d'une partie des paramètres du modèle.

Ainsi, on attend que l'erreur prédictive a posteriori soit plus grand que l'incertitude a posteriori.

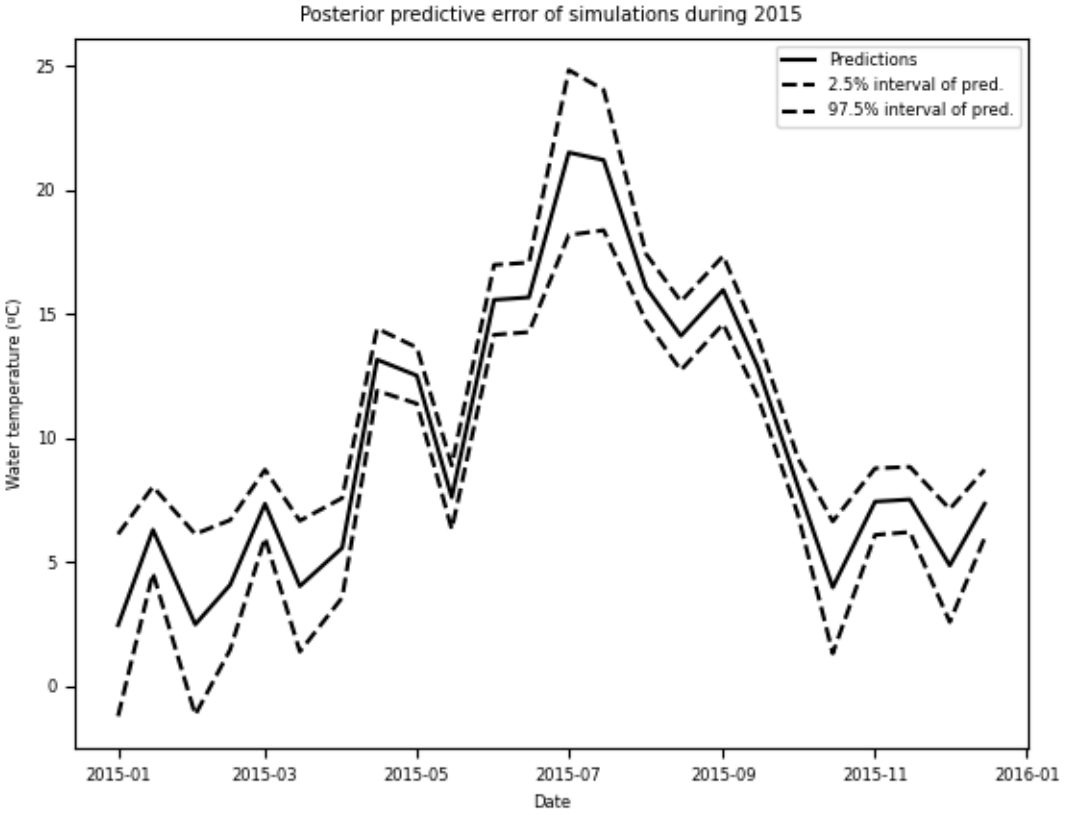
Pourtant, une des difficultés à utiliser l'analyse de la variance de l'erreur est celle d'estimer le nombre de valeurs singuliers à utiliser, ce qui peut introduire un certain dégrée de subjectivité. L'analyse de la variance de l'erreur utilise la décomposition en valeurs singulières (DVS) pour estimer la dimensionnalité effective du système et plusieurs critères ont été proposés pour choisir combien de valeurs singulières il faudrait utiliser :

— Identifier le point avec un angle marqué.
— Utiliser autant de valeurs singulières comme il faut pour représenter au moins un 90 % (ou 99 %) de l'énergie totale.
— Quand les résidus ont une moyenne de zéro et ne sont pas corrélés (bruit blanc), Gavish & Donoho (2014) ont proposé un seuil optimal de $2.858y_{med}$, où y_{med} est la valeur singulière médiane.

Le raisonnement est que le peu de valeurs singulières retenues sont suffisantes pour caractériser la plus grande partie de l'information, alors que les valeurs singulières rejetées représentent majoritairement du bruit.



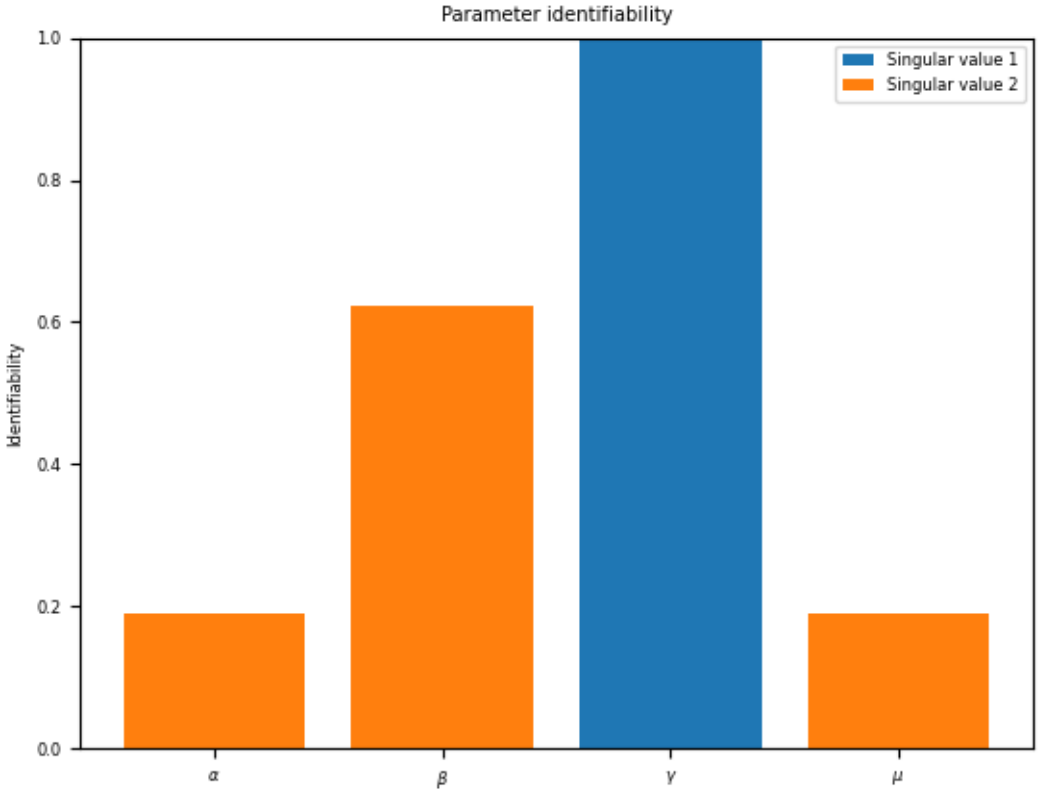
Dans notre cas d'étude, il nous faudrait 2 valeurs singulières d'après le critère du 90 % d'énergie. Mais seulement 1, d'après le seuil strict de Gavish & Donoho. Cependant, ce seuil pourrait ne pas être approprié, car on espère que les résidus soient corrélés. D'ailleurs, la puissance de la deuxième valeur singulière est assez proche de ce seuil. Ainsi, nous utiliserons 2 valeurs singulières dans ce cas.



D'après ces résultats, l'erreur prédictive est plus importante en été et en hiver, en cohérence avec les résultats obtenus pour l'incertitude a posteriori.

2.7.7 Identifiabilité

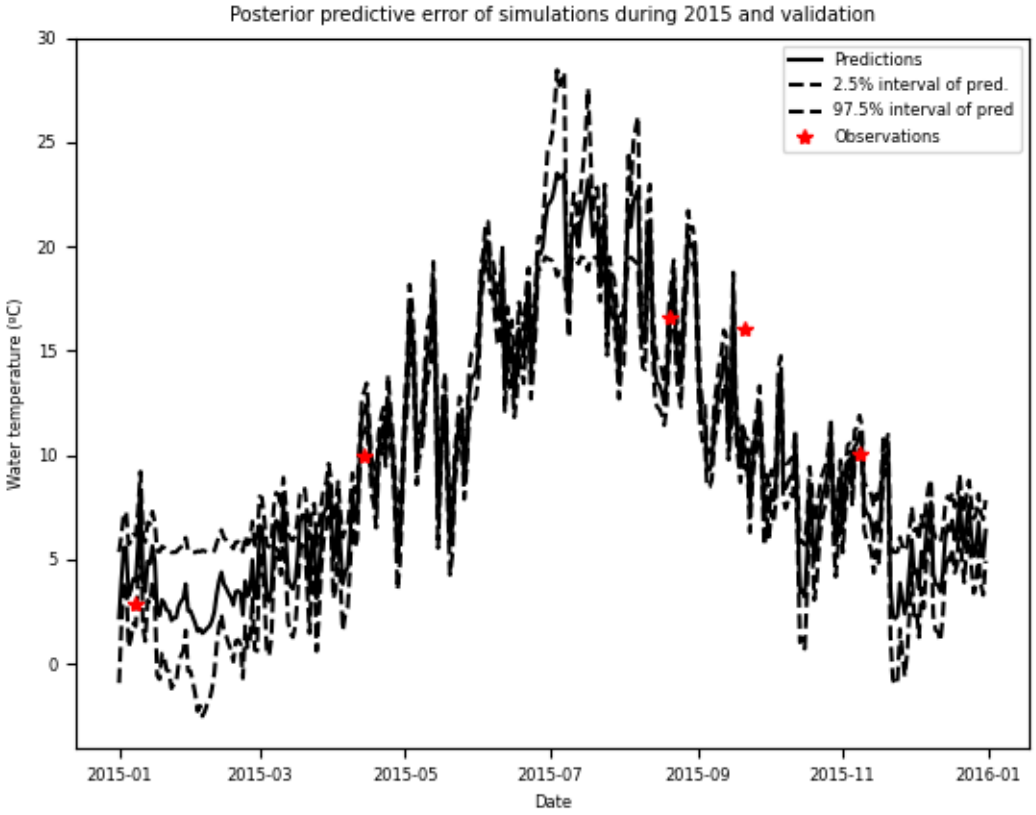
L'analyse de la variance de l'erreur proportionne aussi une estimation de l'identifiabilité des paramètres. D'après les résultats d'identifiabilité des paramètres (à nouveau, en utilisant 2 valeurs singulières), les données de calibration nous permettent de déterminer comme une valeur unique la valeur de γ (identifiabilité proche à un). Par contre, pour les autres paramètres l'information est partagée entre eux, et les données de calibration nous permettent seulement d'estimer une combinaison de ces paramètres. Ceci signifie que la solution obtenue pour les paramètres α, β et μ n'est pas unique, mais une de plusieurs possibilités avec un niveau de l'erreur similaire. Aussi, les paramètres sont corrélés entre eux.



