demandlib

Release 0.1.9

Apr 05, 2022

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Creating heat and power demand profiles from annual values.

- Free software: MIT license

1.1 Installation

```
pip install demandlib
```

You can also install the in-development version with:

```
pip install https://github.com/oemof/demandlib/archive/master.zip
```

1.2 Documentation

https://demandlib.readthedocs.io/

1.3 Development

To run all the 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

Installation

At the command line:

```
pip install demandlib
```

CHAPTER 3

BDEW Load Profiles

Using the demandlib you can create heat and electrical profiles by scaling the BDEW profiles to your desired annual demand. The BDEW profiles are the standard load profiles from BDEW.

3.1 Heat Profiles

3.1.1 Description

Heat profiles are created according to the approach described in the BDEW guideline.

The method was originally established in this PhD Thesis at TU Munich.

The approach for generating heat demand profiles is described in section 4.1 (Synthetic load profile approach).

$$Q_{day}(\theta) = KW \cdot h(\theta) \cdot F \cdot SF$$

KW: Kundenwert (customer value). Daily consumption of customer at $\approx 8°C$, depending on SLP type and Temperature timeseries.
h: h-Wert (h-value) , depending on SLP type and daily mean temperature.
F: Wochentagsfaktor (week day factor), depending on SLP type and day of the week.
T: Daily mean temperature 2 meters above the ground (simple mean or "geometric series", which means a weighted sum over the previous days).
SF: Stundenfaktor (hour factor)

The geometric series approach is meant to account for thermal inertia.

$$\theta = \frac{T_t + 0.5 \cdot T_{t-1} + 0.25 \cdot T_{t-2} + 0.125 \cdot T_{t-3}}{1 + 0.5 + 0.25 + 0.125}$$

Depending on the profile type, different coefficients A, B, C, D for the sigmoid function are used.

$$h(\theta) = \frac{A}{1 + (\frac{B}{\theta - \theta_0})^C} + D$$

$$\theta_0 = 40°C$$

Types of houses:

EFH: Einfamilienhaus (single family house)
MFH: Mehrfamilienhaus (multi family house)
GMK: Metall und Kfz (metal and automotive)
GHA: Einzel- und Großhandel (retail and wholesale)
GKO: Gebietskörperschaften, Kreditinstitute und Versicherungen (Local authorities, credit institutions and insurance companies)
GBD: sonstige betriebliche Dienstleistung (other operational services)
GGA: Gaststätten (restaurants)
GBH: Beherbergung (accommodation)
GWA: Wäschereien, chemische Reinigungen (laundries, dry cleaning)
GGB: Gartenbau (horticulture)
GBA: Backstube (bakery)
GPD: Papier und Druck (paper and printing)
GMF: haushaltsähnliche Gewerbebetriebe (household-like business enterprises)
GHD: Summenlastprofil Gewerbe/Handel/Dienstleistungen (Total load profile Business/Commerce/Services)

Building class:

The parameter `building_class` (German: Baualtersklasse) can assume values in the range 1-11. On pages 42-43 of this document you will find guidance on how to determine `building_class`. You can either use the building class according to table 3 in the linked document or determine `building_class` on your own by identifying the proportion of buildings with a building age prior to 1978 and a building age of 1979 or later in the total building stock and by matching the proportion prior to 1978 with the span given in the column "Altbauanteil" in table 2 of the linked document.

3.1.2 Usage

```
from demandlib import bdew

...
```

3.2 Electrical Profiles

3.2.1 Description

The electrical profiles are the standard load profiles from BDEW. All profiles have a resolution of 15 minutes. They are based on measurements in the German electricity sector. There is a dynamic function (h0_dyn) for the houshold

(h0) profile that better takes the seasonal variance into account [BDEW].

$$F_t = -3,92 \cdot 10^{-10} \cdot t^4 + 3,2 \cdot 10^{-7} \cdot t^3 \backsim 7,02 \cdot 10^{-5} \cdot t^2 + 2,1 \cdot 10^{-3} \cdot t + 1,24$$

With *t* the day of the year as a decimal number.

The following profile types are available. Be aware that the types in Python code are strings in **lowercase**.

