oemof

Release 0.4.0.beta0

May 15, 2020

Contents

CHAPTER 1

Overview: Open Energy Modelling Framework (oemof)

docs	
tests	
package	

The Open Energy Modelling Framework (oemof) is a Python toolbox for energy system modelling and optimisation.

The oemof project aims to be a loose organisational frame for tools in the wide field of (energy) system modelling. Every project is managed by their own developer team but we share some developer and design rules to make it easier to understand each other's tools. All project libraries are free software licenced under the MIT license.

All projects are in different stages of implementation, some even may not have a stable release, but all projects are open to be joined by interested people. We do not belong to a specific institution and everybody is free to join the developer teams and will have the same rights. There is no higher decision level.

This repository is also used to organise everything for the oemof community.

- Webconference dates
- Real life meetings
- Website and Mailinglist
- General communication

You can find recent topics of discussion in the issues.

Overview

1.1 Projects with stable releases

- **oemof-solph** A model generator for energy system modelling and optimisation (LP/MILP).

- **oemof-thermal**
- **cydets** Cycle Detection in Time Series (CyDeTS). An algorithm to detect cycles in times series along with their respective depth-of-cycle (DoC) and duration.

- **demandlib** The demandlib library can be used to create load profiles for elctricity and heat knowing the annual demand. See the documentation of the demandlib for examples and a full description of the library.

- **feedinlib** The feedinlib library serves as an interface between Open Data weather data and libraries to calculate feedin timeseries for fluctuating renewable energy sources.

- **TESPy** Thermal Engineering Systems in Python (TESPy). This package provides a powerful simulation toolkit for thermal engineering plants such as power plants, district heating systems or heat pumps.

- **windpowerlib** The windpowerlib is a library that provides a set of functions and classes to calculate the power output of wind turbines. It was originally part of the feedinlib (windpower and photovoltaic) but was taken out to build up a community concentrating on wind power models.

1.2 Projects in an early state

- **DHNx** District heating system optimisation and simulation models

1.3 Installation

To install the *oemof.solph* package (previously just *oemof*), please use

```
pip install https://github.com/oemof/oemof-solph/archive/master.zip"
```

1.4 Documentation

https://oemof.readthedocs.io/

1.5 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

About oemof

This overview has been developed to make oemof easy to use and develop. It describes general ideas behind and structures of oemof and its modules.

2.1 The idea of an open framework

The Open Energy System Modelling Framework has been developed for the modelling and analysis of energy supply systems considering power and heat as well as prospectively mobility.

Oemof has been implemented in Python and uses several Python packages for scientific applications (e.g. mathematical optimisation, network analysis, data analysis), optionally in combination with a PostgreSQL/PostGIS database. It offers a toolbox of various features needed to build energy system models in high temporal and spatial resolution. For instance, the wind energy feed-in in a model region can be modelled based on weather data, the CO2-minimal operation of biomass power plants can be calculated or the future energy supply of Europe can be simulated.

The framework consists of different libraries. For the communication between these libraries different interfaces are provided. The oemof libraries and their modules are used to build what we call an 'application' (app) which depicts a concrete energy system model or a subprocess of this model. Generally, applications can be developed highly individually by the use of one or more libraries depending on the scope and purpose. The following image illustrates the typical application building process.

It gets clear that applications can be build flexibly using different libraries. Furthermore, single components of applications can be substituted easily if different functionalities are needed. This allows for individual application development and provides all degrees of freedom to the developer which is particularly relevant in environments such as scientific work groups that often work spatially distributed.

Among other applications, the apps 'renpassG!S' and 'reegis' are currently developed based on the framework. 'renpassG!S' enables the simulation of a future European energy system with a high spatial and temporal resolution. Different expansion pathways of conventional power plants, renewable energies and net infrastructure can be considered. The app 'reegis' provides a simulation of a regional heat and power supply system. Another application is 'HESYSOPT' which has been desined to simulate combined heat and power systems with MILP on the component level. These three examples show that the modular approach of the framework allows applications with very different objectives.

