oemof.thermal documentation

oemof developer group

Jun 12, 2023

Getting started

CHAPTER 1

Getting started

oemof.thermal is an oemof library with a focus on thermal energy technologies (heating/cooling). In its original intention it is an extension to the components of the optimization framework oemof.solph. However, some of its functions may be useful for their own.

oemof.thermal is organized like this:

For each technology that is covered, there is a module which holds a collection of useful functions. These functions can be applied to perform pre-calculations of an optimization model or postprocess optimization results. Besides, they may equally well be used stand-alone (totally independent from optimization).

To help setting up more detailed components in a simple way, oemof.thermal provides facades based on the oemof.tabular.facades module. Facades are classes that offer a simpler interface to more complex classes. More specifically, the `Facade` s in this module inherit from *oemof.solph*'s generic classes to serve as more concrete and energy specific interface. The concept of the facades has been derived from oemof.tabular. The idea is to be able to instantiate a `Facade` using only keyword arguments. Under the hood the `Facade` then uses these arguments to construct an *oemof.solph* component and sets it up to be easily used in an `EnergySystem`. Usually, a subset of the attributes of the parent class remains while another part can be addressed by more specific or simpler attributes. In oemof.thermal, some of the technologies have a facade class that can be found in the module oemof.thermal.facades. See the *api reference for the facade module* for further information on the structure of these classes.

For each module, there is a page that explains the scope of the module and its underlying concept. Mathematical symbols for commonly used variables and their names in the code are presented in overview tables. The usage of the functions and some sample results are given. Lastly, notable references to the literature are listed that the reader can refer to if she wants to get more information on the background.

Finally, there are a couple of examples that can give an idea of how the functionality of oemof.thermal can be utilized. Some models have undergone validation whose results you'll find in the section "Model validation".

Contents

1.1 Using oemof.thermal

1.1.1 Installation

Install oemof.thermal from pypi:

```
pip install oemof.thermal
```

1.1.2 Installing the latest (dev) version

Clone oemof.thermal from github:

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

Now you can install your local version of oemof.thermal using pip:

```
pip install -e <path/to/oemof-thermal/root/dir>
```

1.1.3 Examples

We provide examples described in the section *Examples*. Further we developed some complex models with the oemof-thermal components which are described in this section as well.

1.2 Contributing to oemof.thermal

Contributions are welcome. You can write issues to announce bugs or errors or to propose enhancements. Or you can contribute a new approach that helps to model thermal energy systems. If you want to contribute, fork the project at github, develop your features on a new branch and finally open a pull request to merge your contribution to oemof.thermal.

As oemof.thermal is part of the oemof developer group we use the same developer rules, described here.

CHAPTER 2

Examples

In this section we provide several examples to demonstrate how you can use the functions and components of *oemof-thermal*. You can find them in the example folder of the repository. Among them are the following calculations:

- Functionality of solar thermal and concentrating solar collector's facade and efficiency calculation
- Calculation of maximum possible heat output of heat pumps
- Investment decision on thermal storage and capacity

Most of the examples show the usage of *oemof-thermal* together with *oemof-solph*. However, *oemof-thermal* is a stand-alone package and you can use the package and its calculations in any other context as well.

2.1 List of available examples

Compression heat pump and chiller

An example provides an "how to" on the use of the 'calc_cops' function to get the coefficients of performance (COP) of an exemplary air-source heat pump (ASHP). It also shows how to use the pre-calculated COPs in a solph.Transformer. Furthermore, the maximal possible heat output of the heat pump is pre-calculated and varies with the temperature levels of the heat reservoirs. In the example the ambient air is used as low temperature heat reservoir.

In addition to that, the example provides a manual on using the 'calc_cops' function to get the COPs of a heat pump, by plotting the temperature dependency of the COP, and COPs of an exemplary ground-source heat pump (GSHP) using the soil temperature as low temperature heat reservoir.

The Examples can be found here.

Absorption Chiller

The first example shows the behaviour of the coefficient of performance and heat flows such as the cooling capacity for different cooling water temperatures based on the characteristic equation method. The second example underlines the dependence of the temperature of the cooling water on the cooling capacity.

The Examples can be found here.

Concentrating solar power (CSP)

These examples shows the difference between the new approach of the oemof-thermal component and a fix efficiency. The collector's efficiency and irradiance can be calculated with two different loss methods. The examples also shows the functionalitiy of the ParabolicTroughCollector facade.

An application is presented which models a csp plant to meet an electrical demand. The plant itself consists of a parabolic trough collector field, a turbine, and a storage.

The Examples can be found here.

Solar thermal collector

In these examples the functionality of the solar thermal collector is shown. Once with a fixed collector size (aperture area), once with a fixed collector size using the facade and another time with a collector size to be invested. It also provides plots which can be called by the flat_plate_collector_example.py.

The Examples can be found here.

Stratified thermal storage

These example explain how to use the functions of oemof-thermal's stratified thermal storage module to specify a storage in a model that optimizes operation with oemof-solph. Further it is shown how to use the facade class StratifiedThermalStorage.

Furthermore the examples show how to invest into nominal_storage_capacity and capacity (charging/discharging power) with a fixed ratio and independently with no fixed ratio.

The Examples can be found here.

Cogeneration

We further provide an example on different emission allocation methods in cogeneration. This Example can be found here.

2.2 List of available models

In the GitHub organisation of the oemof_heat project you will find more complex models which use the components "solar_thermal_collector" and "concentrating_solar_power" from oemof_thermal.

Solar Cooling Model

The application models a cooling system for a building with a given cooling demand.

Desalination Model

The application models a desalination system with a given water demand.

CHAPTER 3

Absorption chiller

Calculations for absorption chillers based on the characteristic equation method.

3.1 Scope

This module was developed to provide cooling capacity and COP calculations based on temperatures for energy system optimizations with oemof.solph.

3.2 Concept

A characteristic equation model to describe the performance of absorption chillers.

The cooling capacity ($\dot{Q}_E$) is determined by a function of the characteristic temperature difference ($\Delta\Delta t'$) that combines the external mean temperatures of the heat exchangers.

Various approaches of the characteristic equation method exists. Here we use the approach described by Kühn and Ziegler [1]:

$$\Delta\Delta t' = t_G - a \cdot t_{AC} + e \cdot t_E$$

with the assumption

$$t_A = t_C = t_{AC}$$

where t is the external mean fluid temperature of the heat exchangers (G: Generator, AC: Absorber and Condenser, E: Evaporator) and a and e are characteristic parameters.

The cooling capacity ($\dot{Q}_E$) and the driving heat ($\dot{Q}_G$) can be expressed as linear functions of $\Delta\Delta t'$:

$$\dot{Q}_E = s_E \cdot \Delta\Delta t' + r_E$$

$$\dot{Q}_G = s_G \cdot \Delta\Delta t' + r_G$$

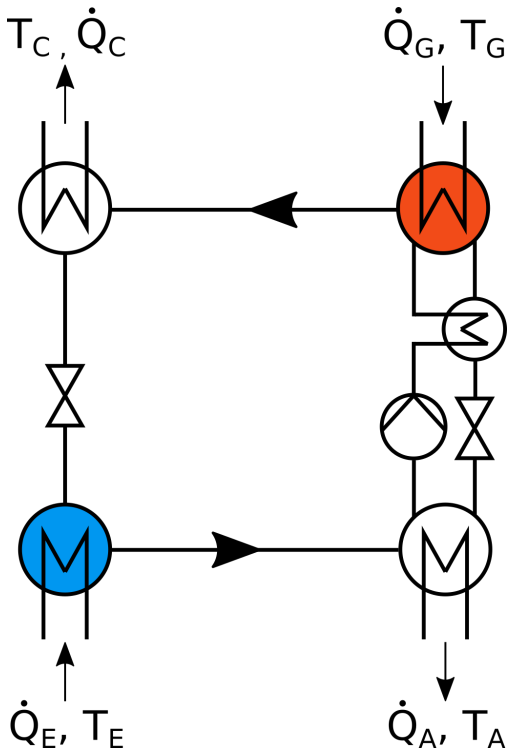


Fig. 1: Fig.1: Absorption cooling (or heating) process.

with the characteristic parameters s_E, r_E, s_G, and r_G.

The COP can then be calculated from $\dot{Q}_E$ and $\dot{Q}_G$:

$$COP = \frac{\dot{Q}_E}{\dot{Q}_G}$$

These arguments are used in the formulas of the function:

symbol	argument	explanation
$\Delta\Delta t'$	`ddt`	Characteristic temperature difference
t_G	`t_hot`	External mean fluid temperature of generator
t_{AC}	`t_cool`	External mean fluid temperature of absorber and condenser
t_E	`t_chill`	External mean fluid temperature of evaporator
a	`coef_a`	Characteristic parameter
e	`coef_e`	Characteristic parameter
s	`coef_s`	Characteristic parameter
r	`coef_r`	Characteristic parameter
$\dot{Q}$	`Q_dots`	Heat flux
$\dot{Q}_E$	`Q_dots_evap`	Cooling capacity (heat flux at evaporator)
$\dot{Q}_G$	`Q_dots_gen`	Driving heat (heat flux at generator)
COP	`COP`	Coefficient of performance

3.3 Usage

The following example shows how to calculate the COP of a small absorption chiller. The characteristic coefficients used in this examples belong to a 10 kW absorption chiller developed and tested at the Technische Universität Berlin [1].

```
import oemof.thermal.absorption_heatpumps_and_chillers as abs_chiller

# Characteristic temperature difference
ddt = abs_chiller.calc_characteristic_temp(
    t_hot=[85],  # in °C
    t_cool=[26],  # in °C
    t_chill=[15],  # in °C
    coef_a=10,
    coef_e=2.5,
    method='kuehn_and_ziegler')

# Cooling capacity
Q_dots_evap = abs_chiller.calc_heat_flux(
    ddts=ddt,
    coef_s=0.42,
    coef_r=0.9,
    method='kuehn_and_ziegler')

# Driving heat
Q_dots_gen = abs_chiller.calc_heat_flux(
    ddts=ddt,
    coef_s=0.51,
    coef_r=2,
    method='kuehn_and_ziegler')

COPs = Q_dots_evap / Q_dots_gen
```

Fig.2 illustrates how the cooling capacity and the COP of an absorption chiller (here the 10 kW absorption chiller mentioned above) depend on the cooling water temperature, i.e. the mean external fluid temperature at absorber and condenser.

You find the code that is behind Fig.2 in our examples: https://github.com/oemof/oemof-thermal/tree/master/examples

You can run the calculations for any other absorption heat pump or chiller by entering the specific parameters (a, e, s, r) belonging to that specific machine. The specific parameters are determined by a numerical fit of the four parameters with testing data or data from the fact sheet (technical specifications from the manufacturer) if temperatures for at least two points of operation are given. You find detailed information in the referenced papers.

This package comes with characteristic parameters for five absorption chillers. Four published by Puig-Arnavat et al. [3]: 'Rotartica', 'Safarik', 'Broad_01' and 'Broad_02' and one published by Kühn and Ziegler [1]: 'Kuehn'. If you like to contribute parameters for other machines, please feel free to contact us or to contribute directly via github.

To model one of the machines provided by this package you can adapt the code above in the following way.

```
import oemof.thermal.absorption_heatpumps_and_chillers as abs_chiller
import pandas as pd
import os

filename_charpara = os.path.join(os.path.dirname(__file__), 'data/characteristic_
↪parameters.csv')
charpara = pd.read_csv(filename_charpara)
```

(continues on next page)

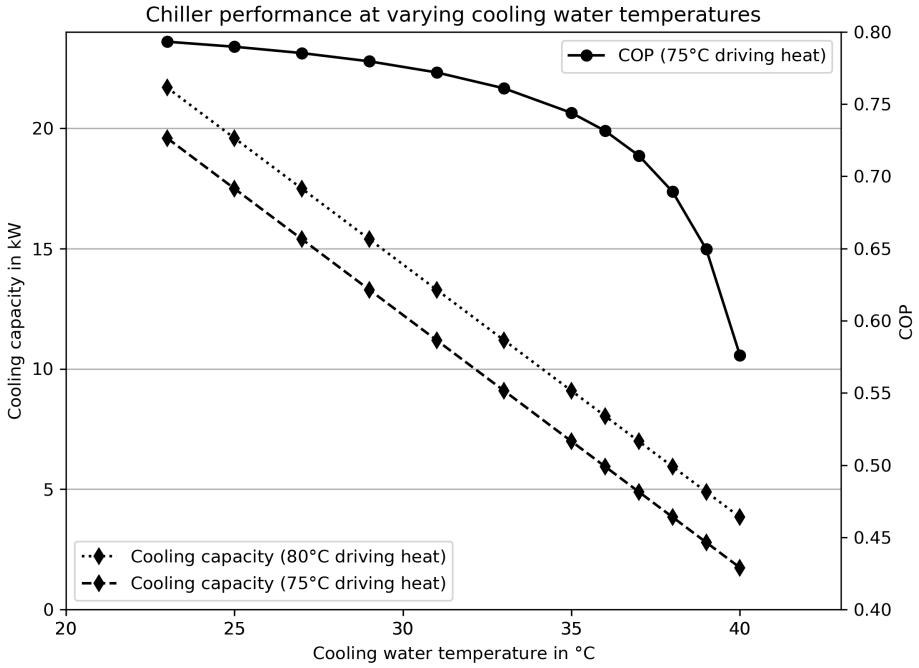


Fig. 2: Fig.2: Dependency of the cooling capacity and the COP of a 10 kW absorption chiller on the cooling water temperature.

(continued from previous page)

```
chiller_name = 'Kuehn'  # 'Rotartica', 'Safarik', 'Broad_01', 'Broad_02'

 # Characteristic temperature difference
ddt = abs_chiller.calc_characteristic_temp(
    t_hot=[85],  # in °C
    t_cool=[26],  # in °C
    t_chill=[15],  # in °C
    coef_a=charpara[(charpara['name'] == chiller_name)]['a'].values[0],
    coef_e=charpara[(charpara['name'] == chiller_name)]['e'].values[0],
    method='kuehn_and_ziegler')

# Cooling capacity
Q_dots_evap = abs_chiller.calc_heat_flux(
    ddts=ddt,
    coef_s=charpara[(charpara['name'] == chiller_name)]['s_E'].values[0],
    coef_r=charpara[(charpara['name'] == chiller_name)]['r_E'].values[0],
    method='kuehn_and_ziegler')

# Driving heat
Q_dots_gen = abs_chiller.calc_heat_flux(
    ddts=ddt,
    coef_s=charpara[(charpara['name'] == chiller_name)]['s_G'].values[0],
    coef_r=charpara[(charpara['name'] == chiller_name)]['r_G'].values[0],
    method='kuehn_and_ziegler')

COPs = [Qevap / Qgen for Qgen, Qevap in zip(Q_dots_gen, Q_dots_evap)]
```

You find information on the machines in [1], [2] and [3]. Please be aware that [2] introduces a slightly different approach (using an improved characteristic equation with $\Delta\Delta t''$ instead of $\Delta\Delta t'$). The characteristic parameters that we use are derived from [1] and therefore differ from those in [2].

3.4 References

[1] A. Kühn, F. Ziegler. "Operational results of a 10 kW absorption chiller and adaptation of the characteristic equation", Proc. of the 1st Int. Conf. Solar Air Conditioning, 6-7 October 2005, Bad Staffelstein, Germany.