Les paramètres avec une identifiabilité plus élevé dans le cas d'étude sont γ et β, qui déterminent la pente et position de la courbe autour de son point d'inflexion. Mais les données disponibles ne permettent de déterminer également bien la position des asymptotes de la courbe en forme de S, indiquée par α et β.

2.7.8 Validation

On peut considérer la validation comme un test pour évaluer le dégrée de cohérence des simulations avec le comportement observé. La figure suivante montre les mesures de température prises en 2015, surimposées aux simulations et à l'incertitude prédictive pour tous les jours dans la même année.

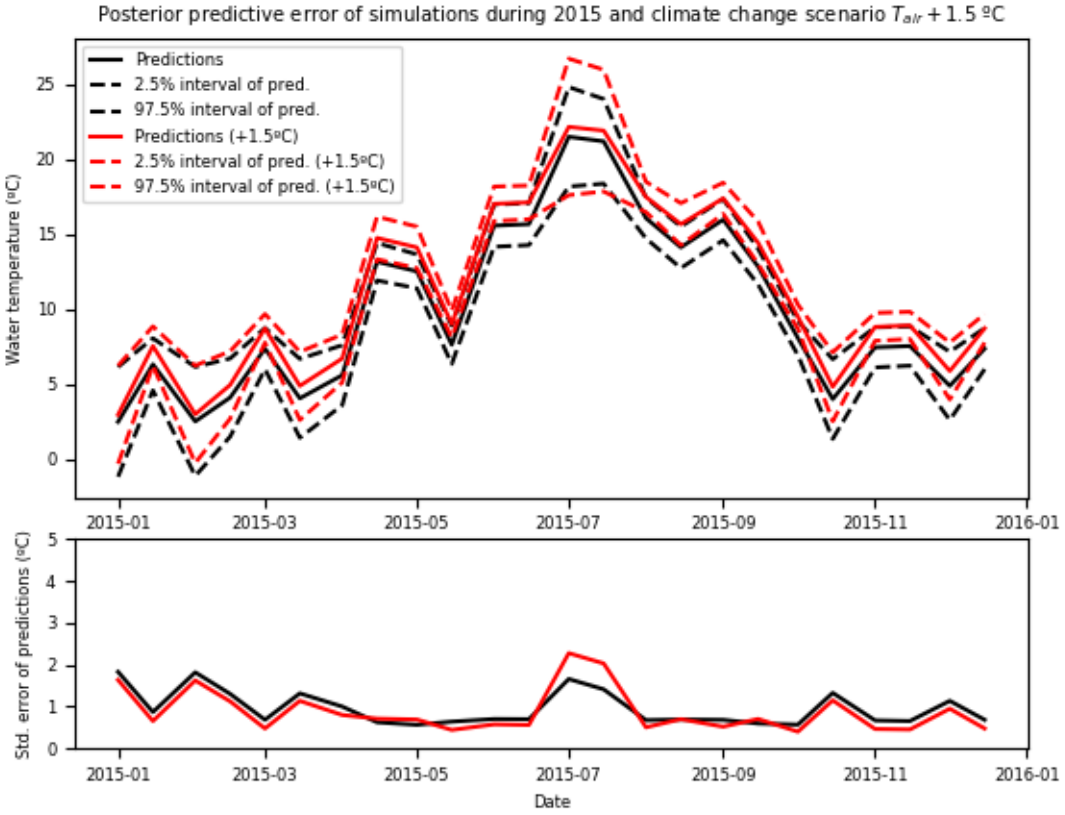


On voit que les prédictions sont assez cohérentes avec les observations : la plupart des mesures sont dans les rangs estimés d'erreur prédictive. Pourtant, le fait que la mesure pour le 21 sept. 2015 soit dehors le range d'erreur prédictive peut indiquer une imperfection du modèle.

En fait, la variabilité de la température de l'eau estimée du printemps et automne semble trop large. La température de l'eau varie plus lentement que la température de l'air, et la température de l'eau est mieux corrélée à la température de l'air des jours antérieurs. Ceci est la justification de la modification proposée par Koch & Grünewald (2010) de l'Éq. (2.1).

2.7.9 Utilisation des résultats dans un scénario de gestion : l'effet du changement climatique

Nous pouvons utiliser les résultats obtenus ici pour analyser un problème de gestion management : quel serait l'effet du changement climatique ? Pour faire ça, nous utiliserons les mêmes températures de l'air que pour 2015 mais incrémentées de 1.5ºC pour simuler la température de l'eau avec l'éq. (2.1).



Le modèle prédits une augmentation moyenne de la température de l'eau de 1,1ºC avec une augmentation de 1.5ºC de la température de l'air. Durant la plupart de l'année l'incertitude prédictive du scénario T_{air} + 1.5alpha` et le fait que le réchauffement fait que les températures d'été sortent du rang de valeurs pour lesquels il y a de l'information. Cette situation remarque un problème critique : la difficulté de prédire la température de l'eau quand il y a les plus grand risque d'avoir des hautes températures.

2.7.10 Une note sur l'utilisation de méthodes d'analyse de l'incertitude linéaires et non-linéaires

Quelques uns des analyses précédents sont basés sur l'utilisation de méthodes linéaires d'analyse de l'incertitude. L'application de telles méthodes à de modèles non-linéaires donne seulement des résultats approximés. Cependant, si bien ces méthodes ne permettent pas une estimation exacte de l'incertitude, elles proportionnent de l'information semi-quantitative suffisamment robuste pour analyser les contributions relatives de différentes sources d'incertitude des paramètres (Moore & Doherty, 2005).

Les méthodes non-linéaires (telles que les méthodes de Monte Carlo) pourrait être utilisées aussi pour avoir des résultats quantitatifs plus appropriés. Pourtant, elles requièrent un majeur nombre de simulations, que peut être prohibitif si le temps d'exécution du modèle est élevé. Ces méthodes permettent d'analyser l'effet de l'incertitude paramétrique comme on a vu dessus (voir *Incertitude a priori*). Avec ces méthodes, l'effet du bruit (erreur de mesure ou erreur structurelle du modèle) peuvent être pris en considération aussi par le biais d'une caractérisation appropriée, par exemple, comme un bruit blanc (si le bruit n'est pas corrélé) ou comme un procès stochastique. Mais pour déterminer les caractéristiques du bruit il peut être nécessaire d'utiliser un jeu de données très riche.

Ainsi, pour choisir entre une méthode linéaire ou non-linéaire il faut prendre en considération le temps de calcul, le coût computationnel de la méthode et les données disponibles.

2.8 Références

— Bennett, N.D. ; Croke, B.F.W. ; Guariso, G. ; Guillaume, J.H.A. ; Hamilton, S.H. ; Jakeman, A.J. ; Marsili-Libelli, S. ; Newham, L.T.H. ; Norton, J.P. ; Perrin, C. ; Pierce, S.A. ; Robson, B. ; Seppelt, R. ; Voinov, A.A. ; Fath, B.D. ; Andreassian, V. (2013) Characterising performance of environmental models. *Environmental Modelling & Practice* 40, 1-20.

— Beck M.B. (1987) Water Quality Modeling : A Review of the Analysis of Uncertainty. *Water Resources Research* 23(8), 1393-1442.

— Beven, K. ; Binley, A. (1992) The future of distributed models : model calibration and uncertainty prediction. *Hydrological Processes* 6, 279-298.

— Beven, K. ; Freer, J. (2001) Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology. *Journal of Hydrology* 249, 11-29.

— Beven, K. ; Young, P. (2013) A guide to good practice in modeling semantics for authors and referees. *Water Resources Research* 49, 5092-5098.

— Boschetti, F. ; Grigg, N.J. ; Enting, I. (2011) Modelling = conditional prediction. *Ecological Complexity* 8, 86-91.

— Doherty, J. ; Hunt, R.J. (2009) Two statistics for evaluating parameter identifiability and error reduction. *Journal of Hydrology* 366, 119-127.

— Gavish, M. ; Donoho, D.L. (2014) The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory* 60(8), 5040-5053.

— Guillaume, J.H.A. ; Jakeman, J.D. ; Marsili-Libelli, S. ; Asher, M. ; Brunner, P. ; Croke, B. ; Hill, M.C. ; Jakeman, A.J. ; Keesman, K.J. ; Razavi, S. ; Stigter, J.D. (2019) Introductory overview of identifiability analysis : A guide to evaluating whether you have the right type of data for your modeling purpose. *Environmental Modelling and Software* 119, 418-432.

— Hall, J.W. ; Boyce, S.A. ; Wang, Y. ; Dawson, R.J. ; Tarantola, S. ; Saltelli, A. (2009) Sensitivity analysis for hydraulic models. *Journal of Hydraulic Engineering* 135(11), 959-969.

— Jakeman, A.J. ; Letcher, R.A. ; Norton, J.P. (2006) Ten iterative steps in development and evaluation of environmental models. *Environmental Modelling and Software* 21, 602-614.

— Koch, H. ; Grünewald, U. (2010) Regression models for daily stream temperature simulation : case studies for the river Elbe, Germany. *Hydrological Processes* 24, 3826-3836.

— Moore, C. ; Doherty, J. (2005) Role of the calibration process in reducing model predictive error. *Water Resources Research* 41, W05020, 14 p.

— Oreskes, N. ; Shrader-Frechette, K. ; Belitz, K. (1994) Verification, validation, and confirmation of numerical models in the Earth sciences. *Science* 263, 641-646.

— Prats, J. ; Reynaud, N. ; Rebière, D. ; Peroux, T. ; Tormos, T. ; Danis, P.-A. (2018) LakeSST : Lake Skin Surface Temperature in French inland water bodies for 1999-2016 from Landsat archives. *Earth System Science Data* 10, 727-743.

— Quintana-Seguí, P. ; Le Migno, P. ; Durand, Y. ; Martin, E. ; Habets, F. ; Baillon, M. ; Canellas, C. ; Franchisteguy, L. ; Morel, S. (2008) Analysis of near-surface atmospheric variables : validation of the SAFRAN analysis over France. *Journal of Applied Meteorology* 47, 92-107.

— Saltelli, A. ; Tarantola, S. ; Campolongo, F. ; Ratto, M. (2004) *Sensitivity Analysis in Practice. A Guide to Assessing Scientific Models.* John Wiley & Sons, Ltd, Chichester. 219 p.

— Saltelli, A. ; Ratto, M. ; Andres, T. ; Campolongo, F. ; Cariboni, J. ; Gatelli, D. ; Saisana, M. ; Tarantola, S. (2008) *Global Sensitivity Analysis. The Primer.* John Wiley & Sons, Ltd, Chichester. 292 p.

