Documentation okplm\_fr 0.2.4

# Bienvenue à la documentation de okplm ![¶](#welcome-to-okplm-s-documentation)

|  |  |
| --- | --- |
| [logo1](_images/logo-afb_1_2.jpg) | [logo2](_images/Segula_Technologies_Logo.jpg) |

## Présentation[¶](#presentation)

### Qu’est-ce que c’est le paquet okplm ?[¶](#what-is-the-okplm-package)

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:sup: ] comme données d’entrée. Le modèle OKP est le résultat de la combinaison des modèles développés par Ottosson & Abrahamsson (1998) et Kettle *et al.* (2004). Le développement du modèle a été financé par l’AFB (Agence Française pour la Biodiversité, anciennement 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 par défaut des paramètres obtenues pour les plans d’eau français. Ces valeurs sont le résultat de la paramétrisation en fonction des caractéristiques des lacs (latitude, altitude, profondeur maximale, surface, volume) proposée par Prats & Danis (2019). Cette paramétrisation a été obtenue après analyse de données issues des réseaux nationaux de surveillance mis en place pour la mise en application de la Directive Cadre européenne sur l’Eau pour 414 plans d’eau français de surface supérieure à 0,06 km2. La table suivante présente les gammes des valeurs des caractéristiques de ces plans d’eau et donc le domaine d’applicabilité de la paramétrisation par défaut.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Min.** | **Max.** |
| Altitude (m) | 0,0 | 2841 |
| Latitude (ºN) | 41,47 | 50,87 |
| Profondeur max. (m) | 0,8 | 310 |
| Surface max. (km2) | 0,06 | 580 |
| Volume max. (hm3) | 0,12 | 89000 |

Les valeurs des paramètres peuvent aussi être définies par l’utilisateur. Il est possible de trouver les valeurs de certains paramètres pour d’autres régions comme pour des lacs suédois dans les travaux d’Ottosson & Abrahamsson (1998) ou pour la température de l’épilimnion de lacs du sud-ouest de Groenland dans les travaux de Kettle et al. (2004).

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

### Auteurs[¶](#authors)

* Jordi Prats-Rodríguez ([jprats@segula.es](mailto:jprats@segula.es))
* Pierre-Alain Danis ([pierre-alain.danis@afbiodiversite.fr](mailto:pierre-alain.danis@afbiodiversite.fr))

### Références[¶](#references)

* 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](https://doi.org/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](https://doi.org/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. Final report. 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](https://doi.org/10.1051/kmae/2019001).

## Installation[¶](#installation)

### Besoins et dépendances[¶](#requirements-and-dependencies)

Il est nécessaire d’avoir une version de Python 3.5 ou une version 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

Sous Windows, pour utiliser pip pour l’installation des paquets Python utiles, vérifier que le chemin d’accès au fichier exécutable de pip est bien ajouté à votre variable d’environnement PATH. L’emplacement peut varier selon que vous installez Python pour un utilisateur ou pour tous les utilisateurs (p. ex., %USERPROFILE%\AppData\roaming\Python\Python37\scripts ou C:\Users\MyUserName\AppData\Local\Programs\Python\Python37\Scripts).

L’application okplm dépend du paquet 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 (pour, par exemple, faire des modifications), vous aurez besoin du paquet sphinx. Pour installer sphinx faites:

pip install sphinx

Le paquet sphinx travaille par défaut avec des 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 une 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

### Clonage du référentiel[¶](#cloning-the-repository)

Vous devez cloner le paquet okplm avec git. Pour ça, placez-vous dans un dossier approprié (p. ex., pathtoprojectsfolder) où copier le code du projet dans un sous-répertoire et clonez le projet okplm :

cd pathtoprojectsfolder  
git clone https://gitlab.irstea.fr/alamode/okplm

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

### Installation de okplm[¶](#installing-okplm)

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

Avec septuptools :

cd pathtorepertoryokplm  
python setup.py install

Avec pip :

cd pathtorepertoryokplm  
pip install -U .

### Compilation de la documentation du projet[¶](#compilation-of-the-project-documentation)

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.

#### Documentation en anglais[¶](#documentation-in-english)

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.

