SEGULA
TECHNOLOGIES

okplm: User and Developer Manual

Release 1.0.0

J. Prats-Rodríguez **P.-A. Danis**

Jan 26, 2023

CONTENTS:

CHAPTER

ONE

PRESENTATION

1.1 What is the `okplm` package?

`okplm` (Ottosson-Kettle-Prats lake model) is a package in Python 3 used to simulate the epilimnion and hypolimnion temperature of freshwater bodies using the Ottosson-Kettle-Prats (OKP) lake model (Prats & Danis, 2019).

The OKP model simulates water temperature at the daily frequency using air temperature [in ºC] and solar radiation [in W m^2] as forcing data. The OKP model is the result of the combination of the models presented by Ottosson & Abrahamsson (1998) and Kettle *et al.* (2004). The development of the model was funded by the AFB (French Agency for Biodiversity, previously ONEMA, French National Office for Water and Aquatic Environments) and a preliminary version was presented in the report by Prats & Danis (2015). The definitive version was published by Prats & Danis (2019).

To calculate water temperatures with `okplm` the values of a series of parameters need to be defined. It is possible to use the default parameter values obtained for French water bodies. These values are the result of the parameterisation as a function of lake characteristics (latitude, altitude, maximum depth, surface area, volume) proposed by Prats & Danis (2019). This parameterisation was derived by analysing data from the French national measuring networks set up for the application of the European Water Framework Directive for 414 French water bodies with surface area larger than 0.06 km^2. The following table summarises the range of values of the characteristics of these water bodies and thus the domain of applicability of the default parameterisation.

Variable	Min.	Max.
Altitude (m)	0.0	2841
Latitude (ºN)	41.47	50.87
Max. depth (m)	0.8	310
Max. surface (km^2)	0.06	580
Max. volume (hm^3)	0.12	89000

Parameter values can also be defined by the user. It is possible to find the values of certain parameters for other geographical settings such as Swedish lakes in the work by Ottosson & Abrahamsson (1998) or for the epilimnion temperature of southwest Greenland lakes in the work by Kettle *et al.* (2004).

Finally, if field data is available, the parameter values can be calibrated by the user using this package.

1.2 Authors

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

1.3 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.
- 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. 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.

CHAPTER

TWO

INSTALLATION

2.1 Requirements and dependencies

You need Python 3.5 or later to run the `okplm` package. You can have multiple Python versions (2.x and 3.x) installed on the same system without problems using, for example, virtual environments.

You may use `pip` to install the required packages.

Note: In Windows, to use `pip` to install the required packages, please verify that the path to `pip`'s executable file has been added correctly to your environment variable `PATH`. The address may vary depending on whether you install it for one or all users (e.g., `%USERPROFILE%\AppData\roaming\Python\Python37\scripts` or `C:\Users\MyUserName\AppData\Local\Programs\Python\Python37\Scripts`).

The application `okplm` depends on the Python package `numpy`. Make sure it is installed before using `okplm`.

If you do not have to compile the documentation, `sphinx` is not necessary to use `okplm`, since a precompiled pdf copy of the documentation is included in the folder `docs`.

If you need to compile the documentation from source (e.g., to modify it), you will need the package `sphinx`. To install `sphinx` just make:

```
pip install sphinx
```

The package `sphinx` works by default with reStructuredText documents. However, it can also recognise Markdown by installing the Markdown parser `recommonmark`.

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

To create multilingual documentation, you will need the package `sphinx-intl`. The installation process is as above:

```
pip install sphinx-intl
```

To compile pdf documents (only in Linux), you will also need to have installed latex and the Python package `latexmk`. To install latex, type:

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

And to install `latexmk`, type:

```
pip install latexmk.py
```

2.2 Cloning the repository

You need to clone the okplm package with git. For this go to an appropriate directory (e.g., pathtoprojectsfolder) where to copy the project's code in a subfolder and clone the okplm project:

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

This command creates the okplm repertory in the folder pathtoprojectsfolder.

To install the development branch of the project, after cloning the okplm package, change to the dev branch:

```
cd okplm
git checkout dev
```

2.3 Installing okplm

To install okplm, go to the repertory created during the cloning of the package okplm (e.g., pathtorepertoryokplm) and install it using pip:

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

2.4 Compilation of the project documentation

The source files for the project user manual are stored in the folder pathtorepertoryokplm/sphinx-doc/source. Sphinx also extracts data from the project modules docstrings.