— Saltelli, A. ; Aleksankina, K. ; Becker, W. ; Fennell, P. ; Ferretti, F. ; Holst, N. ; Li, S. ; Wu, Q. (2019) Why so many published sensitivity analyses are false : A systematic review of sensitivity analysis practices. *Environmental Modelling and Software* 114, 29-39.

CHAPITRE 3

Installation

3.1 Requirements and dependencies

Il est nécessaire d'avoir une version Python 3.6 ou ultérieure pour faire tourner le paquet `cuspy`. Il est possible d'avoir plusieurs versions de Python (2.x et 3.x) installées sur le même système d'exploitation en utilisant, par exemple, les environnements virtuels.

Vous pouvez utiliser `pip` pour installer les paquets requis.

Note : Sous Windows, pour utiliser `pip` pour installer les paquets requis, veuillez vérifier que le chemin au fichier exécutable de `pip` a été ajouté correctement à votre variable d'environnement `PATH`. L'adresse peut varier en fonction de si vous l'installez pour un ou plusieurs utilisateurs (p. ex., `%USERPROFILE%/AppData/roaming/Python/Python37/scripts` ou `C:/Users/MyUserName/AppData/Local/Programs/Python/Python37/Scripts`).

Le logiciel `cuspy` dépend des paquets Python `numpy`, `pandas` et `pyemu`. Le paquet `pyemu` dépend aussi de `matplotlib` pour tracer des graphiques. Assurez-vous qu'ils sont installés avant l'utilisation de `cuspy`.

De plus, `cuspy` utilise les exécutables de PEST++. Les exécutables de PEST++ pour Windows are sont disponibles sur https://github.com/usgs/pestpp/bin/win, mais pour Linux vous devrez les compiler (voir les instructions dessous, *Compilation des exécutables de PEST++*).

Si vous ne devez pas compiler la documentation, `sphinx` n'est pas nécessaire pour utiliser `cuspy`, puisque une copie déjà compilée de la documentation en pdf est incluse dans le dossier `docs`.

Si vous avez besoin de compiler la documentation depuis le code source (p. ex., parce que vous avez fait des modifications), vous aurez besoin du paquet `sphinx`. Pour installer `sphinx` simplement faites

```
pip install sphinx
```

Le paquet `sphinx` travaille par défaut avec documents de type reStructuredText. Mais il peut aussi reconnaître le formatage Markdown si on installe le parseur Markdown `recommonmark`.

```
pip install --upgrade recommonmark
```

Pour créer documentation multilingue, vous avez besoin du paquet `sphinx-intl`. Le procès d'installation est comme ci-dessus :

```
pip install sphinx-intl
```

Pour compiler documents pdf (seulement sous Linux), il vous faudra installer aussi latex et le paquet Python `latexmk`. Pour installer latex, tapez :

```
sudo apt-get install texlive-full
```

Et pour installer `latexmk`, tapez :

```
pip install latexmk.py
```

Sur Ubuntu vous pouvez utiliser :

```
sudo apt install latexmk
```

3.2 Clonage du répertoire

Vous devez cloner les paquets `okplm` et `pestpp` avec git. Pour ça, placez-vous dans un dossier approprié (p. ex., `pathtoprojectsfolder`) où copier dans un sous-dossier le code du projet et clonez le projet : Par exemple, pour `cuspy` :

```
cd pathtoprojectsfolder
git clone https://github.com/inrae/ALAMODE-cuspy
```

Cette commande crée le répertoire `okplm` dans le dossier `pathtoprojectsfolder`.

Pour installer la branche de développement du projet, changez à la branche `dev` après avoir cloné le paquet `cuspy` package :

```
cd cuspy
git checkout dev
```

Pour cloner le code source de `pestpp`, faites :

```
cd pathtoprojectsfolder
git clone https://github.com/usgs/pestpp
```

3.3 Compilation des exécutables de PEST++

Si vous travaillez sous Linux, vous devrez compiler les exécutables de PEST++. Pour ça vous avez besoin d'avoir `gcc`, `gfortran` et les librairies `lapack` et `blas` installés sur votre ordinateur. Pour les installer vous pouvez utiliser :

```
sudo apt-get install gcc gfortran
sudo apt-get install liblapack-dev libblas-dev
```

Pour compiler les exécutables de PEST++, allez au dossier `pestpp/src` et faites :

```
make clean
STATIC=no make install
```

Les exécutables compilés se trouvent alors dans le dossier `pestpp/bin/linux`.

3.4 Installation de cuspy

Pour installer `cuspy`, allez au répertoire créé pendant le clonage de `cuspy` (p. ex., `pathtorepertorycuspy`) et installe-le avec `pip`.

```
cd pathtorepertorycuspy
pip install -U .
```

3.5 Compilation de la documentation du projet

Les fichiers source du manuel d'utilisation du projet sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/source`. Sphinx extrait aussi des données des docstrings des modules du projet.

3.5.1 Documentation en anglais

Pour compiler le manuel d'utilisation en anglais en html allez au dossier `pathtorepertorycuspy/sphinx-doc` et tapez :

```
make html
```

Les fichiers html de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build/html`.

Vous pouvez compiler aussi le manuel d'utilisation en pdf avec :

```
make latexpdf
```

Les fichiers source de la documentation sont convertis à latex et après à pdf. Les fichiers latex et pdf de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build/latex`.

3.5.2 Documentation en français

Pour compiler le manuel d'utilisation en français en html allez au dossier `pathtorepertorycuspy/sphinx-doc` et tapez :

```
sphinx-build -b html -aE -D language='fr' -c source/locale/fr source build_fr/html
```

Les fichiers html de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build_fr/html`.

Pour compiler la documentation en pdf, tapez les commandes suivantes :

```
sphinx-build -b latex -aE -D language='fr' -c source/locale/fr source build_fr/latex
cd build_fr/latex
make
```

Les fichiers source de la documentation sont convertis à latex et après à pdf. Les fichiers latex et pdf de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build_fr/latex`.

CHAPITRE 4

Guide de style de codage

4.1 Code Python

Nous avons suivi la Guide de style pour code Python PEP 08.

4.2 Docstrings

Nous avons suivi les Conventions Docstrings – PEP 257 et les recommandations de la *Google Python Style Guide* pour docstrings dans des fonctions (point 3.8.1).

D'accord avec la Licence Publique Générale GNU, une notice de copyright est incluse dans chaque module.

4.3 Documentation

Quelques documents d'aide ont été écrits en format markdown (README.md, DISCLAIMER.md).

Le manuel d'utilisation a été généré avec `sphinx` depuis fichiers en format markup reStructuredText . La documentation traduite en français a été créé avec `sphinx-intl`.

CHAPITRE 5

Traduction de la documentation du projet

Le package `cuspy` a été écrit originalement en anglais et traduit au français. Cette section décrit le procès pour traduire le *Guide d'Utilisation et Développement*, qui utilise des fichiers pot et po et le package `sphinx-intl`. L'utilisation de cette méthode permet de traduire seulement les sections modifiées quand il y a des modifications de la modification originelle. Cette méthode facilite aussi la traduction des docstrings et de la documentation des modules. Les instructions de traduction sont basées sur https://sphinx-doc.org/en/master/usage/advanced/intl.html.

5.1 Pas préliminaires

Avant de commencer la traduction il est conseillé de relire attentivement les fichiers source pour éliminer des erreurs linguistiques ou de format.

Un glossaire de termes techniques a été créé pour améliorer la consistance de la traduction. Il est situé dans le dossier `sphinx-doc/glossaries`.

5.2 Création de fichiers POT et PO

La traduction des fichiers de documentation est basé sur l'extraction du texte traduisible en fichiers pot (portable object template) et la création de texte traduit en fichiers po (portable object). Pour ça, le package `sphinx-intl` est utilisé.

Pour extraire les messages traduisibles du document vers des fichiers pot, changez au dossier `sphinx-doc` dans le terminal et faites :

```
make gettext
```

Les fichiers pot créés sont stockés dans le dossier `build/gettext`. Ils contiennent le texte traduisible coupé en segments.

Alors vous devez générer les fichiers po pour chaque langue cible. Par exemple, pour la langue française vous devez faire

```
sphinx-intl update -p build/gettetxt/ -l fr
```

Les fichiers po créés sont stockés dans le dossier `source/locale/fr/LC_MESSAGES`.

Les fichiers po contiennent paires de segments de texte dans les langues source (msgid) et cible (msgstr).

Avant de démarrer la traduction, la valeur de msgstr est vide :

```
#: ../../source/style.rst:24
msgid "The user manual has been generated using ``sphinx`` from files using␣
↪`reStructuredText <http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.
↪html>`_ markup language."
msgstr ""
```

5.3 Traduction

Ainsi pour traduire le texte vous devez éditer les fichiers po et taper le texte traduit à côté msgstr :

```
#: ../../source/style.rst:24
msgid "The user manual has been generated using ``sphinx`` from files using␣
↪`reStructuredText <http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.
↪html>`_ markup language."
msgstr "Le guide utilisateur a été créé avec ``sphinx`` depuis fichiers utilisant le␣
↪language markup `reStructuredText <http://www.sphinx-doc.org/en/master/usage/
↪restructuredtext/index.html>`_."
```

Attention : Faites attention à maintenir le formatage reST dans la version traduite.

Vous pouvez faire la traduction manuellement modifiant les fichiers po. Par contre, il est plus efficient d'utiliser un éditeur PO (p. ex., GNOME Traduction Editor ou PO edit) ou des logiciels de traduction assistée par ordinateur (TAO) comme OmegaT.

5.4 Compilation de la documentation traduite

La documentation en français est sauvegardée dans le dossier `build_fr`. Pour compiler le manuel d'utilisation en français en html allez au dossier `pathtorepertorycuspy/sphinx-doc` et tapez :

```
sphinx-build -b html -aE -D language='fr' -c source/locale/fr source build_fr/html
```

Les fichiers html de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build_fr/html`.

Pour compiler la documentation en pdf, tapez les commandes suivantes :

```
sphinx-build -b latex -aE -D language='fr' -c source/locale/fr source build_fr/latex
cd build_fr/latex
make
```

Les fichiers source de la documentation sont convertis à latex et après à pdf. Les fichiers latex et pdf de sortie sont stockés dans le dossier `pathtorepertorycuspy/sphinx-doc/build_fr/latex`.

5.5 Mise à jour de la documentation traduite

Si la documentation du projet est modifiée, il faut créer des nouveaux fichiers pot d'accord avec la procédure décrite ci-dessus. Pour appliquer les changements aux fichiers po, faites

```
sphinx-intl update -p build/gettext -l fr
```

Après, vous devrez traduire seulement les segments modifiés.

CHAPITRE 6

Modules

6.1 Module `input_output`

Fonctions pour lire et écrire des données.