[2] A. Kühn, C. Özgür-Popanda, and F. Ziegler. "A 10 kW indirectly fired absorption heat pump : Concepts for a reversible operation," in Thermally driven heat pumps for heating and cooling, Universitätsverlag der TU Berlin, 2013, pp. 173–184. [http://dx.doi.org/10.14279/depositonce-4872]

[3] Maria Puig-Arnavat, Jesús López-Villada, Joan Carles Bruno, Alberto Coronas. Analysis and parameter identification for characteristic equations of single- and double-effect absorption chillers by means of multivariable regression. In: International Journal of Refrigeration, 33 (2010) 70-78.

CHAPTER 4

Cogeneration

4.1 Scope

The module is designed to hold functions that are helpful when modeling components that generate more than one type of output.

4.2 Concept

Currently there are three different methods that can be used to allocate the emissions to the two outputs of a unit that produces electricity and heat.

IEA method

$EM_{el} = EM \cdot \frac{\eta_{el}}{\eta_{el}+\eta_{th}}$,

$EM_{th} = EM \cdot \frac{\eta_{th}}{\eta_{el}+\eta_{th}}$.

Efficiency method

$EM_{el} = EM \cdot \frac{\eta_{th}}{\eta_{el}+\eta_{th}}$,

$EM_{th} = EM \cdot \frac{\eta_{el}}{\eta_{el}+\eta_{th}}$.

Finnish method

$EM_{el} = EM \cdot (1 - PEE) \frac{\eta_{el}}{\eta_{el,REF}}$,

$EM_{th} = EM \cdot (1 - PEE) \frac{\eta_{th}}{\eta_{th,REF}}$,

with

$PEE = 1 - \frac{1}{\frac{\eta_{th}}{\eta_{th,ref}} + \frac{\eta_{el}}{\eta_{el,ref}}}$.

Reference: Mauch, W., Corradini, R., Wiesmeyer, K., Schwentzek, M. (2010). Allokationsmethoden für spezifische CO2-Emissionen von Strom und Waerme aus KWK-Anlagen. Energiewirtschaftliche Tagesfragen, 55(9), 12–14.

4.3 Usage

```
em_el, em_heat = allocate_emissions(
        total_emissions=200,  # in CO2 equivalents
        eta_el=0.3,
        eta_th=0.5,
        method=method,
        eta_el_ref=0.525,
        eta_th_ref=0.82
)
```

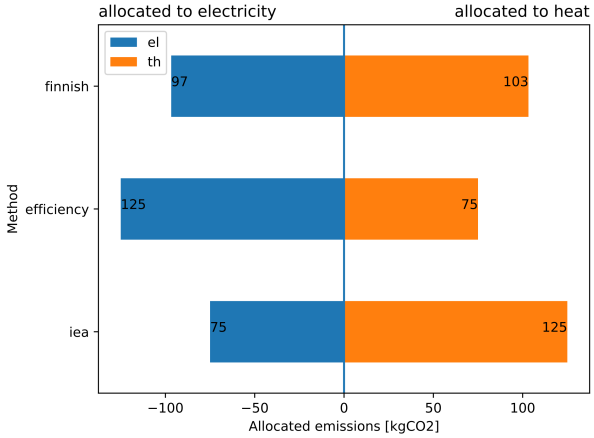


Fig. 1: Fig. 1: The above figure illustrates the allocation of emissions using the different allocation methods.

CHAPTER 5

Compression heat pump and chiller

Simple calculations for compression heat pumps and chillers.

5.1 Scope

This module was developed to provide COP calculations based on temperatures for energy system optimizations with oemof.solph.

A time series of pre-calculated COPs can be used as input for a transformer (an oemof.solph component) in an energy system optimization. Discover more possibilities to use this module with our examples: https://github.com/oemof/oemof-thermal/tree/dev/examples

5.2 Concept

Compression heat pumps and chillers increase the temperature of a flow using a compressor that consumes electric power. The inlet heat flux comes from a low temperature source (T_low) and the outlet has the temperature level of the high temperature sink (T_high). The same cycle can be used for heating (heat pump) or cooling (chiller).

The efficiency of the heat pump cycle process can be described by the Coefficient of Performance (COP). The COP describes the ratio of useful heat $\dot{Q}_{\mathrm{useful}}$ ($\dot{Q}_{\mathrm{in}}$ or $\dot{Q}_{\mathrm{out}}$) per electric work P_{el} consumed:

$$COP = \frac{\dot{Q}_{\mathrm{useful}}}{P_{\mathrm{el}}}$$

The Carnot efficiency COP_{Carnot} describes the maximum theoretical efficiency (ideal process). It depends on the temperature difference between source and sink:

$$COP_{\mathrm{Carnot,HP}} = \frac{T_{\mathrm{high}}}{T_{\mathrm{high}} - T_{\mathrm{low}}}$$

for heat pumps and

$$COP_{\mathrm{Carnot,chiller}} = \frac{T_{\mathrm{low}}}{T_{\mathrm{high}} - T_{\mathrm{low}}}$$

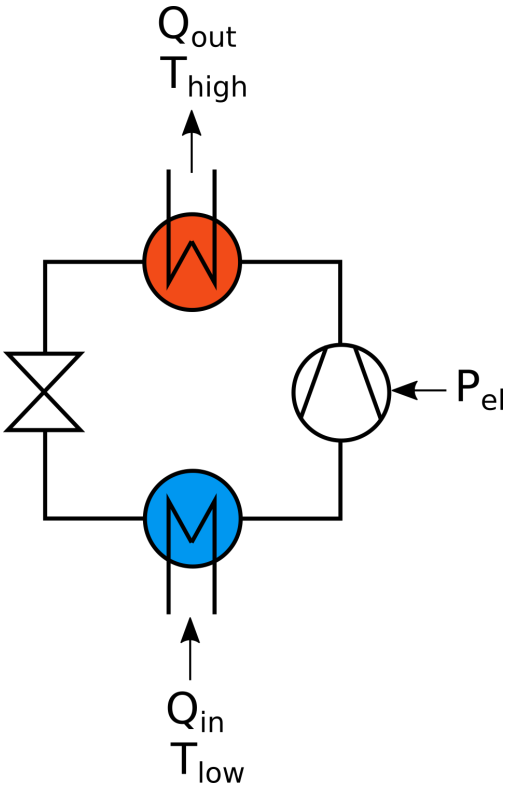


Fig. 1: Fig.1: The heat pump cycle and its two temperature levels.

for chillers.

To determine the COP of a real machine a scale-down factor (the quality grade η) is applied on the Carnot efficiency:

$$COP = \eta \cdot COP_{\text{Carnot}}$$

with

$$0 \leq \eta \leq 1$$

Typical values of quality grades are 0.4 for air-source heat pumps, 0.55 for ground-source ("brine-to-water") heat pumps using a ground heat exchanger, and 0.5 for heat pumps using groundwater as source.[1]

For high temperature heat pumps Arpagaus finds quality grades between 0.4 and 0.6.[2]

Fig.2 illustrates how the temperature difference affects the COP and how the choice of the quality grade allows to model different types of heat pumps.

5.3 Usage

These arguments are input to the functions:

[1] VDE ETG Energietechnik, VDE-Studie "Potenziale für Strom im Wärmemarkt bis 2050 - Wärmeversorgung in flexiblen Energieversorgungssystemen mit hohen Anteilen an erneuerbaren Energien". 2015. (http://www.energiedialog2050.de/BASE/DOWNLOADS/VDE_ST_ETG_Warmemarkt_RZ-web.pdf)

[2] C. Arpagaus, Hochtemperatur-Wärmepumpen - Marktübersicht, Stand der Technik und Anwendungsbeispiele. Berlin, Offenbach: VDE-Verlag, 2019.

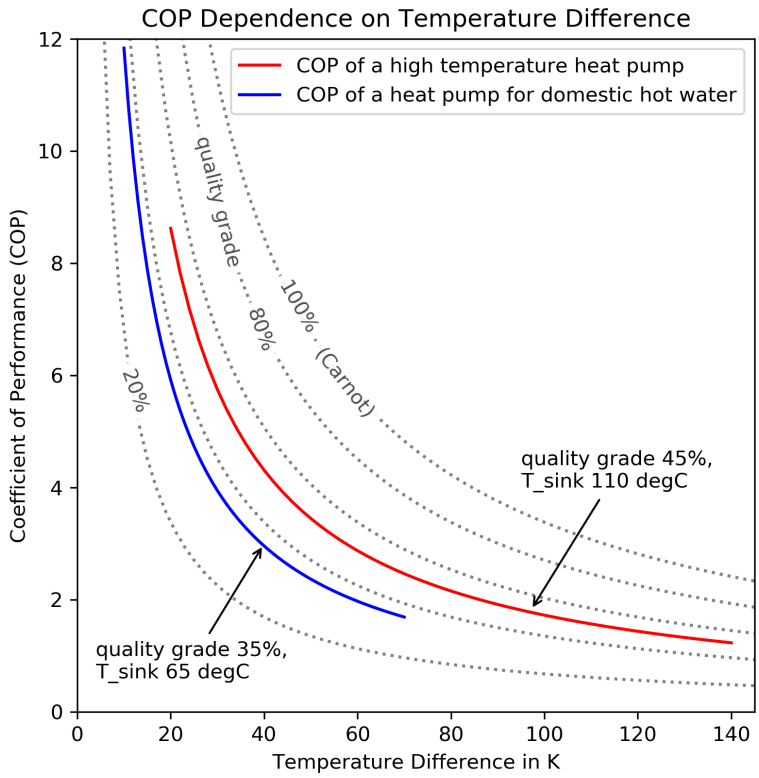


Fig. 2: Fig.2: COP dependence on temperature difference (Output of example *cop_dependence_on_temperature_difference.py*).

symbol	attribute	explanation
COP	`cop`	Coefficient of Performance
T_{high}	`temp_high`	Temperature of the high temp. heat reservoir
T_{low}	`temp_low`	Temperature of the low temp. heat reservoir
η	`quality_grade`	Quality grade
T_{icing}	`temp_threshold_icing`	Temperature below which icing occurs
f_{icing}	`factor_icing`	COP reduction caused by icing

The Coefficient of Performance (COP) is calculated using 'calc_cops()'

```
COP = calc_cops(temp_high,
                temp_low,
                quality_grade,
                temp_threshold_icing,
                factor_icing,
                mode)
```

mode='heat_pump'

$COP = \eta \cdot \frac{T_{\text{high}}}{T_{\text{high}} - T_{\text{low}}}$

$COP = f_{\text{icing}} \cdot \eta \cdot \frac{T_{\text{high}}}{T_{\text{high}} - T_{\text{low}}}$

mode='chiller'

$COP = \eta \cdot \frac{T_{\text{low}}}{T_{\text{high}} - T_{\text{low}}}$

The maximum cooling capacity can be calculated using 'calc_max_Q_dot_chill()'.

```
Q_dot_chill_max = calc_max_Q_dot_chill(nominal_conditions, cops)
```

$\dot{Q}_{\text{chilled,max}} = \frac{COP_{\text{actual}}}{COP_{\text{nominal}}}$

The maximum heating capacity can be calculated using 'calc_max_Q_dot_heat()'

```
Q_dot_heat_max = calc_max_Q_dot_heat(nominal_conditions, cops)
```

$\dot{Q}_{\text{hot,max}} = \frac{COP_{\text{actual}}}{COP_{\text{nominal}}}$

The quality grade at nominal point of operation can be calculated using 'calc_chiller_quality_grade()'

Do NOT use this function to determine the input for *calc_cops()*!

```
quality_grade = calc_chiller_quality_grade(nominal_conditions)
```

$\eta = \frac{\dot{Q}_{\text{chilled,nominal}}}{P_{\text{el}}} / \frac{T_{\text{low,nominal}}}{T_{\text{high,nominal}} - T_{\text{low,nominal}}}$

5.4 References

CHAPTER 6

Concentrating solar power

Module to calculate the usable heat of a parabolic trough collector

6.1 Scope

This module was developed to provide the heat of a parabolic trough collector based on temperatures and collectors location, tilt and azimuth for energy system optimizations with oemof.solph.

In https://github.com/oemof/oemof-thermal/tree/dev/examples you can find an example on how to use the modul to calculate a CSP power plant. A time series of pre-calculated heat flows can be used as input for a source (an oemof.solph component), and a transformer (an oemof.solph component) can be used to hold electrical power consumption and further thermal losses of the collector in an energy system optimization. In addition, you will find an example which compares this precalculation with a calculation using a constant efficiency.

6.2 Concept

The pre-calculations for the concentrating solar power calculate the heat of the solar collector based on the direct horizontal irradiance (DHI) or the direct normal irradiance (DNI) and information about the collector and its location. The losses can be calculated in 2 different ways.

The direct normal radiation (E_{dir}) is reduced by geometrical losses ($\dot{Q}_{loss,geom}$) so that only the collector radiation (E^*_{coll}) hits the collector. Before the thermal power is absorbed by the absorber tube, also optical losses ($\dot{Q}_{loss,opt}$), which can be reflection losses at the mirror, transmission losses at the cladding tube and absorption losses at the absorber tube, occur. The absorber finally loses a part of the absorbed heat output through thermal losses ($\dot{Q}_{loss,therm}$).

The processing of the irradiance data is done by the pvlib, which calculates the direct irradiance on the collector. This irradiance is reduced by dust and dirt on the collector with:

$$E_{coll} = E^*_{coll} \cdot X^{3/2}$$

The efficiency of the collector is calculated depending on the loss method with

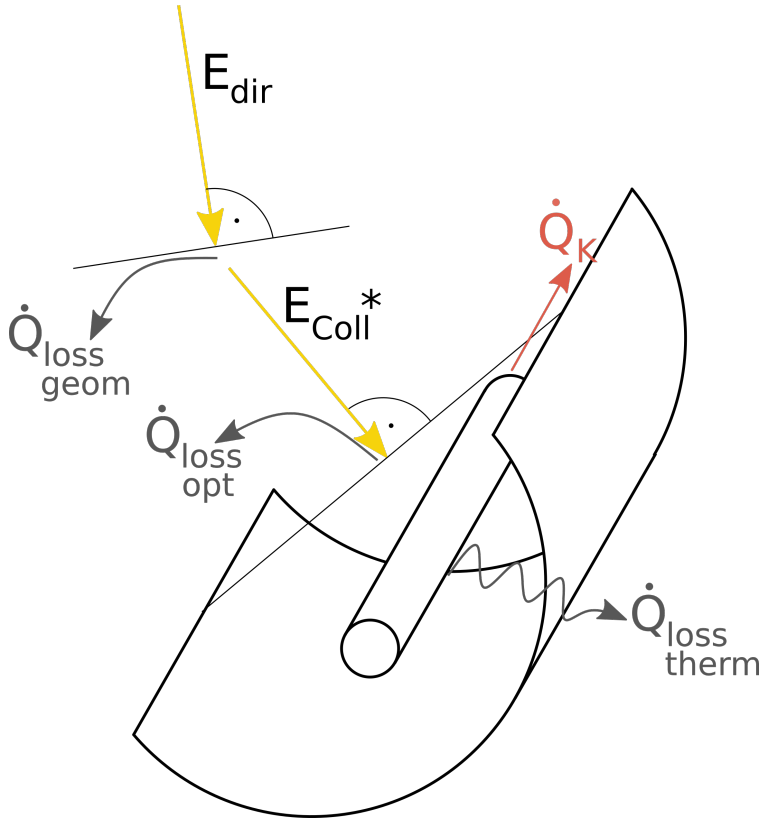


Fig. 1: Fig.1: The energy flows and losses at a parabolic trough collector.

method ‘Janotte’:

$\eta_C = \eta_0 \cdot \kappa(\Theta) - c_1 \cdot \frac{\Delta T}{E_{coll}} - c_2 \cdot \frac{\Delta T^2}{E_{coll}}$

method ‘Andasol’:

$\eta_C = \eta_0 \cdot \kappa(\Theta) - \frac{c_1}{E_{coll}}$

with the incident angle modifier, which is calculated depending on the loss method:

method ‘Janotte’:

$\kappa(\Theta) = 1 - a_1 \cdot |\Theta| - a_2 \cdot |\Theta|^2$

method ‘Andasol’:

$\kappa(\Theta) = 1 - a_1 \cdot |\Theta| - a_2 \cdot |\Theta|^2 - a_3 \cdot |\Theta|^3 - a_4 \cdot |\Theta|^4 - a_5 \cdot |\Theta|^5 - a_6 \cdot |\Theta|^6$

In the end, the irradiance on the collector is multiplied with the efficiency to get the collector’s heat.