#### Documentation en français[¶](#documentation-in-french)

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 documentation sont convertis vers latex et après vers pdf. Les fichiers latex et pdf de sortie sont sauvegardés dans le dossier pathtorepertoryokplm/sphinx-doc/build\_fr/latex.

## Guide de style de codage[¶](#coding-style-guide)

### Code Python[¶](#python-code)

Nous avons suivi la [Guide de style pour code Python PEP 08](https://www.python.org/dev/peps/pep-0008/).

### Docstrings[¶](#docstrings)

Nous avons suivi les [Conventions Docstrings – PEP 257](https://www.python.org/dev/peps/pep-0257/) et les recommandations de la *Google Python Style Guide* pour docstrings dans des fonctions (point [3.8.1](http://google.github.io/styleguide/pyguide.html" \l "383-functions-and-methods)).

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

### Documentation[¶](#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](http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html) . La documentation traduite en français a été créé avec sphinx-intl.

## Traduction de la documentation du projet[¶](#translation-of-the-project-s-documentation)

Le package okplm a été écrit originalement en anglais et traduit en français. Cette section décrit la procédure 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 version 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>.

### Pas préliminaires[¶](#preliminary-steps)

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.

### Création de fichiers POT et PO[¶](#creation-of-pot-and-po-files)

La traduction des fichiers de documentation est basée 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 ""

### Traduction[¶](#translation)

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 efficace d’utiliser un éditeur PO (p. ex., GNOME Traduction Editor ou PO edit) ou des logiciels de traduction assistée par ordinateur (TAO) comme OmegaT.

### Compilation de la documentation traduite[¶](#compilation-of-the-translated-documentation)

La documentation en français est sauvegardée dans le dossier build\_fr. Pour compiler le manuel d’utilisation en français comme fichiers 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 documentation sont convertis vers latex et après vers pdf. Les fichiers latex et pdf de sortie sont sauvegardés dans le dossier pathtorepertoryokplm/sphinx-doc/build\_fr/latex.

### Mise à jour de la documentation traduite[¶](#update-of-the-translated-documentation)

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.

## Modules[¶](#modules)

### Module parameter\_constants[¶](#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.

### Module parameter\_functions[¶](#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 sont issues de 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*)[¶](#parameter_functions.estimate_par_a)

Estime le paramètre A.

Paramètres

* **var\_vals** – un dictionnaire qui indique la valeur des variables indépendantes “latitude” (dégrées Nord), “altitude” (m) et “surface” (m2).
* **par\_cts** – un dictionnaire qui indique la valeur des constantes “A1” à “A4”.

Retourne

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*)[¶](#parameter_functions.estimate_par_alpha)

Estime le paramètre alpha.

Paramètres

* **var\_vals** – un dictionnaire qui indique la valeur des variables indépendantes “altitude” (m), “surface” (m2), et “volume” (m3).
* **par\_cts** – un dictionnaire qui indique la valeur des constantes “ALPHA1” à “ALPHA4”.

Retourne

La valeur estimée du paramètre \(\alpha\) 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*)[¶](#parameter_functions.estimate_par_b)

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”.

Retourne

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*)[¶](#parameter_functions.estimate_par_beta)

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”.

Retourne

La valeur estimée du paramètre \(\beta\) 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*)[¶](#parameter_functions.estimate_par_c)

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”.

Retourne

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*)[¶](#parameter_functions.estimate_par_e)

Estime le paramètre E.

Paramètres

* **var\_vals** – un dictionnaire qui indique la valeur des variables indépendantes “surface” (m2) et “volume” (m3).
* **par\_cts** – un dictionnaire qui indique la valeur des constantes “E1” à “E3”.

Retourne

La valeur estimée du paramètre *C* 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*)[¶](#parameter_functions.estimate_parameters)

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” (m2), “volume” (m3).
* **par\_cts** – un dictionnaire qui indique la valeur des constantes “ALPHA1”-“ALPHA4”, “BETA1”-“BETA3”, “A1”-“A4”, “B1”-“B2”, “C1”-“C2”, “D”, “E1”-“E3”.

Retourne

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)

### Module input\_output[¶](#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*)[¶](#input_output.read_dict)

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.

Retourne

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

input\_output.write\_dict(*x\_dict*, *path*)[¶](#input_output.write_dict)

Écrit dictionnaire à un fichier.

