SEGULA
TECHNOLOGIES

okplm : Guide d'Utilisation et Développement

Version 1.0.0

J. PRATS-RODRÍGUEZ **P.-A. DANIS**

janv. 26, 2023

Contenu :

CHAPITRE 1

Présentation

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

`okplm` (modèle de lacs d'Ottosson-Kettle-Prats) est un paquet en Python 3 utilisé pour simuler la température de l'épilimnion et de l'hypolimnion des plans d'eau douce avec le modèle de lacs d'Ottosson-Kettle-Prats (OKP) (Prats & Danis, 2019).

Le modèle OKP simule la température de l'eau à fréquence journalière utilisant la température de l'air [en ºC] et le rayonnement solaire [en W m^2] comme données d'entrée. Le modèle OKP est le résultat de la combinaison des modèles présentés par Ottosson & Abrahamsson (1998) et Kettle *et al.* (2004). Le développement du modèle a été financé par l'AFB (Agence Française pour la Biodiversité, antérieurement ONEMA, Office National de l'Eau et des Milieux Aquatiques) et une version préliminaire a été présentée dans le rapport par Prats & Danis (2015). La version définitive a été publiée par Prats & Danis (2019).

Pour calculer la température de l'eau avec `okplm` il faut définir les valeurs d'une série de paramètres. Il est possible d'utiliser les valeurs des paramètres par défaut obtenus pour les plans d'eau français. Ces valeurs sont le résultat de la paramétrisation en fonction des caractéristiques des lacs (latitude, altitude, profondeur maximale, aire de surface, volume) proposée par Prats & Danis (2019). Cette paramétrisation a été obtenue après analyse de données des réseaux nationaux de mesure établis pour l'application de la Directive Cadre de l'Eau pour 414 plans d'eau français plus larges de 0,06 km^2. La table suivante résume les caractéristiques de ces plans d'eau et définit les rangs espérés d'applicabilité de la paramétrisation par défaut.

Variable	Min.	Max.
Altitude (m)	0,0	2841
Latitude (ºN)	41,47	50,87
Max. profondeur (m)	0,8	310
Max. surface (km^2)	0,06	580
Max. volume (hm^3)	0,12	89000

Sinon, des valeurs des paramètres définies par l'utilisateur peuvent être utilisées. Il est possible de trouver des valeurs des paramètres pour d'autres régions dans les travaux d'Ottosson & Abrahamsson (1998) (lacs suèdes) et Kettle *et al.* (2004) (température de l'épilimnion pour lacs du sud-ouest de Groenland).

Enfin, si des données de terrain sont disponibles, les valeurs des paramètres peuvent être calées par l'utilisateur avec le paquet.

1.2 Auteurs

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

1.3 Références

— Kettle, H. ; Thompson, R. ; Anderson, N. J. ; Livingstone, D. M. (2004) Empirical modeling of summer lake surface temperatures in southwest Greenland. *Limnology and Oceanography*, 49 (1), 271-282, doi : 10.4319/lo.2004.49.1.0271.
— Ottosson, F. ; Abrahamsson, O. (1998) Presentation and analysis of a model simulating epilimnetic and hypolimnetic temperatures in lakes. *Ecological Modelling*, 110, 233-253, doi : 10.1016/S0304-3800(98)00067-2.
— Prats, J. ; Danis, P.-A. (2015) Optimisation du réseau national de suivi pérenne in situ de la température des plans d'eau : apport de la modélisation et des données satellitaires. Rapport final. Irstea-Onema, Aix-en-Provence. 93 p. https://www.documentation.eauetbiodiversite.fr/notice/optimisation-du-reseau-national-de-suivi-perenne-in-situ-de-la-temperature-des-plans-d-eau-apport-de0
— Prats, J. ; Danis, P.-A. (2019) An epilimnion and hypolimnion temperature model based on air temperature and lake characteristics. *Knowledge and Management of Aquatic Ecosystems*, 420, 8, doi : 10.1051/kmae/2019001.

CHAPITRE 2

Installation

2.1 Besoins et dépendances

Il est nécessaire d'avoir une version Python 3.5 ou ultérieure pour faire tourner le paquet `okplm`. 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 : Dans Windows, pour utiliser `pip` pour installer les paquets requis, veuillez vérifier que le chemin à l'exécutable de `pip` a été ajouté à la variable d'environnement `PATH`. L'adresse peut varier selon le cas, si vous l'installez pour un ou tous les utilisateurs (e.g., `%USERPROFILE%\AppData\roaming\Python\Python37\scripts` ou `C:\Users\MyUserName\AppData\Local\Programs\Python\Python37\Scripts`).

L'application `okplm` dépend du package Python `numpy`. Assurez-vous qu'il est installé avant l'utilisation de `okplm`.

