



Université
de Rennes



Fondation
Université de Rennes

Documentation LakeRes

Suivi de thèse de Bastien Boivin

Version Draft – Document de Travail

Auteur : Bastien Boivin

Email (pro) : bastien.boivin@univ-rennes.fr

Email (perso) : bastien.boivin@proton.me

Directeur de thèse :

Jean-Raynald de Dreuzy, Directeur de recherche CNRS, Géosciences Rennes

Co-directeur de thèse :

Luc Aquilina, Professeur des universités, Géosciences Rennes

Partenaire industriel :

Jean-Yves Gaubert, Directeur du pôle R&D, Eau du Bassin Rennais

Rennes, 30 avril 2025

Table des matières

Table des matières	1
Table des figures	3
Liste des tableaux	4
1 Introduction	5
1.1 Objectifs du document	6
1.2 Contexte du projet	6
1.3 Guide d'utilisation	6
2 Bibliographie	7
2.1 Modflow	8
2.1.1 Modflow NWT	8
2.1.2 Package DRN (drain)	8
2.1.3 Package SFR (streamflow-routing)	8
2.2 Fuite du Lac (Leakage)	8
3 Données	9
3.1 DEM	10
3.1.1 BD-ALTI-75m	10
3.2 Climatiques (passé)	10
3.3 Projections climatiques	10
3.4 Hydrologie	10
3.4.1 Stations de jaugeage	10
3.4.2 Hydrographie	10

3.4.3	Intermittence	10
3.5	Géologie	10
3.6	Données EBR	10
3.6.1	Abaque Bathymétrie	10
3.6.2	Données journalières	10
3.6.3	Scénarios de gestion	10
4	Code - EBR	11
4.1	App EBR commun.py	12
4.1.1	Chargements des bibliothèques, modules et du dossier racines	12
4.1.2	LogManager	12
4.2	Initialisation de la classe climatiques	12
4.2.1	Réanalyse Surfex	12
4.2.2	Méthode de création d'un csv pour données climatiques	12
4.3	Paramétrisation	13
4.3.1	Simplex	13
5	HydroModPy	16
5.1	watershed_root.py	17
5.2	toolbox.py	17
5.2.1	class LogManager	17
6	Patch	20
6.1	DeprecationWarnings	21
6.2	Suppression des fichiers.chk	21
	Références bibliographiques	22

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1	Objectifs du document	6
1.2	Contexte du projet	6
1.3	Guide d'utilisation	6

1.1 Objectifs du document

Ce document a pour but de fournir une documentation technique dans le cadre de mon doctorat. Il est conçu pour expliquer les concepts, les méthodes et les résultats de mes recherches, en passant par la bibliographie, les résultats, les concepts ainsi que l'explication du code développé au sein d'HydroModPy, initié par Alexandre Coche.

1.2 Contexte du projet

1.3 Guide d'utilisation

Chapitre 2

Bibliographie

2.1	Modflow	8
2.1.1	Modflow NWT	8
2.1.2	Package DRN (drain)	8
2.1.3	Package SFR (streamflow-routing)	8
2.2	Fuite du Lac (Leakage)	8

2.1 Modflow

2.1.1 Modflow NWT

Modflow NWT est une version de Modflow qui intègre un solveur non linéaire pour simuler des conditions de flux d'eau souterraine. Il est particulièrement utile pour modéliser des aquifères avec des conditions de recharge variable et des niveaux d'eau fluctuants.

2.1.2 Package DRN (drain)

2.1.3 Package SFR (streamflow-routing)

2.2 Fuite du Lac (Leakage)

Chapitre 3

Données

3.1	DEM	10
3.1.1	BD-ALTI-75m	10
3.2	Climatiques (passé)	10
3.3	Projections climatiques	10
3.4	Hydrologie	10
3.4.1	Stations de jaugeage	10
3.4.2	Hydrographie	10
3.4.3	Intermittence	10
3.5	Géologie	10
3.6	Données EBR	10
3.6.1	Abaque Bathymétrie	10
3.6.2	Données journalières	10
3.6.3	Scénarios de gestion	10