Les fonctions de ce module servent à lire et écrire des fichiers d'entrée et sortie.

Ce module contient les fonctions suivantes :

- *`get_obs_data()`* : Obtient des données observées.
- *`process_sweep_out()`* : Traite fichier `sweep_out.csv`.
- *`write_dict()`* : Écrit un dictionnaire à un fichier.
- *`write_dummy_pred_file()`* : Écrit un fichier de prédictions avec des valeurs fictives.
- *`write_ins_file()`* : Écrit le fichier PEST d'instructions.
- *`write_pest_files()`* : Écrit les fichiers de PEST.
- `write_tpl_file()` : Écrit le fichier de paramètres modèle de PEST.
- *`write_tpl_file()`* : Écrit le fichier de paramètres modèle de PEST.

`input_output.`**`get_obs_data`**(*obs_file*, *start_date*, *end_date*, *weights=1*, *groups='obs'*, *obsnames='default'*, *delimiter=' '*)

Obtient données observées

Cette fonction sert à lire des données (observations ou prédictions) depuis un fichier. Il est utilisé aussi pour grouper des données et pour assigner des noms et poids aux données. Dans la terminologie utilisée par PEST les « observations » correspondent à mesures déjà réalisées (d'habitude avec un poids > 0), et les « prédictions » (ou « prévisions ») correspondent aux données que nous voulons prédire (avec poids = 0).

Paramètres

- **`obs_file`** – chemin du fichier d'observations (il devrait avoir le même format que le fichier de sortie du modèle). Les valeurs manquantes peuvent être indiquées avec NA, na ou NaN. La première colonne contient les dates avec le format « AAAA-mm-jj ».
- **`start_date`** – date de début de la simulation avec le format "AAAA-mm-jj".
- **`end_date`** – date de finalisation de la simulation avec le format "AAAA-mm-jj".
- **`weights`** – str ou float. Si c'est str, l'argument indique le chemin à un fichier avec les mêmes format et nom des colonnes que obs_file et qui contient les poids pour chaque mesure. Si c'est float, il indique un poids à appliquer à toutes les mesures.

— **groups** – peut être un chemin, une chaîne de caractères ou "default". si c'est un chemin, les noms des groupes sont lus depuis un fichier avec les mêmes format et noms de colonnes que obs_file. Si c'est "default", les observations sont groupées par nom de variable. Si c'est une autre str, toutes les observations sont assignées au même groupe.
— **obsnames** – peut être un chemin, une chaîne de caractères ou "default". Si c'est un chemin, les noms des observations sont lus depuis un fichier avec les mêmes format et noms des colonnes que obs_file. Si c'est "default", les noms des observations ont le format xxxx_AAAAmmjj (xxxx est le nom de la variable correspondante, et AAAAmmjj est une chaîne de caractères qui représente la date). Si c'est une autre str, les noms des observations ont le format obsnamesNN, où NN est l'index de l'observation dans le dataframe créé.
— **delimiter** – délimiteur des colonnes utilisé dans les fichiers lus par la fonction.

Renvoie Un dataframe pandas qui contient les indexes des files et colonnes des observations dans le fichier de sortie, les noms et groupes des observations, et leur poids.

input_output.**process_sweep_out**(*fname_in*, *var_names*, *folder_out*, *ptl_avg=None*)

Traite le fichier `sweep_out.csv`.

Paramètres
— **fname_in** – Chemin du fichier `sweep_out.csv`, créé quand on réalise des simulations de Monte Carlo simulations.
— **var_names** – Liste des noms des variables.
— **folder_out** – Dossier où écrire les données traitées.
— **ptl_avg** – Dictionnaire de noms des variables pour lesquelles des moyennes partielles seront calculées. Les clés indiquent les variables utilisées pour grouper les données par percentiles ; les valeurs sont listes des noms des variables pour lesquelles des moyennes partielles seront calculées pour chaque groupe de la variable clé.

Renvoie Une série de fichiers texte qui contiennent la simulation de base, le minimum, le maximum et les percentiles 5%, 25%, 50%, 75% et 95% des simulations dans le fichier `sweep_out.csv`. Les noms des fichiers de sortie suivent le modèle « mc_uncert_<v>.txt », où *v* est le nom de la variable de sortie telle quelle est utilisée dans `sweep_out.csv`. De plus, si *ptl_avg* est spécifiée, une autre série de fichiers est créée, contenant les moyennes partielles d'une variable (*vpa*) par groupe de percentile d'une autre variable (*v*). Dans ce cas, les fichiers de sortie suivent le modèle « mc_pavg_<vpa>_by_<v>.txt ».

input_output.**write_dict**(*x_dict*, *path*)

Écrit dictionnaire à un fichier.

Cette fonction écrit un dictionnaire Python à un fichier. Le fichier est organisé en deux colonnes, la première contient les clés et la seconde contient les valeurs.

Paramètres
— **x_dict** – un dictionnaire Python.
— **path** – chemin du fichier texte où écrire les données.

Renvoie Un fichier situé à « path » où le contenu du dictionnaire data est écrit.

input_output.**write_dummy_pred_file**(*fname*, *time_stamps*, *colnames*, *time_colname='date'*, *dummy_value=0*, *sep=' '*)

Écrit un fichier de prédictions avec des valeurs fictives.

Paramètres
— **fname** – Nom du fichier.
— **time_stamps** – Liste ou array de données de temps correspondantes aux données dans le fichier de prédictions.
— **colnames** – Noms des colonnes du fichier de prédictions.
— **time_colname** – Nom de la colonne qui contient les données de temps (données par le biais de l'argument *time_stamps*).
— **dummy_value** – Valeur utilisée pour remplir le fichier de prédictions (à exception des données de temps).

— **sep** – Séparateur de colonnes du fichier de sortie.

Renvoie Un fichier avec le même format que le fichier de sortie, rempli avec des données fictives.

Note : Un fichier de prédictions rempli avec des données fictives peut être utile quand on veut obtenir des résultats de simulation pour une période de temps pour lequel il n'y a pas d'observations disponibles.

---

input_output.**write_ins_file**(*fname*, *obs_inds*, *pred_inds*, *ncols*, *nl_header=1*, *field_wd=10*, *delimiter=' '*)

Écrit le fichier PEST d'instructions.

Cette fonction sert à écrire un fichier d'instructions de PEST(++), un fichier texte utilisé pour spécifier comment PEST doit lire le fichier de sortie d'un modèle. Des instructions détaillées sur comment formater un fichier d'instructions peuvent être trouvées dans White et al. (2019).

Paramètres

- **fname** – un nom de fichier. Il est conseillé d'utiliser l'extension .ins.
- **obs_inds** – liste de tuples (x, y, obsname) qui indique l'index des files (x), l'index des colonnes (y) et le nom (obsname) qui correspond aux observations dans le fichier de sortie du modèle.
- **pred_inds** – liste de tuples (x, y, obsname) qui indique l'index des files (x), l'index des colonnes (y) et le nom (obsname) qui correspond aux prédictions dans le fichier de sortie du modèle.
- **ncols** – nombre de colonnes du fichier de sortie.
- **nl_header** – nombre de lignes de l'entête du fichier.
- **field_wd** – largeur du champ en nombre de caractères. Il doit être au moins si long que le nom de l'observation/prédiction.
- **delimiter** – caractère délimiteur utilisé dans le fichier de sortie.

Renvoie Un fichier d'instruction PEST(++) (un fichier texte qui spécifie comment traiter la sortie d'un modèle pour lire les résultats de simulation qui correspondent aux observations et prédictions d'intérêt).

Références

— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

input_output.**write_par_data_file**(*parnme*, *parval1*, *parlbnd*, *parubnd*, *partrans='none'*, *parchglim='relative'*, *pargp=None*, *scale=1.0*, *offset=0.0*, *dercom=1*, *file_path='par_data.csv'*)

Écrit le fichier de données des paramètres.

Paramètres

- **parnme** – Liste de noms des paramètres.
- **parval1** – Liste des valeurs initiales des paramètres.
- **parlbnd** – Liste des seuils inférieurs des paramètres.
- **parubnd** – Liste des seuils supérieurs des paramètres.
- **partrans** – Transformation appliquée à tous les paramètres ou liste de transformations appliquées pour chaque paramètre. Les valeurs possibles sont « none », « log », « fixed » or « tied ».
- **parchglim** – Type de limite de changement du paramètre (valeur à appliquer à tous les paramètres ou liste de valeurs). Les valeurs possibles sont : « relative » et « factor ».
- **pargp** – Liste de noms des groupes de paramètres. Si c'est None, les noms des groupes de paramètres sont les mêmes que les noms des paramètres (un groupe de paramètres pour chaque paramètre).
- **scale** – Multiplicateur ou liste de multiplicateurs des valeurs des paramètres.
- **offset** – Écart ou liste d'écarts à appliquer aux valeurs des paramètres.
- **dercom** – Numéro (ou liste de numéros) de la ligne de commande utilisée pour calculer les dérivées des paramètres. C'est d'habitude égal à 1.

— **file_path** – Chemin du fichier de données des paramètres où écrire les données.

input_output.**write_pest_files**(*start_date*, *end_date*, *model_command*, *par_file='par.txt'*, *output_file='output.txt'*, *par_data_file='par_data.txt'*, *obs_file=None*, *pred_file=None*, *obs_weights=1*, *obs_groups='default'*, *obs_names='default'*, *pred_groups='default'*, *pred_names='default'*, *tpl_file='par.tpl'*, *ins_file='res_file.ins'*, *pst_file='pest.pst'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *delimiter=' '*)

Écrit les fichiers de PEST.

La fonction *write_pest_files()* sert à écrire les fichiers nécessaires pour tourner des programmes de PEST(++). Ces fichiers incluent le fichier de contrôle, le fichier d'instructions et le fichier de paramètres modèle. Pour plus d'information sur les fichiers utilisés par PEST++, veuillez voire White et al. (2019).