Si vous ne devez pas compiler la documentation, `sphinx` n'est pas nécessaire pour utiliser `okplm`, 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
```

2.2 Clonage du référentiel

Vous devez cloner le paquet `okplm` avec git. Pour ça, placez-vous dans un dossier approprié (e.g., `pathtoprojrctsfolder`) où copier le code du projet et clonez le projet `okplm` :

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

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

Pour installer la branche de développement du projet, après avoir cloné le paquet `okplm`, passez à la branche `dev` :

```
cd okplm
git checkout dev
```

2.3 Installation de `okplm`

Pour installer `okplm`, allez dans le répertoire créé lors du clonage du paquet `okplm` (p. ex., ``pathtorepertoryokplm`) et installez-le en utilisant `pip` :

```
cd pathtorepertoryokplm
pip install -U .
```

2.4 Compilation de la documentation du projet

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

2.4.1 Documentation en anglais

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

```
make html
```

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

2.4.2 Documentation en français

Pour compiler le manuel d'utilisation en français en html allez au dossier `pathtorepertoryokplm/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 `pathtorepertoryokplm/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 sauvegardés dans le dossier `pathtorepertoryokplm/sphinx-doc/build_fr/latex`.

CHAPITRE 3

Guide de style de codage

3.1 Code Python

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

3.2 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).

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

3.3 Documentation

Quelques documents d'aide ont été écrits en format markdown (README.md, package_description.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 4

Traduction de la documentation du projet

Le package `okplm` a été écrit originalement en anglais et traduit en 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/1.8/intl.html.

4.1 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`.

4.2 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, placez-vous dans le 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 ""
```

4.3 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 : Il faut veiller 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 Translation Editor ou PO edit) ou des logiciels de traduction assistée par ordinateur (TAO) comme OmegaT.

4.4 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 `pathtorepertoryokplm/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 `pathtorepertoryokplm/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 sauvegardés dans le dossier `pathtorepertoryokplm/sphinx-doc/build_fr/latex`.

4.5 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 5

Modules

5.1 Module `parameter_constants`

Constantes du modèle de lacs OKP.

Ce module définit les constantes des équations utilisées pour estimer les valeurs des paramètres du modèle OKP en fonction des caractéristiques des plans d'eau. Les constantes sont définies dans Prats & Danis (2019).

Références

— Prats, J. ; Danis, P.-A. (2019) An epilimnion and hypolimnion temperature model based on air temperature and lake characteristics. *Knowledge and Management of Aquatic Ecosystems*, 420, 8, doi : 10.1051/kmae/2019001.

5.2 Module `parameter_functions`

Fonctions pour calculer les valeurs des paramètres.

Ce module contient les définitions des fonctions utilisées pour estimer la valeur des paramètres utilisés par le modèle de lacs OKP. Ces équations ont été obtenues par Prats & Danis (2019).

Les fonctions incluses dans ce module sont :
- estimate_par_a : estime le paramètre A.
- estimate_par_alpha : estime le paramètre alpha.
- estimate_par_b : estime le paramètre B.
- estimate_par_beta : estime le paramètre beta.
- estimate_par_c : estime le paramètre C.
- estimate_par_e : estime le paramètre E.
- estimate_parameters : estime les valeurs des paramètres d'OKP.

Références

— Prats, J. ; Danis, P.-A. (2019) An epilimnion and hypolimnion temperature model based on air temperature and lake characteristics. *Knowledge and Management of Aquatic Ecosystems*, 420, 8, doi : 10.1051/kmae/2019001.

parameter_functions.**estimate_par_a**(*var_vals*, *par_cts*)

Estime le paramètre A.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur des variables indépendantes "latitude" (dégrées Nord), "altitude" (m) et "surface" (m^2).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "A1" à "A4".

Renvoie La valeur estimée du paramètre *A* d'après l'Éq. (21) dans Prats & Danis (2019, p.6).

Exemple

```
var_vals = {'latitude': 44.233,
            'altitude': 2232,
            'surface': 528425}
par_cts = {'A1': 39.9,
           'A2': -0.484,
           'A3': -4.52E-3,
           'A4': -0.167}
a = estimate_par_a(var_vals=var_vals, par_cts=par_cts)
```

parameter_functions.**estimate_par_alpha**(*var_vals*, *par_cts*)

Estime le paramètre alpha.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur des variables indépendantes "altitude" (m), "surface" (m^2), et "volume" (m^3).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "ALPHA1" à "ALPHA4".

Renvoie La valeur estimée du paramètre α d'après l'Éq. (23) dans Prats & Danis (2019, p. 6).

Exemple

```
var_vals = {'altitude': 2232,
            'surface': 528425,
            'volume': 9775853}
par_cts = {'ALPHA1': 0.52,
           'ALPHA2': -3.0E-4,
           'ALPHA3': 0.25,
           'ALPHA4': -0.36}
alpha = estimate_par_alpha(var_vals=var_vals, par_cts=par_cts)
```

parameter_functions.**estimate_par_b**(*var_vals*, *par_cts*)

Estime le paramètre B.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur de la variable indépendante "zmax" (m).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "B1" à "B2".

Renvoie La valeur estimée du paramètre *B* d'après l'Éq. (24) dans Prats & Danis (2019, p.6).

Exemple

```
var_vals = {'zmax': 51}
par_cts = {'B1': 1.058,
           'B2': -0.0010}
b = estimate_par_b(var_vals=var_vals, par_cts=par_cts)
```

parameter_functions.**estimate_par_beta**(*par_e*, *par_cts*)

Estime le paramètre beta.

Paramètres

— **par_e** – valeur du paramètre E [0 - 1].

— **par_cts** – un dictionnaire qui indique la valeur des constantes "BETA1" à "BETA3".