3.1 DEM

3.1.1 BD-ALTI-75m

3.2 Climatiques (passé)

3.3 Projections climatiques

3.4 Hydrologie

3.4.1 Stations de jaugeage

3.4.2 Hydrographie

3.4.3 Intermittence

3.5 Géologie

3.6 Données EBR

3.6.1 Abaque | Bathymétrie

3.6.2 Données journalières

3.6.3 Scénarios de gestion

Chapitre 4

Code - EBR

4.1	App EBR commun.py	12
4.1.1	Chargements des bibliothèques, modules et du dossier racines	12
4.1.2	LogManager	12
4.2	Initialisation de la classe climatiques	12
4.2.1	Réanalyse Surfex	12
4.2.2	Méthode de création d'un csv pour données climatiques	12
4.3	Paramétrisation	13
4.3.1	Simplex	13

4.1 App EBR commun.py

4.1.1 Chargements des bibliothèques, modules et du dossier racines

Cette section permet l'importation de l'ensemble des librairies utilisées par le code, dont celles de Python, celles de librairies externes et les codes d'HydroModPy fonctionnant en POO (programmation orientée objet). Ces différentes librairies sont toutes incluses dans l'environnement `Hydromodpy-0.1` préalablement installé.

En amont de ces librairies, une section `## Filtrer les avertissements` est à renseigner à chaque début de code afin que les alertes de `DeprecationWarnings` ne s'affichent pas, voir 6.1.

4.1.2 LogManager

La `class LogManager` permet de gérer l'interface verbale entre l'utilisateur et le code, en faisant remonter des logs selon différentes classes avec plus ou moins de précisions et de messages selon le mode choisi. Pour paramétrer le `LogManager`, voir la section 5.2.1.

4.2 Initialisation de la classe climatiques

4.2.1 Réanalyse Surfex

4.2.2 Méthode de création d'un csv pour données climatiques

En temps normal, HydroModPy (à l'échelle de la France) fonctionne automatiquement avec les données SIM2. Pour la Bretagne, la recharge et le runoff sont modifiés à partir des données de réanalyse. Ici, des données ISBA brutes issues du serveur FTP de Météo-France sont utilisées directement.

Ce procédé nécessite de fusionner des fichiers NetCDF à chaque itération, ce qui est coûteux en calcul. De plus, les données de réanalyses doivent être extraites dans chaque dossier de sortie, sauf si elles sont externalisées au préalable.

Une méthode plus simple consiste à exécuter une dernière fois la méthode classique, puis à créer un `DataFrame` pour exporter l'ensemble des données climatiques, comme ci-dessous :

```

1  #=====
2  # Exportation des données climatiques
3  # =====
4  # df_climatic = pd.DataFrame({
5  #     'recharge': BV.climatic.recharge,
6  #     'runoff': BV.climatic.runoff,
7  #     'precip': BV.climatic.precip,
8  #     'evt': BV.climatic.evt,
9  #     'etp': BV.climatic.etp,
10 #     't': BV.climatic.t,
11 # })
12 # df_climatic.to_csv(os.path.join(data_path, 'Meteo', 'Historiques SIM2', 'climatic_data.csv'))

```

Ensuite, toute la classe climatique peut être mise en commentaire afin de ne garder que la lecture du CSV précédemment créé, comme ci-dessous :

```

1 df_climatic = pd.read_csv(
2     os.path.join(data_path, 'Meteo', 'Historiques SIM2', 'climatic_data.csv'),
3     index_col=0, parse_dates=True
4 )
5 df_climatic.index = pd.to_datetime(df_climatic.index)
6 df_climatic = df_climatic.loc[
7     (df_climatic.index >= pd.Timestamp("01/01/{}".format(first_year))) &
8     (df_climatic.index <= pd.Timestamp("31/12/{}".format(last_year)))
9 ]
10
11 agg_dict = {
12     'recharge': 'sum',
13     'runoff': 'sum',
14     'precip': 'sum',
15     'evt': 'sum',
16     'etp': 'sum',
17     't': 'mean'
18 }
19 df_climatic = df_climatic.resample(freq_input).agg(agg_dict)
20
21 BV.climatic.recharge = df_climatic['recharge']
22 BV.climatic.runoff = df_climatic['runoff']
23 BV.climatic.precip = df_climatic['precip']
24 BV.climatic.evt = df_climatic['evt']
25 BV.climatic.etp = df_climatic['etp']
26 BV.climatic.t = df_climatic['t']
27
28 first_clim = BV.climatic.recharge[0]
29 BV.climatic.update_first_clim(first_clim)