2.4.1 Documentation in English

To compile the user manual in English as html files go to the folder pathtorepertoryokplm/sphinx-doc and type:

```
make html
```

The output html files are saved in the folder pathtorepertoryokplm/sphinx-doc/build/html

You can also compile the user manual as a pdf file making:

```
make latexpdf
```

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder pathtorepertoryokplm/sphinx-doc/build/latex.

2.4.2 Documentation in French

To compile the user manual in French as html files go to the folder `pathtorepertoryokplm/sphinx-doc` and type:

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

The output html files are saved in the folder `pathtorepertoryokplm/sphinx-doc/build_fr/html`.

To compile the pdf documentation, type the following commands:

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

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder `pathtorepertoryokplm/sphinx-doc/build_fr/latex`.

CHAPTER

THREE

CODING STYLE GUIDE

3.1 Python code

We have followed the PEP 08 – Style Guide for Python Code.

3.2 Docstrings

We have followed the PEP 257 – Docstring Conventions and the recommendations of the *Google Python Style Guide* for docstrings in functions (point 3.8.1).

According to the GNU General Public License, a copyright notice is included in every module.

3.3 Documentation

Some help documents have been written using markdown (README.md, package_description.md).

The user manual has been generated using `sphinx` from files using reStructuredText markup language. The documentation translated into French has been created using `sphinx-intl`.

CHAPTER

FOUR

TRANSLATION OF THE PROJECT'S DOCUMENTATION

The `okplm` package is originally written in English and translated into French. This section describes the process to translate the *User and Developer Guide* by using pot and po files and the package `sphinx-intl`. Using this method only the modified sections of text need to be translated again when modifications are made to the original documentation. It also facilitates the translation of docstrings and module documentation. The translation instructions are based on https://sphinx-doc.org/en/1.8/intl.html.

4.1 Preliminary steps

Before starting the translation it is advisable to check the source files thoroughly to eliminate any language or format issues.

A glossary of technical terms has been created to improve the consistency of the translation. It is located in the folder `sphinx-doc/glossaries`.

4.2 Creation of POT and PO files

The translation of the documentation files is based on the extraction of the translatable text into pot (portable object template) files and the creation of translated text in po (portable object) files. For this, the package `sphinx-intl` is used.

To extract the document's translatable messages into pot files, change to the `sphinx-doc` directory in the terminal and do:

```
make gettext
```

The resulting pot files are saved in the folder `build/gettext`. They contain the translatable text broken down into segments.

Then you need to generate the po files for each target language. For example, for French you need to do

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

The resulting po files are stored in the folder `source/locale/fr/LC_MESSAGES`.

The po files contain pairs of segments of text in the source (msgid) and target language (msgstr).

Before starting the translation, the value of msgstr is empty:

```
#: ../../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 Translation

Thus to translate the text you need to edit the po files and type de translated text beside 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: Be careful to maintain the reST formatting in the translated version.

You can do the translation by hand by modifying the po files. However, it is more efficient to use a PO editor (e.g., GNOME Translation Editor or PO edit) or computer assisted translation (CAT) software such as OmegaT.

4.4 Compilation of the translated documentation

The documentation in French is saved in the folder `build_fr`. To compile the user manual in French as html files go to the folder `pathtorepertoryokplm/sphinx-doc` and type:

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

The output html files are saved in the folder `pathtorepertoryokplm/sphinx-doc/build_fr/html`.

To compile the pdf documentation, type the following commands:

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

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder `pathtorepertoryokplm/sphinx-doc/build_fr/latex`.

4.5 Update of the translated documentation

If the project's documentation is updated, it is necessary to create new pot files following the procedure described above. To apply the changes to the po files, do

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

Then, you only need to translate the modified segments.

CHAPTER

FIVE

MODULES

5.1 Module parameter_constants

OKP lake model constants.

This module defines the constants of the equations used to estimate the parameter values of the model OKP as a function of water body characteristics. The constants are defined in Prats & Danis (2019).

References

- 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

Functions to calculate parameter values.

This module contains the definition of the functions used to estimate the value of the parameters used by the OKP lake model. These equations were derived in Prats & Danis (2019).

The functions included in this module are:

- estimate_par_a: estimate parameter A.
- estimate_par_alpha: estimate parameter alpha.
- estimate_par_b: estimate parameter B.
- estimate_par_beta: estimate parameter beta.
- estimate_par_c: estimate parameter C.
- estimate_par_e: estimate parameter E.
- estimate_parameters: estimate OKP parameter values.

References

- 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*)

Estimate the parameter A.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variables 'latitude' (degrees North), 'altitude' (m) and 'surface' (m^2).
- **par_cts** – a dictionary indicating the value of the parameter constants 'A1' to 'A4'.

Returns The estimated value of the parameter *A* according to Eq. (21) in Prats & Danis (2019, p.6).

Example