Renvoie La valeur estimée du paramètre β d'après l'Éq. (27) dans Prats & Danis (2019, p. 9).

Exemple

```
par_e = 0.24
par_cts = {'BETA1': 1.0,
           'BETA2': 0.13,
           'BETA3': 0.95}
beta = estimate_par_beta(par_e=par_e, par_cts=par_cts)
```

parameter_functions.**estimate_par_c**(*var_vals*, *par_cts*)

Estime le paramètre C.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur de la variable indépendante "altitude" (m).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "C1" à "C2".

Renvoie La valeur estimée du paramètre *C* d'après l'Éq. (25) dans Prats & Danis (2019, p. 6).

Exemple

```
var_vals = {'altitude': 2232}
par_cts = {'C1': 1.12E-3,
           'C2': -3.62E-6}
c = estimate_par_c(var_vals=var_vals, par_cts=par_cts)
```

parameter_functions.**estimate_par_e**(*var_vals*, *par_cts*)

Estime le paramètre E.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur des variables indépendantes "surface" (m^2) et "volume" (m^3).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "E1" à "E3".

Renvoie La valeur estimée du paramètre *E* d'après l'Éq. (28) dans Prats & Danis (2019, p. 10).

Exemple

```
var_vals = {'surface': 528425,
            'volume': 9775853}
par_cts = {'E1': 0.10,
           'E2': 2.0,
           'E3': -1.8}
e = estimate_par_e(var_vals=var_vals, par_cts=par_cts)
```

parameter_functions.**estimate_parameters**(*var_vals*, *par_cts*)

Estime les valeurs des paramètres d'OKP.

Paramètres

— **var_vals** – un dictionnaire qui indique la valeur des variables indépendantes "latitude" (dégrées Nord), "altitude" (m), "zmax" (m), "surface" (m^2), "volume" (m^3).

— **par_cts** – un dictionnaire qui indique la valeur des constantes "ALPHA1"-"ALPHA4", "BETA1"-"BETA3", "A1"-"A4", "B1"-"B2", "C1"-"C2", "D", "E1"-"E3".

Renvoie Un dictionnaire des valeurs des paramètres du modèle OKP d'après l'Éq. (21, 23-25, 27-28) dans Prats & Danis (2019, p. 6-10).

Exemple

```
var_vals = {'latitude': 44.233,
            'altitude': 2232,
            'surface': 528425,
            'volume': 9775853,
            'zmax': 51}
par_cts = {'A1': 39.9, 'A2': -0.484, 'A3': -4.52E-3, 'A4': -0.167,
           'ALPHA1': 0.52, 'ALPHA2': -3.0E-4, 'ALPHA3': 0.25,
           'ALPHA4': -0.36,
           'B1': 1.058, 'B2': -0.0010,
           'BETA1': 1.0, 'BETA2': 0.13, 'BETA3': 0.95,
           'C1': 1.12E-3, 'C2': -3.62E-6,
           'D': 0.51,
           'E1': 0.10, 'E2': 2.0, 'E3': -1.8}
pars = estimate_parameters(var_vals=var_vals, par_cts=par_cts)
```

5.3 Module input_output

Fonctions pour lire et écrire des données.

Les fonctions de ce module servent à lire les fichiers de configuration et d'entrée du modèle de lacs OKP, ainsi qu'à écrire les résultats dans un fichier texte.

Ce module contient les fonctions suivantes :

— read_dict : lit un fichier de lac ou de paramètres dans un dictionnaire.

— write_dict : écrit dictionnaire dans un fichier.

input_output.**read_dict**(*path*)

Lit fichier de lac ou de paramètres et stocke dans un dictionnaire.

Paramètres **path** – chemin du fichier texte. Le fichier devrait être organisé en deux colonnes séparées par un espace ; la première colonne contient les noms des clés et la deuxième colonne contient ses valeurs.

Renvoie Un dictionnaire Python créé depuis les paires clé-valeur dans le fichier texte.

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

Écrit dictionnaire à un fichier.

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.

5.4 Module time_functions

Fonctions pour opérations temporelles.

Ce module contient fonctions pour des opérations temporelles (p. ex., calcul de la moyenne sur une période ou sélection de rangs de dates).

Les fonctions incluses sont :

- daily_f : applique fonction sur des périodes journalières.
- monthly_f : applique fonction sur des périodes mensuelles.
- select_daterange : retourne les indices entre deux dates.
- weekly_f : applique fonction sur des périodes hebdomadaires.

time_functions.**daily_f**(*t*, *x*, *funcname*)

Applique une fonction à valeurs avec fréquence infra-journalière pour obtenir des valeurs journalières.

Paramètres

- **t** – séquence de type datetime à fréquence infra-journalière. Les données manquantes ne sont pas permises.
- **x** – séquence de données de la même longitude que t.
- **funcname** – la fonction à utiliser (sum, numpy.mean, etc.).

Renvoie Un tuple de deux séquences/arrays (t_day, y_day). La séquence t_day est une séquence de type datetime à fréquence journalière. La séquence y_day est la séquence des données de sortie, résultat de l'application de funcname aux valeurs x pour chaque jour. Si les données d'entrée disponibles pour un jour sont moins de 90% des mesures possibles pour un jour, une valeur nan est retournée.

time_functions.**monthly_f**(*t*, *x*, *funcname*, *input_type*)

Applique une fonction à valeurs avec fréquence journalière/infra-journalière pour obtenir des valeurs mensuelles.