Table 1: German (original) [Wikipedia]

Typ	Beschreibung	Erläuterung
G0	Gewerbe allgemein	Gewogener Mittelwert der Profile G1-G6
G1	Gewerbe werktags 8–18 Uhr	z.B. Büros, Arztpraxen, Werkstätten, Verwaltungseinrichtungen
G2	Gewerbe mit starkem bis überwiegendem Verbrauch in den Abendstunden	z.B. Sportvereine, Fitnessstudios, Abendgaststätten
G3	Gewerbe durchlaufend	z.B. Kühlhäuser, Pumpen, Kläranlagen
G4	Laden/Friseur	
G5	Bäckerei mit Backstube	
G6	Wochenendbetrieb	z.B. Kinos
G7	Mobilfunksendestation	durchgängiges Bandlastprofil
L0	Landwirtschaftsbetriebe allgemein	Gewogener Mittelwert der Profile L1 und L2
L1	Landwirtschaftsbetriebe mit Milchwirtschaft/Nebenerwerbs-Tierzucht	
L2	Übrige Landwirtschaftsbetriebe	
H0/H0_dyn	Haushalt/Haushalt dynamisiert	

Table 2: British English (translation)

type	description	explanation
G0	General trade/business/commerce	Weighted average of profiles G1-G6
G1	Business on weekdays 8 a.m. - 6 p.m.	e.g. offices, doctors' surgeries, workshops, administrative facilities
G2	Businesses with heavy to predominant consumption in the evening hours	e.g. sports clubs, fitness studios, evening restaurants
G3	Continuous business	e.g. cold stores, pumps, sewage treatment plants
G4	Shop/barber shop	
G5	Bakery with bakery	
G6	Weekend operation	e.g. cinemas
G7	Mobile phone transmitter station	continuous band load profile
L0	General farms	Weighted average of profiles L1 and L2
L1	Farms with dairy farming/part-time livestock farming	
L2	Other farms	
H0/H0_dyn	Household/dynamic houshold	

Further information in German language is available at the BDEW.

3.2.2 Usage

```
from demandlib import bdew
e_slp = bdew.ElecSlp(year=2020)
```

(continues on next page)

(continued from previous page)

```
# get all available types
print(e_slp.get_profiles().columns)

# get the "h0" and "g0" profile
profiles = e_slp.get_profiles("h0", "g0")

# get scaled profiles
scaled_profiles = e_slp.get_scaled_profiles({"h0": 3000, "g0": 5000})

# get scaled profiles with power values instead of energy values
# a conversion_factor of 4 will convert Wh, kWh etc. to W, kW
e_slp.get_scaled_power_profiles({"h0": 3000, "g0": 5000}, conversion_factor=4)

# add holidays, holidays are treated as Sundays
holidays = {
    datetime.date(2010, 1, 1): "New year",
    datetime.date(2010, 10, 3): "Day of German Unity",
}
e_slp = bdew.ElecSlp(year=2010, holidays=holidays)

# holiday dictionaries can be created using workalendar
# https://github.com/workalendar/workalendar
```

CHAPTER 4

Further Profiles

We implemented further profiles (one until now) to represent further demand sectors which are not covered by the BDEW load profiles.

4.1 Industrial Electrical Profile

4.1.1 Description

The industrial electrical profile uses a step function.

4.1.2 Usage

CHAPTER 5

Reference

Implementation of the standard load profiles

class demandlib.bdew.elec_slp.**ElecSlp**(*year*, *seasons=None*, *holidays=None*)

Bases: object

Generate electrical standardized load profiles based on the BDEW method.

Parameters

- **year** (*integer*) – Year of the demand series.
- **Optional Parameters**
- **————————-**
- **seasons** (*dictionary*) – Describing the time ranges for summer, winter and transition periods. The seasons dictionary will update the existing one, so only changed keys have to be defined. Make sure not to create time gaps. The "h0_dyn" will not work with changed seasons, so you have to use your own smoothing curve to create a "h0_dyn" profile.
- **holidays** (*dictionary or list*) – The keys of the dictionary or the items of the list should be datetime objects of the days that are holidays.

all_load_profiles(*time_df*, *holidays=None*)

create_bdew_load_profiles(*dt_index*, *slp_types*, *holidays=None*)

Calculates the hourly electricity load profile in MWh/h of a region.

create_dynamic_h0_profile()

Use the dynamisation function of the BDEW to smoothen the seasonal edges. Functions resolution is daily.

$$F_t = -3,92 \cdot 10^{-10} \cdot t^4 + 3,2 \cdot 10^{-7} \cdot t^{3\smile} 7,02 \cdot 10^{-5} \cdot t^2 + 2,1 \cdot 10^{-3} \cdot t + 1,24$$

With *t* the day of the year as a decimal number.

Adjustment of accuracy: from -3,92 to -3.916649251

date_time_index

get_profile(*ann_el_demand_per_sector*)

DEPRECATED: Use *get_scaled_power_profiles()* instead

Parameters **ann_el_demand_per_sector** (*dictionary*) – Key: sector, value: annual value

Returns **pandas.DataFrame** (*Table with all profiles*)

get_profiles(**args*)

Get all or the selected profiles. To select profiles you can pass the name of the types as strings. The profiles are normalised to 1.