```
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*)

Estimate the parameter alpha.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variables 'altitude' (m), 'surface' (m^2), and 'volume' (m^3).
- **par_cts** – a dictionary indicating the value of the parameter constants 'ALPHA1' to 'ALPHA4'.

Returns The estimated value of the parameter α according to Eq. (23) in Prats & Danis (2019, p. 6).

Example

```
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*)

Estimate the parameter B.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variable 'zmax' (m).
- **par_cts** – a dictionary indicating the value of the parameter constants 'B1' to 'B2'.

Returns The estimated value of the parameter *B* according to Eq. (24) in Prats & Danis (2019, p.6).

Example

```
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*)

Estimate the parameter beta.

Parameters

- **par_e** – value of the parameter E [0 - 1].
- **par_cts** – a dictionary indicating the value of the parameter constants 'BETA1' to 'BETA3'.

Returns The estimated value of the parameter β according to Eq. (27) in Prats & Danis (2019, p. 9).

Example

```
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*)

Estimate the parameter C.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variable 'altitude' (m).
- **par_cts** – a dictionary indicating the value of the parameter constants 'C1' to 'C2'.

Returns The estimated value of the parameter *C* according to Eq. (25) in Prats & Danis (2019, p. 6).

Example

```
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*)

Estimate the parameter E.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variables 'surface' (m^2) and 'volume' (m^3).
- **par_cts** – a dictionary indicating the value of the parameter constants 'E1' to 'E3'.

Returns The estimated value of the parameter *E* according to Eq. (28) in Prats & Danis (2019, p. 10).

Example

```
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*)

Estimate the OKP parameter values.

Parameters

- **var_vals** – a dictionary indicating the value of the independent variables 'latitude' (degrees North), 'altitude' (m), 'zmax' (m), 'surface' (m^2), 'volume' (m^3).
- **par_cts** – a dictionary indicating the value of the parameter constants 'ALPHA1'-'ALPHA4', 'BETA1'-'BETA3', 'A1'-'A4', 'B1'-'B2', 'C1'-'C2', 'D', 'E1'-'E3'.

Returns A dictionary of the OKP model parameter values according to Eq. (21, 23-25, 27-28) in Prats & Danis (2019, p. 6-10).

Example

```
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

Functions to read and write data.

The functions in this module are used to read the configuration and input data files of the OKP lake model, as well as for writing the results to a text file.

This module contains the following functions:

- read_dict: read lake or parameter file to dictionary.
- write_dict: write dictionary to file.

input_output.**read_dict**(*path*)
Read lake or parameter file to dictionary.

Parameters **path** – path of text file. The file should be structured in two columns separated by a space; the first column contains key names and the second column contains their values.

Returns A Python dictionary created from the key-value pairs in the text file.

input_output.**write_dict**(*x_dict*, *path*)
Write dictionary to file.

Parameters

- **x_dict** – a Python dictionary.
- **path** – path of the text file to write the data.

Returns A file located at "path" where the dictionary data is written.

5.4 Module time_functions

Functions for time-related operations.

This module contains functions for time operations (e.g., time averaging or selection of date ranges).

The included functions are:

- daily_f: apply function on daily periods.
- monthly_f: apply function on monthly periods.
- select_daterange: return indices between two dates.
- weekly_f: apply function on weekly periods.

time_functions.**daily_f**(*t*, *x*, *funcname*)
Apply a function using subdaily values as args to obtain daily values.

Parameters

- **t** – datetime sequence at subdaily frequency. Missing timestamps are not allowed.
- **x** – data sequence, the same length of t.
- **funcname** – the function to be used (sum, numpy.mean, etc.).

Returns A tuple of two sequences/arrays (t_day, y_day). The sequence t_day is a datetime sequence at daily frequency. The sequence y_day is the output data sequence, result of applying funcname to the x values for each day. If the input data for a given day is less than 90% of the measurements for a day, a nan value is returned.

time_functions.**monthly_f**(*t*, *x*, *funcname*, *input_type*)
Apply a function using daily/subdaily values to obtain monthly values.