Paramètres

- **t** – séquence de type datetime à fréquence journalière ou infra-journalière. Il ne devrait pas y avoir des données manquantes.
- **x** – séquence de données de la même longitude que t.
- **funcname** – la fonction à utiliser (sum, numpy.mean, etc.).
- **input_type** – type de données d'entrée, « daily » (journalières) or « subdaily » (infra-journalières).

Renvoie Un tuple de deux séquences/arrays (t_mon, y_mon). La séquence t_mon est une séquence de type datetime à fréquence hebdomadaire. La date indique le début de chaque mois. La séquence y_mon est la séquence des données de sortie, résultat de l'application de funcname aux valeurs x pour chaque mois. S'il y a au moins 3 jours avec des données manquantes dans un mois, une valeur nan est retournée.

time_functions.**select_daterange**(*t*, *t_start*, *t_end*)

Retourne les indices des dates comprises entre deux dates.

Paramètres

- **t** – liste ou array de dates en format datetime.

— **t_start** – date initiale en format datetime.
— **t_end** – date finale en format datetime.

Renvoie Un array des indices des dates dans t comprises entre t_start et t_end (ou égales).

time_functions.**weekly_f**(*t*, *x*, *funcname*, *input_type*)
Applique une fonction à valeurs avec fréquence journalière/infra-journalière pour obtenir des valeurs hebdomadaires.

Paramètres
— **t** – séquence de type datetime à fréquence journalière ou infra-journalière. Il ne devrait pas y avoir des données manquantes.
— **x** – séquence de données de la même longitude que t.
— **funcname** – la fonction à utiliser (sum, numpy.mean, etc.).
— **input_type** – type de données d'entrée, « daily » (journalières) or « subdaily » (infra-journalières).

Renvoie Un tuple de deux séquences/arrays (t_week, y_week). La séquence t_week est une séquence de type datetime à fréquence hebdomadaire. La date indique le début de chaque semaine. La séquence y_week est la séquence des données de sortie, résultat de l'application de funcname aux valeurs x pour chaque semaine. S'il n'y a pas de données disponibles pour tous les jours d'une semaine, une valeur nan est retournée.

5.5 Module okp_model

Fonctions du modèle de lacs OKP.

Ce module contient les fonctions utilisées pour calculer la température de l'épilimnion et de l'hypolimnion d'après le modèle OKP, décrit dans Prats & Danis (2019).

Les fonctions incluses sont :
— calc_epilimnion_temperature : calcule la température de l'épilimnion.
— calc_hypolimnion_temperature : calcule la température de l'hypolimnion.
— fit_sinusoidal : cale une fonction sinusoïdale.
— main : analyse les arguments en ligne de commande et exécute le modèle OKP.
— run_okp : exécute le modèle OKP.
— water_density : calcule la densité de l'eau.

Références

— Prats, J. ; Danis, P.-A. (2019) An epilimnion and hypolimnion temperature model based on air temperature and lake characteristics. *Knowledge and Management of Aquatic Ecosystems*, 420, 8, doi : 10.1051/kmae/2019001.

okp_model.**calc_epilimnion_temperature**(*tair*, *sr*, *par_vals*, *periodicity='daily'*)
Calcule la température de l'épilimnion.

Paramètres
— **tair** – température de l'air journalière (ºC).
— **sr** – rayonnement solaire journalier (W/m^2).
— **par_vals** – a dictionary with values for the parameters ALPHA, A, B, C, at_factor and sw_factor.
— **periodicity** – fréquence des données météorologiques d'entrée et de la simulation ; les valeurs possibles sont "daily" (journalière), "weekly" (hebdomadaire), "monthly" (mensuelle).

Renvoie La température simulée de l'épilimnion en dégrées C.

okp_model.**calc_hypolimnion_temperature**(*tepi*, *par_vals*, *periodicity='daily'*)
Calcule la température de l'hypolimnion.

Paramètres
- **tepi** – température de l'épilimnion journalière (ºC).
- **par_vals** – un dictionnaire avec valeurs des paramètres BETA, A, D et E.
- **periodicity** – fréquence des données d'entrée de la température de l'épilimnion et de la simulation ; les valeurs possibles sont "daily" (journalière), "weekly" (hebdomadaire), "monthly" (mensuelle).

Renvoie La température simulée de l'hypolimnion en ºC.

okp_model.**fit_sinusoidal**(*x*, *y*, *period*)

Cale une fonction sinusoïdale aux données.

Cette fonction cale une fonction sinusoïdale du type :

$y = m + a\sin(2\pi x/period + ph)$

Paramètres
- **x** – array de données de temps.
- **y** – array de données de réponse.
- **period** – longueur de la période en unités de temps.

Renvoie Un tuple (m, a, ph) des trois coefficients d'une fonction sinusoïdale donnant la valeur moyenne (m), l'amplitude de la fonction sinusoïdale (a), et la phase de la fonction sinusoïdale (ph).

okp_model.**main**()

Analyse les arguments en ligne de commande arguments et exécute le modèle OKP.