```



Remarque : Il est conseillé d'exporter le fichier en données journalières, puis de procéder à la réanalyse (hebdomadaire, mensuelle, etc.) lors de l'import. La sélection automatique des dates minimale et maximale peut être réalisée à l'aide des arguments déjà renseignés.

4.3 Paramétrisation

4.3.1 Simplex

Le Simplex de Nelder-Mead est un algorithme d'optimisation non-linéaire. Son principe repose sur la construction d'une figure géométrique à $N + 1$ sommets dans un espace à N dimensions, où N représente le nombre de paramètres à calibrer.

Formulation de Nelder-Mead

L'algorithme fonctionne en minimisant le volume du simplexe à travers une série d'opérations géométriques jusqu'à convergence vers le jeu de paramètres optimal. À chaque itération, les valeurs de la fonction objectif aux sommets sont ordonnées :

$$f(x_1) \leq f(x_2) \leq \dots \leq f(x_{N+1}) \quad (4.1)$$

Quatre opérations géométriques principales sont appliquées (GAO et HAN, 2012) :

- *Réflexion* : création d'un point x_r en reflétant le plus mauvais sommet x_{N+1} par rapport au centroïde des autres sommets
- *Expansion* : si le point réfléchi est prometteur, on étend dans cette direction (point x_e)
- *Contraction* : si le point réfléchi n'améliore pas suffisamment la solution, on contracte le simplexe vers l'intérieur (point x_c)
- *Rétrécissement* : si aucune des opérations précédentes n'améliore la solution, on rétrécit l'ensemble du simplexe vers le meilleur sommet x_1

La version adaptative proposée par GAO et HAN (2012) améliore l'algorithme classique en ajustant dynamiquement les paramètres de réflexion (ρ), d'expansion (χ), de contraction (γ) et de rétrécissement (σ) en fonction de la dimension du problème :

$$\rho = 1, \quad \chi = 1 + \frac{2}{N}, \quad \gamma = 0.75 - \frac{0.5}{N}, \quad \sigma = 1 - \frac{1}{N} \quad (4.2)$$

Intégration dans le flux d'exécution

L'algorithme Simplex est intégré juste avant le lancement des simulations MODFLOW, entre la mise à jour des paramètres du modèle et l'exécution standard de la simulation. Une condition booléenne permet de contrôler l'activation de la calibration automatique :

```
if run_simplex then : ## executer la calibration
```

Une fois la paramétrisation terminée, le meilleur jeu de paramètres est utilisé pour mettre à jour les valeurs initiales et lancer la simulation standard.

Paramètres calibrés

Dans notre implémentation actuelle, trois paramètres clés sont calibrés :

- Épaisseur de l'aquifère (e)
- Porosité (ϕ)
- Conductivité hydraulique (K)

Le Simplex est particulièrement adapté pour calibrer entre 2 et 5 paramètres maximum, ce qui correspond bien à notre cas d'utilisation. Cette efficacité pour un nombre limité de paramètres est l'une des raisons pour lesquelles la méthode est privilégiée dans notre contexte hydrogéologique (GAO et HAN, 2012).

Fonction objectif et sélection des données

La fonction objectif utilisée pour l'évaluation de performance est le critère de Nash-Sutcliffe (NSE) :

$$\text{NSE} = 1 - \frac{\sum_{t=1}^T (Q_{obs,t} - Q_{sim,t})^2}{\sum_{t=1}^T (Q_{obs,t} - \bar{Q}_{obs})^2} \quad (4.3)$$

où $Q_{obs,t}$ représente le débit observé au temps t , $Q_{sim,t}$ le débit simulé, et \bar{Q}_{obs} la moyenne des débits observés.