Paramètres

— **start_date** – date de début des simulations avec le format str "AAAA-mm-jj" ou similaire à un format datetime.
— **end_date** – date de finalisation des simulations avec le format str "AAAA-mm-jj" ou similaire à un format datetime.
— **model_command** – instruction utilisée pour faire tourner le modèle.
— **par_file** – chemin du fichier de paramètres du modèle. Il est écrit par la fonction en utilisant les valeurs des paramètres dans *par_data_file*.
— **output_file** – chemin du fichier de sortie du modèle.
— **par_data_file** – chemin d'un fichier qui contient des données sur les paramètres nécessaires à configurer la section dédiée aux paramètres dans le fichier de contrôle de PEST.
— **obs_file** – chemin du fichier d'observations. Il a le même format que *output_file*.
— **pred_file** – chemin du fichier de prédictions. Il a le même format que *output_file*.
— **obs_weights** – poids assignés aux observations. Il peut être str ou float. Si c'est str, l'argument indique le chemin à un fichier avec les mêmes format et nom des colonnes que obs_file et qui contient les poids pour chaque mesure. Si c'est float, il indique un poids à appliquer à toutes les mesures.
— **obs_groups** – groupes auxquels chaque observation est assignée. Peut être un chemin, une chaîne de caractères ou "default". Si c'est un chemin, les noms des groupes sont lus depuis un fichier avec les mêmes format et noms de colonnes que *obs_file*. Si c'est "default", les observations sont groupées par nom de variable. Si c'est une autre str, toutes les observations sont assignées au même groupe.
— **obs_names** – noms des observations. Peut être un chemin, une chaîne de caractères ou "default". Si c'est un chemin, les noms des observations sont lus depuis un fichier avec les mêmes format et noms des colonnes que *obs_file*. Si c'est "default", les noms des observations ont le format « xxxx_AAAAmmjj » (xxxx est le nom de la variable correspondante, et AAAAmmjj est une chaîne de caractères qui représente la date). Si c'est une autre str, les noms des observations ont le format « obsnamesNN », où NN est l'index de l'observation dans le dataframe créé.
— **pred_groups** – groupes auxquels chaque prédiction est assignée. Peut être un chemin, une chaîne de caractères ou "default". Si c'est un chemin, les noms des groupes sont lus depuis un fichier avec les mêmes format et noms de colonnes que *obs_file*. Si c'est "default", les prédictions sont groupées par nom de variable. Si c'est une autre str, toutes les prédictions sont assignées au même groupe.
— **pred_names** – noms des prédictions. Peut être un chemin, une chaîne de caractères ou "default". Si c'est un chemin, les noms des prédictions sont lus depuis un fichier avec les mêmes format et noms des colonnes que *pred_file*. Si c'est "default", les noms des prédictions ont le format « xxxx_AAAAmmjj » (xxxx est le nom de la variable correspondante, et AAAAmmjj est une chaîne de caractères qui représente la date). Si c'est une autre str, les noms des prédictions ont le format « pred_namesNN », où NN est l'index de la prédiction dans le dataframe créé.
— **tpl_file** – chemin du fichier de paramètres modèle. Il est écrit par la fonction en utilisant

les noms des paramètres dans *par_data_file*.
- **ins_file** – chemin du fichier d'instructions de pest. Il est écrit par la fonction en utilisant les données dans *obs_file* et *pred_file*.
- **pst_file** – chemin du fichier de contrôle PEST. Il est écrit par la fonction en utilisant les données que lui ont été passées.
- **control_data** – un dictionnaire où sont définies les de la section de contrôle du fichier de contrôle PEST.
- **svd_data** – un dictionnaire utilisé pour configurer des options de la section SVD du fichier de contrôle PEST.
- **reg_data** – un dictionnaire utilisé pour configurer des options de la section de régularisation du fichier de contrôle PEST.
- **pestpp_opts** – un dictionnaire utilisé pour configurer des options de PEST++ du fichier de contrôle PEST.
- **delimiter** – caractère délimiteur utilisé dans *output_file*, *obs_file* et *pred_file*.

Renvoie Une série de fichiers PEST (fichier de contrôle, fichier d'instructions et fichier de paramètres modèle) et un fichier de paramètres.

Références

— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

input_output.**write_tpl_file**(*fname*, *par_names*)

Écrit le fichier de paramètres modèle de PEST.

Cette fonction écrit un fichier de paramètres modèle de PEST, un fichier texte utilisé pour spécifier le format du fichier de paramètres. Le fichier de paramètres est un fichier texte où sont écrites les valeurs des paramètres utilisés par le modèle.

Paramètres
- **fname** – un nom de fichier. Il est conseillé d'utiliser l'extension .tpl.
- **par_names** – liste de noms des paramètres dans l'ordre dans lequel ils se présentent dans le fichier de paramètres.

Renvoie Un fichier de paramètres modèle de PEST(++) (un fichier texte qui spécifie le format du fichier de paramètres).

Références

— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

6.2 Module functions

Fonctions d'utilité.

La fonction dans cette module sert à lancer l'exécutable de PEST++.

Ce module contient les fonctions suivantes :
- *launch_pestpp()* : lance l'exécutable PEST++

functions.**launch_pestpp**(*pst_file*, *pestpp_folder*, *pestpp_cmd='pestpp-glm'*, *parallel=False*)

Lance l'exécutable PEST++.

Cette fonction lance l'exécutable PEST++ sollicité, soit faisant les calcules en séquentiellement ou en parallèle. Les exécutables de PEST++ et leurs utilisations sont :
- PESTPP-GLM : inversion avec un grand nombre de paramètres et optimisation globale (c'est-à-dire, optimisation de paramètres, calibration).
- PESTPP-SEN : analyse de sensitivité globale.
- PESTPP-OPT : optimisation contrainte sous incertitude (White et al. 2018).

— PESTPP-IES : lissage itératif d'ensemble (White 2018).
— PESTPP-SWP : simulations de Monte Carlo.

Pour plus d'information sur les exécutables de PEST++, veuillez voire White et al. (2019).

Paramètres
— **pst_file** – chemin du fichier de contrôle pest (.pst).
— **pestpp_folder** – dossier qui contient les exécutables de PEST++.
— **pestpp_cmd** – Exécutable de PEST++. Les exécutables de PEST++ sont : « pestpp-glm », « pestpp-sen », « pestpp-opt », « pestpp-ies » et « pestpp-swp ».
— **parallel** – parallélise les calculs.

Renvoie Le retour de l'instruction de PEST++ es montrée sur l'écran. De plus, les fichiers de sortie de *pestpp_cmd* sont écrit dans le dossier qui contient le *pst_file*.

Références

— White, J.T. (2018) A model-independent iterative ensemble smoother for efficient history matchingh and uncertainty quantification in very high dimensions. *Environmental Modelling and Software* 109, 191-201.
— White, J.T. ; Fienen, M.N. ; Barlow, P.M. ; Welter, D.E. (2018) A tool for efficient, model-independent management optimization under uncertainty. *Environmental Modelling and Software* 100, 2013-221.
— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

6.3 Module analyses

Fonctions pour réaliser des analyses basées sur PEST++.

Les fonctions dans ce module réalisent les différents types d'analyse implémentées dans le paquet.

Ce module contient les fonctions suivantes :
— *calibration()* : calibre le modèle.
— *gsa()* : Analyse de Sensitivité Globale.
— *ies()* : Lissage Itératif d'Ensemble.
— *linear_uncertainty()* : incertitude prédictive.
— *monte_carlo()* : simulation de Monte Carlo.

analyses.**calibration**(*method='glm'*, *reg=False*, *pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *pestpp_folder='..'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=False*)

Calibre le modèle (méthodes GLM ou DE).

Paramètres
— **method** – méthode utilisée pour calibrer le modèle. C'estI "glm" pour l'algorithme de Gauss-Levenberg-Marquardt, "de" pour évolution différentielle.
— **reg** – utilise la régularisation de Tikhonov.
— **pst_file0** – chemin du fichier pest originel.
— **pst_file1** – chemin du fichier pest modifié.
— **pestpp_folder** – dossier qui contient les exécutables de PEST++.
— **control_data** – un dictionnaire où sont définies les options de la section de contrôle du fichier de contrôle pest.
— **svd_data** – un dictionnaire utilisé pour configurer des options de la section SVD du fichier de contrôle PEST.
— **reg_data** – un dictionnaire utilisé pour configurer des options de la section de régularisation du fichier de contrôle PEST.
— **pestpp_opts** – un dictionnaire utilisé pour configurer les options de la méthode DE.
— **parallel** – parallélise les exécutions du modèle.

Renvoie Une instance PEST où les valeurs des paramètres correspondent aux valeurs calibrées. Un fichier PEST (*pst_file1*) et les fichiers associés qui contiennent les résultats du procès de calibration sont créés aussi.

Note : Cette fonction appelle l'exécutable de PEST++ *pestpp-glm*.

Références

— Doherty, J. (2010) *Methodologies and Software for PEST-Based Model Predictive Uncertainty Analysis.* Watermark Numerical Computing. 157 p.

— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`analyses.gsa`(*method='morris'*, *pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *pestpp_folder='..'*, *parallel=True*)

Réalise une analyse de sensitivité globale (méthode de Morris ou de Sobol).

Paramètres

— **method** – "morris" pour la méthode de Morris (un paramètre à la fois) ou "sobol" pour la méthode de Sobol.

— **pst_file0** – chemin du fichier pest originel.

— **pst_file1** – chemin du fichier pest modifié.

— **control_data** – un dictionnaire où sont définies les options de la section de contrôle du fichier de contrôle pest.

— **svd_data** – un dictionnaire utilisé pour configurer des options de la section SVD du fichier de contrôle PEST.

— **reg_data** – un dictionnaire utilisé pour configurer des options de la section de régularisation du fichier de contrôle PEST.

— **pestpp_opts** – un dictionnaire utilisé pour configurer les options de la méthode GSA.

— **pestpp_folder** – dossier qui contient les exécutables de PEST++.

— **parallel** – parallélise les exécutions du modèle.

Renvoie Un fichier pst modifié et ses fichiers associés qui contiennent les résultats.

Note : Cette fonction appelle l'exécutable de PEST++ *pestpp-sen*.

Références

— Saltelli, A. ; Ratto, M. ; Andres, T. ; Campolongo, F. ; Cariboni, J. ; Gatelli, D. ; Saisana, M. ; Tarantola, S. (2008) *Global Sensitivity Analysis. The Primer.* John Wiley & Sons, Ltd. 292 p.

— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`analyses.ies`(*pst_file0*, *pst_file1*, *pestpp_folder*, *n_reals=50*, *parcov=None*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=True*)

Lissage Itératif d'Ensemble.

Paramètres

— **pst_file0** – chemin du fichier pest originel.

— **pst_file1** – chemin du fichier pest modifié.

— **pestpp_folder** – dossier qui contient les exécutables de PEST++.

— **n_reals** – nombre de réalisations.

— **parcov** – nom du fichier de covariance des paramètres a priori. Si None, la covariance des paramètres est estimée depuis le rang de valeurs des paramètres.

— **control_data** – un dictionnaire où sont définies les options de la section de contrôle du fichier de contrôle pest.

— **svd_data** – un dictionnaire utilisé pour configurer des options de la section SVD du fichier de contrôle PEST.
— **reg_data** – un dictionnaire utilisé pour configurer des options de la section de régularisation du fichier de contrôle PEST.
— **pestpp_opts** – un dictionnaire qui contient options additionnelles à passer à *pestpp-ies*.
— **parallel** – parallélise les calculs.

Renvoie Un fichier de contrôle PEST modifié et ses fichiers de sortie associés.