Pour obtenir aide sur cette fonction tapez « run_okp -h » en ligne de commande.

okp_model.**run_okp**(*output_file*, *meteo_file*, *par_file*, *lake_file=None*, *start_date=None*, *end_date=None*, *periodicity='daily'*, *output_periodicity=None*, *validation_data_file=None*, *validation_res_file=None*)

Exécute le modèle OKP.

Paramètres
- **output_file** – chemin du fichier de sortie.
- **meteo_file** – chemin du fichier de données météorologiques.
- **par_file** – chemin du fichier de paramètres.
- **lake_file** – chemin fichier de données de lac (optionnel, seulement nécessaire si par_file n'est pas muni).
- **start_date** – date de début de la simulation avec le format "AAAA-mm-jj".
- **date** (*end*) – date de finalisation de la simulation avec le format "AAAA-mm-jj".
- **periodicity** – fréquence des données météorologiques d'entrée et de la simulation ; les valeurs possibles sont "daily" (journalière), "weekly" (hebdomadaire), "monthly" (mensuelle).
- **output_periodicity** – fréquence des données de sortie (implémentée seulement pour simulations journalières) ; les valeurs possibles sont "daily" (journalière), "weekly" (hebdomadaire), "monthly" (mensuelle). Utilisée seulement si la fréquence des simulations est "daily".
- **validation_data_file** – chemin du fichier contenant données observées pour calculer les statistiques d'erreur. Si validation_data_file est défini, il faut définir aussi validation_res_file. Si c'est None, les statistiques d'erreur ne sont pas calculées. La validation est implémentée uniquement pour simulations journalières.
- **validation_res_file** – chemin du fichier où écrire les résultats de la validation. Il nécessite la définition d'un fichier validation_data_file valide.

Renvoie Un fichier texte nommé output_file est écrit. Si par_file n'existe pas, il est créé aussi par cette fonction. Si des données de validation sont fournies, le fichier validation_res_file contenant information sur les statistiques d'erreur est créé aussi.

okp_model.**water_density**(*temp*)

Calcule la densité de l'eau en fonction de la température.

Paramètres **temp** – température de l'eau (ºC).

Renvoie La densité de l'eau (kg/m^3) est calculée avec la formule de Markofsky & Harleman (1971).

Références

— Markofsky, M. and Harleman, D. R. F. (1971) *A predictive model for thermal stratification and water quality in reservoirs.* Environmental Protection Agency.

5.6 Module `validation`

Fonctions la validation des résultats des simulations.

Ce module contient uniquement une fonction, utilisée pour valider les résultats de simulation :
— error_statistics : calcule statistiques d'erreur.

`validation.`**`error_statistics`**(*t_sim*, *v_sim*, *t_obs*, *v_obs*)

Calcule statistiques d'erreur.

Paramètres
— **`t_sim`** – array de temps des données simulées.
— **`v_sim`** – array avec des valeurs simulées.
— **`t_obs`** – array de temps des données observées.
— **`v_obs`** – valeurs observées.

Renvoie Un tuple de six indicateurs de performance (n, sd, r, me, mae, rmse), correspondant au nombre de mesures (n), la déviation standard (sd), le coefficient de corrélation (r), l'erreur moyenne (me), l'erreur absolue moyenne (mae), et la racine carrée de l'erreur quadratique moyenne (rmse).

CHAPITRE 6

Utilisation

6.1 Données d'entrée

Le modèle lit les données d'entrée et de configuration depuis trois fichiers texte, un obligatoire (`meteo_file`) et deux optionnels (`lake_file` et `par_file`). Les données de terrain utilisées pour la validation peuvent être lues depuis un fichier texte (`validation_data_file`). Une fois vous avez créé les fichiers d'entrée, vous pouvez utiliser le paquet `okplm` comme une application en ligne de commande ou un module Python.

6.1.1 Fichier `meteo_file`

Fichier d'entrée obligatoire. Il contient données de température de l'air et rayonnement solaire.

Le fichier est organisé en trois colonnes séparées par espaces vides :

- date : date avec le format "aaaa-mm-jj".
- tair : température de l'air journalière moyenne (ºC).
- sr : rayonnement solaire journalier moyen (W/m^2).

```
date tair sr
2015-01-01 -5.3 71.4
2015-01-02 -4.6 71.5
2015-01-03 -5.9 72.2
2015-01-04 -8.5 69.4
2015-01-05 -9.0 73.1
...
```

Il est possible de munir les données météorologiques avec trois fréquences différentes : journalière, hebdomadaire et mensuelle.

6.1.2 Fichier `lake_file`

Fichier d'entrée optionnel. Il contient les caractéristiques du lac (profondeur, surface, volume, altitude, latitude).

Le fichier est organisé en deux colonnes, séparées par espaces. La première colonne contient les noms des variables et la deuxième contient leurs valeurs. Le noms des variables sont :

- name : nom ou code du lac (optionnel, pour identifier le lac)
- zmax : profondeur du lac (m)
- surface : aire de surface du lac (m^2)
- volume : volume du lac (m^3)
- altitude : altitude sur le niveau de la mer (m)
- latitude : latitude (º)
- type : type de lac ; il peut être "L" pour les lacs (exutoire en surface) ou "R" pour les réservoirs (exutoire submergé)

Les paires de noms-valeurs peuvent être spécifiés dans n'importe quel ordre.