Parameters

- **t** – datetime sequence at daily or subdaily frequency. There should not be missing timestamps.
- **x** – data sequence, the same length of t.
- **funcname** – the function to be used (sum, numpy.mean, etc.).
- **input_type** – type of input data, "daily" or "subdaily".

Returns A tuple of two sequences/arrays (t_mon, y_mon). The sequence t_mon is a datetime sequence at monthly frequency. The date indicates the beginning of each month. The sequence y_mon is the output data sequence, result of applying funcname to the x values for each month. If data there are at least 3 days with missing data in a month, a nan value is returned.

time_functions.**select_daterange**(*t*, *t_start*, *t_end*)
Return indices of dates comprised between two dates.

Parameters

- **t** – list or array of dates in datetime format.
- **t_start** – initial date in datetime format.
- **t_end** – last date in datetime format.

Returns An array of indices of dates in t comprised between t_start and t_end (or equal).

time_functions.**weekly_f**(*t*, *x*, *funcname*, *input_type*)
Apply a function using daily/subdaily values to obtain weekly values.

Parameters

- **t** – datetime sequence at daily/subdaily frequency. There should not be missing timestamps.
- **x** – data sequence, the same length of t.
- **funcname** – the function to be used (sum, numpy.mean, etc.).
- **input_type** – type of input data, "daily" or "subdaily".

Returns A tuple of two sequences/arrays (t_week, y_week). The sequence t_week is a datetime sequence at weekly frequency. The date indicates the beginning of each week. The sequence y_week is the output data sequence, result of applying funcname to the x values for each week. If data is not available for all days for a given week, a nan value is returned.

5.5 Module okp_model

OKP lake model functions.

This module contains the functions used to calculate epilimnion and hypolimnion temperature corresponding to the OKP model, described in Prats & Danis (2019).

The included functions are:

- calc_epilimnion_temperature: calculate epilimnion temperature.
- calc_hypolimnion_temperature: calculate hypolimnion temperature.
- fit_sinusoidal: fit a sinusoidal function.

- main: parse command line arguments and run the OKP model.
- run_okp: run the OKP model.
- water_density: calculate water density.

References

- 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'*)
Calculate epilimnion temperature.

Parameters

- **tair** – daily air temperature (ºC).
- **sr** – daily solar radiation (W/m^2).
- **par_vals** – a dictionary with values for the parameters ALPHA, A, B, C, at_factor and sw_factor.
- **periodicity** – periodicity of the input meteorological data and of the simulation; it can take the values 'daily', 'weekly', 'monthly'.

Returns The daily simulated epilimnion temperature in degrees C.

okp_model.**calc_hypolimnion_temperature**(*tepi*, *par_vals*, *periodicity='daily'*)
Calculate hypolimnion temperature.

Parameters

- **tepi** – daily epilimnion temperature (ºC).
- **par_vals** – a dictionary with values for the parameters BETA, A, D and E.
- **periodicity** – periodicity of the input epilimnion temperature data and of the simulation; it can take the values 'daily', 'weekly', 'monthly'.

Returns The daily simulated hypolimnion temperature in ºC.

okp_model.**fit_sinusoidal**(*x*, *y*, *period*)
Fit a sinusoidal function to data.

This function fits a sinusoidal function of the form:

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

Parameters

- **x** – array of time data.
- **y** – array of response data.
- **period** – length of the period in time units.

Returns A tuple (m, a, ph) of the three coefficients of a sinusoidal function providing the mean value (m), the amplitude of the sinusoidal function (a), and the phase of the sinusoidal function (ph).

okp_model.**main**()
Parse command line arguments and run the OKP model.

To obtain help on this function type "run_okp -h" in the command line.

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*)

Run the OKP model.

Parameters

- **output_file** – path of the output file.
- **meteo_file** – path of the meteorological data file.
- **par_file** – path of the parameter file.
- **lake_file** – path of the lake data file (optional, it is only necessary if par_file is not provided).
- **start_date** – date of start of the simulation in the format 'YYYY-mm-dd'.
- **date** (*end*) – date of end of the simulation in the format 'YYYY-mm-dd'.
- **periodicity** – periodicity of the input meteorological data and of the simulation; it can take the values 'daily', 'weekly', 'monthly'.
- **output_periodicity** – periodicity of the output data (only implemented for daily simulations); it can take the values 'daily', 'weekly', 'monthly'. It is only used if the periodicity of the simulations is 'daily'.
- **validation_data_file** – path of the file containing observational data to calculate error statistics. If validation_data_file is defined, you need to define also validation_res_file. If None, error statistics are not calculated. Validation is only implemented for 'daily' simulations.
- **validation_res_file** – path of the file where validation results will be written. It requires the definition of a valid validation_data_file.