Note : Cette fonction appelle l'exécutable de PEST++ *pestpp-ies*.

Références

— White, J. T. (2018) A model-independent iterative ensemble smoother for efficient history-matching and uncertainty quantification in very high dimensions. *Environmental Modelling and Software* 109, 191-201.
— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

analyses.**linear_uncertainty**(*analysis*, *pst_file0*, *pst_file1*, *pestpp_folder*, *predictions=None*)
Réalise les calculs linéaires d'incertitude.

Paramètres
— **analysis** – Type d'analyse d'incertitudes prédictives. Il peut prendre les valeurs "schur", pour l'analyse du complément de Schur pour la propagation conditionnelle d'incertitudes ; "err_var", pour l'analyse de la variance de l'erreur ; ou "prior", pour l'estimation de l'incertitude a priori.
— **pst_file0** – chemin du fichier pest originel.
— **pst_file1** – chemin du fichier pest modifié.
— **pestpp_folder** – dossier qui contient les exécutables de PEST++.
— **predictions** – liste de noms des prédictions. Si None, les prédictions sont lues depuis le fichier *pst_file0* comme observations avec un poids nul.

Renvoie Un objet `LinearAnalysis` (`analysis='prior'`), un objet `Schur` (`analysis='schur'`) ou un objet `ErrVar` (`analysis='err_var'`). Un fichier de contrôle PEST modifié (*pst_file1*) et une matrice de Jacobi sont aussi créés pendant le procès.

Note : On suppose que le modèle a déjà été calibré. Ainsi, le fichier de contrôle PEST originel *pst_file0* devrait être dans le même dossier que les fichiers créés pendant le procès de calibration.

Références

— Doherty, J. (2010) *Methodologies and Software for PEST-Based Model Predictive Uncertainty Analysis.* Watermark Numerical Computing. 157 p.
— White, J.T. ; Fienen, M.N. ; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling and Software* 85, 217-228.

analyses.**monte_carlo**(*pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *dist_type='post'*, *distribution='gaussian'*, *n_samples=100*, *how_dict=None*, *csv_in='sweep_in.csv'*, *pestpp_folder='..'*, *add_base=False*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=True*, *process_swp_out=False*)
Simulations de Monte Carlo.

Paramètres
— **pst_file0** – chemin du fichier pest originel.
— **pst_file1** – chemin du fichier pest modifié.
— **dist_type** – type de distribution ; "prior" pour une distribution a priori ou "post" pour une distribution a posteriori.

— **distribution** – "uniform", "triangular", "gaussian", "mixed". Ignoré si `type = post` (la distribution normale est utilisée).
— **n_samples** – nombre d'échantillons à extraire.
— **how_dict** – dictionnaire de noms des paramètres (clés) et distributions (valeurs). Seulement utilisée si `distribution = mixed`.
— **csv_in** – nom du fichier où sauvegarder les échantillons de valeurs des paramètres.
— **pestpp_folder** – dossier qui contient les exécutables de PEST++.
— **add_base** – ajoute les valeurs des paramètres du fichier *pst_file0* comme une réalisation.
— **control_data** – un dictionnaire où sont définies les options de la section de contrôle du fichier de contrôle pest.
— **svd_data** – un dictionnaire utilisé pour configurer des options de la section SVD du fichier de contrôle PEST.
— **reg_data** – un dictionnaire utilisé pour configurer des options de la section de régularisation du fichier de contrôle PEST.
— **pestpp_opts** – un dictionnaire utilisé pour configurer les options de *pestpp-swp*.
— **parallel** – parallélise les calculs.
— **process_swp_out** – option pour traiter le fichier de résultats `sweep_out.csv`. Les noms des groupes d'observations devraient correspondre aux noms des variables auxquelles les observations appartiennent.

Renvoie Un fichier de contrôle PEST modifié et ses fichiers de sortie associés.

Note : Cette fonction appelle l'exécutable de PEST++ *pestpp-swp*.

Références

— White, J.T. ; Fienen, M.N. ; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling and Software* 85, 217-228.
— White, J. ; Welter, D. ; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

CHAPITRE 7

Utilisation

Le paquet `cuspy` peut être appliqué à une large variété de modèles, à condition que :

— le modèle puisse être exécuté avec la ligne de commandes (p. ex., par le biais d'un exécutable disponible dans le chemin ou un appel à un script).

— la communication entre le modèle et l'utilisateur (données de sortie, valeurs des paramètres) se fait par le moyen de fichiers texte (ou le script qui appelle le modèle lit des données en forme binaire et créé la sortie en forme de texte).

De plus, le paquet a été spécialement désigné pour travailler avec des modèles dynamiques avec un retour sur plusieurs colonnes, la première desquelles est la date. Il est possible de travailler avec des modèles avec un autre type de sortie, mais il faut une majeure implication de l'utilisateur dans la préparation des fichiers d'instructions PEST.

7.1 Application en ligne de commande

7.2 Module Python

Pour utiliser `cuspy` comme un module Python vous pouvez simplement l'importer et utiliser les fonctions qu'il contient :

```
import cuspy
```

Vous pouvez inclure les commandes précédentes dans un script Python (voyez les scripts d'exemple dans le dossier `tests`). Pour exécuter un script python script en ligne de commande, tapez :

```
python path_to_script
```

7.3 Le procès d'analyse

7.3.1 Préparation du modèle

Avant de commencer les analyses, vous devez préparer tous les fichiers d'entrée et les scripts dont le modèle a besoin.

En particulier, il doit être possible d'exécuter le modèle en utilisant une commande en ligne de commandes comme la suivante :

```
run_model -x arg_x -y arg_y
```

Alternativement, si le modèle a des sorties binaires ou plusieurs fichiers de sortie, vous pourrez avoir besoin de créer un script pour traiter le retour du modèle de façon qu'il soit conforme au format conseillé (fichier texte séparé par des espaces avec la date sur la première colonne). Par exemple :

```
python script.py
```

De plus, le modèle doit lire les valeurs des paramètres depuis un fichier texte organisé en deux colonnes séparées par des espaces et qui contiennent les noms des paramètres et leurs valeurs. Par exemple :

```
A 6.2
B 1.007
C -0.007
D 0.51
E 0.24
ALPHA 0.07
BETA 0.13
mat -0.4
at_factor 1.0
sw_factor 1.0
```

7.3.2 Création des fichiers PEST

Le premier étage dans l'utilisation du paquet `cuspy` est la création des fichiers PEST. Vous pouvez faire ça manuellement, ou vous pouvez utiliser la fonction `write_pest_files()`. Le suivant est un exemple de la commande pour créer les fichiers PEST pour le modèle OKP.

```
cuspy.write_pest_files(start_date=start_date, end_date=end_date,
                       par_file='par.txt', output_file='output.txt',
                       par_data_file='par_data.csv',
                       obs_file='obs.txt', pred_file='pred.txt',
                       model_command='run_okp',
                       control_data={'noptmax': 0, 'numlam': 10},
                       svd_data={'maxsing': len(par_names)},
                       pestpp_opts={'parcov': uncertainty_file},
                       tpl_file='par.tpl', ins_file='res_file.ins',
                       pst_file='test0.pst')
```

Dans l'exemple antérieur, on peut noter deux points :

- `start_date` et `end_date` correspondent aux dates initiale et finale de la simulation.
- `par_file` et `output_file` sont les noms du fichier de paramètres du modèle et des fichiers de sortie.

— `par_data_file`, `obs_file` et `pred_file` sont des fichiers texte séparés par des espaces utilisés pour passer les données des paramètres et des observations a la fonction. Ces fichiers-ci doivent être préparés par l'utilisateur.
— `model_command` est la commande de ligne de commandes utilisée pour exécuter le modèle. Dans ce cas, le modèle OKP est exécuté avec la commande `run_okp` du paquet `okplm`.
— `control_data`, `svd_data` et `pestpp_opts` sont des dictionnaires utilisés pour configurer les valeurs des variables dans les sections dédiées aux données de contrôle, à la décomposition en valeurs singulières et aux données de contrôle de PEST++ du fichier de contrôle PEST.
— `tpl_file`, `ins_file` and `pst_file` sont les noms des fichiers PEST créés par la fonction. Le fichier `tpl_file` est un fichier modèle du fichier de paramètres `par_file` utilisé par PEST++ pour le modifier. Le `ins_file` est un fichier qui indique à PEST++ quelles valeurs du fichier `output_file` correspondent aux observations et quelles aux prédictions. Le `pst_file` contrôle l'exécution des exécutables de PEST++.

On décrit à continuation quelques-uns des fichiers utilisés par `write_pest_files()`. Pour plus d'information sur les autres arguments (incluant quelques-uns non-utilisés ci-dessus), veuillez voire la description de la fonction dans la section Modules.

Le fichier de contrôle (`pst_file`)

Le fichier de contrôle PEST contient toute l'information nécessaire pour contrôler l'exécution des exécutables de PEST++. Il est écrit automatiquement par la fonction `write_pest_files()` et il contient les sections suivantes :

— **control data** (obligatoire)
— automatic user intervention (optionnelle)
— **singular value decomposition** (optionnelle)
— lsqr (optionnelle)
— sensitivity reuse (optionnelle)
— svd assist (optionnelle)
— **parameter groups** (obligatoire)
— **parameter data** (obligatoire)
— **observation groups** (obligatoire)
— **observation data** (obligatoire)
— derivatives command line (optionnelle)
— **model command line** (obligatoire)
— **model input** (obligatoire)
— **model output** (obligatoire)
— prior information (optionnelle)
— predictive analysis (optionnelle)
— **regularization** (optionnelle)
— pareto (optionnelle)
— **PEST++ variables** (optionnelle) [1]

Pour le moment les fonctions de `cuspy` peuvent être utilisées pour configurer les sections en graisse, et elles incluent toutes les sections obligatoires et quelques-unes d'optionnelles. Si c'est nécessaire, les autres sections peuvent être configurées manuellement. Pour des détails sur la structure du fichier et ses variables, veuillez voire le manuel d'utilisation de PEST++.

1. Les déclarations des variables de PEST++ (la chaîne « ++ » suivie par le nom et valeur de la variable) peuvent être localisées n'importe où dans le fichier de contrôle de PEST, mais elles sont placées à la fin du fichier par `write_pest_files()`.

Le fichier de données des paramètres (par_data_file)

Dans l'exemple ci-dessus, par_data_file est un fichier texte séparé par des espaces qui contiennent les données de la section des paramètres du fichier de contrôle de PEST. Ce fichier doit être donné par l'utilisateur, sur la base de ses connaissances antérieures des valeurs des paramètres optimales et des rangs de variation acceptables.