Par exemple, pour le Lac d'Allos (ALL04) :

```
name ALL04
altitude 2232
latitude 44.233
zmax 51
surface 528424.501
volume 9775853.276
type L
```

Il est nécessaire de fournir soit `lake_file` soit `par_file`. Si `par_file` est donné, le programme utilise les valeurs des paramètres dans `par_file`. En cas contraire, il faut donner `lake_file` et le programme calcule les paramètres du modèle depuis les caractéristiques du lac spécifiées dans `lake_file`.

6.1.3 Fichier `par_file`

Fichier d'entrée optionnel. Il contient la valeur des paramètres du modèle.

Le fichier est organisé en deux colonnes, séparées par espaces. La première colonne contient les noms des paramètres et la deuxième contient leurs valeurs. Les noms des paramètres sont :

- A : paramètre *A*. Il correspond à la température moyenne annuelle de l'épilimnion (ºC).
- B : paramètre *B*. Il module l'effet de la température de l'air.
- C : paramètre *C*. Il module l'effet du rayonnement solaire.
- D : paramètre *D*. Une valeur constante de 0.51 par défaut.
- E : paramètre *E*. Il a relation avec le ratio entre la température de l'hypolimnion et la température de l'épilimnion. Il est égal à un quand le plan d'eau n'est pas stratifié.
- ALPHA : paramètre α, facteur de lissage exponentiel de la température de l'air.
- BETA : paramètre β, facteur de lissage exponentiel de la température de l'épilimnion.
- mat : température moyenne annuelle de l'air (ºC).
- at_factor : facteur multiplicatif de la température de l'air
- sw_factor : facteur multiplicatif du rayonnement solaire.

Par exemple, pour le Lac d'Allos (ALL04) :

```
A 6.20
B 1.007
C -0.0070
D 0.51
E 0.24
```

(suite sur la page suivante)

(suite de la page précédente)

```
ALPHA 0.07
BETA 0.13
mat -0.41
at_factor 1.0
sw_factor 1.0
```

Pour la définition des paramètres et plus d'information voyez Prats & Danis (2019).

Si `par_file` n'est pas donnée par l'utilisateur, le modèle calcule les valeurs des paramètres en fonction des caractéristiques du lac définies dans `lake_file` d'accord avec la paramétrisation de Prats & Danis (2019) pour les plans d'eau douce français. Les valeurs des paramètres ainsi estimés sont écrits dans `par_file`.

The parameters `at_factor` and `sw_factor` are multiplicative factors of input meteorological data, that can be useful for sensitivity analyses. Par défaut ils prennent une valeur de 1.0.

6.1.4 Fichier `validation_data_file`

Fichier de données observées optionnel utilisé pour calculer indicateurs de performance.

Le fichier est organisé en (au maximum) trois colonnes séparées par espaces vides :
- date : date avec le format "aaaa-mm-jj".
- tepi : température de l'épilimnion (ºC) (optionnelle).
- thyp : température de l'hypolimnion (ºC) (optionnelle).

```
date tepi thyp
2015-01-10 0.0 3.9
2015-03-08 0.0 4.0
2015-04-04 2.0 4.0
2015-06-11 8.5 5.2
2015-06-12 8.0 5.3
2015-06-13 9.2 5.4
2015-08-18 13.7 6.8
2015-10-23 7.0 4.9
2015-10-29 1.2 4.0
2015-12-31 0.2 4.0
```

Le fichier contient données pour les dates où il y a des mesures disponibles. Vous pouvez fournier des données seulement pour une d'entre tepi et thyp :

```
date tepi
2015-01-10 0.0
2015-03-08 0.0
2015-04-04 2.0
2015-06-11 8.5
2015-06-12 8.0
2015-06-13 9.2
2015-08-18 13.7
2015-10-23 7.0
2015-10-29 1.2
2015-12-31 0.2
```

6.1.5 Début et fin de la simulation

Quand vous appelez la fonction `run_okp()`, vous pouvez spécifier les dates de début et fin de simulation avec les arguments `start_date` et `end_date` (ou `-s` et `-e` en ligne de commande). Le format des dates est "aaaa-mm-jj".

Si la date de début et fin ne sont pas définies, la durée de la simulation est déterminée par la longueur de `meteo_file`.

6.2 Application en ligne de commande

Pour exécuter `okplm` en ligne de commande, changez au dossier qui contient les fichiers d'entrée et faites :

```
run_okp
```

Alternativement, vous pouvez indiquer le dossier des données d'entrée. P. ex. :

```
run_okp -f C:/users/yourself/data/lake_data
```

Veuillez noter que le logiciel comprends le remplacement du tilde "~", de façon que vous pouvez utiliser aussi :

```
run_okp -f ~/data/lake_data
```

Par défaut, le modèle cherche les fichiers nommés `meteo.txt` (données météorologiques), `lake.txt` (données du lac) et `par.txt` (valeurs des paramètres du modèle). Vous pouvez spécifier d'autres noms de fichier avec les arguments optionnels `-m`, `-l` et `-p`, respectivement. P. ex.,

```
run_okp -m meteorology.txt
```

De façon similaire, les résultats sont écrit par défaut dans `output.txt`, mais vous pouvez définir un autre nom avec `-o`.