$\dot{Q}_{coll} = E_{coll} \cdot \eta_C$

The three values $\dot{Q}_{coll}$, η_C and E_{coll} are returned. Losses which occur after the heat absorption in the collector (e.g. losses in pipes) have to be taken into account in a later step (see the example).

These arguments are used in the formulas of the function:

sym-bol	argument	explanation
E_{coll}	`collector_irradiance`	Irradiance on collector considering all losses including losses because of dirtiness
E^*_{coll}	`irradiance_on_collector`	Irradiance which hits collectors surface before losses because of dirtiness are considered
X	`cleanliness`	Cleanliness of the collector (between 0 and 1)
κ	`iam`	Incidence angle modifier
a_1	`a_1`	Parameter 1 for the incident angle modifier
a_2	`a_2`	Parameter 2 for the incident angle modifier
a_3	`a_3`	Parameter 3 for the incident angle modifier
a_4	`a_4`	Parameter 4 for the incident angle modifier
a_5	`a_5`	Parameter 5 for the incident angle modifier
a_6	`a_6`	Parameter 6 for the incident angle modifier
Θ	`aoi`	Angle of incidence
η_C	`eta_c`	Collector efficiency
c_1	`c_1`	Thermal loss parameter 1
c_2	`c_2`	Thermal loss parameter 2
ΔT	`delta_t`	Temperature difference (collector to ambience)
η_0	`eta_0`	Optical efficiency of the collector
$\dot{Q}_{coll}$	`collector_heat`	Collector’s heat

6.3 Usage

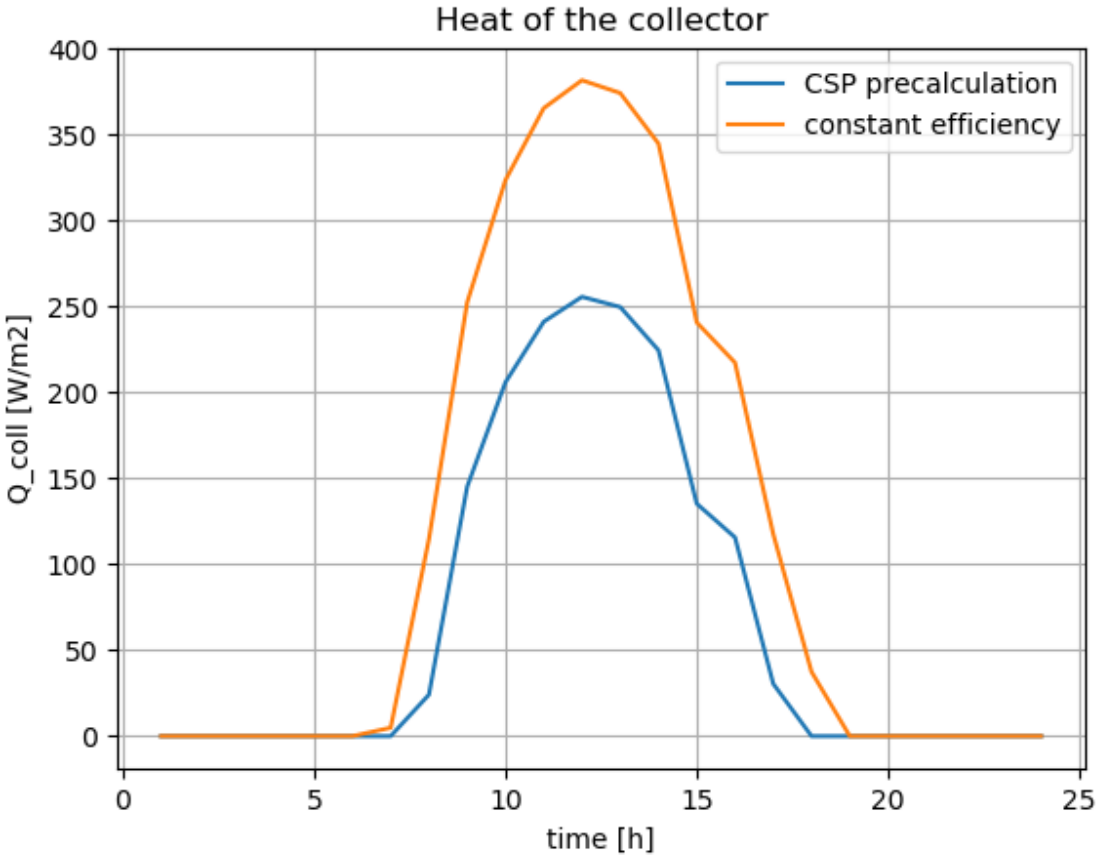
It is possible to use the precalculation function as stand-alone function to calculate the collector values $\dot{Q}_{coll}$, η_C and E_{coll}. Or it is possible to use the ParabolicTroughCollector facade to model a collector with further losses (e.g. in pipes or pumps) and the electrical consumption of pipes within a single step. Please note: As the unit of the input irradiance is given as power per area, the outputs $\dot{Q}_{coll}$ and E_{coll} are given in the same unit. If these values are used in an oemof source, the unit of the nominal value must be an area too.

6.3.1 Precalculation function

Please see the API documentation of the *concentrating_solar_power* module for all parameters which have to be provided, also the ones that are not part of the described formulas above. The data for ambient temperature and irradiance must have the same time index. Depending on the method, the irradiance must be the horizontal direct irradiance or the direct normal irradiance. Be aware of the correct time index regarding the time zone, as the utilized pvlib need the correct time stamp corresponding to the location (latitude and longitude).

```
data_precalc = csp_precalc(
    latitude, longitude,
    collector_tilt, collector_azimuth, cleanliness,
    eta_0, c_1, c_2,
    temp_collector_inlet, temp_collector_outlet, dataframe['t_amb'],
    a_1, a_2,
    E_dir_hor=dataframe['E_dir_hor']
    )
```

The following figure shows the heat provided by the collector calculated with this functions and the loss method “Janotte” in comparison to the heat calculated with a fix efficiency.



The results of this precalculation can be used in an oemof energy system model as output of a source component. To model the behaviour of a collector, it can be complemented with a transformer, which holds the electrical consumption of pumps and peripheral heat losses (see the the example csp_plant_collector.py).

6.3.2 ParabolicTroughCollector facade

Instead of using the precalculation, it is possible to use the ParabolicTroughCollector facade, which will create an oemof component as a representative for the collector. It calculates the heat of the collector in the same way as the precalculation do. Additionally, it integrates the calculated heat as an input into a component, uses an electrical input for pumps and gives a heat output, which is reduced by the defined additional losses. As given in the example, further parameters are required in addition to the ones of the precalculation. Please see the API documentation of the *ParabolicTroughCollector* class of the facade module for all parameters which have to be provided.

See example_csp_facade.py for an application example. It models the same system as the csp_plant_example.py, but uses the ParabolicTroughCollector facade instead of separate source and transformer.

```
from oemof import solph
    >>> from oemof.thermal.facades import ParabolicTroughCollector
    >>> bth = solph.Bus(label='thermal_bus')
    >>> bel = solph.Bus(label='electrical_bus')
    >>> collector = ParabolicTroughCollector(
    ...     label='solar_collector',
    ...     heat_bus=bth,
    ...     electrical_bus=bel,
    ...     electrical_consumption=0.05,
    ...     additional_losses=0.2,
    ...     aperture_area=1000,
    ...     loss_method='Janotte',
    ...     irradiance_method='horizontal',
    ...     latitude=23.614328,
    ...     longitude=58.545284,
    ...     collector_tilt=10,
    ...     collector_azimuth=180,
    ...     x=0.9,
    ...     a_1=-0.00159,
    ...     a_2=0.0000977,
    ...     eta_0=0.816,
    ...     c_1=0.0622,
    ...     c_2=0.00023,
    ...     temp_collector_inlet=435,
    ...     temp_collector_outlet=500,
    ...     temp_amb=input_data['t_amb'],
    ...     irradiance=input_data['E_dir_hor']
    )
```

6.4 References

[1] Janotte, N; et al: Dynamic performance evaluation of the HelioTrough collector demon-stration loop - towards a new benchmark in parabolic trough qualification, SolarPACES 2013

[2] William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. “pvlib python: a python package for modeling solar energy systems.” Journal of Open Source Software, 3(29), 884, (2018). https://doi.org/10.21105/joss.00884

CHAPTER 7

Solar thermal collector

Module to calculate the usable heat of a flat plate collector.

7.1 Scope

This module was developed to provide the heat of a flat plate collector based on temperatures and collector's location, tilt and azimuth for energy systems optimizations with oemof.solph.

In https://github.com/oemof/oemof-thermal/tree/dev/examples you can find an example, how to use the modul to calculate a system with flat plate collector, storage and backup to provide a given heat demand. The time series of the pre-calculated heat is output of a source (an oemof.solph component) representing the collector, and a transformer (an oemof.solph component) is used to hold electrical power consumption and further thermal losses of the collector in an energy system optimization. In addition, you will find a plot, which compares this precalculation with a calculation with a constant efficiency.

7.2 Concept

The precalculations for the solar thermal collector calculate the heat of the solar collector based on global and diffuse horizontal irradiance and information about the collector and the location. The following scheme shows the calculation procedure.

The processing of the irradiance data is done by the pvlib, which calculates the total in-plane irradiance according to the azimuth and tilt angle of the collector.

The efficiency of the collector is calculated with

$$\eta_C = \eta_0 - a_1 \cdot \frac{\Delta T}{E_{coll}} - a_2 \cdot \frac{\Delta T^2}{E_{coll}}$$

with

$$\Delta T = T_{coll,in} + \Delta T_n - T_{amb}$$

In the end, the irradiance on the collector is multiplied with the efficiency to get the collectors heat.

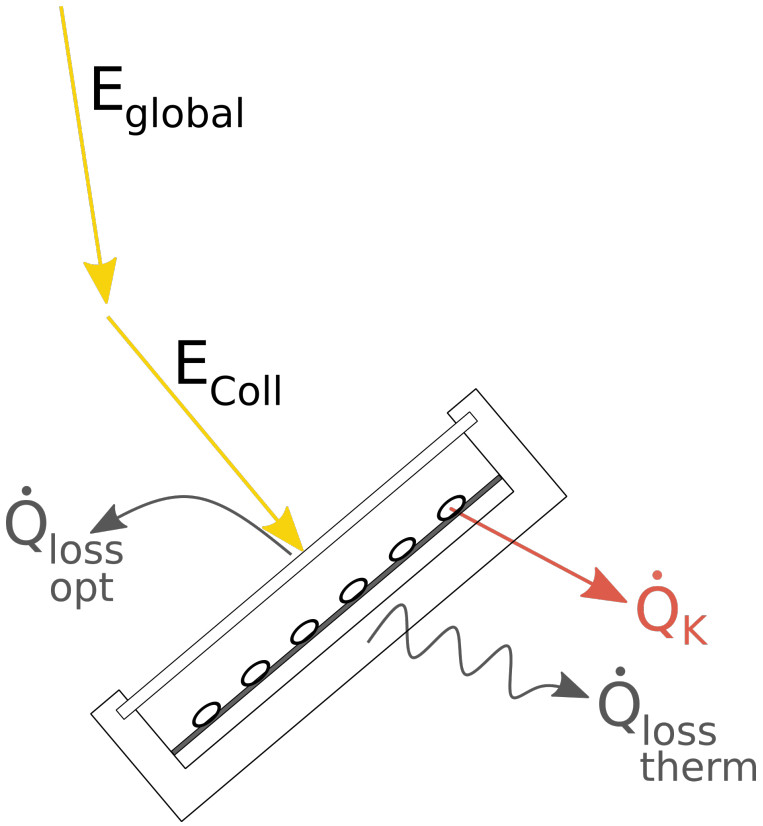


Fig. 1: Fig.1: The energy flows and losses at a flat plate collector.

$$\dot{Q}_{coll} = E_{coll} \cdot \eta_C$$

The three values $\dot{Q}_{coll}$, η_C and E_{coll} are returned in a dataframe. Losses, which occur after the heat absorption in the collector (e.g. losses in pipes) have to be taken into account in the component, which uses the precalculation (see the example).

These arguments are used in the formulas of the function:

symbol	argument	explanation
E_{coll}	`collector_irradiance`	Irradiance on collector after all losses
η_C	`eta_c`	Collectors efficiency
a_1	`a_1`	Thermal loss parameter 1
a_2	`a_2`	Thermal loss parameter 2
ΔT	`delta_t`	Temperature difference (collector to ambience)
$T_{coll,in}$	`temp_collector_inlet`	Collectors inlet temperature
ΔT_n	`delta_temp_n`	Temperature difference between collector inlet and mean temperature
T_{amb}	`temp_amb`	Ambient temperature
η_0	`eta_0`	Optical efficiency of the collector
$\dot{Q}_{coll}$	`collector_heat`	Collectors heat

7.3 Usage

It is possible to use the precalculation function as stand-alone function to calculate the collector values $\dot{Q}_{coll}$, η_C and E_{coll}. Or it is possible to use the SolarThermalCollector facade to model a collector with further losses (e.g. in pipes or pumps) and the electrical consumption of pipes within a single step. Please note: As the unit of the input irradiance is given as power per area, the outputs $\dot{Q}_{coll}$ and E_{coll} are given in the same unit. If these values are used in an oemof source, the unit of the nominal value must be an area too.