2.2 Application Examples

Some applications are publicly available and continuously developed. Examples and a screenshot gallery can be found on oemof's official homepage.

2.3 Why are we developing oemof?

Energy system models often do not have publicly accessible source code and freely available data and are poorly documented. The missing transparency slows down the scientific discussion on model quality with regard to certain problems such as grid extension or cross-border interaction between national energy systems. Besides, energy system models are often developed for a certain application and cannot (or only with great effort) be adjusted to other requirements.

The Center for Sustainable Energy Systems (ZNES) Flensburg together with the Reiner Lemoine Institute (RLI) in Berlin and the Otto-von-Guericke-University of Magdeburg (OVGU) are developing the Open Energy System Modelling Framework (oemof) to address these problems by offering a free, open and clearly documented framework for energy system modelling. This transparent approach allows a sound scientific discourse on the underlying models and data. In this way the assessment of quality and significance of undertaken analyses is improved. Moreover, the modular composition of the framework supports the adjustment to a large number of application purposes.

The open source approach allows a collaborative development of the framework that offers several advantages:

- **Synergies** - By developing collaboratively synergies between the participating institutes can be utilized.
- **Debugging** - Through the input of a larger group of users and developers bugs are identified and fixed at an earlier stage.
- **Advancement** - The oemof-based application profits from further development of the framework.

2.4 Why should I contribute?

- You do not want to start at the very beginning. - You are not the first one, who wants to set up a energy system model. So why not start with existing code?
- You want your code to be more stable. - If other people use your code, they may find bugs or will have ideas to improve it.
- Tired of 'write-only-code'. - Developing as part of a framework encourages you to document sufficiently, so that after years you may still understand your own code.

- You want to talk to other people when you are deadlocked. - People are even more willing to help, if they are interested in what you are doing because they can use it afterwards.
- You want your code to be seen and used. We try to make oemof more and more visible to the modelling community. Together it will be easier to increase the awareness of this framework and therefore for your contribution.

We know, sometimes it is difficult to start on an existing concept. It will take some time to understand it and you will need extra time to document your own stuff. But once you understand the libraries you will get lots of interesting features, always with the option to fit them to your own needs.

If you first want to try out the collaborative process of software development you can start with a contribution on a low level. Fixing typos in the documentation or rephrasing sentences which are unclear would help us on the one hand and brings you nearer to the collaboration process on the other hand.

For any kind of contribution, please fork the oemof repository to your own github account and make changes as described in the github guidelines: https://guides.github.com/activities/hello-world/

Just contact us if you have any questions!

2.5 Join oemof with your own approach or project

Oemof is designed as a framework and there is a lot of space for own ideas or own libraries. No matter if you want a heuristic solver library or different linear solver libraries. You may want to add tools to analyse the results or something we never heard of. You want to add a GUI or your application to be linked to. We think, that working together in one framework will increase the probability that somebody will use and test your code (see *Why should I contribute?*).

Interested? Together we can talk about how to transfer your ideas into oemof or even integrate your code. Maybe we just link to your project and try to adept the API for a better fit in the future.

Also consider joining our developer meetings which take place every 6 months (usually May and December). Just contact us!

CHAPTER 3

Installation

To install the *oemof.solph* package (previously just *oemof*), please use

```
pip install https://github.com/oemof/oemof-solph/archive/master.zip"
```

CHAPTER 4

Usage

Open Energy Modelling Framework - Python toolbox for energy system modelling and optimisation.

The omeof project aims to be a loose organisational frame for tools in the wide field of (energy) system modelling.

Current oemof libraries

- *oemof-solph*
- *oemof-thermal*
- *cydets*
- *demandlib*
- *feedinlib*
- *tespy*

4.1 oemof-solph

The solph library is designed to create and solve linear or mixed-integer linear optimization problems. It is based on optimization modelling language pyomo.

To use solph at least one linear solver has to be installed on your system. Solph is tested with the open source solver *cbc* and the *gurobi* solver (free for academic use). The open *glpk* solver recently showed some odd behaviour, but it should work for small systems.