Returns A text file named output_file is written. If the par_file does not exist, it is also created by this function. If validation data is provided, the file validation_res_file containing information on error statistics is created too.

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

Calculate the water density as a function of temperature.

Parameters **temp** – water temperature (ºC).

Returns The water density (kg/m^3) calculated using the formula by Markofsky & Harleman (1971).

References

- 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

Functions for the validation of simulation results.

This module contains only one function, used to validate simulation results:

- error_statistics: calculate error statistics.

validation.**error_statistics**(*t_sim*, *v_sim*, *t_obs*, *v_obs*)
Calculate error statistics.

Parameters

- **t_sim** – time array of the simulated data.
- **v_sim** – array with simulated values.
- **t_obs** – time array of observed data.
- **v_obs** – observed values.

Returns A tuple of six performance indicators (n, sd, r, me, mae, rmse), corresponding to the number of measurements (n), the standard deviation (sd), the correlation coefficient (r), the mean error (me), the mean absolute error (mae), and the root mean square error (rmse).

CHAPTER

SIX

USAGE

6.1 Input data

The model reads input and configuration data from three text files, one obligatory (`meteo_file`) and two optional (`lake_file` and `par_file`). Field data used for validation may also be read from a text file (`validation_data_file`). Once you have created the input files, you can use the `okplm` package as a command line application or a Python module.

6.1.1 File `meteo_file`

Obligatory input file. It contains air temperature and solar radiation data.

The file is structured in three columns separated by white spaces:

- date: date in the format 'yyyy-mm-dd'.
- tair: mean daily air temperature (ºC).
- sr: mean daily solar radiation (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
...
```

Meteorological data may be provided at three different frequencies: daily, weekly and monthly.

6.1.2 File `lake_file`

Optional input file. It contains lake characteristics (depth, surface, volume, altitude, latitude).

The file is structured in two columns, separated by blanks. The first column contains the names of the variables and the second one contains their values. The variable names are:

- name: lake name or code (optional, to identify the lake)
- zmax: lake depth (m)
- surface: lake surface area (m^2)
- volume: lake volume (m^3)

- altitude: altitude above sea level (m)
- latitude: latitude (º)
- type: lake type; it can be 'L' for lakes (surface outlet) or 'R' for reservoirs (submerged outlet)

The pairs of names-values may be specified in any order.

For example, for the Lake Allos (ALL04):

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

Providing either `lake_file` or `par_file` is necessary. If the `par_file` is given, the program uses parameter values in `par_file`. Otherwise, the `lake_file` is required and the program calculates the model parameters from the lake characteristics included in the `lake_file`.

6.1.3 File `par_file`

Optional input file. It contains the value of the model parameters.

The file is structured in two columns, separated by blanks. The first column contains the names of the parameters and the second one contains their values. The parameter names are:

- A: parameter *A*. It corresponds to the mean annual epilimnion temperature (ºC).
- B: parameter *B*. It modulates the effect of air temperature.
- C: parameter *C*. It modulates the effect of solar radiation.
- D: parameter *D*. A constant value of 0.51 by default.
- E: parameter *E*. It is related to the ratio between hypolimnion temperature and epilimnion temperature. It is equal to one when the water body is not stratified.
- ALPHA: parameter α, exponential smoothing factor of the air temperature.
- BETA: parameter β, exponential smoothing factor of the epilimnion temperature.
- mat: mean annual air temperature (ºC).
- at_factor: multiplicative factor of air temperature.
- sw_factor: multiplicative factor of shortwave radiation.

For example, for the Lake 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
```

(continues on next page)

(continued from previous page)

```
at_factor 1.0
sw_factor 1.0
```

For the definition of the parameters and further information see Prats & Danis (2019).

If `par_file` is not given by the user, the model calculates the parameter values as a function of the lake characteristics defined in `lake_file` according to the parameterisation by Prats & Danis (2019) for French freshwater bodies. The values of the parameters thus estimated are written to `par_file`.