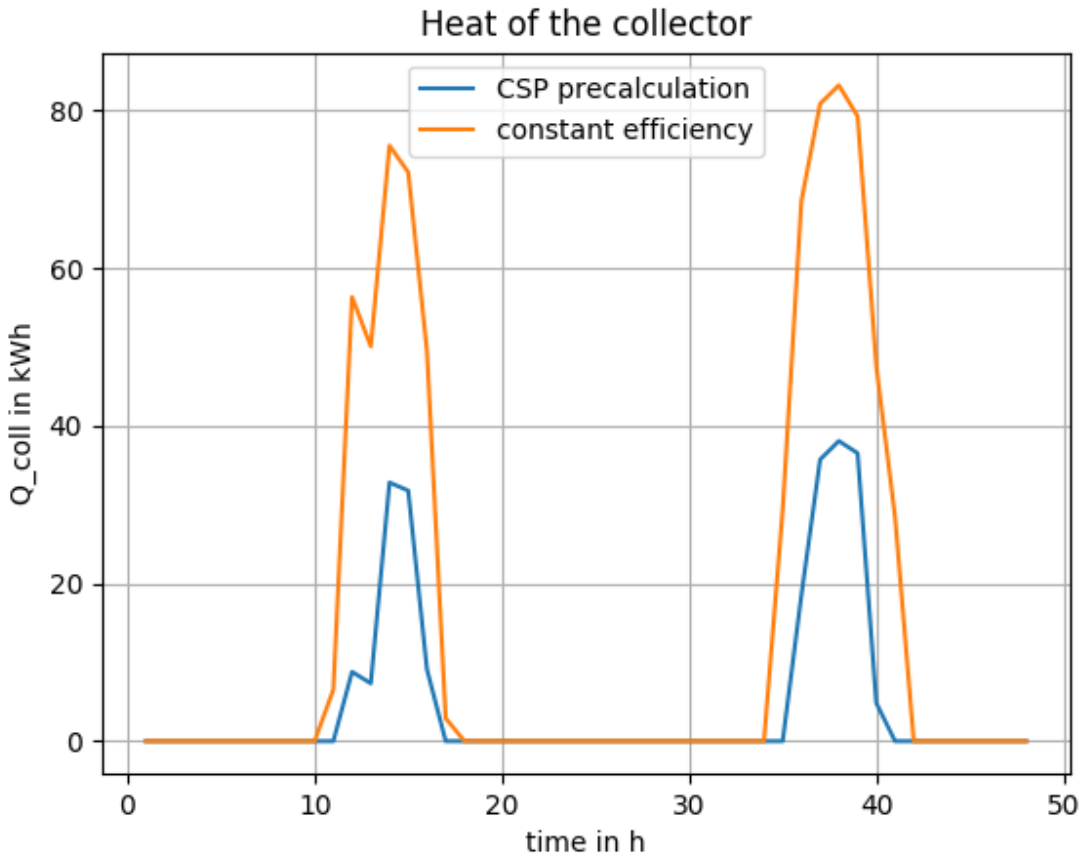
7.3.1 Solar thermal collector precalculations

Please see the API documentation of the *`solar_thermal_collector`* module for all parameters which have to be provided, also the ones that are not part of the described formulas above. The data for the irradiance and the ambient temperature must have the same time index. Be aware of the correct time index regarding the time zone, as the utilized pvlib needs the correct time stamp corresponding to the location.

```
precalc_data = flat_plate_precalc(
    latitude,
    longitude,
    collector_tilt,
    collector_azimuth,
    eta_0,
    a_1,
    a_2,
    temp_collector_inlet,
    delta_temp_n,
    irradiance_global=input_data['global_horizontal_W_m2'],
    irradiance_diffuse=input_data['diffuse_horizontal_W_m2'],
    temp_amb=input_data['temp_amb'],
)
```

The input_data must hold columns for the global and diffuse horizontal irradiance and the ambient temperature.

The following figure shows the heat provided by the collector calculated with this function in comparison to the heat calculated with a fix efficiency.



The results of this precalculation can be used in an oemof energy system model as output of a source component. To model the behaviour of a collector, it can be complemented with a transformer, which holds the electrical consumption of pumps and peripheral heat losses (see the the examples flat_plate_collector_example.py and flat_plate_collector_example_investment.py).

7.3.2 SolarThermalCollector facade

Instead of using the precalculation, it is possible to use the SolarThermalCollector facade, which will create an oemof component as a representative for the collector. It calculates the heat of the collector in the same way as the precalculation do. Additionally, it integrates the calculated heat as an input into a component, uses an electrical input for pumps and gives a heat output, which is reduced by the defined additional losses. As given in the example, further parameters are required in addition to the ones of the precalculation. Please see the API documentation of the *SolarThermalCollector* class of the facade module for all parameters which have to be provided.

See flat_plate_collector_example_facade.py for an application example. It models the same system as the flat_plate_collector_example.py, but uses the SolarThermalCollector facade instead of separate source and transformer.

```
from oemof import solph
from oemof.thermal.facades import SolarThermalCollector
bth = solph.Bus(label='thermal')
bel = solph.Bus(label='electricity')
collector = SolarThermalCollector(
    label='solar_collector',
    heat_out_bus=bth,
    electricity_in_bus=bel,
    electrical_consumption=0.02,
    peripheral_losses=0.05,
    aperture_area=1000,
    latitude=52.2443,
    longitude=10.5594,
    collector_tilt=10,
    collector_azimuth=20,
    eta_0=0.73,
    a_1=1.7,
    a_2=0.016,
    temp_collector_inlet=20,
    delta_temp_n=10,
    irradiance_global=input_data['global_horizontal_W_m2'],
    irradiance_diffuse=input_data['diffuse_horizontal_W_m2'],
    temp_amb_col=input_data['temp_amb'],
)
```

CHAPTER 8

Stratified thermal storage

8.1 Scope

This module was developed to implement a simplified model of a large-scale sensible heat storage with ideal stratification for energy system optimization with oemof.solph.

8.2 Concept

A simplified 2-zone-model of a stratified thermal energy storage.

- We assume a cylindrical storage of (inner) diameter d and height h, with two temperature regions that are perfectly separated.
- The temperatures are assumed to be constant and correspond to the feed-in/return temperature of the heating system.
- Heat conductivity of the storage has to be passed as well as a timeseries of outside temperatures for the calculation of heat losses.
- There is no distinction between outside temperature and ground temperature.
- A single value for the thermal transmittance U is assumed, neglecting the fact that the storage's lateral surface is bent and thus has a higher thermal transmittance than a flat surface. The relative error introduced here gets smaller with larger storage diameters.
- Material properties are constant.

The equation describing the storage content at timestep t is the following:

$$Q_t = Q_{t-1}\left(1 - U\frac{4}{d\rho c}\Delta t\right) - U\frac{4Q_N}{d\rho c\Delta T_{HC}}\Delta T_{C0}\Delta t - U\frac{\pi d^2}{4}\left(\Delta T_{H0} + \Delta T_{C0}\right)\Delta t + \dot{Q}_{in,t}\eta_{in}\Delta t - \frac{\dot{Q}_{out,t}}{\eta_{out}}\Delta t,$$

which is of the form

$$Q_t = Q_{t-1}(1-\beta) - \gamma Q_N - \delta + \dot{Q}_{in,t}\eta_{in}\Delta t - \frac{\dot{Q}_{out,t}}{\eta_{out}}\Delta t,$$

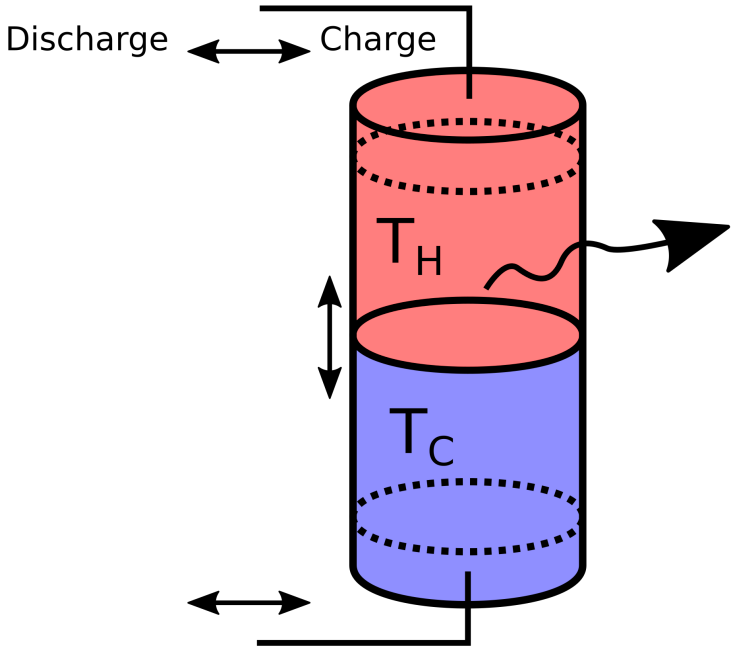


Fig. 1: Fig. 1: Schematic of the simplified model of a stratified thermal storage with two perfectly separated bodies of water with temperatures T_H and T_C. When charging/discharging the storage, the thermocline moves down or up, respectively. Losses to the environment through the surface of the storage depend on the size of the hot and cold zone.

with

$$\beta = U \frac{4}{d\rho c} \Delta t$$
$$\gamma = U \frac{4}{d\rho c \Delta T_{HC}} \Delta T_{C0} \Delta t$$
$$\delta = U \frac{\pi d^2}{4} \Big(\Delta T_{H0} + \Delta T_{C0} \Big) \Delta t.$$

The three terms represent:

- δ, constant heat losses through the top and bottom surfaces,
- $\gamma \cdot Q_N$, losses through the total lateral surface assuming the storage to be empty (storage is at T_C, and ΔT_{C0} is the driving temperature difference), depending on the height of the storage,
- $\beta \cdot Q_{t-1}$, additional losses through lateral surface that belong to the hot part of the water body, depending on the state of charge.

In the case of investment, the diameter d is given and the height can be adapted to adapt the nominal capacity of the storage. With this assumption, all relations stay linear.

Because of the space that diffuser plates for charging/discharging take up, it is assumed that the storage can neither be fully charged nor discharged, which is parametrised as a minimal/maximal storage level (indicated by the dotted lines in Fig. 1).

These parameters are part of the stratified thermal storage module:

sym-bol	attribute	type	explanation
h	`height`		Height [m] (if not investment)
d	`diameter`		Diameter [m]
A	`surface`		Storage surface [m2]
V	`volume`		Storage volume [m3]
ρ	`density`		Density of storage medium [kg/m3]
c	`heat_capacity`		Heat capacity of storage medium [J/(kg*K)]
T_H	`temp_h`		Hot temperature level [deg C]
T_C	`temp_c`		Cold temperature level [deg C]
T_0	`temp_env`		Environment temperature timeseries [deg C]
Q_t	attribute of oemof-solph component		Stored thermal energy at time t [MWh]
$\dot{Q}_{in,t}$	attribute of oemof-solph component		Energy flowing in at time t
Q_N	`nominal_storage_capacity`		Maximum amount of stored thermal energy [MWh]
U	`u_value`		Thermal transmittance [W/(m2*K)]
s_{iso}	`s_iso`		Thickness of isolation layer [mm]
λ_{iso}	`lamb_iso`		Heat conductivity of isolation material [W/(m*K)]
α_i	`alpha_inside`		Heat transfer coefficient inside [W/(m2*K)]
α_o	`alpha_outside`		Heat transfer coefficient outside [W/(m2*K)]
β	`loss_rate`		Relative loss of storage content within one timestep [-]
γ	`fixed_losses_relative`		Fixed losses as share of nominal storage capacity [-]
δ	`fixed_losses_absolute`		Fixed absolute losses independent of storage content or nominal storage capacity [MWh]
η_{in}	`inflow_conversion_factor`		Charging efficiency [-]
η_{out}	`outflow_conversion_factor`		Discharging efficiency [-]

8.3 Usage

8.3.1 StratifiedThermalStorage facade

Using the StratifiedThermalStorage facade, you can instantiate a storage like this:

```
from oemof.solph import Bus
from oemof.thermal.facades import StratifiedThermalStorage

bus_heat = Bus('heat')

thermal_storage = StratifiedThermalStorage(
    label='thermal_storage',
    bus=bus_heat,
    diameter=2,
    height=5,
    temp_h=95,
    temp_c=60,
    temp_env=10,
    u_value=u_value,
    min_storage_level=0.05,
    max_storage_level=0.95,
    capacity=1,
```

(continues on next page)

(continued from previous page)

```
    efficiency=0.9,
    marginal_cost=0.0001
)
```

The non-usable storage volume is represented by the parameters `min_storage_level` and `max_storage_level`.

To learn about all parameters that can be passed to the facades, have a look at the API documentation of the *`StratifiedThermalStorage`* class of the facade module.

For the storage investment mode, you still need to provide `diameter`, but leave `height` and `capacity` open and set `expandable=True`.

There are two options to choose from:

1. Invest into `nominal_storage_capacity` and `capacity` (charging/discharging power) with a fixed ratio. Pass `invest_relation_input_capacity` and either `storage_capacity_cost` or `capacity_cost`.
2. Invest into `nominal_storage_capacity` and `capacity` independently with no fixed ratio. Pass `storage_capacity_cost` and `capacity_cost`.

In many practical cases, thermal storages are dimensioned using a rule of thumb: The storage should be able to provide its peak thermal power for 6-7 hours. To apply this in a model, use option 1.

```
thermal_storage = StratifiedThermalStorage(
    label='thermal_storage',
    bus=bus_heat,
    diameter=2,
    temp_h=95,
    temp_c=60,
    temp_env=10,
    u_value=u_value,
    expandable=True,
    capacity_cost=0,
    storage_capacity_cost=400,
    minimum_storage_capacity=1,
    invest_relation_input_capacity=1 / 6,
    min_storage_level=0.05,
    max_storage_level=0.95,
    efficiency=0.9,
    marginal_cost=0.0001
)
```

If you do not want to use a rule of thumb and rather let the model decide, go with option 2. Do so by leaving out `invest_relation_input_capacity` and setting `capacity_cost` to a finite value. Also have a look at the examples, where both options are shown.

A 3rd and 4th option, investing into `nominal_storage_capacity` but leaving `capacity` fixed or vice versa, can not be modelled with this facade (at the moment). It seems to be a case that is not as relevant for thermal storages as the others. If you want to model it, you can do so by performing the necessary pre-calculations and using oemof.solph's `GenericStorage` directly.

Warning: For this example to work as intended, please use oemof-solph v0.4.0 or higher to ensure that the GenericStorage has the attributes `fixed_losses_absolute` and `fixed_losses_relative`.

The following figure shows a comparison of results of a common storage implementation using only a loss rate vs. the stratified thermal storage implementation (source code).

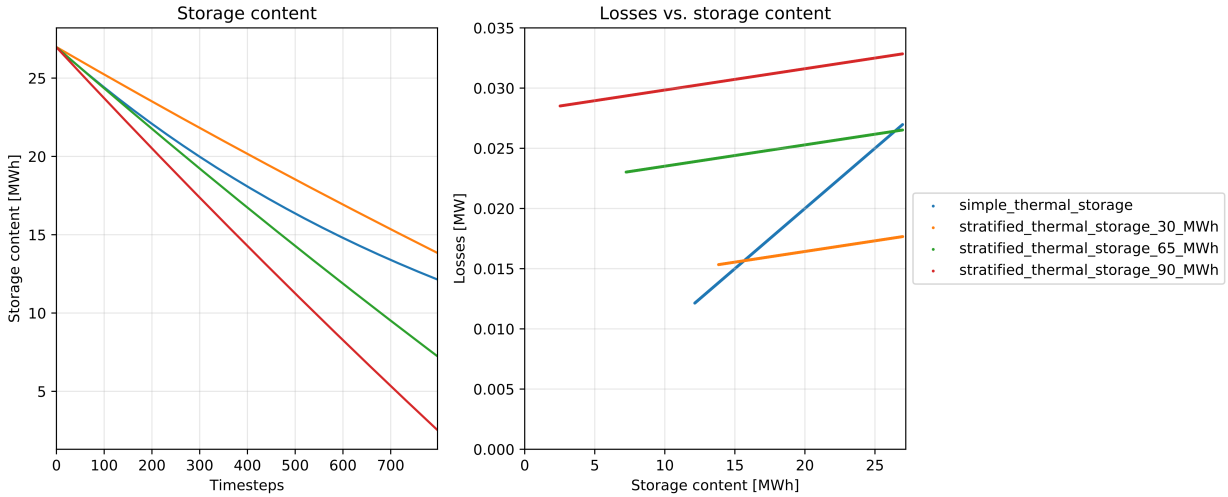


Fig. 2: Fig. 2: Example plot showing the difference between `StratifiedThermalStorages` of different storage capacities and a simpler model with a single loss rate. The left panel shows the loss of thermal energy over time. The right panel shows losses vs. storage content.