On montre un exemple ci-dessous.

```
parnme partrans parchglim parval1 parlbnd parubnd pargp scale offset dercom
a none relative 6.2 4.7 7.7 a 1.0 0.0 1
b none relative 1.007 0.847 1.167 b 1.0 0.0 1
c none relative -0.0069 -0.015 0.001 c 1.0 0.0 1
d none relative 0.51 0.0 1.0 d 1.0 0.0 1
e none relative 0.24 0.0 1.0 e 1.0 0.0 1
alpha none relative 0.07 0.0 0.23 alpha 1.0 0.0 1
beta none relative 0.13 0.0 1.0 beta 1.0 0.0 1
mat none relative 2.4 1.4 3.4 mat 1.0 0.0 1
at_factor fixed relative 1.0 0.9 1.1 at_factor 1.0 0.0 1
sw_factor fixed relative 1.0 0.9 1.1 sw_factor 1.0 0.0 1
```

La première ligne du fichier contient les noms des colonnes, décrits ci-dessous :

- parnme : nom du paramètre (jusqu'à 200 caractères, sans distinction de majuscules et minuscules).
- partrans : transformation du paramètre. Elle prend une de quatre valeurs possibles :
  - 'none' : aucune transformation n'est appliquée au paramètre.
  - 'log' : on applique une transformation logarithmique au paramètre.
  - 'fixed' : paramètre constant ou non-ajustable.
  - 'tied' : le paramètre est lié à un autre paramètre (option non implémentée dans cuspy).
- parchglim : limite de changement du paramètre. Elle est utilisée par les exécutables de PEST++ *pestpp-glm* et (en option) *pestpp-ies* pour limiter le changement qu'un paramètre peut subir à chaque itération du procès d'optimisation. Elle peut prendre les valeurs :
  - 'relative' : une limite relative est utilisée, soit, la valeur du paramètre b doit avérer la condition $|b - b_0|/|b_0| \leq r$, où b_0 est la valeur du paramètre au début de l'itération et r est la valeur de la variable de contrôle de relparmax (=10 par défaut).
  - 'factor' : un facteur est utilisé comme limite, soit, la valeur du paramètre b doit être dans les limites $|b_0/f| \leq b \leq |fb_0|$, où b_0 est la valeur du paramètre au début de l'itération et f est la valeur de la variable de contrôle de PEST facparmax (=10 par défaut).
- parval1 : valeur initiale du paramètre. Elle devrait être la meilleure estimation de la valeur du paramètre avant de la calibration sur la base de l'avis d'expert.
- parlbnd et parubnd : seuils inférieur et supérieur de la valeur du paramètre. Si un fichier d'incertitude est donné [2], ils représentent les valeurs minimales et maximales que le paramètre peut prendre. Si un fichier d'incertitudes n'est pas donné, ils représentent l'intervalle de confidence du 95 % (une largeur de 4 déviations standard) [3] de la valeur du paramètre.
- pargp : nom du groupe de paramètres auquel appartient le paramètre.
- scale et offset : multiplicateur (1.0 par défaut) et écart (0.0 par défaut) qu'on applique au paramètre avant de l'écrire dans le fichier d'entrée du modèle. Ils peuvent être utilisés pour modifier le rang d'un paramètre pour un de plus intéressant (p. ex., si le paramètre prend des valeurs négatives et on veut utiliser une transformation logarithmique).
- dercom : entier qui indique la ligne de la section « model command line » du fichier de contrôle de PEST utilisée pour calculer les dérivées pour un paramètre donné. D'habitude il y a une seule commande (dercom=1).

2. Un fichier d'incertitude peut être donné avec la variable de contrôle de PEST++ parcov.
3. La largeur de l'intervalle peut être modifiée avec la variable de contrôle de PEST++ par_sigma_range.

Les observations

Les fichiers `obs.txt` et `pred.txt` contiennent des observations ou mesures. Ils doivent avoir le même format que le fichier de sortie du modèle, mais ils incluent seulement les observations disponibles. Il faut donner au moins un de ces fichiers.

Une façon d'interpréter les données dans `obs.txt` est de les considérer comme les données utilisées dans le procès de calibration. Les observations dans `obs.txt` prennent un poids positif (1 par défaut) et s'il y a des données manquantes (codées comme NA, NaN ou nan), les files correspondantes seront ignorées. Les poids des observations peuvent être modifiées avec l'argument `obs_weights`. Ce-ci est d'utilité quand quelques mesures sont plus incertaines que d'autres.

Par contre, les observations dans le fichier `pred.txt` ou prédictions, peuvent être vues comme les données de validation ou les dates pour lesquelles on désire d'avoir une prévision. Elles prennent un poids nul et leur valeur n'est pas utilisée dans les calculs.

Les observations peuvent être assignées à des différents groupes avec les arguments `obs_groups` et `pred_groups`. Si on assigne les observations à différents groupes, on obtient une fonction d'objectifs multiples. Par exemple, pour le modèle OKP, qu'estime la température de l'épilimnion et de l'hypolimnion, les mesures pour chaque de ces deux compartiments peuvent être placées dans de groupes différents.

Pour des modèles plus complexes ce-ci est encore de majeur utilité. Par exemple, pour le modèle GLM, l'utilisateur pourrait avoir des mesures de température prises dans le lac, température de l'eau à l'exutoire, températures à la surface mesurées par satellite, mesures du niveau d'eau, salinité mesurée dans le lac et salinité à l'exutoire. Tous ces différents types de mesures pourraient être assignés à des différents groupes. Ou, si l'étude s'intéresse d'un seul aspect, un seul de ces jeux de données (et un seul groupe) pourrait être utilisé pour calibrer le modèle.

Cependant, Doherty & Welter (2010) recommandent d'utiliser des fonctions d'objectifs multiples comme à moyen pour réduire l'influence de l'erreur structurale du modèle sur la calibration.

Le fichier d'incertitude

On peut passer de l'information à PEST++ sur l'incertitude a priori des paramètres par le moyen d'un fichier d'incertitude avec la variable de contrôle de PEST++ `parcov`. Ce fichier peut prendre plusieurs formes :

- Un fichier d'incertitude des paramètres (.unc). Il contient des déviations standard pour des paramètres individuels que ne sont corrélés à d'autres paramètres et/ou indique le nom de la matrice de covariances pour les autres paramètres.
- Un unique fichier de matrice de covariances a priori (.cov). Il contient une matrice de covariances de tous les paramètres du modèle. Le fichier est écrit d'après les spécifications du fichier de matrice de PEST.
- Un fichier binaire de matrice de covariances (.jco or .jcb). C'est un fichier utilisé par PEST++ pour écrire une matrice de Jacobi.

Pour plus de renseignements sur la spécification de ces types de fichier voyez le manuel d'utilisation de PEST++.

Si un fichier d'incertitude n'est pas donné, l'incertitude des paramètres est estimée depuis les limites des paramètres en supposant que l'intervalle couvre 4 déviations standard.

7.3.3 Réalisation des analyses

Ce paquet vous permet de réaliser plusieurs types d'analyses :

- Calibration du modèle (ou estimation des paramètres). Il y a quatre options possibles :
  - Algorithme de Gauss-Levenberg-Marquardt (GLM), en utilisant la fonction `calibration()`. Est algorithme de descente de gradient pour des cas surdéterminés.
  - Algorithme GLM avec régularisation de Tikhonov, en utilisant la fonction `calibration()`. Pour des cas sous-déterminés.
  - Évolution Différentielle (DE), en utilisant la fonction `calibration()`. C'est un algorithme global avec un temps de calcul plus long que les deux options antérieures.

  - Lissage itératif d'ensemble, en utilisant la fonction `ies()`. Il est aussi un algorithme global, spécialement utile pour des modèles non-linéaires avec un grand nombre de paramètres, qui obtient plusieurs jeux de valeurs des paramètres qui peuvent être considérés comme des valeurs calibrées.
- Analyse d'incertitudes a priori (avant calibration) et a posteriori (après calibration). Il y a les options suivantes :
  - Analyse linéaire d'incertitudes, avec la fonction `linear_uncertainty()`. Approprié pour des modèles linéaires ou presque linéaires. Il peut être appliquée à des modèles non-linéaires si le comportement du modèle dans l'espace des paramètres est approximativement linéaire ($R^2 > 0.7$). Il a besoin d'un court temps de calcul une fois le modèle calibré. Les options suivantes sont disponibles :
    - Analyse linéaire d'incertitudes a priori.
    - Analyse du complément de Schur pour la propagation d'incertitude conditionnelle.
    - Analyse de la variance de l'erreur. Il inclue aussi l'estimation de l'identifiabilité.
  - Analyse non-linéaire d'incertitudes Préférable quand le modèle est non-linéaire. Pourtant, les temps de calcul sont plus longs.
    - Simulations de Monte Carlo, avec la fonction `monte_carlo()`. D'habitude il a besoin d'un grand nombre de simulations (>1000).
    - Lissage itératif d'ensemble, en utilisant la fonction `ies()`. Il a besoin d'un nombre de simulations beaucoup moindre que les méthodes de Monte Carlo.
- Analyse de sensitivité Il y a deux options :
  - Analyse de sensitivité locale. Court temps de calcul. Les calculs sont réalisés pendant la calibration en utilisant la fonction `calibration()`.
  - Analyse de sensitivité globale (GSA). Appliqué en utilisant la fonction `gsa()`. Deux méthodes implémentées :
    - Méthode de Morris (un paramètre à la fois). Court temps de calcul.
    - Méthode de Sobol (tous les paramètres varient au même temps). Long temps de calcul.

Dans tous les cas les calculs peuvent être parallélisés en utilisant l'argument `parallel=True`.

La table suivante montre le coût computationnel de plusieurs fonctions de `cuspy` en nombre d'exécutions du modèle et de temps de calcul. Les valeurs de temps de calcul sont basées sur les cas du dossier `tests` et ont été obtenues pour une machine virtuelle Ubuntu avec 4 processeurs de 3067 MHz. Ces tests consistent en l'application de différentes fonctions `cuspy` au modèle OKP de température de l'eau avec 8 paramètres ajustables.