The parameters `at_factor` and `sw_factor` are multiplicative factors of input meteorological data, that can be useful for sensitivity analyses. By default they take a value of 1.0.

6.1.4 File `validation_data_file`

Optional observational data file used for the calculation of performance indicators.

The file is structured in (at most) three columns separated by white spaces:

- date: date in the format 'yyyy-mm-dd'.
- tepi: epilimnion temperature (ºC) (optional).
- thyp: hypolimnion temperature (ºC) (optional).

```
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
```

The file contains data for the dates for which measurements are available. You may provide data for only one of tepi or 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 Start and end of simulation

When calling the function `run_okp()`, you can specify the dates of start and end of the simulation by using the arguments `start_date` and `end_date` (or `-s` and `-e` in the command line). The format of the dates is 'yyyy-mm-dd'.

If no start and end date are defined, the length of the simulation is determined by the length of the `meteo_file`.

6.2 Command line application

To run `okplm` in the command line, change to the directory containing the input files and make:

```
run_okp
```

Alternatively, you can indicate the input data folder. E.g.:

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

Please note that the software understands the tilde '~' expansion, so that you may use instead:

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

By default, the model looks for the files named `meteo.txt` (meteorological data), `lake.txt` (lake data) and `par.txt` (values of the model parameters). You can specify other names using the optional arguments `-m`, `-l` and `-p`, respectively. E.g.,

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

Similarly, the results are written by default to `output.txt`, but you can use define another name using `-o`.

You can limit the length of the simulation by specifying the start and end dates:

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

To tell the model the frequency of the input meteorological data and of the simulation you may use `-d` (daily), `-w` (weekly) or `-n` (monthly). E.g.

```
run_okp -w
```

By default the program assumes the input data is provided at a daily time step.

For daily simulations, the output can be given at daily, weekly of monthly frequencies with the arguments `--daily_output`, `--weekly_output` and `--monthly_output`.

It is also possible to obtain error statistics of the daily simulations by providing an observation data file (e.g., `obs.txt`) and the name of the validation results file (e.g., `err_stats.txt`):

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

If these file names are not provided, validation statistics are not calculated.

For obtaining help on the usage of the application, write:

```
run_okp -h
```

6.3 Python module

To use `okplm` as a Python module you can simply import it and use the functions within:

```
import okplm
```

To run the model, first define the names of the different input and output files. For example:

```
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')
```

Remember you may use the tilde '~' expansion.

Then, type:

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

You may also define a start, an end date and a periodicity for the 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')
```

The output of daily simulations can be given at `daily`, `weekly` or `monthly` frequency using the argument `output_periodicity`.

If you provide a file containing observational data (`validation_data_file`) and a file name where to write the validation results (`validation_res_file`), error statistics are calculated and written to the specified file.

Other useful functions are `okplm.read_dict()` and `okplm.write_dict()`, which can be used to read and write the lake data and parameter files.

You can include the previous commands in a Python script (see the example script `test_script.py`). To run a python script from the command line, type:

```
python path_to_script
```

6.4 Output data

The OKP model produces three types of output data:

- water temperature simulations, saved to the file `output_file`.
- estimated parameter values (if not provided by the user), saved to the file `par_file` described above.
- indicators of simulation performance (if validation data is provided), saved to the file `validation_res_file`.

6.4.1 File `output_file`

Main output file. It contains the simulated epilimnion and hypolimnion temperature at the requested output periodicity.

The file is structured in three columns separated by white spaces:

- date: date in the format 'yyyy-mm-dd'.
- tepi: epilimnion temperature (ºC).
- thyp: hypolimnion temperature (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 File `validation_res_file`

Optional output file. It contains performance statistics of the simulation, calculated if `validation_data_file` and `validation_res_file` are defined.

The file is structured in six columns separated by white spaces:

- n: number of measurements.
- sd: standard deviation.
- r: correlation coefficient.
- me: mean error.
- mae: mean absolute error.
- rmse: root mean square error.

The first line of results corresponds to the epilimnion, and the second line of results corresponds to the 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
```

CHAPTER

SEVEN

TESTS

The folder `tests` contains 2 scripts to test the functionalities of the package *okplm*:

- test_okp_model.py: to test the function `run_okp()`, main function used to run the simulations.
- test_validation.py: to test the function `error_statistics()`, function used for the validation of simulation results.