8.3.2 Implicit calculations

In the background, the StratifiedThermalStorage class uses the following functions. They can be used independent of the facade class as well.

The thermal transmittance is pre-calculated using *calculate_u_value*.

The dimensions of the storage are calculated with *calculate_storage_dimensions*

```
volume, surface = calculate_storage_dimensions(height, diameter)
```

$$V = \pi \frac{d^2}{4} \cdot h$$

$$A = \pi d h + \pi \frac{d^2}{2}$$

The nominal storage capacity is pre-calculated using *calculate_capacities*.

```
nominal_storage_capacity = calculate_capacities(
    volume, temp_h, temp_c, heat_capacity, density
)
```

$$Q_N = V \cdot c \cdot \rho \cdot (T_H - T_C)$$

Loss terms are precalculated by the following function.

```
loss_rate, fixed_losses_relative, fixed_losses_absolute = calculate_losses(
    u_value, diameter, temp_h, temp_c, temp_env,
    time_increment, heat_capacity, density)
```

$$\beta = U \frac{4}{d \rho c} \Delta t$$

$$\gamma = U \frac{4}{d \rho c \Delta T_{HC}} \Delta T_{C0} \Delta t$$

$$\delta = U \frac{\pi d^2}{4} \Big(\Delta T_{H0} + \Delta T_{C0} \Big) \Delta t$$

To calculate the thermal transmittance of the storage hull from material properties, you can use the following function.

```
u_value = calculate_storage_u_value(s_iso, lamb_iso, alpha_inside, alpha_outside)
```

$$U = \frac{1}{\frac{1}{\alpha_i} + \frac{s_{iso}}{\lambda_{iso}} + \frac{1}{\alpha_a}}$$

CHAPTER 9

Compression Heat Pump and Chiller

9.1 Scope

The validation of the compression heat pump and chiller has been conducted within the GRECO project in collaboration with the oemof_heat project. Monitored data of the two components in combination with PV without storage has been provided by Universidad Politécnica de Madrid (UPM). An open access publication containing further description of the data and experiments done in [2] is planned for November 2020. Both, heat pump and chiller, are working with air to air technology. The set of data contains amongst others external and internal temperatures of the components and a calculated Coefficient of performance (COP) / Energy Efficiency Ratio (EER) value. The code used for the validation can be found here.

9.2 Method

In order to calculate the COP and EER using oemof-thermal the temperature of the heat source, the temperature of the heat sink and the quality grade are required. The quality grade describes the relation between the actual coefficient of performance and the coefficient of performance of the Carnot process. Please see the USER'S GUIDE on compression heat pumps and chillers for further information.

The monitored coefficients from UPM are compared with the coefficients calculated using different quality grades to evaluate which quality grade fits best the examined chiller and heat pump. For the heat pump, the temperature of the external input into the evaporator is the heat source and the temperature of the internal output from the condenser is the heat sink. In case of the chiller the heat source is the external temperature input into the condenser and the heat sink the internal temperature output from the evaporator. The monitored coefficients are calculated as the ratio between the thermal capacity and the electrical capacity.

The data set contains data points where the solar modules provide an electrical power less than 100 watts and where the compressor of the installation is turned off. These data points are excluded from calculations since they differ from the ones under operational behavior of the component. For this purpose the data is preprocessed in order to attain only data points with electrical power greater or equal to 100 watts and with the integral fan turned on. In comparison to the chiller's monitored data, the data of the heat pump contains a higher number of excluded data points. It further has multiple downtimes, which lead to reversed temperatures of heat sink and heat source. These data points are also excluded from the validation.

The functionalities for calculating the COP and EER of oemof-thermal are made for a stationary process, while the data provided by UPM includes mostly data from non-stationary periods as different control modes are explored. To balance the fluctuating values we decided to analyze average hourly values.

Various types of charts are used for the validation of the calculated COP and EER. For the validation the residual, which corresponds to the difference of the monitored and calculated coefficients, is used. For both, chiller and heat pump, correlations with the residual are shown in various types of charts: the histograms, the correlation between calculated and monitored coefficients, the root mean square error (RMSE), the relation between the residuals and temperature hub as well as the relation between residuals and monitored coefficients.

9.3 Results of the chiller

Typical EER of chillers used for cooling are around 4 to 5 [1]. By adhering to these reference values we conclude that EERs with quality grades ranging from 0.25 to 0.4 give fitting results.

The RMSE for the validated quality grades presents the standard deviation of the residuals. Among the range of quality grades for the chiller, the RMSE is smallest for the quality grade 0.30 with 0.573 and larger for 0.25 with 0.758 and for 0.35 with 1.181 (cf. Tab.1).

Tab.1: Root mean square error for different quality grades of the chiller

Quality grade	RSME
0.05	3.831
0.10	3.030
0.15	2.236
0.20	1.460
0.25	0.758
0.30	0.573
0.35	1.181
0.40	1.943
0.45	2.732
0.50	3.531

The correlation with quality grades below 0.30 show an underestimation of the coefficients. In contrast, calculated EER at quality grades above 0.30 indicate an overestimation. Fig.1 shows the estimated correlation of EERs in the middle together with under- and overestimation to the left and right:

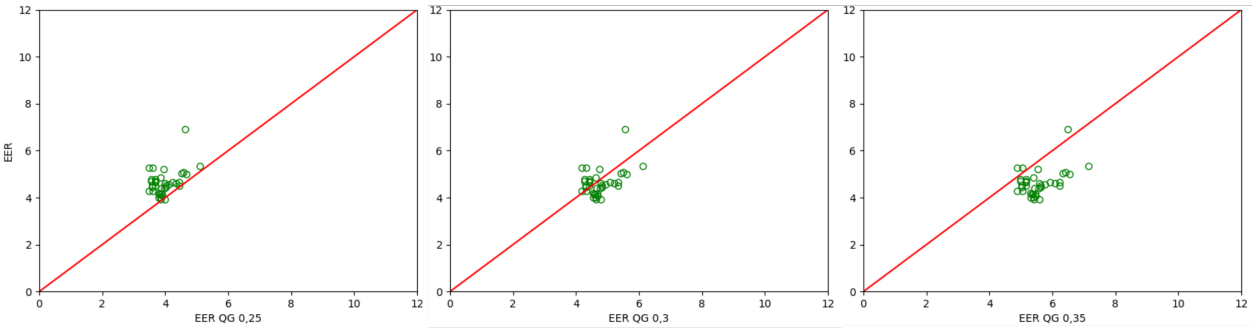


Fig. 1: Fig.1: Correlation between monitored and calculated EER with underestimation showing a quality grade of 0.25 (left), quality grade of 0.30 with least error (middle) and with overestimation connected to a quality grade of 0.35 (right)

Fig.2 shows the residual over monitored EER for quality grades of 0.25, 0.30 and 0.35. In Fig.3 the residual is plotted over the temperature hub for the three quality grades. From both graphs can be derived that the residual is minimal for a quality grade of 0.30. Furthermore they indicate a dependence of the residuals to both parameters. Smaller temperature hubs cause larger residuals, while larger temperature differences lead to smaller residuals. In general, residuals decrease with rising quality grades.

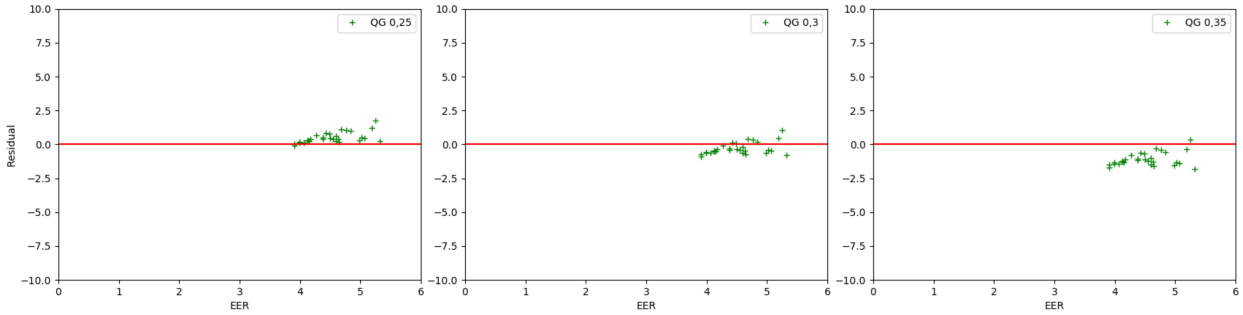


Fig. 2: Fig.2: Correlation between residual and monitored EER showing a quality grade of 0.25 (left), quality grade of 0.30 with least error (middle) and a quality grade of 0.35 (right)

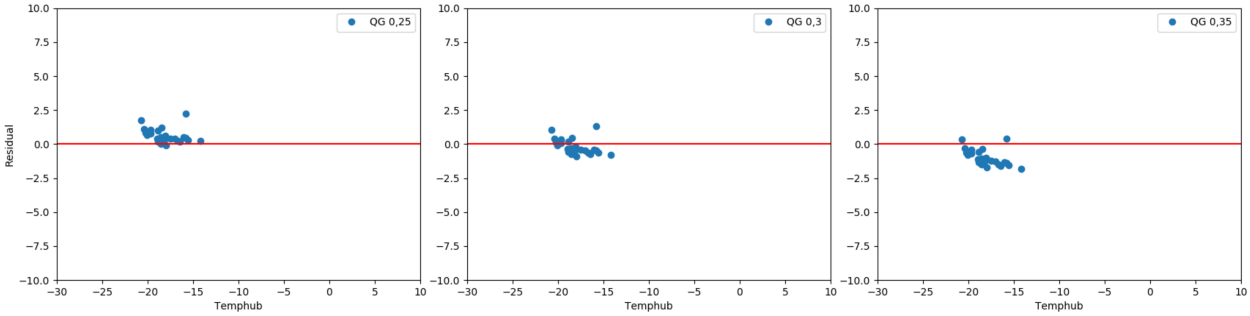


Fig. 3: Fig.3: Correlation between residual and temperature hub showing a quality grade of 0.25 (left), quality grade of 0.30 with least error (middle) and a quality grade of 0.35 (right)

The histogram in Fig.4 depicts that most of the calculated coefficients have small deviations with the quality grade of 0.30 (middle). Based on the left graph it gets clear that the average calculated EER decreases with lower quality grades due to the shift to the right of the histogram. As seen in the right graph of Fig.4 the average calculated EER increases with higher quality grades due to the shift to the left of the histogram.

The outliers in the monitored data could be due to the start-up and shutdown of the prototypes' compressor.

An examination of the complete data set of the chiller shows a linear dependence of the residuals to the monitored EER. In Fig.5 this linearity can be seen for a quality grade of 0.05 (left graph) and a quality grade of 0.50 (right graph). It is striking that the linearity dependence is higher for smaller quality grades such as 0.05 (cf. left graph in Fig.5). The dispersion of residuals in areas of lower as well as higher monitored EER increases with larger quality grades.

9.4 Results of the heat pump

The RMSE calculated using the heat pump's monitored data is smallest for the quality grade 0.35 with 0.991 and larger for 0.30 with 1.123 and for 0.40 with 1.206 (cf. Tab.1).

Tab.2: Root mean square error for different quality grades of the heat pump

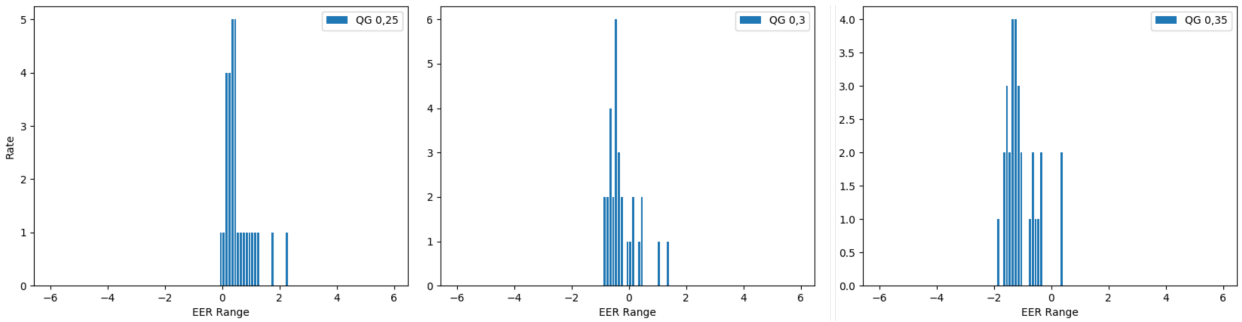


Fig. 4: Fig.4: Histogram of residuals with underestimation showing a quality grade of 0.25 (left), quality grade of 0.30 with least error (middle) and with overestimation connected to a quality grade of 0.35 (right)

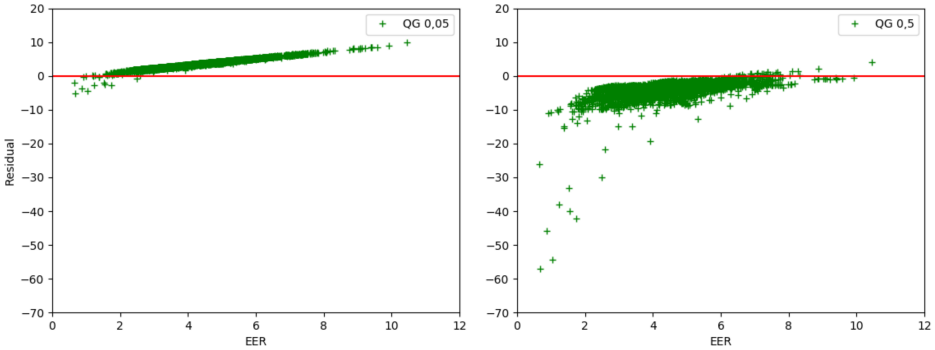


Fig. 5: Fig.5: Correlation between residual and monitored EER of the complete data set with a quality grade of 0.05 (left) and a quality grade of 0.50 (right)

Quality grade	RSME
0.05	3.726
0.10	3.140
0.15	2.566
0.20	2.015
0.25	1.512
0.30	1.123
0.35	0.991
0.40	1.206
0.45	1.635
0.50	2.155

The comparison of the smallest RSME of both chiller and heat pump indicates that the monitored data of the heat pump contains higher deviations.