Une fonction dédiée en début de script permet de sélectionner les chroniques de données qui seront utilisées pour la calibration. L'utilisateur peut définir une période spécifique en précisant les années minimale et maximale dans la plage commune des données de simulation et d'observation. Cette approche permet d'optimiser le modèle sur une période représentative tout en excluant potentiellement des périodes atypiques qui pourraient biaiser la calibration.

Critères de convergence

Suivant les recommandations de GAO et HAN (2012), la convergence de l'algorithme est évaluée par deux critères :

- L'écart-type des valeurs de la fonction objectif aux sommets du simplexe est inférieur à un seuil prédéfini
- Le diamètre maximal du simplexe (distance maximale entre deux sommets) est inférieur à une tolérance définie

Lorsque ces deux critères sont satisfaits simultanément, l'algorithme s'arrête et retourne le meilleur jeu de paramètres identifié (GAO et HAN, 2012).

Chapitre 5

HydroModPy

5.1	watershed_root.py	17
5.2	toolbox.py	17
5.2.1	class LogManager	17

5.1 watershed_root.py

5.2 toolbox.py

5.2.1 class LogManager

Le `LogManager` est conçue pour configurer et gérer la journalisation de l'application de manière flexible et adaptable.

Initialisation du LogManager : Pour intégrer le `LogManager` dans un script, il suffit d'insérer les lignes suivantes :

```
1 # Initialisation du LogManager en mode developpement
2 log_manager = toolbox.LogManager(
3     mode="dev", # Utilisez mode="verbose" pour afficher les logs INFO et superieurs, et mode="quiet"
4     # pour afficher les logs WARNING et superieurs
5     log_dir=root_dir, # Specifiez le repertoire de journalisation
6     overwrite=False, # Utilisez overwrite=True (par default) pour ecraser les fichiers de log
7     # existants
8     verbose_libraries=True # Utilisez verbose_libraries=True pour afficher les logs des bibliotheques
9     # (avertissements et superieurs, generalement masques)
10 )
```

Mode de fonctionnement :

- Mode `dev` :
 - Console : Affiche tous les messages de niveau DEBUG et supérieur (DEBUG, INFO, WARNING, ERROR, CRITICAL).
 - Format : `\%([levelname)s] [\%(name)s] [\%(module)s:\%(lineno)d] \%(message)s`
- Mode `verbose` :
 - Console : Affiche tous les messages de niveau INFO et supérieur (INFO, WARNING, ERROR, CRITICAL).
 - Format : `\%([levelname)s] \%(message)s`
- Mode `quiet` :
 - Console : Affiche uniquement les messages de niveau WARNING et supérieur (WARNING, ERROR, CRITICAL).
 - Format : `\%([levelname)s] \%(message)s`

Gestion des bibliothèques externes :

Par défaut, le `LogManager` supprime les logs provenant de certaines bibliothèques externes pour éviter un terminal (kernel) surchargé. Voici la liste des bibliothèques dont les logs sont réduits au niveau CRITICAL :

```
1 libraries_to_silence = [
2     "fiona",
3     "rasterio",
4     "urllib3",
```

```
5 "geopy",
6 "matplotlib",
7 "PIL"
8 ]
```

Vous pouvez activer les logs des bibliothèques externes en définissant `verbose_libraries=True` lors de l'initialisation. Dans ce cas, les messages de niveau WARNING et supérieur seront affichés pour ces bibliothèques.

Sauvegarde des Logs :

- **Fichier de log** : Un fichier `dev.log` est automatiquement sauvegardé dans le dossier `dev.log` à la racine du projet.
- **Format** : Les logs sont enregistrés dans le format `dev` pour inclure la provenance des messages (fichier et numéro de ligne).
- **Écrasement** : Par défaut, le fichier est écrasé à chaque nouvelle exécution. Pour ajouter les logs successifs, utilisez `overwrite=False`.

Logique des niveaux de Logging :

Les scripts situés dans `src/` ont été mis à jour pour respecter la logique suivante :

- `logging.debug` : Points d'étape détaillés (peut générer beaucoup de lignes, notamment dans les boucles).
- `logging.info` : Messages classiques équivalents aux `print`.
- `logging.warning` : Avertissements nécessitant une attention particulière de l'utilisateur ou signalant une erreur mineure sans arrêt du code.
- `logging.error` : Erreurs mettant fin à l'exécution du script.
- `logging.critical` : Actuellement non utilisé.