Try *print(get_profiles().columns* to get all valid types.

Returns **pandas.DataFrame** (*Table with all or the selected profiles.*)

Examples

```
>>> from demandlib import bdew
>>> e_slp = bdew.ElecSlp(year=2020)
>>> ", ".join(sorted(e_slp.get_profiles().columns))
'g0, g1, g2, g3, g4, g5, g6, h0, h0_dyn, l0, l1, l2'
>>> e_slp.get_profiles("h0", "g0").head()
                           h0        g0
2020-01-01 00:00:00  0.000017  0.000016
2020-01-01 00:15:00  0.000015  0.000015
2020-01-01 00:30:00  0.000014  0.000015
2020-01-01 00:45:00  0.000012  0.000014
2020-01-01 01:00:00  0.000012  0.000013
```

```
>>> e_slp.get_profiles("h0", "g0").sum()
h0    1.0
g0    1.0
dtype: float64
```

get_scaled_power_profiles(*ann_el_demand_per_sector*, *conversion_factor=4*)

Get profiles scaled by there annual value. Each value represents the average power of an interval. Therefore, it is not possible to sum up the array. A conversion factor is used to calculate power units from energy units. By default the conversion factor is *4*. As the interval of each profile is 15 minutes a conversion factor of *4* will convert energy units like Wh, kWh, MWh etc. to power units like W, kW, MW etc..

Parameters

- **ann_el_demand_per_sector** (*dict*) – The annual demand in an energy unit for each type.
- **conversion_factor** (*float*) – Factor to convert the energy unit of the annual value to the power unit of each interval.

Returns **pandas.DataFrame** (*Table with scaled profiles.*)

Examples

```
>>> from demandlib import bdew
>>> e_slp = bdew.ElecSlp(year=2020)
>>> e_slp.get_scaled_power_profiles({"h0": 3000, "g0": 5000}).head()
                           g0        h0
```

(continues on next page)

(continued from previous page)

```
2020-01-01 00:00:00  0.320338  0.202627
2020-01-01 00:15:00  0.305866  0.182365
2020-01-01 00:30:00  0.291590  0.164500
2020-01-01 00:45:00  0.278682  0.149633
2020-01-01 01:00:00  0.268122  0.138602
>>> cf = 4
>>> spp = e_slp.get_scaled_power_profiles({"h0": 3000, "g0": 5000},
...                                       conversion_factor=cf)
>>> spp.sum()
g0    20000.0
h0    12000.0
dtype: float64
>>> spp.div(cf).sum()
g0    5000.0
h0    3000.0
dtype: float64
```

get_scaled_profiles(*ann_el_demand_per_sector*)

Get profiles scaled by there annual value.

Parameters **ann_el_demand_per_sector** (*dict*) – The annual demand in an energy unit for each type.

Returns **pandas.DataFrame** (*Table with scaled profiles.*)

Examples

```
>>> from demandlib import bdew
>>> e_slp = bdew.ElecSlp(year=2020)
>>> e_slp.get_scaled_profiles({"h0": 3000, "g0": 5000}).head()
                           g0        h0
2020-01-01 00:00:00  0.080084  0.050657
2020-01-01 00:15:00  0.076466  0.045591
2020-01-01 00:30:00  0.072897  0.041125
2020-01-01 00:45:00  0.069671  0.037408
2020-01-01 01:00:00  0.067030  0.034650
```

```
>>> e_slp.get_scaled_profiles({"h0": 3000, "g0": 5000}).sum()
g0    5000.0
h0    3000.0
dtype: float64
```

Implementation of the bdew heat load profiles

class demandlib.bdew.heat_building.**HeatBuilding**(*df_index*, ***kwargs*)

Bases: object

Parameters **year** (*int*) – year for which the profile is created

Variables

- **datapath** (*string*) – path to the bdew basic data files (csv)
- **temperature** (*pandas.Series*) – Series containing hourly temperature data
- **annual_heat_demand** (*float*) – annual heat demand of building in kWh

- **building_class** (*int*) – class of building according to bdew classification possible numbers are: 1 - 11
- **shlp_type** (*string*) – type of standardized heat load profile according to bdew possible types are: GMF, GPD, GHD, GWA, GGB, EFH, GKO, MFH, GBD, GBA, GMK, GBH, GGA, GHA
- **wind_class** (*int*) – wind classification for building location (0=not windy or 1=windy)
- **ww_incl** (*boolean*) – decider whether warm water load is included in the heat load profile

get_bdew_profile()
Calculation of the hourly heat demand using the bdew-equations

get_normalized_bdew_profile()
Calculation of the normalized hourly heat demand

get_sf_values(*filename='shlp_hour_factors.csv'*)
Determine the h-values

Parameters **filename** (*string*) – name of file where sigmoid factors are stored

get_sigmoid_parameters(*filename='shlp_sigmoid_factors.csv'*)
Retrieve the sigmoid parameters from csv-files