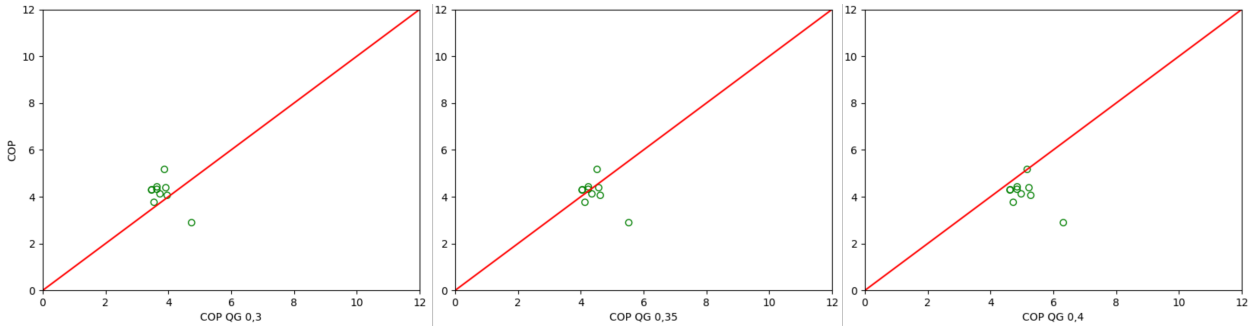


Fig. 6: Fig.6: Correlation between monitored and calculated COP with underestimation showing a quality grade of 0.30 (left), quality grade of 0.35 with least error (middle) and with overestimation connected to a quality grade of 0.40 (right)

Just as with the chiller, the correlations indicate an underestimation at lower quality grades and an overestimation at larger quality grades.

Fig.7 shows the residual over monitored COP for quality grades of 0.30, 0.35 and 0.40. In Fig.8 the residual is plotted over the temperature hub for the three quality grades. From both graphs can be derived that the residual is minimal for a quality grade of 0.35. As in the cooler's results the dependency of residuals and both parameter is observable: Residuals decrease with rising quality grades.

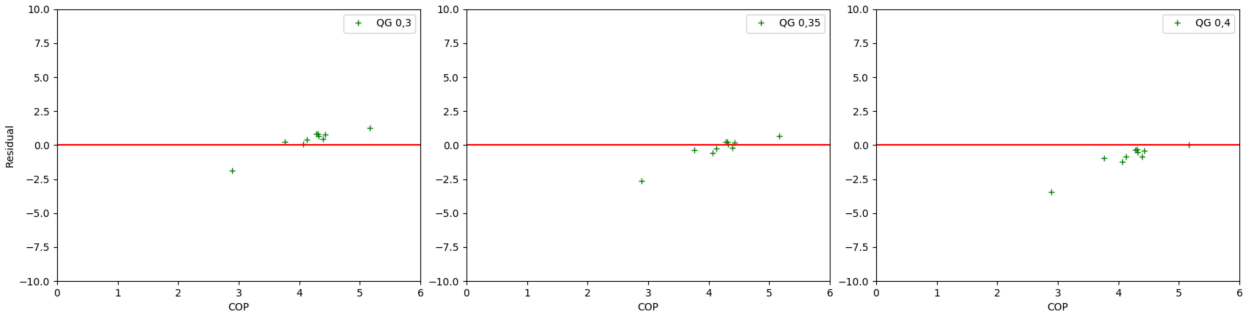


Fig. 7: Fig.7: Correlation between residual and monitored COP showing a quality grade of 0.30 (left), quality grade of 0.35 with least error (middle) and a quality grade of 0.40 (right)

In Fig.9 the histograms of the heat pump are shown. The peak of the histograms shifts to the right with smaller quality

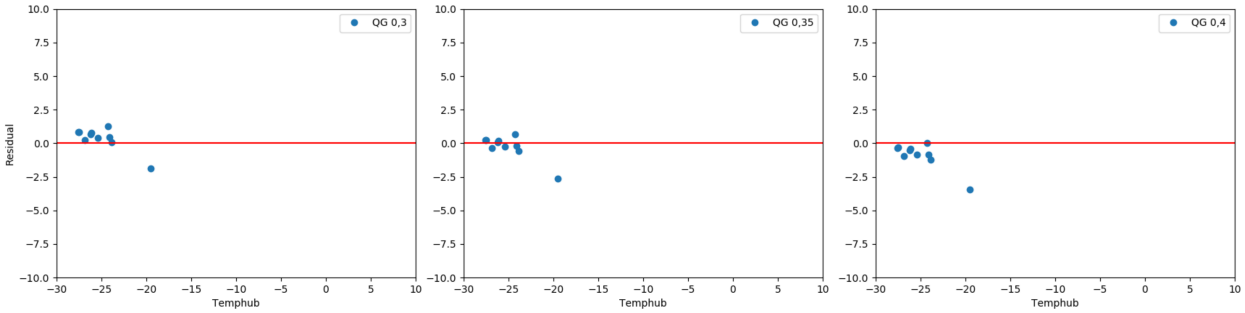


Fig. 8: Fig.8: Correlation between residual and temperature hub showing a quality grade of 0.30 (left), a quality grade of 0.35 with least error (middle) and a quality grade of 0.40 (right)

grades (cf. left graph in Fig.9) and to the left with larger quality grades (cf. right graph in Fig.9). The values of the coefficients fluctuate more compared to the chiller.

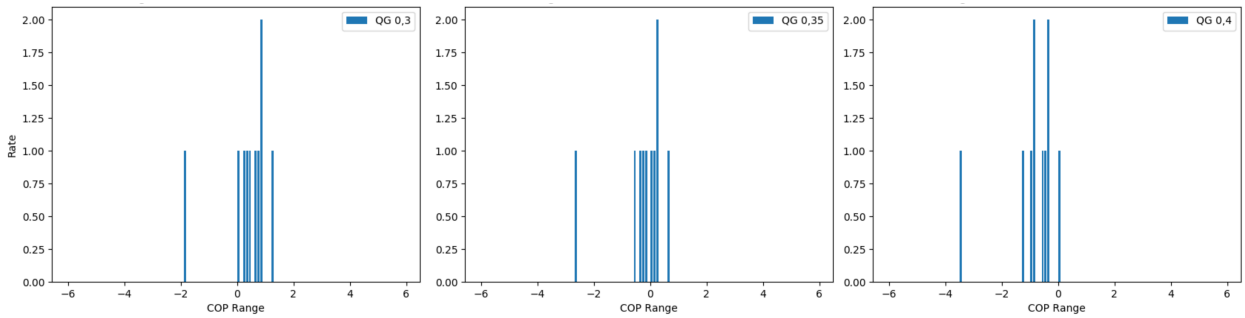


Fig. 9: Fig.9: Histogram of residuals with underestimation showing a quality grade of 0.30 (left), quality grade of 0.35 with least error (middle) and with overestimation connected to a quality grade of 0.40 (right)

Looking at the whole preprocessed monitored data, a linear dependence of the residuals to monitored COP values can be identified. The linear dependency for two quality grades 0.05 (left) and 0.5 (right) is depicted in Fig.10. Just as with the chiller the linearity dependence is higher for smaller quality grades such as 0.05 (cf. left graph in Fig.10). The dispersion of residuals in areas of lower as well as higher monitored COP increases with larger quality grades.

9.5 References

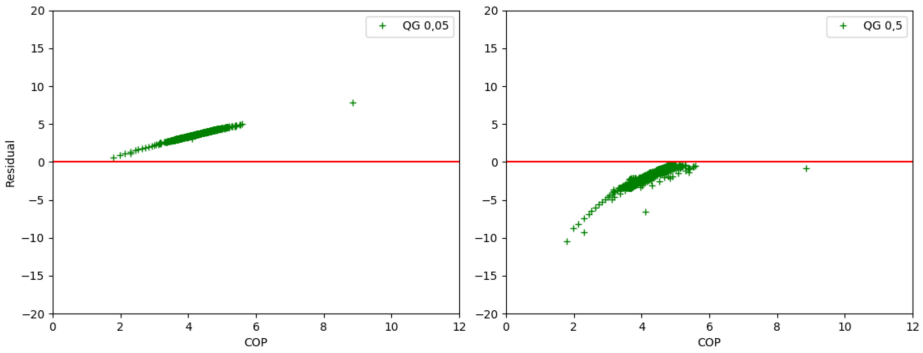


Fig. 10: Fig.10: Correlation between residual and monitored COP of the complete data set with a quality grade of 0.05 (left) and a quality grade of 0.50 (right)

CHAPTER 10

Stratified thermal storage

10.1 Scope

The validation of the stratified thermal storage has been conducted within the oemof_heat project. Measurement data of a reference storage has been provided by the energy supplier Naturstrom AG. The set of data contains the storage geometry (height, diameter, insulation thickness), temperatures at top and bottom of the storage and a time series of the storage level.

10.2 Method

In order to calculate the storage level using the StratifiedThermalStorage component from oemof-thermal the storage geometry, the temperatures of the hot and cold layers (top and bottom), the temperature of the environment, the heat conductivity of the insulation and the heat transfer coefficients inside and outside of the storage surface are required. Tab.1 shows the required input parameter and the respective values of the reference storage. For some parameters assumptions had to be made.

Name	Value
Data of the reference storage	
height	2.96 m
diameter	1.15 m
insulation thickness	100 mm
temperature of hot layer	82°C
temperature of cold layer	55°C
Own Assumptions	
temperature of environment	25°C
conductivity of insulation	0.039 W/(m*K)
heat transfer coef. inside	7 W/(m2*K)
heat transfer coef. outside	4 W/(m2*K)

Tab.1: Input parameters used for the model validation

Please see the USER'S GUIDE on the stratified thermal storage for further information.

The level of the reference storage is not measured directly but is determined from the temperatures at different heights T_i in the storage.

$$level = \frac{T_{\mathrm{mean}} - T_{\mathrm{cold}}}{T_{\mathrm{hot}} - T_{\mathrm{cold}}}$$

where T_{mean} is the arithmetic mean temperature of the storage.

$$T_{\mathrm{mean}} = \frac{\sum_{i=1}^{n} T_i}{n}$$

where n is the amount of temperature sensors.

10.3 Measurement data

The measurement data come from an energy system that contains several identical storages. Here, only a single storage is calculated to keep the model simple.

The validation aims on checking how accurately the losses of the storage are predicted by the model. This does not include losses during the charging and discharging (inflow_conversion_factor and outflow_conversion_factor). Therefore a short time series of measurement data (see Tab.2) is used for the validation where no charging or discharging occurs.

Time	Level in %
0.0	78.50
0.25	78.21
0.5	78.38
0.75	78.00
1.0	78.25
1.25	77.79
1.5	77.75
1.75	77.04
2.0	77.17
2.25	77.63
2.5	78.00
2.75	77.71
3.0	77.79
3.15	77.29
3.5	77.00
3.75	76.38
4.0	77.33
4.25	77.21
4.5	77.00
4.75	77.29
5.0	77.08
5.25	76.54
5.5	76.33

Tab.2: Measured storage level.

10.4 Results

Fig.1 shows the measured and the calculated storage level over a period of 5 1/2 hours. The reference storage loses around 1.5% of its content in 5 hours. The calculated losses are slightly less.

The trajectory of the measured level is not straight like the calculated one but appears to fluctuate. This is caused by the way the level is determined. It is derived from temperatures measured at discrete points. Slight changes of the measured local temperatures lead to the fluctuating level signal.

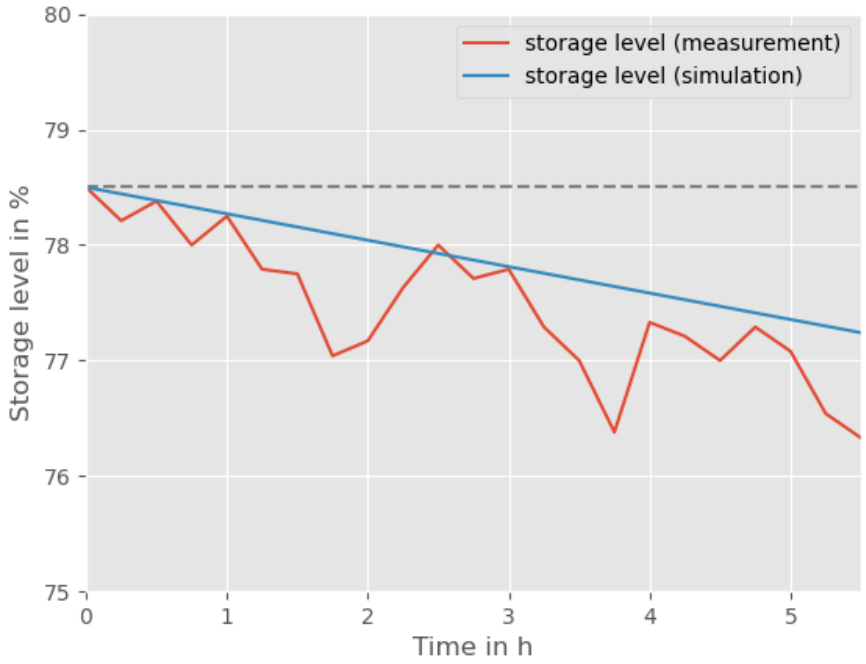


Fig. 1: Fig.1: Measured storage level (red) and calculated storage level (blue).

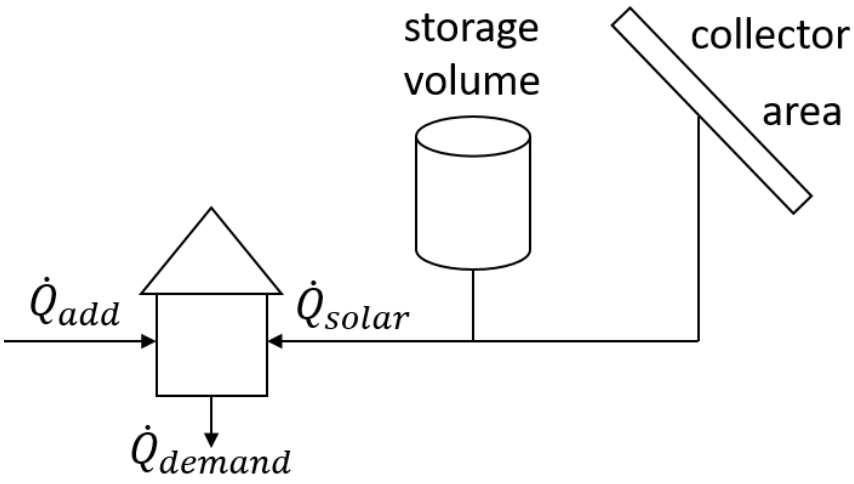
The model allows an approximation of the losses from simple storage geometry data in periods without charging or discharging.

You can reproduce Fig.1 and the calculation with the example `model_validation.py` in the examples section on GitHub.

CHAPTER 11

Aggregation of domestic decentral solar thermal systems

In this section the aggregation of consumers using solar thermal systems is discussed. Depending on the consumers energy usage the ratio of heat load and collector size or storage capacity differs. If you want to model a district with domestic solar systems, it is difficult to aggregate them because of the different sizes of the components and thus different points in time, when the backup heating has to start. The picture shows a scheme of such a system:



11.1 Concept

Instead of aggregate all storages and all collectors of the system, which would be really unaccurate, we want to classify two different types of domestic solar systems:

- systems for hot water provision
- systems for hot water provision and space heating support

The system shown in the picture can be characterized by two ratios:

$ratio1 = \frac{V_{storage}}{A_{collector}}$ and $ratio2 = \frac{A_{collector}}{\dot{Q}_{demand,max}}$

or, also possible:

$ratio1 = \frac{V_{storage}}{A_{collector}}$ and $ratio2 = \frac{A_{collector}}{Q_{demand}}$

Normally there are typical ratios which are used when dimensioning these types of systems. So it is possible to aggregate the houses within one type by defining these ratios. This would be more accurate than an aggregation of all installed systems. In case of special system modifications, also more than two groups could be defined.