CHAPTER

EIGHT

EXAMPLES OF USAGE OF THE PACKAGE `OKPLM`

You can test the application with examples provided. The package contains four data examples in the folder `examples`. An associated Python script (`test_script.py`) and a module based on the `plotly` package (`plotly_fonctions.py`) to plot results are available in the folder `tests`.

8.1 The data

In all cases the lake data in the `lake.txt` file corresponds to the Lake Allos (lake code = ALL04). The meteorological data (`meteo.txt`) is synthetic data based on meteorological data. Air temperature has been created using a seasonal component and an ARMA model, while solar radiation data corresponds to the seasonal component only.

Three of the cases are used to exemplify the usage of `okplm` for three different periodicities. The necessary data for these tests can be found in the folders:

- synthetic_case_daily : daily data
- synthetic_case_weekly: weekly data
- synthetic_case_monthly: monthly data

8.2 The simulations

For these three case the input is meteorological data (`meteo.txt`) and lake data (`lake.txt`). Thus, the model parameters are calculated from lake characteristics given in `lake.txt` with the following command:

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

In the case of the simulation with daily data, the script `test_script.py` gives example of validation of the model results with an observation data file `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)
```

The remaining test case (synthetic_case_par_given) exemplifies the case when the parameter file is given. The data are in the folder:

- examples/synthetic_case_given : daily data

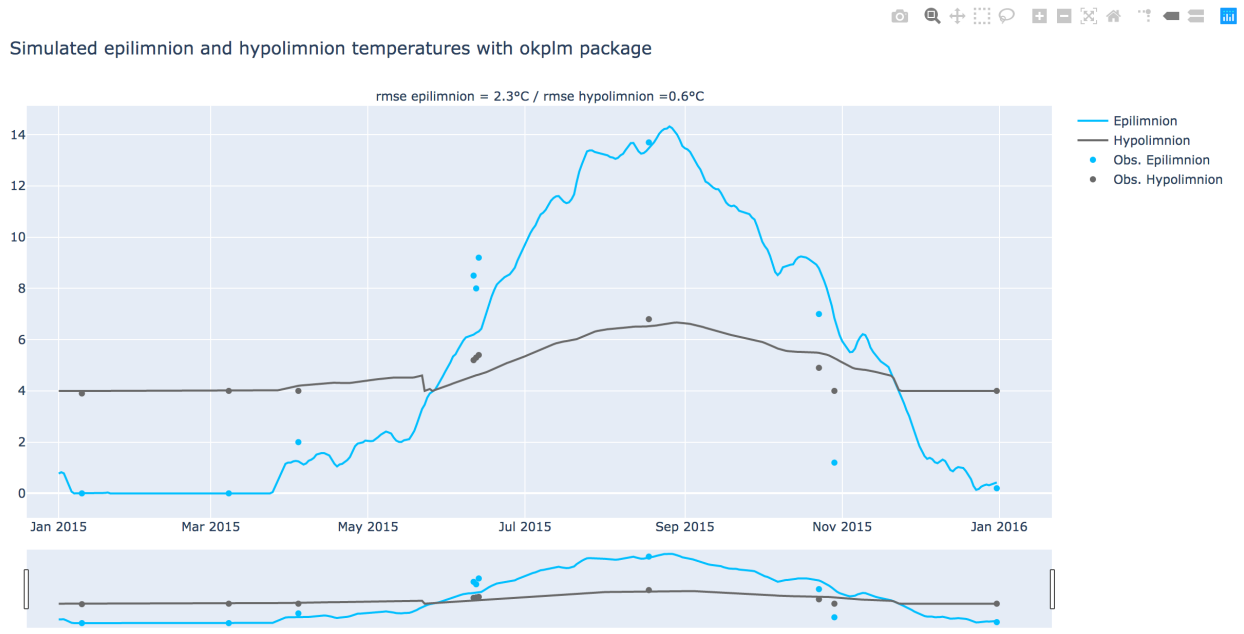
The model uses the given parameter values (`par.txt`) and the daily meteorological data (`meteo.txt`). The file `lake.txt` is not necessary.

8.3 Plotting of results

You can plot the simulation results with the function `plotlyoutput()` in the module `tests/plotly_functions.py`, in addition to validation criteria (n, sd, r, me, mae et rmse) and observations, if they are available, with the instruction :

```
plotlyoutput(folder)
```

The results are then presented as an html file such as:



PYTHON MODULE INDEX

INDEX