Parameters **filename** (*string*) – name of file where sigmoid factors are stored

get_temperature_interval()
Appoints the corresponding temperature interval to each temperature in the temperature vector.

get_weekday_parameters(*filename='shlp_weekday_factors.csv'*)
Retrieve the weekday parameter from csv-file

Parameters **filename** (*string*) – name of file where sigmoid factors are stored

weighted_temperature(*how='geometric_series'*)
A new temperature vector is generated containing a multi-day average temperature as needed in the load profile function.

Parameters **how** (*string*) – string which type to return ("geometric_series" or "mean")

Notes

Equation for the mathematical series of the average temperature[1]:

$$T = \frac{T_D + 0.5 \cdot T_{D-1} + 0.25 \cdot T_{D-2} + 0.125 \cdot T_{D-3}}{1 + 0.5 + 0.25 + 0.125}$$

with T_D **= Average temperature on the present day** T_{D-i} = Average temperature on the day - i

References

Implementation of the bdew standard load profiles for electric power.

class demandlib.particular_profiles.**IndustrialLoadProfile**(*dt_index*, *holidays=None*)

Bases: object

Generate an industrial heat or electric load profile.

[1] BDEW, # noqa: E501 BDEW Documentation for heat profiles.

simple_profile(*annual_demand*, ***kwargs*)
Create industrial load profile

Parameters **annual_demand** (*float*) – Total demand.

Other Parameters

- **am** (*datetime.time*) – beginning of workday
- **pm** (*datetime.time*) – end of workday
- **week** (*list*) – list of weekdays
- **weekend** (*list*) – list of weekend days
- **profile_factors** (*dictionary*) – dictionary with scaling factors for night and day of weekdays and weekend days

CHAPTER 6

Contributing

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

6.1 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.

6.2 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.

6.3 Feature requests and feedback

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

6.4 Development

To set up *demandlib* for local development:

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

```
git clone git@github.com:YOURGITHUBNAME/demandlib.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.

6.4.1 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`).
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`.

6.4.2 Tips

To run a subset of tests:

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

To run all the test environments in *parallel*:

```
tox -p auto
```

CHAPTER 7

Authors

(alphabetic order)

- Birgit Schachler
- Caroline Möller
- Guido Plessmann
- Hendrik Huyskens
- Jann Launer
- Patrik Schönfeldt
- Pyosch
- Steffen Wehkamp
- Stephen Bosch
- Uwe Krien

CHAPTER 8

Changelog

8.1 v0.1.9 (2021-?-?)

8.1.1 New features

-

8.1.2 Bug fixes

-

8.1.3 Other changes

-

8.2 v0.1.8 (2021-01-27)

8.2.1 Bug fixes

- FutureWarning for “dyn_function_h0” was raised instead of printed

8.3 v0.1.7 (2021-01-27)

8.3.1 New features

- Add dynamic h0 profile calculation (The implementation is not optimised for performance. Thus, it is not used by default.)

8.3.2 Bug fixes

- Fix improper use of pandas.dataframe.merge (demandlib will now work with pandas>=1.2)

8.3.3 Other changes

- Update deprecated pd.datetime to datetime.datetime
- Add (integration) tests and coverage as CI
- Split BDEW profile generation into submodules

8.4 v0.1.6 (2019-01-30)

8.4.1 General

- Update requirements
- Fix typos

8.5 v0.1.5 (2018-09-05)

8.5.1 New features

- Add function *get_normalized_bdew_profile(self)* to get a normalised profile. You could also use an annual_demand of one to get the same results.

8.5.2 Bug fixes

- Fix y-label of the heat example plot.

8.5.3 Other changes

- Make matplotlib optional in examples.

8.6 v0.1.4 (2018-05-30)

8.6.1 Code

- fix temperature bug
- fix Code style

8.6.2 Documentation

- Documentation improvements.

8.7 v0.1.1 (2016-11-30)

8.7.1 New features

- Examples callable by command-line script

8.7.2 Bug fixes

- Path specs when installed via pip

8.7.3 Other changes

- Fix versioning

8.8 0.1.0 (2016-10-04)

8.8.1 New features

- Implementation of BDEW synthetic load profiles
- Synthetic load profiles for heating sector
- Self-made industry demand profile similar to BDEW profiles

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index