11.2 Nomenclature

symbol	explanation
$\dot{Q}_{add}$	Heat flow from external to consumer
$\dot{Q}_{demand}$	Heat flow demand of the consumer
$\dot{Q}_{solar}$	Heat flow from collector
$\dot{Q}_{demand,max}$	Maximal value of the consumer's heat flow demand
Q_{demand}	Total demand
$V_{storage}$	Storage volume
$A_{collector}$	Collector surface

CHAPTER 12

What's New

Discover noteable new features and improvements in each release.

Releases

12.1 v0.0.6 (June 12, 2023)

12.1.1 Bug fixes

- Made compatible to oemof.solph v0.5

12.1.2 Contributors

Patrik Schönfeldt

12.2 v0.0.5 (November 12, 2021)

12.2.1 API Changes

- Explicitly define public API

12.2.2 Other changes

- Updated dependencies

12.2.3 Contributors

- Patrik Schönfeldt
- Sasan Rasti

12.3 v0.0.4 (October 14, 2020)

12.3.1 New features

- We have validated the following components: compression heat pump, compression chiller, stratified thermal storage and concentrating solar power (CSP). The documentation of the CSP validation is postponed due to data licensing issues.
- An excess sink has been defined in the facades of the solar thermal and the concentrating solar power component to prevent energy systems from being unsolvable because of heat overproduction.

12.3.2 Documentation

- Improvements in the documentation of the stratified thermal storage have been made.
- We improved and extended the examples section in the documentation.

12.3.3 Bug fixes

- A warning for missing required parameters has been implemented in the concentrating solar power component.

12.3.4 Other changes

- We have cleaned the examples of the stratified thermal storage component and made it more transparent.

12.3.5 Contributors

- Caroline Möller
- Franziska Pleißner
- Jakob Wolf

- Jann Launer
- Marie-Claire Gering
- Patrik Schönfeldt
- Felix Janiak

12.4 v0.0.3 (July 3, 2020)

12.4.1 API changes

- The repository has been updated to oemof.solph v0.4.1

12.4.2 New components

- **We have developed a new component: Absorption chillers** The module contains functions to calculate the characteristic equation, the heat fluxes at the evaporator and generator and some further specification of the chiller's capacity.

12.4.3 Documentation

- The documentation has been improved.
- The descriptions in the examples have been cleaned up.

12.4.4 Contributors

- Caroline Möller
- Franziska Pleißner
- Jakob Wolf
- Jann Launer
- Marie-Claire Gering
- Patrik Schönfeldt

12.5 v0.0.2 (April 30, 2020)

12.5.1 API changes

- The arguments of `compression_heatpumps_and_chillers.calc_cops()` changed: `consider_icing` was removed. To consider icing in the calculation `factor_icing` (default value: `None`) has to be set not None.
- concentrating solar power and solar thermal collector: The input for irradiance is now a time indexed series instead of a series and separate information about the date.

12.5.2 New features

- We introduced new facade classes that simplify instantiating components. These facades are now ready to be used: `SolarThermalCollector`, `ParabolicTroughCollector` and `StratifiedThermalStorage`. There is in each case an example which shows how the facade is used.
- Concentrating solar power: losses can be calculated with two methods.
- The function `compression_heatpumps_and_chillers.calc_cops()` raises Errors in case of wrong argument type or size.
- We have added a function that implements methods for emission allocation. It is part of a new module `cogeneration.py`.

12.5.3 Documentation

- The documentation has been revised.
- The schematic pictures of the components have been improved.
- Documentation has been extended, in particular for the new facade classes.
- Badges showing build status, docs build status, test coverage and zenodo DOI have been added to the README.rst.

12.5.4 Bug fixes

- The path to the readin data in csp_collector_plot_example.py has been corrected. The files could not be found because it was given in uppercase letters.
- Wrong path in api/oemof-thermal.rst automodule has been fixed.
- The default values for the density and capacity have been corrected in stratified_thermal_storage.py. As a simplification the values are calculated with CoolProp for a constant temperature of 80 °C.
- Figures and images of svg format did not render in pdfLaTeX (Issue #46). As a solution to the problem png files have been added and included in the docs.
- The title of the documentation was ‘oemof heat documentation’ and not ‘oemof.thermal documentation’. Target name and title of documents, that are output within conf.py, have been changed for LaTeX, manual page, Texinfo and Epub.

12.5.5 Tests

- The repository now has function tests, see oemof-thermal/tests/test_functions.py
- The new facade classes are tested with constraint tests, see oemof-thermal/tests/test_constraints.py

12.5.6 Other changes

- Warnings in the docs and the examples of the stratified thermal storage have been added if the minimum required oemof version for oemof.thermal is not v3.3 while using the attributes fixed_losses_relative and fixed_losses_absolute.
- The separation in the csv files of the concentrating solar power has been changed from “;” to “,”.

12.5.7 Contributors

- Caroline Möller
- Franziska Pleißner
- Jakob Wolf
- Jann Launer
- Marie-Claire Gering

12.6 v0.0.1 (December 19, 2019)

First release by the oemof developing group.

CHAPTER 13

API

13.1 cogeneration module

This module is designed to hold functions for pre- and postprocessing for combined heat and power plants.

This file is part of project oemof (github.com/oemof/oemof-thermal). It's copyrighted by the contributors recorded in the version control history of the file, available from its original location: oemof-thermal/src/oemof/thermal/chp.py

SPDX-License-Identifier: MIT

`oemof.thermal.cogeneration.`**`allocate_emissions`**(*total_emissions*, *eta_el*, *eta_th*, *method*, ***kwargs*)

Function to allocate emissions caused in cogeneration to the products electrical energy and heat according to specified method.

IEA method

$EM_{el} = EM \cdot \frac{\eta_{el}}{\eta_{el}+\eta_{th}}$,

$EM_{th} = EM \cdot \frac{\eta_{th}}{\eta_{el}+\eta_{th}}$.

Efficiency method

$EM_{el} = EM \cdot \frac{\eta_{th}}{\eta_{el}+\eta_{th}}$,

$EM_{th} = EM \cdot \frac{\eta_{el}}{\eta_{el}+\eta_{th}}$.

Finnish method

$EM_{el} = EM \cdot (1 - PEE) \frac{\eta_{el}}{\eta_{el,REF}}$,

$EM_{th} = EM \cdot (1 - PEE) \frac{\eta_{th}}{\eta_{th,REF}}$,

with

$PEE = 1 - \frac{1}{\frac{\eta_{th}}{\eta_{th,ref}} + \frac{\eta_{el}}{\eta_{el,ref}}}$.

Reference: Mauch, W., Corradini, R., Wiesmeyer, K., Schwentzek, M. (2010). Allokationsmethoden für spezifische CO2-Emissionen von Strom und Waerme aus KWK-Anlagen. Energiewirtschaftliche Tagesfragen, 55(9), 12–14.

Parameters

- **total_emissions** (*numeric*) – Total absolute emissions to be allocated to electricity and heat [in CO2 equivalents].
- **eta_el** (*numeric*) – Electrical efficiency of the cogeneration [-].
- **eta_th** (*numeric*) – Thermal efficiency of the cogeneration [-].
- **method** (*str*) – Specification of method to use. Choose from ['iea', finnish', 'efficiency'].
- ****kwargs** – For the finnish method, *eta_el_ref* and *eta_th_ref* have to be passed.

Returns

- **allocated_emissions_electricity** (*numeric*) – total emissions allocated to electricity according to specified *method* [in CO2 equivalents].
- **allocated_emissions_heat** (*numeric*) – total emissions allocated to heat according to specified *method* [in CO2 equivalents].

13.2 concentrating_solar_power module

This module is designed to hold functions which are necessary for the CSP.

This file is part of project oemof (github.com/oemof/oemof-thermal). It's copyrighted by the contributors recorded in the version control history of the file, available from its original location: oemof-thermal/src/oemof/thermal/concentrating_solar_power.py

SPDX-License-Identifier: MIT

oemof.thermal.concentrating_solar_power.**calc_collector_irradiance**(*irradiance_on_collector*, *cleanliness*)

Subtracts the losses of dirtiness from the irradiance on the collector

$E_{coll} = E^*_{coll} \cdot X^{3/2}$

Parameters

- **irradiance_on_collector** (*series of numeric*) – Irradiance which hits collectors surface.
- **x** (*numeric*) – Cleanliness of the collector (between 0 and 1).

Returns **collector_irradiance** – Irradiance on collector after all losses.

Return type series of numeric

oemof.thermal.concentrating_solar_power.**calc_eta_c**(*eta_0*, *c_1*, *c_2*, *iam*, *temp_collector_inlet*, *temp_collector_outlet*, *temp_amb*, *collector_irradiance*, *loss_method*)

Calculates collectors efficiency depending on the loss method

method 'Janotte':

$\eta_C = \eta_0 \cdot \kappa(\Theta) - c_1 \cdot \frac{\Delta T}{E_{coll}} - c_2 \cdot \frac{\Delta T^2}{E_{coll}}$

method 'Andasol':

$\eta_C = \eta_0 \cdot \kappa(\Theta) - \frac{c_1}{E_{coll}}$

Parameters

- **eta_0** (*numeric*) – Optical efficiency of the collector.
- **c_1** (*numeric*) – Thermal loss parameter 1. Required for both loss methods.
- **c_2** (*numeric*) – Thermal loss parameter 2. Required for loss method 'Janotte'.
- **iam** (*series of numeric*) – Incidence angle modifier.
- **temp_collector_inlet** (*numeric, in °C*) – Collectors inlet temperature.
- **temp_collector_outlet** (*numeric, in °C*) – Collectors outlet temperature.
- **temp_amb** (*series of numeric, in °C*) – Ambient temperature.
- **collector_irradiance** (*series of numeric*) – Irradiance on collector after all losses.
- **loss_method** (*string, default 'Janotte'*) – Valid values are: 'Janotte' or 'Andasol'. Describes, how the thermal losses and the incidence angle modifier are calculated.

Returns collectors efficiency

Return type series of numeric

oemof.thermal.concentrating_solar_power.**calc_heat_coll**(*eta_c*, *collector_irradiance*)

$\dot{Q}_{coll} = E_{coll} \cdot \eta_C$

Parameters

- **eta_c** (*series of numeric*) – collectors efficiency.
- **collector_irradiance** (*series of numeric*) – Irradiance on collector after all losses.

Returns collectors heat

Return type series of numeric

oemof.thermal.concentrating_solar_power.**calc_iam**(*a_1*, *a_2*, *a_3*, *a_4*, *a_5*, *a_6*, *aoi*, *loss_method*)

Calculates the incidence angle modifier depending on the loss method

method 'Janotte':

$\kappa(\Theta) = 1 - a_1 \cdot |\Theta| - a_2 \cdot |\Theta|^2$

method 'Andasol':

$\kappa(\Theta) = 1 - a_1 \cdot |\Theta| - a_2 \cdot |\Theta|^2 - a_3 \cdot |\Theta|^3 - a_4 \cdot |\Theta|^4 - a_5 \cdot |\Theta|^5 - a_6 \cdot |\Theta|^6$

Parameters

- **a_2, a_3, a_4, a_5, a_6** (*a_1,*) – Parameters for the incident angle modifier. For loss method 'Janotte' a_1 and a_2 are required, for 'Andasol' a_1 to a_6 are required.
- **aoi** (*series of numeric*) – Angle of incidence.
- **loss_method** (*string, default 'Janotte'*) – Valid values are: 'Janotte' or 'Andasol'. Describes, how the thermal losses and the incidence angle modifier are calculated.

Returns **Incidence angle modifier**

Return type series of numeric

oemof.thermal.concentrating_solar_power.**calc_irradiance**(*surface_tilt*, *surface_azimuth*, *apparent_zenith*, *azimuth*, *irradiance*, *irradiance_method*)

Parameters

- **surface_tilt** (*series of numeric*) – Panel tilt from horizontal.
- **surface_azimuth** (*series of numeric*) – Panel azimuth from north.
- **apparent_zenith** (*series of numeric*) – Solar zenith angle.
- **azimuth** (*series of numeric*) – Solar azimuth angle.
- **irradiance** (*series of numeric*) – Solar irraciance (dni or E_direct_horizontal).
- **irradiance_method** (*str*) – Describes, if the horizontal direct irradiance or the direct normal irradiance is given and used for calculation.

Returns **irradiance_on_collector** – Irradiance which hits collectors surface.

Return type series of numeric

oemof.thermal.concentrating_solar_power.**csp_precalc**(*lat*, *long*, *collector_tilt*, *collector_azimuth*, *cleanliness*, *eta_0*, *c_1*, *c_2*, *temp_collector_inlet*, *temp_collector_outlet*, *temp_amb*, *a_1*, *a_2*, *a_3=0*, *a_4=0*, *a_5=0*, *a_6=0*, *loss_method='Janotte'*, *irradiance_method='horizontal'*, ***kwargs*)

Calculates collectors efficiency and irradiance according to [1] and the heat of the thermal collector. For the calculation of irradiance pvlib [2] is used.

$Q_{coll} = E_{coll} \cdot \eta_C$

functions used

- pvlib.solarposition.get_solarposition
- pvlib.tracking.singleaxis
- calc_irradiance
- calc_collector_irradiance
- calc_iam
- calc_eta_c
- calc_heat_coll

Parameters

- **lat** (*numeric*) – Latitude of the location.
- **long** (*numeric*) – Longitude of the location.
- **collector_tilt** (*numeric*) – The tilt of the collector.

- **collector_azimuth** (*numeric*) – The azimuth of the collector. Azimuth according to pvlib in decimal degrees East of North.
- **cleanliness** (*numeric*) – Cleanliness of the collector (between 0 and 1).
- **a_2, a_3, a_4, a_5, a_6** (*a_1,*) – Parameters for the incident angle modifier. For loss method 'Janotte' a_1 and a_2 are required, for 'Andasol' a_1 to a_6 are required.
- **eta_0** (*numeric*) – Optical efficiency of the collector.
- **c_1** (*numeric*) – Thermal loss parameter 1. Required for both loss methods.
- **c_2** (*numeric*) – Thermal loss parameter 2. Required for loss method 'Janotte'. If loss method 'Andasol' is used, set it to 0.
- **temp_collector_inlet** (*numeric or series with length periods*) – Collectors inlet temperature.
- **temp_collector_outlet** (*numeric or series with length periods*) – Collectors outlet temperature.
- **temp_amb** (*time indexed series*) – Ambient temperature time series.
- **loss_method** (*string, default 'Janotte'*) – Valid values are: 'Janotte' or 'Andasol'. Describes, how the thermal losses and the incidence angle modifier are calculated.
- **irradiance_method** (*string, default 'horizontal'*) – Valid values are: 'horizontal' or 'normal'. Describes, if the horizontal direct irradiance or the direct normal irradiance is given and used for calculation.
- **(depending on irradiance_method)** (*E_dir_hor/dni*) – Irradiance for calculation.

Returns

data – Dataframe containing the following columns

- collector_irradiance
- eta_c
- collector_heat

collector_irradiance is the irradiance which reaches the collector after all losses (incl. cleanliness).

Return type pandas.DataFrame

Comment

Series for ambient temperature and irradiance must have the same length and the same time index. Be aware of the time one.

Proposal of values

If you have no idea, which values your collector have, here are values, which were measured in [1] for a collector: a1: -0.00159, a2: 0.0000977, eta_0: 0.816, c1: 0.0622, c2: 0.00023.

Reference

[1] Janotte, N; et al: Dynamic performance evaluation of the HelioTrough collector demon-stration loop - towards a new benchmark in parabolic trough qualification, SolarPACES 2013

[2] William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. "pvlib python: a python package for modeling solar energy systems." Journal of Open Source Software, 3(29), 884, (2018). https://doi.org/10.21105/joss.00884

13.3 compression_heatpumps_and_chillers module

This module provides functions to calculate compression heat pumps and compression chillers.