Vous pouvez limiter la durée de la simulation spécifiant les dates de début et fin :

```
run_okp -s 2014-01-01 -e 2015-12-31
```

Pour communiquer au modèle quelle est la fréquence des données météorologiques d'entrée et de la simulation vous pouvez utiliser `-d` (journalière), `-w` (hebdomadaire) or `-n` (mensuelle). P. ex.

```
run_okp -w
```

Par défaut le logiciel présume que les données d'entrée sont données à un pas de temps journalier.

Pour simulations journalières, la sortie peut être donnée à fréquence journalière, hebdomadaire ou mensuelle avec les arguments `--daily_output`, `--weekly_output` et `--monthly_output`.

Il est possible aussi d'obtenir statistiques d'erreur des simulations journalières si on fournit un fichier de données observées (p. ex., `obs.txt`) et le nom du fichier des résultats de validation (p. ex., `err_stats.txt`) :

```
run_okp -a obs.txt -b err_stats.txt
```

Si ces noms de fichier ne sont pas données, les statistiques de validation ne sont pas calculées.

Pour avoir une aide sur l'utilisation de l'application, écrivez :

```
run_okp -h
```

6.3 Module Python

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

```
import okplm
```

Pour exécuter le modèle, d'abord définissez les noms des différents fichiers d'entrée et de sortie. Par exemple :

```
import os.path

folder = path_to_data_repertory
output_file = os.path.join(folder, 'output.txt')
meteo_file = os.path.join(folder, 'meteo.txt')
par_file = os.path.join(folder, 'par.txt')
lake_file = os.path.join(folder, 'lake.txt')
```

À nouveau, vous pouvez utiliser le tilde "~".

Après, tapez :

```
okplm.run_okp(output_file=output_file, meteo_file=meteo_file,
              par_file=par_file, lake_file=lake_file)
```

Vous pouvez aussi définir le début, la date de fin et la périodicité des simulations :

```
okplm.run_okp(output_file=output_file, meteo_file=meteo_file,
              par_file=par_file, lake_file=lake_file, start_date='2014-01-01',
              end_date='2015-12-31', periodicity='weekly')
```

La sortie des simulations journalières peut être donnée à fréquence journalière (`daily`), hebdomadaire (`weekly`) ou mensuelle (`monthly`) avec l'argument `output_periodicity`.

Si vous fournissez un fichier contenant données observées (`validation_data_file`) et le nom d'un fichier où écrire les résultats de validation (`validation_res_file`), les statistiques d'erreur sont calculées et écrites dans le fichier spécifié.

D'autres fonctions utiles sont `okplm.read_dict()` et `okplm.write_dict()`, qui peuvent être utilisées pour lire et écrire les fichiers de lac et des paramètres.

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

```
python path_to_script
```

6.4 Données de sortie

Le modèle OKP produit trois types de données de sortie :

- simulations de température de l'eau, sauvegardées au fichier `output_file`.
- valeurs estimées des paramètres (si non fournies par utilisateur), sauvegardées au fichier `par_file` décrit ci-dessus.
- indicateurs de la performance des simulations (s'il y a des données de validation), sauvegardés dans le fichier `validation_res_file`.

6.4.1 Fichier `output_file`

Fichier de sortie principale. Il contient la température simulée de l'épilimnion et de l'hypolimnion avec la périodicité de sortie sollicitée.

Le fichier est organisé en trois colonnes séparées par espaces vides :
- date : date avec le format "aaaa-mm-jj".
- tepi : température de l'épilimnion (ºC).
- thyp : température de l'hypolimnion (ºC).

```
date tepi thyp
2015-01-01 0.7736048264277242 4.0
2015-01-02 0.8253707698690544 4.001647106544848
2015-01-03 0.780854030568508 4.001663640357946
2015-01-04 0.5561293109277756 4.0
2015-01-05 0.31121842955433243 4.0
2015-01-06 0.06755075725464099 4.0
2015-01-07 0.0 4.0
2015-01-08 0.0 4.0
2015-01-09 0.0 4.0
...
```

6.4.2 Fichier `validation_res_file`

Fichier de sortie optionnel. Il contient statistiques de la simulation, calculées si `validation_data_file` et `validation_res_file` sont définis.

Le fichier est organisé en six colonnes séparées par espaces vides :
- n : nombre de mesures.
- sd : déviation standard.
- r : coefficient de corrélation.
- me : erreur moyenne.
- mae : erreur absolue moyenne.
- rmse : racine carrée de l'erreur quadratique moyenne.

La première ligne des résultats correspond à l'épilimnion, et la deuxième ligne des résultats correspond à l'hypolimnion.

```
n sd r me mae rmse
10 2.285 0.871 -0.025 1.555 2.286
10 0.596 0.757 -0.018 0.452 0.597
```

CHAPITRE 7

Tests

Le dossier `tests` contient 2 scripts permettant de tester les fonctionnalités du paquet `okplm` :

- test_okp_model.py : pour tester la `run_okp()`, fonction principale utilisée pour lancer les simulations.
- test_validation.py : pour tester la fonction `error_statistics()`, fonction utilisée pour la validation des résultats de simulation.