The formulation of the energy system is based on the oemof-network library but contains additional components such as storages. Furthermore the network class are enhanced with additional parameters such as efficiencies, bounds, cost and more. See the API documentation for more details. Try the examples to learn how to build a linear energy system.

4.2 oemof-thermal

Coming soon...

4.3 cydets

Cycle Detection in Time Series (CyDeTS). An algorithm to detect cycles in times series along with their respective depth-of-cycle (DoC) and duration.

4.4 demandlib

The demandlib library can be used to create load profiles for elctricity and heat knowing the annual demand. See the documentation of the demandlib for examples and a full description of the library.

4.5 feedinlib

The feedinlib library serves as an interface between Open Data weather data and libraries to calculate feedin timeseries for fluctuating renewable energy sources.

It is currently under revision (see here for further information). To begin with it will provide an interface to the pvlib and windpowerlib and functions to download MERRA2 weather data and open_FRED weather data. See documentation of the feedinlib for a full description of the library.

4.6 tespy

Thermal Engineering Systems in Python (TESPy) allows you to calculate stationary operation in order to design the process of thermal energy systems. From that point it is possible to simulate the offdesign behavior of your plant using underlying characteristics for each of the plants components. The package inlcudes basic components, such as turbines, pumps, compressors, heat exchangers, pipes, mixers and splitters as well as some advanced components (e. g. including combustion, derivatives of heat exchangers, drum).

CHAPTER 5

Reference

5.1 oemof

CHAPTER 6

Developing oemof

Oemof is developed collaboratively and therefore driven by its community. While advancing as a user of oemof, you may find situations that demand solutions that are not readily available. In this case, your solution may be of help to other users as well. Contributing to the development of oemof is good for two reasons: Your code may help others and you increase the quality of your code through the review of other developers. Read also these arguments on why you should contribute.

A first step to get involved with development can be contributing a component that is not part of the current version that you defined for your energy system. We have a module oemof.solph.custom that is dedicated to collect custom components created by users. Feel free to start a pull request and contribute.

Another way to join the developers and learn how to contribute is to help improve the documentation. If you find that some part could be more clear or if you even find a mistake, please consider fixing it and creating a pull request.

New developments that provide new functionality may enter oemof at different locations. Please feel free to discuss contributions by creating a pull request or an issue.

In the following you find important notes for developing oemof and elements within the framework. On whatever level you may want to start, we highly encourage you to contribute to the further development of oemof. If you want to collaborate see description below or contact us.

6.1 Install the developer version

To avoid problems make sure you have fully uninstalled previous versions of oemof. It is highly recommended to use a virtual environment. See this virtualenv tutorial for more help. Afterwards you have to clone the repository. See the github documentation to learn how to clone a repository. Now you can install the cloned repository using pip:

```
pip install -e /path/to/the/repository
```

Newly added required packages (via PyPi) can be installed by performing a manual upgrade of oemof. In that case run:

```
pip install --upgrade -e /path/to/the/repository
```

6.2 Contribute to the documentation

If you want to contribute by improving the documentation (typos, grammar, comprehensibility), please use the developer version of the dev branch at readthedocs.org. Every fixed typo helps.

6.3 Contribute to new components

You can develop a new component according to your needs. Therefore you can use the module oemof.solph.custom which collects custom components created by users and lowers the entry barrier for contributing.

Your code should fit to the *Issue-Management* and the docstring should be complete and hold the equations used in the constraints. But there are several steps you do not necessarily need to fulfill when contributing to oemof.solph.custom: you do not need to meet the *Naming Conventions*. Also compatiblity to the results-API must not be guaranteed. Further you do not need to test your components or adapt the documentation. These steps are all necessary once your custom component becomes a constant part of oemof (oemof.solph.components) and are described here: *Generally the following steps are required when changing, adding or removing code*. But in the first step have a look at existing custom components created by other users in oemof.solph.custom and easily create your own if you need.

6.4 Collaboration with pull requests

To collaborate use the pull request functionality of github as described here: https://guides.github.com/activities/hello-world/

6.4.1 How to create a pull request

- Fork the oemof repository to your own github account.
- Change, add or remove code.