Paramètres

* **x\_dict** – un dictionnaire Python.
* **path** – chemin du fichier texte où écrire les données.

Retourne

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

### Module time\_functions[¶](#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*)[¶](#time_functions.daily_f)

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.).

Retourne

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*)[¶](#time_functions.monthly_f)

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).

Retourne

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*)[¶](#time_functions.select_daterange)

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.

Retourne

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*)[¶](#time_functions.weekly_f)

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).

Retourne

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.

### Module okp\_model[¶](#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'*)[¶](#okp_model.calc_epilimnion_temperature)

Calcule la température de l’épilimnion.

Paramètres

* **tair** – température de l’air journalière (ºC).
* **sr** – rayonnement solaire journalier (W/m2).
* **par\_vals** – un dictionnaire avec valeurs des paramètres ALPHA, A, B et C.
* **periodicity** – fréquence des données météorologiques d’entrée et de la simulation ; les valeurs possibles sont “daily” (journalière), “weekly” (hebdomadaire), “monthly” (mensuelle).

Retourne

La température simulée de l’épilimnion en dégrées C.

okp\_model.calc\_hypolimnion\_temperature(*tepi*, *par\_vals*, *periodicity='daily'*)[¶](#okp_model.calc_hypolimnion_temperature)

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).

Retourne

La température simulée de l’hypolimnion en ºC.

okp\_model.fit\_sinusoidal(*x*, *y*, *period*)[¶](#okp_model.fit_sinusoidal)

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.

Retourne

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()[¶](#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*)[¶](#okp_model.run_okp)

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) ; es valeurs possibles sont “daily” (journalière), “weekly” (hebdomadaire), “monthly” (mensuelle).
* **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.

Retourne

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*)[¶](#okp_model.water_density)

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

Paramètres

**temp** – température de l’eau (ºC).

Retourne

La densité de l’eau (kg/m3) 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.

### Module validation[¶](#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*)[¶](#validation.error_statistics)

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.

Retourne

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).

## Utilisation[¶](#usage)

### Données d’entrée[¶](#input-data)

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.

#### Fichier meteo\_file[¶](#file-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/m2).

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.

#### Fichier lake\_file[¶](#file-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 (m2)
* volume : volume du lac (m3)
* 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.

#### Fichier par\_file[¶](#file-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 \(\alpha\), facteur de lissage exponentiel de la température de l’air.
* BETA : paramètre \(\beta\), facteur de lissage exponentiel de la température de l’épilimnion.
* mat : température moyenne annuelle de l’air (ºC).

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

A 6.20  
B 1.007  
C -0.0070  
D 0.51  
E 0.24  
ALPHA 0.07  
BETA 0.13  
mat -0.41

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 estimées sont écrites dans par\_file.

#### Fichier validation\_data\_file[¶](#file-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

#### Début et fin de la simulation[¶](#start-and-end-of-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.

### Application en ligne de commande[¶](#command-line-application)

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

### Module Python[¶](#python-module)

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

### Données de sortie[¶](#output-data)

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.

#### Fichier output\_file[¶](#file-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  
...

#### Fichier validation\_res\_file[¶](#file-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

## Tests[¶](#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.

## Exemples d’utilisation du paquet okplm[¶](#examples-of-usage-of-the-package-okplm)

Vous pouvez tester le logiciel à l’aide des exemples de fichiers fournis. Le paquet contient quatre exemples de données dans le 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.