CHAPITRE 8

Exemples d'utilisation du paquet `okplm`

Vous pouvez tester l'application à l'aide des exemples de fichiers fournis. Le paquet contient quatre jeux de données d'exemple dans les dossier `examples` : un exemple de script Python (`tests/test_script.py`), et un module basée sur le paquet `plotly` (`tests/plotly_fonctions.py`) et permettant de visualiser les résultats.

8.1 Les données

Dans ces exemples, les données du lac dans le fichier `lake.txt` correspondent à celles du lac d'Allos. (code du lac = ALL04). Les données météorologiques (`meteo.txt`) sont des données synthétiques d'une base de données météorologiques. La température de l'air a été créée à l'aide d'une température avec une composante saisonnière et un modèle ARMA, tandis que les données de rayonnement solaire correspondent seulement à la composante saisonnière.

Trois tests peuvent être réalisés avec `test_script.py` pour illustrer l'utilisation de `okplm` pour trois différentes fréquences. Les données nécessaires pour ces trois tests sont dans les dossiers :

- synthetic_case_daily : données journalières
- synthetic_case_weekly : données hebdomadaires
- synthetic_case_monthly : données mensuelles

8.2 Les simulations

Pour ces trois cas, les données météorologiques (`meteo.txt`) et les données lacustres (`lake.txt`) sont fournies en entrée au modèle. Par conséquent, les paramètres du modèle sont calculés à partir des caractéristiques du lac dans `lake.txt` avec la formulation suivante :

```
okplm.run_okp(output_file, meteo_file, par_file, lake_file,
              periodicity=periodchoice)
```

Dans le cas d'une simulation avec des données journalières, le script `test_script.py` donne un exemple de validation des résultats du modèle avec une fichier d'observation `obs.txt`.

```
okplm.run_okp(output_file, meteo_file, par_file, lake_file,
              periodicity=periodchoice,
              validation_data_file=validation_data_file,
              validation_res_file=validation_res_file)
```

Le dernier cas de test (`synthetic_case_par_given`) est le cas où un fichier utilisateur des paramètres du modèle est donné en entrée. Les données sont dans le dossier :

— examples/synthetic_case_given : données journalières

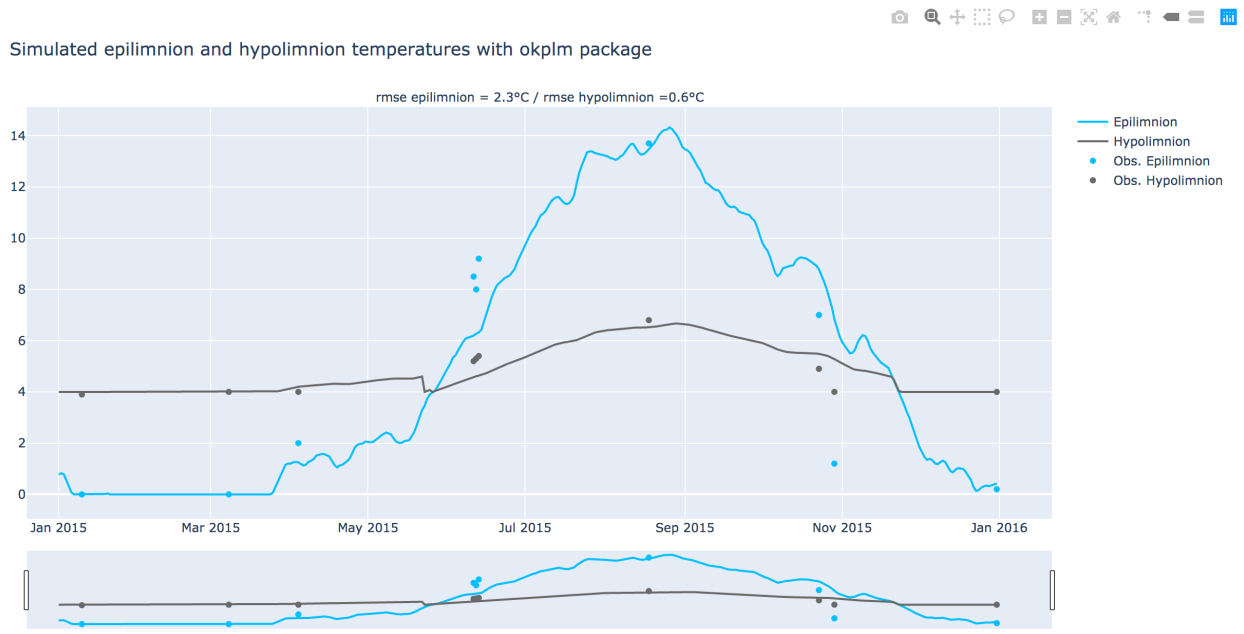
Le modèle utilise donc les valeurs de paramètre (`par.txt`) et les données météorologiques journalières (`meteo.txt`). Le fichier `lake.txt` n'est pas nécessaire.

8.3 Visualisation des résultats

La fonction `plotlyoutput()` dans le script `plotly_functions.py` permet de visualiser les résultats des simulations ainsi que, le cas échéant, les valeurs des critères de validation (n, sd, r, me, mae et rmse) et les observations avec la formulation suivante :

```
plotlyoutput(folder)
```

Les résultats sont alors présentés sous la forme d'un fichier html de ce type :



Index des modules Python

Index