Exceptions :

Certains `print` sont conservés pour des raisons spécifiques :

- Affichage du logo d'HydroModPy.
- Décompte des étapes (ex. "Étape 1/51") afin de ne pas surcharger le terminal.

Actuellement, les `print` dans les fichiers d'exécution, comme les exemples, n'ont pas été mis à jour. Il reste à discuter si nous les conservons en tant que `print` ou si nous les remplaçons par des logs de niveau `logging.info()`.

Changement de syntaxe pour le Logging

La syntaxe utilisée pour les messages de logs a été modifiée, car le module `logging` ne permet pas d'insérer directement plusieurs variables dans une chaîne de caractères, comme c'est possible avec un simple `print` (par exemple : `print("Exemple" + A + B)` ou `print("Exemple", A, B)`). Pour formater les messages dans le contexte de logging, deux approches sont possibles :

- Utilisation des f-strings :

- `logging.debug(f"Etape : {i} / {len(x)}")`
- Utilisation des Spécificateurs de Format, associés aux variables dans l'ordre :
 - `logging.debug("Etape : %s / %s", i, len(x))`
 - * Liste des principaux spécificateurs utiles :
 - `%s` : Pour les chaînes de caractères.
 - `%d` : Pour les entiers.
 - `%f` : Pour les nombres à virgule flottante.

Chapitre 6

Patch

6.1	DeprecationWarnings	21
6.2	Suppression des fichiers.chk	21

6.1 DeprecationWarnings

Les `DeprecationWarning` sont affichés dans le kernel lorsque des méthodes ou définitions d'une bibliothèque Python sont appelées et que ces dernières vont être supprimées dans une prochaine version. HydroModPy étant actuellement basé sur une version 3.8.10 de Python (version actuelle 3.13), beaucoup de `DeprecationWarning` apparaissent. Pour éviter cela, les quatre lignes ci-dessous sont à inclure en début de script.

 Supprimer l'affichage de ces messages ne pose aucun problème de fonctionnement à l'exécution du code.

```
1 # Filtrer les avertissements (avant les imports)
2 import warnings
3 warnings.filterwarnings('ignore', category=DeprecationWarning)
4
5 import pkg_resources # A placer apres DeprecationWarning car elle meme obsolète...
6 warnings.filterwarnings('ignore', message='.*pkg_resources.*')
7 warnings.filterwarnings('ignore', message='.*declare_namespace.*')
```

6.2 Suppression des fichiers.chk

À ce jour, je n'ai trouvé aucune information dans la bibliographie de Flopy permettant de désactiver la création des fichiers `*.chk`. Ces fichiers sont générés directement par le solveur et non par Flopy lui-même. Seules des variantes faites maison permettent de contourner la création de ces fichiers. Deux solutions sont donc possibles :

1. La première serait de simplement ajouter ces fichiers dans le `.gitignore` pour éviter leur synchronisation.
2. Sinon, créer un script qui supprime tous les fichiers se terminant par `*.chk`, sous la forme d'une fonction `def` dans la `toolbox`, appelée à la fin des `post-traitements` de **Modflow** et **Modpath**.

```
1 clean_root = [dirname(root_dir), self.watershed_folder]
2 for clean_root in clean_root:
3     for dirpath, dirnames, filenames in os.walk(clean_root):
4         print(dirpath, filenames, dirnames)
5         for filename in filenames:
6             if filename.endswith('.chk'):
7                 os.remove(os.path.join(dirpath, filename))
8                 print(f"Delete {filename} file")
```

Références bibliographiques

GAO, F. et HAN, L. (jan. 2012). « Implementing the Nelder-Mead Simplex Algorithm with Adaptive Parameters ». Dans : *Computational Optimization and Applications* 51.1, 259-277. DOI : [10.1007/s10589-010-9329-3](https://doi.org/10.1007/s10589-010-9329-3). URL : <http://link.springer.com/10.1007/s10589-010-9329-3> (visité le 27/03/2025).