- Commit your changes.
- Create a pull request and describe what you will do and why. Please use the pull request template we offer. It will be shown to you when you click on "New pull request".
- Wait for approval.

6.4.2 Generally the following steps are required when changing, adding or removing code

- Read the *Issue-Management* and *Naming Conventions* and follow them
- Add new tests according to what you have done
- Add/change the documentation (new feature, API changes . . .)
- Add a whatsnew entry and your name to Contributors
- Check if all *Tests* still work.

6.5 Tests

Run the following test before pushing a successful merge.

```
nosetests -w "/path/to/oemof" --with-doctest
```

6.6 Issue-Management

A good way for communication with the developer group are issues. If you find a bug, want to contribute an enhancement or have a question on a specific problem in development you want to discuss, please create an issue:

- describing your point accurately
- using the list of category tags
- addressing other developers

If you want to address other developers you can use @name-of-developer, or use e.g. @oemof-solph to address a team. Here you can find an overview over existing teams on different subjects and their members.

Look at the existing issues to get an idea on the usage of issues.

6.7 Style guidelines

We mostly follow standard guidelines instead of developing own rules. So if anything is not defined in this section, search for a PEP rule and follow it.

6.7.1 Docstrings

We decided to use the style of the numpydoc docstrings. See the following link for an example.

6.7.2 Code commenting

Code comments are block and inline comments in the source code. They can help to understand the code and should be utilized "as much as necessary, as little as possible". When writing comments follow the PEP 0008 style guide: https://www.python.org/dev/peps/pep-0008/#comments.

6.7.3 PEP8 (Python Style Guide)

- We adhere to PEP8 for any code produced in the framework.
- We use pylint to check your code. Pylint is integrated in many IDEs and Editors. Check here or ask the maintainer of your IDE or Editor
- Some IDEs have pep8 checkers, which are very helpful, especially for python beginners.

6.7.4 Quoted strings

As there is no recommendation in the PEP rules we use double quotes for strings read by humans such as logging/error messages and single quotes for internal strings such as keys and column names. However one can deviate from this rules if the string contains a double or single quote to avoid escape characters. According to PEP 257 and numpydoc we use three double quotes for docstrings.

```
logging.info("We use double quotes for messages")

my_dictionary.get('key_string')

logging.warning('Use three " to quote docstrings!'  # exception to avoid escape␣
↪characters
```

6.8 Naming Conventions

- We use plural in the code for modules if there is possibly more than one child class (e.g. import transformers AND NOT transformer). If there are arrays in the code that contain multiple elements they have to be named in plural (e.g. *transformers = [T1, T2,...]*).
- Please, follow the naming conventions of pylint
- Use talking names
  - Variables/Objects: Name it after the data they describe (power_line, wind_speed)
  - Functions/Method: Name it after what they do: **use verbs** (get_wind_speed, set_parameter)

6.9 Using git

6.9.1 Branching model

So far we adhere mostly to the git branching model by Vincent Driessen.

Differences are:

- instead of the name `origin/develop` we call the branch `origin/dev`.

- feature branches are named like `features/*`
- release branches are named like `releases/*`

6.9.2 Commit message

Use this nice little commit tutorial to learn how to write a nice commit message.

6.10 Documentation

The general implementation-independent documentation such as installation guide, flow charts, and mathematical models is done via ReStructuredText (rst). The files can be found in the folder */oemof/doc*. For further information on restructured text see: http://docutils.sourceforge.net/rst.html.

6.11 How to become a member of oemof

And last but not least, here you will find all information about how to become a member of the oemof organisation and of developer teams.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

7.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.2 Documentation improvements

oemof could always use more documentation, whether as part of the official oemof docs, in docstrings, or even on the web in blog posts, articles, and such.

7.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/oemof/oemof/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

7.4 Development

To set up *oemof* for local development:

1. Fork oemof (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:oemof/oemof.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

7.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

7.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 8

Authors

- oemof developer group - https://oemof.org

CHAPTER 9

Changelog

9.1 0.4.0.beta0 (2020-04-04)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

O