### Les données[¶](#the-data)

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

### Les simulations[¶](#the-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

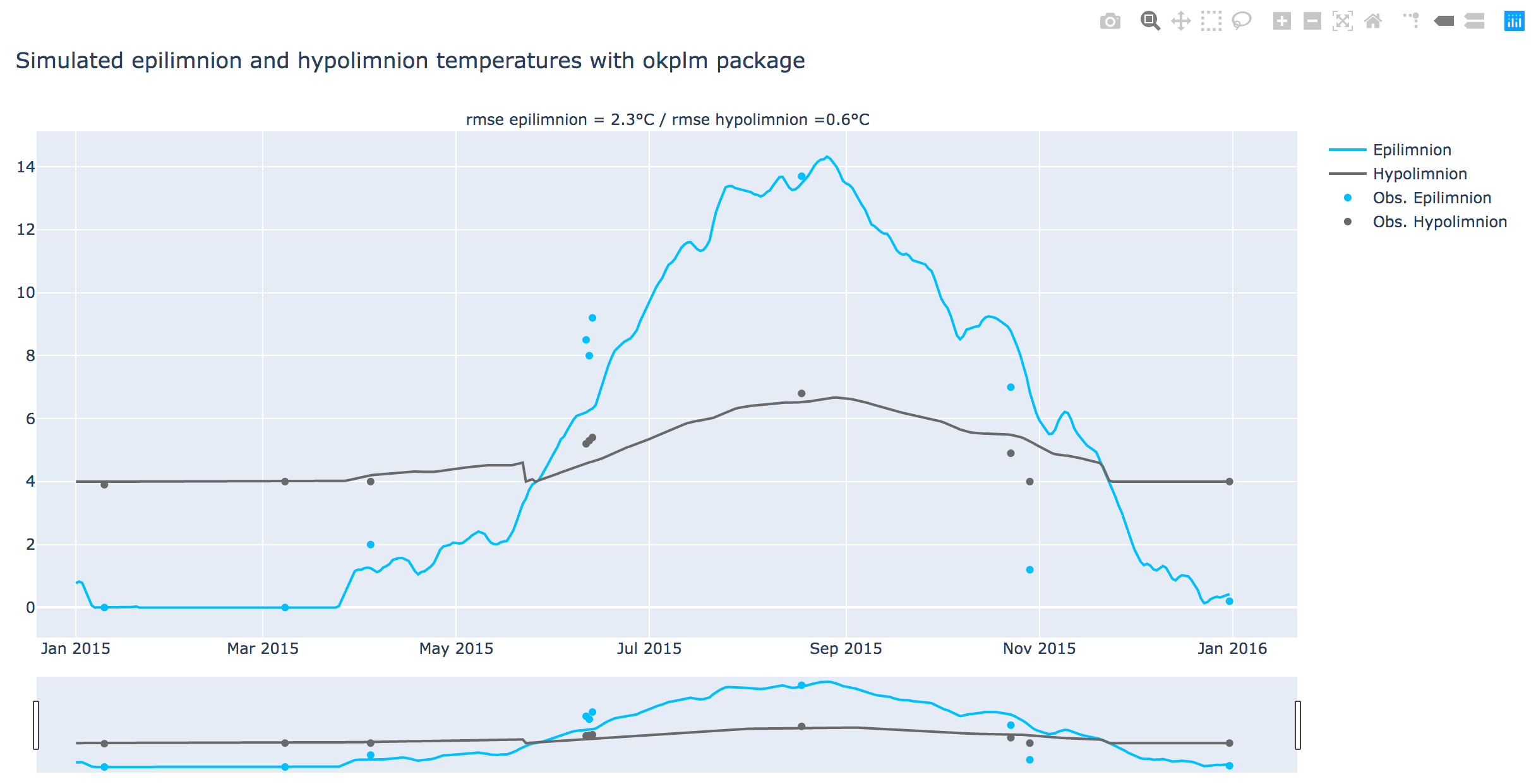
Le modèle utilise donc les valeurs de paramètre (par.txt) et les données météorologiques journalières (meteo.txt). Le fichier lake.txt n’est pas nécessaire.

### Visualisation des résultats[¶](#plotting-of-results)

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 :



# Indices et tableaux[¶](#indices-and-tables)

* [Index](genindex.html)
* [Index du module](py-modindex.html)
* [Page de recherche](search.html)

# [okplm\_fr](index.html#document-index)

### Navigation

Contenu :

* [Présentation](index.html#document-presentation)
* [Installation](index.html#document-installation)
* [Guide de style de codage](index.html#document-style)
* [Traduction de la documentation du projet](index.html#document-translation)
* [Modules](index.html#document-modules)
* [Utilisation](index.html#document-usage)
* [Tests](index.html#document-tests)
* [Exemples d’utilisation du paquet okplm](index.html#document-examples)

### Related Topics

* [Documentation overview](index.html#document-index)

©2019, Segula Technologies - Agence Française pour la Biodiversité. | Powered by [Sphinx 2.2.0](http://sphinx-doc.org/) & [Alabaster 0.7.12](https://github.com/bitprophet/alabaster)