Tableau 7.1 – Coût computationnel de différents types d'analyse

Fonction	Détails	Parallélisé	N. exécutions	Temps calcul (s)
`calibration()`	`method='glm'`, `noptmax=10`	Oui	95	101,3
`calibration()`	`method='glm'`, `noptmax=10`	Non	95	192,1
`calibration()`	`method='glm'`, `noptmax=10`, `reg=True`	Oui	145	146,6
`calibration()`	`method='glm'`, `noptmax=10`, `reg=True`	Non	145	292,6
`calibration()`	`method='de'`, `de_max_gen=20`	Oui	801	707,3
`calibration()`	`method='de'`, `de_max_gen=20`	Non	801	1694,7
`gsa()`	`method='morris'`	Oui	36	37,0
`gsa()`	`method='morris'`	Non	36	72,8
`gsa()`	`method='sobol'`, `gsa_sobol_samples=50`	Oui	500	413,4
`gsa()`	`method='sobol'`, `gsa_sobol_samples=50`	Non	500	1009,1
`monte_carlo()`	`n_samples=50`[4]	Oui	50	50,9
`monte_carlo()`	`n_samples=50`[?]	Non	50	103,5
`ies()`	`noptmax=10`, `n_reals=50`	Oui	788	672,2
`ies()`	`noptmax=10`, `n_reals=50`	Non	788	1590,0

4. Le nombre de simulations (`n_reals`) utilisé dans des expériences de Monte Carlo est d'habitude de l'ordre de milliers ou plus. En conséquence le temps de calcul est beaucoup plus grand que celui montré ici.

7.3.4 Calibration avec GLM ou DE

Pour calibrer un modèle vous pouvez utiliser la fonction `calibration()`. Par exemple :

```
pst = cuspy.calibration(method='glm', reg=False,
                        pst_file0='test0.pst',
                        pst_file1='test1.pst',
                        control_data={'noptmax': 10})
```

L'instance PEST retournée par la fonction peut être utilisées pour accéder aux résultats de la calibration. Pour obtenir les valeurs des paramètres calibrées, tapez :

```
pst.write_par_summary_table()
```

Pour obtenir les valeurs simulées et les résidus, tapez :

```
pst.res
```

Et pour en obtenir un résumé, tapez :

```
pst.write_obs_summary_table()
```

Vous pouvez tracer les résultats en utilisant :

```
pst.plot()
```

De plus, un a certain nombre de fichiers utiles sont écrits par PEST++. Veuillez voire le manuel d'utilisation de PEST++ pour vous renseigner sur ces fichiers.

7.3.5 Analyse de sensitivité globale

Deux méthodes peuvent être utilisées pour réaliser un analyse de sensitivité globale : méthode de Morris et méthode de Sobol. Vous devez utiliser la fonction `gsa()`, et choisir la méthode désirée ("sobol" ou "morris") et configurer la méthode en utilisant l'argument *pestpp_opts*. Par exemple :

```
cuspy.gsa(method='sobol', pst_file0='pest.pst', pst_file1='pest3.pst',
          pestpp_folder=pestpp_folder,
          pestpp_opts={'gsa_sobol_samples': 50, 'gsa_sobol_par_dist': 'unif'})
```

7.3.6 Analyse linéaire d'incertitudes

Une fois la fonction a été calibrée, vous pouvez réaliser un analyse linéaire d'incertitudes avec la fonction `linear_uncertainty()`. Par exemple :

```
la = cuspy.linear_uncertainty(analysis='prior', pst_file0='test6.pst',
                              pst_file1='test6b.pst',
                              pestpp_folder=pestpp_folder)
```

Incertitude a priori

L'objet `LinearAnalysis` retourné par la fonction peut être utilisé pour accéder aux résultats de l'analyse d'incertitudes. Pour obtenir l'incertitude a priori des prédictions, tapez :

```
la.prior_forecast
```

Incertitude a posteriori

Pour réaliser un analyse du complément de Schur, vous devez le choisir avec `analysis='schur'`. Par exemple :

```
la_sc = cuspy.linear_uncertainty(analysis='schur', pst_file0='test6.pst',
                                 pst_file1='test6b.pst',
                                 pestpp_folder=pestpp_folder)
```

Avec l'analyse du complément de Schur vous pouvez obtenir l'incertitude a posteriori si vous tapez :

```
la_sc.posterior_forecast
```

Pour obtenir les incertitudes a priori et a posteriori pour les prédictions d'intérêt et la réduction de l'incertitude due à la calibration ("schur"), tapez :

```
forecast_sum = la_sc.get_forecast_summary()
```

De façon similaire, vous pouvez obtenir information sur la réduction de l'incertitude des paramètres par la calibration si vous tapez :

```
parameter_sum = la_sc.get_parameter_summary()
```

Valeur des données

Il est possible d'analyser la valeur des données avec l'objet `Schur` retourné par la fonction `linear_uncertainty()`. Les instructions suivantes retournent la variance des prédictions après l'ajout d'une nouvelle observation ou groupe d'observations :

```
la_sc.get_added_obs_importance()
la_sc.get_added_obs_group_importance()
```

Un analyse similaire peut être fait par l'élimination d'observations ou groupes d'observations avec les méthodes `get_removed_obs_importance()` et `get_removed_obs_group_importance()`.

Pour obtenir la contribution de chaque paramètre à l'incertitude de prédiction, tapez :

```
la_sc.get_par_contribution()
```

Erreur prédictive

Pour analyser l'erreur prédictive avec un analyse de la variance de l'erreur vous devez le configurer avec `analysis='err_var'`. Par exemple :

```
la_ev = cuspy.linear_uncertainty(analysis='err_var', pst_file0='test6.pst',
                                 pst_file1='test6b.pst',
                                 pestpp_folder=pestpp_folder)
```

L'objet `ErrVar` retourné par la fonction `linear_uncertainty()` peut être utilisé pour accéder aux résultats de l'analyse de la variance de l'erreur :

```
la_ev.get_errvar_dataframe()
```

Identifiabilité

Avec l'objet `ErrVar`, il est possible aussi d'accéder aux résultats de l'analyse d'identifiabilité avec :

```
la_ev.get_identifiability_dataframe()
```

7.3.7 Simulations de Monte Carlo

Pour réaliser des simulations de Monte Carlo avec le paquet `cuspy` vous pouvez utiliser la fonction `monte_carlo()`. Par exemple :

```
cuspy.monte_carlo(pst_file0='test1.pst', pst_file1='test4.pst',
                  dist_type='post', distribution='uniform',
                  n_samples=5000, pestpp_folder=pestpp_folder,
                  csv_in='sweep_in.csv', parallel=parallel)
```

Les jeux de valeurs des paramètres extraits de la distribution spécifiée sont sauvegardés dans le fichier *csv_in* (`sweep_in.csv` par défaut). Les résultats de simulation sont sauvegardés aussi dans un fichier csv, qui s'appelle `sweep_out.csv` par défaut. Le nom du fichier de sortie csv peut être modifié en utilisant l'argument *pestpp_opts* :

```
cuspy.monte_carlo(pst_file0='test1.pst', pst_file1='test4.pst',
                  dist_type='post', distribution='uniform',
                  n_samples=5000, pestpp_folder=pestpp_folder,
                  csv_in='sweep_in.csv', parallel=parallel,
                  pestpp_opts={'sweep_output_csv_file': 'sweep_out.csv'})
```

Les simulations de Monte Carlo peuvent être utiles à plusieurs finalités, incluses calibration, analyse de sensitivité global et analyse d'incertitude, en fonction du traitement qu'on fait des résultats de simulations. Aucun type particulier d'analyse n'est implémenté dans ce paquet, mais ils peuvent être implémentés facilement par le biais du traitement des fichiers csv d'entrée et de sortie (*csv_in* et *sweep_output_csv_file*).

7.3.8 Lissage Itératif d'Ensemble

Pour appliquer la méthode, vous devez utiliser la fonction `ies()`. Par exemple :

```
cuspy.ies(pst_file0='test0.pst', pst_file1='test5.pst',
          pestpp_folder=pestpp_folder, n_reals=50,
          control_data={'noptmax': 10})
```

L'implémentation de la méthode IES utilise l'algorithme GLM (appelé avec *pestpp-glm*), qui peut être configuré en utilisant les arguments *pst_file0* et *control_data*. La méthode IES peut être configurée avec l'argument *pestpp_opts*.

Le retour de la fonction `ies()` peut être trouvé dans une série de fichiers csv. Les jeux de valeurs des paramètres optimisées se trouvent dans le fichier `cas.N.par.csv`, où « cas » est le nom du fichier PEST *pst_file1* et « N » est le numéro de la dernière itération. Les résultats de simulation pour chaque jeu de paramètres se trouve dans le fichier `case.N.obs.csv`. Les résultats de la fonction d'objectifs pour toutes les itérations se trouve dans les fichiers `case.phi.actual.csv` (fonction d'objectifs calculée par rapport aux différences entre prédictions et observations), `case.phi.meas.csv` (fonction d'objectifs calculée par rapport aux différences entre prédictions et observations avec du bruit), `case.phi.group.csv` (fonction d'objectifs calculée par paramètre et groupe d'observation) et `case.phi.regul.csv` (fonction d'objectifs avec régularisation).

7.4 Références

— Doherty, J. ; Welter, D. (2010) A short exploration of structural noise. *Water Resources Research*, 46, W05525.

CHAPITRE 8

Données d'exemple pour tester le paquet cuspy

Vous pouvez tester le logiciel à l'aide des données d'exemple fourniesdans le dossier `example_data` et le modèle OKP implementé dans le paquet `okplm`. Plusieurs scripts qu'utilisent ces données et modèle sont disponibles aussi dans le dossier `tests`.

8.1 Les données

Les données de lac dans le fichier `lake.txt` correspondent au lac d'Allos (code de lac = ALL04).

Les données météorologiques (`meteo.txt`) sont des données synthetiques créées depuis des données météorologiques. La température de l'air a été créée à l'aide d'une composante saisonnière et un modèle ARMA, tandis que les données de rayonnement solaire correspondent seulement à la composante saisonnière

Les observations dans les fichiers `obs.txt` et `pred.txt` sont fitives et leur seul but est de vérifier le correct fonctionnement du paquet et d'aider dans l'apprentissage de son utilisation.

Dans le dossier il y a aussi trois fichiers qui contiennent les noms des observations (`obs_names.txt`), les noms des groupes d'observations (`obs_groups.txt`) et les poids des observations (`obs_weights.txt`).

8.2 Les scripts

Le dossier `tests` contient plusieurs scripts permettant de tester les fonctionnalités du paquet. Chaque script démontre comment faire différents types d'analyses.

Les scripts inclus sont :

- `test_0_input.py` : il démontre la gestion des fichiers d'entrée.
- `test_1_overdet.py` : il donne des exemples de calibration dans le cas surdéterminé, avec les algorithmes GLM ou DE.
- `test_2_calib_reg.py` : il donne un exemple de calibration dans le cas sous-déterminé, en utilisant la régularisation de Tikhonov.
- `test_3_sensitivity.py` : il exemplifie la réalisation d'une analyse de sensitivité globale.
- `test_4_monte_carlo.py` : il montre comment faire des simulations de Monte Carlo simulations.

— `test_5_ies.py` : il montre une application du Lissage Itératif d'Ensemble (Iterative Ensemble Smoothing).
— `test_6_uncertainty.py` : il montre un exemple de calibration et d'analyse d'incertitudes.

Index des modules Python

Index