This file is part of project oemof (github.com/oemof/oemof-thermal). It's copyrighted by the contributors recorded in the version control history of the file, available from its original location: oemof-thermal/src/oemof/thermal/compression_heatpumps_and_chillers.py

SPDX-License-Identifier: MIT

oemof.thermal.compression_heatpumps_and_chillers.**calc_chiller_quality_grade**(*nominal_conditions*)

Calculates the quality grade for a given point of operation.

Note: This function is rather experimental. Please do not use it to estimate the quality grade of a real machine. A single point of operation might not be representative!

Parameters **nominal_conditions** (*dict*) – Dictionary describing one operating point (e.g., operation under STC) of the chiller by its cooling capacity, its electricity consumption and its COP ('nominal_Q_chill', 'nominal_el_consumption' and 'nominal_cop')

Returns **q_grade** – Quality grade

Return type numerical value

oemof.thermal.compression_heatpumps_and_chillers.**calc_cops**(*mode*, *temp_high*, *temp_low*, *quality_grade*, *temp_threshold_icing=2*, *factor_icing=None*)

Calculates the Coefficient of Performance (COP) of heat pumps and chillers based on the Carnot efficiency (ideal process) and a scale-down factor.

Note: Applications of air-source heat pumps should consider icing at the heat exchanger at air-temperatures around $2°C$. Icing causes a reduction of the efficiency.

Parameters

- **temp_high** (*list or pandas.Series of numerical values*) – Temperature of the high temperature reservoir in $°C$
- **temp_low** (*list or pandas.Series of numerical values*) – Temperature of the low temperature reservoir in $°C$
- **quality_grade** (*numerical value*) – Factor that scales down the efficiency of the real heat pump (or chiller) process from the ideal process (Carnot efficiency), where

  a factor of 1 means teh real process is equal to the ideal one.
- **factor_icing** (*numerical value*) – Sets the relative COP drop caused by icing, where 1 stands for no efficiency-drop.

- **mode** (*string*) – Two possible modes: “heat_pump” or “chiller” (default ‘None’)
- **t_threshold** – Temperature in $°C$ below which icing at heat exchanger occurs (default 2)

Returns **cops** – List of Coefficients of Performance (COPs)

Return type list of numerical values

oemof.thermal.compression_heatpumps_and_chillers.**calc_max_Q_dot_chill**(*nominal_conditions*, *cops*)

Calculates the maximal cooling capacity (relative value) of a chiller.

Note: This function assumes the cooling capacity of a chiller can exceed the rated nominal capacity (e.g., from the technical specification sheet). That means: The value of `max_Q_chill` can be greater than 1. Make sure your actual chiller is capable of doing so. If not, use 1 for the maximal cooling capacity.

Parameters

- **nominal_conditions** (*dict*) – Dictionary describing one operating point (e.g., operation under STC) of the chiller by its cooling capacity, its electricity consumption and its COP (‘nominal_Q_chill’, ‘nominal_el_consumption’ and ‘nominal_cop’)
- **cops** (*list of numerical values*) – Actual COP

Returns **max_Q_chill** – Maximal cooling capacity (relative value). Value is equal or greater than 0 and can be greater than 1.

Return type list of numerical values

oemof.thermal.compression_heatpumps_and_chillers.**calc_max_Q_dot_heat**(*nominal_conditions*, *cops*)

Calculates the maximal heating capacity (relative value) of a heat pump.

This function assumes the heating capacity of a heat pump can exceed the rated nominal capacity (e.g., from the technical specification sheet). That means: The value of `max_Q_hot` can be greater than 1. Make sure your actual heat pump is capable of doing so. If not, use 1 for the maximal heating capacity.

13.4 solar_thermal_collector module

This module is designed to hold functions for calculating a solar thermal collector.

This file is part of project oemof (github.com/oemof/oemof-thermal). It’s copyrighted by the contributors recorded in the version control history of the file, available from its original location: oemof-thermal/src/oemof/thermal/solar_thermal_collector.py

SPDX-License-Identifier: MIT

oemof.thermal.solar_thermal_collector.**calc_eta_c_flate_plate**(*eta_0*, *a_1*, *a_2*, *temp_collector_inlet*, *delta_temp_n*, *temp_amb*, *collector_irradiance*)

Calculates collectors efficiency

$\eta_C = \eta_0 - a_1 \cdot \frac{\Delta T}{E_{coll}} - a_2 \cdot \frac{\Delta T^2}{E_{coll}}$

with

$\Delta T = T_{coll,in} + \Delta T_n - T_{amb}$

Parameters

- **eta_0** (*numeric*) – Optical efficiency of the collector.
- **a_1** (*numeric*) – Thermal loss parameter 1.
- **a_2** (*numeric*) – Thermal loss parameter 2.
- **temp_collector_inlet** (*numeric, in °C*) – Collectors inlet temperature.
- **delta_temp_n** (*numeric*) – Temperature difference between collector inlet and mean temperature.
- **temp_amb** (*series of numeric, in °C*) – Ambient temperature.
- **collector_irradiance** (*series of numeric*) – Irradiance on collector after all losses.

Returns **eta_c** – collectors efficiency

Return type series of numeric

oemof.thermal.solar_thermal_collector.**flat_plate_precalc**(*lat*, *long*, *collector_tilt*, *collector_azimuth*, *eta_0*, *a_1*, *a_2*, *temp_collector_inlet*, *delta_temp_n*, *irradiance_global*, *irradiance_diffuse*, *temp_amb*)

Calculates collectors heat, efficiency and irradiance of a flat plate collector.

$\dot{Q}_{coll} = E_{coll} \cdot \eta_C$

Parameters

- **lat** (*numeric*) – Latitude of the location.
- **long** (*numeric*) – Longitude of the location.
- **collector_tilt** (*numeric*) – The tilt of the collector.
- **collector_azimuth** (*numeric*) – The azimuth of the collector. Azimuth according to pvlib in decimal degrees East of North.
- **eta_0** (*numeric*) – Optical efficiency of the collector.
- **a_2** (*a_1,*) – Thermal loss parameters.
- **temp_collector_inlet** (*numeric or series with length of periods*) – Collectors inlet temperature.
- **delta_temp_n** (*numeric*) – Temperature difference between collector inlet and mean temperature.
- **irradiance_global** (*time indexed series*) – Global horizontal irradiance.
- **irradiance_diffuse** (*time indexed series*) – Diffuse irradiance.
- **temp_amb** (*time indexed series*) – Ambient temperature.

Returns

data – DataFrame containing the followiing columns:

- col_ira: The irradiance on the tilted collector.
- eta_c: The efficiency of the collector.
- collector_heat: The heat power output of the collector.

Return type pandas.DataFrame

13.5 stratified_thermal_storage module

This module is designed to hold functions for calculating stratified thermal storages.

This file is part of project oemof (github.com/oemof/oemof-thermal). It's copyrighted by the contributors recorded in the version control history of the file, available from its original location: oemof-thermal/src/oemof/thermal/stratified_thermal_storage.py

SPDX-License-Identifier: MIT

oemof.thermal.stratified_thermal_storage.**calculate_capacities**(*volume*, *temp_h*, *temp_c*, *heat_capacity=4195.52*, *density=971.803*)

Calculates the nominal storage capacity, minimum and maximum storage level of a stratified thermal storage.

$Q_N = V \cdot c \cdot \rho \cdot (T_H - T_C)$

Parameters

- **volume** (*numeric*) – Volume of the storage [m3]
- **temp_h** (*numeric*) – Temperature of hot storage medium [deg C]
- **temp_c** (*numeric*) – Temperature of cold storage medium [deg C]
- **heat_capacity** (*numeric*) – Average specific heat capacity of storage medium [J/(kg*K)] Default values calculated with CoolProp for a temperature of 80 °C as a simplifying assumption
- **density** (*numeric*) – Average density of storage medium [kg/m3] Default values calculated with CoolProp for a temperature of 80 °C as a simplifying assumption

Returns **nominal_storage_capacity** – Maximum amount of stored thermal energy [MWh]

Return type numeric

oemof.thermal.stratified_thermal_storage.**calculate_losses**(*u_value*, *diameter*, *temp_h*, *temp_c*, *temp_env*, *time_increment=1*, *heat_capacity=4195.52*, *density=971.803*)

Calculates loss rate and fixed losses for a stratified thermal storage.

$\beta = U \frac{4}{d \rho c} \Delta t$

$\gamma = U \frac{4}{d \rho c \Delta T_{HC}} \Delta T_{C0} \Delta t$

$\delta = U \frac{\pi d^2}{4} \Big(\Delta T_{H0} + \Delta T_{C0} \Big) \Delta t$

Parameters

- **u_value** (*numeric*) – Thermal transmittance of storage envelope [W/(m2*K)]
- **diameter** (*numeric*) – Diameter of the storage [m]
- **temp_h** (*numeric*) – Temperature of hot storage medium [deg C]
- **temp_c** (*numeric*) – Temperature of cold storage medium [deg C]
- **temp_env** (*numeric*) – Temperature outside of the storage [deg C]
- **time_increment** (*numeric*) – Time increment of the `oemof.solph.EnergySystem` [h]
- **heat_capacity** (*numeric*) – Average specific heat capacity of storage medium [J/(kg*K)] Default values calculated with CoolProp for a temperature of 80 °C as a simplifying assumption
- **density** (*numeric*) – Average density of storage medium [kg/m3] Default values calculated with CoolProp for a temperature of 80 °C as a simplifying assumption

Returns

- **loss_rate** (*numeric (sequence or scalar)*) – The relative loss of the storage capacity between two consecutive timesteps [-]
- **fixed_losses_relative** (*numeric (sequence or scalar)*) – Losses independent of state of charge between two consecutive timesteps relative to nominal storage capacity [-]
- **fixed_losses_absolute** (*numeric (sequence or scalar)*) – Losses independent of state of charge and independent of nominal storage capacity between two consecutive timesteps [MWh]

`oemof.thermal.stratified_thermal_storage.`**calculate_storage_dimensions**(*height*, *diameter*)

Calculates volume and total surface of a hot water storage.

$V = \pi \frac{d^2}{4} \cdot h$

$A = \pi dh + \pi \frac{d^2}{2}$

Parameters

- **height** (*numeric*) – Height of the storage [m]
- **diameter** (*numeric*) – Diameter of the storage [m]

Returns

- **volume** (*numeric*) – Volume of storage
- **surface** (*numeric*) – Total surface of storage [m2]

`oemof.thermal.stratified_thermal_storage.`**calculate_storage_u_value**(*s_iso*, *lamb_iso*, *alpha_inside*, *alpha_outside*)

Calculates the thermal transmittance (U-value) of a thermal storage.

$U = \frac{1}{\frac{1}{\alpha_i} + \frac{s_{iso}}{\lambda_{iso}} + \frac{1}{\alpha_a}}$

Parameters

- **s_iso** (*numeric*) – Thickness of isolation layer [mm]

- **lamb_iso** (*numeric*) – Thermal conductivity of isolation layer [W/(m*K)]
- **alpha_inside** (*numeric*) – Heat transfer coefficient at the inner surface of the storage [W/(m2*K)]
- **alpha_outside** (*numeric*) – Heat transfer coefficient at the outer surface of the storage [W/(m2*K)]

Returns **u_value** – Thermal transmittance (U-value) [W/(m2*K)]

Return type numeric

13.6 facades module

Adapted from oemof.tabular's facades

Facade's are classes providing a simplified view on more complex classes. More specifically, the *Facade* s in this module inherit from *oemof.solph*'s generic classes to serve as more concrete and energy specific interface.

The concept of the facades has been derived from oemof.tabular. The idea is to be able to instantiate a *Facade* using only keyword arguments. Under the hood the *Facade* then uses these arguments to construct an *oemof.solph* component and sets it up to be easily used in an `EnergySystem`. Usually, a subset of the attributes of the parent class remains while another part can be addressed by more specific or simpler attributes.

Note The mathematical notation is as follows:

- Optimization variables (endogenous) are denoted by x
- Optimization parameters (exogenous) are denoted by c
- The set of timesteps T describes all timesteps of the optimization problem

SPDX-License-Identifier: MIT

class oemof.thermal.facades.**Facade**(**args*, ***kwargs*)

Bases: `oemof.network.network.Node`

Parameters **_facade_requires_** (*list of str*) – A list of required attributes. The constructor checks whether these are present as keyword arguments or whether they are already present on self (which means they have been set by constructors of subclasses) and raises an error if he doesn't find them.

update()

class oemof.thermal.facades.**ParabolicTroughCollector**(**args*, ***kwargs*)

Bases: `oemof.solph.network.transformer.Transformer`, *oemof.thermal.facades.Facade*

Parabolic trough collector unit

Parameters

- **heat_bus** (*oemof.solph.Bus*) – An oemof bus instance in which absorbs the collectors heat.
- **electrical_bus** (*oemof.solph.Bus*) – An oemof bus instance which provides electrical energy to the collector.
- **electrical_consumption** (*numeric*) – Specifies how much electrical energy is used per provided thermal energy.
- **additional_losses** (*numeric*) – Specifies how much thermal energy is lost in peripheral parts like pipes and pumps.

- **aperture_area** (*numeric*) – Specify the ares or size of the collector.

See the API of csp_precalc in oemof.thermal.concentrating_solar_power for the other parameters.

Examples

```
>>> from oemof import solph
>>> from oemof.thermal.facades import ParabolicTroughCollector
>>> bth = solph.Bus(label='thermal_bus')
>>> bel = solph.Bus(label='electrical_bus')
>>> collector = ParabolicTroughCollector(
...     label='solar_collector',
...     heat_bus=bth,
...     electrical_bus=bel,
...     electrical_consumption=0.05,
...     additional_losses=0.2,
...     aperture_area=1000,
...     loss_method='Janotte',
...     irradiance_method='horizontal',
...     latitude=23.614328,
...     longitude=58.545284,
...     collector_tilt=10,
...     collector_azimuth=180,
...     x=0.9,
...     a_1=-0.00159,
...     a_2=0.0000977,
...     eta_0=0.816,
...     c_1=0.0622,
...     c_2=0.00023,
...     temp_collector_inlet=435,
...     temp_collector_outlet=500,
...     temp_amb=input_data['t_amb'],
...     irradiance=input_data['E_dir_hor']
... )
```

build_solph_components()

class oemof.thermal.facades.**SolarThermalCollector**(**args*, ***kwargs*)

Bases: oemof.solph.network.transformer.Transformer, *oemof.thermal.facades.Facade*

Solar thermal collector unit

heat_out_bus: oemof.solph.Bus An oemof bus instance which absorbs the collectors heat.

electrical_in_bus: oemof.solph.Bus An oemof bus instance which provides electrical energy to the collector.

electrical_consumption: numeric Specifies how much electrical energy is used per provided thermal energy.

peripheral_losses: numeric Specifies how much thermal energy is lost in peripheral parts like pipes and pumps as percentage of provided thermal energy.

aperture_area: numeric Specifies the size of the collector as surface area.

See the API of flat_plate_precalc in oemof.thermal.solar_thermal_collector for the other parameters.

```
>>> from oemof import solph
>>> from oemof.thermal.facades import SolarThermalCollector
>>> bth = solph.Bus(label='thermal')
```

(continues on next page)

(continued from previous page)

```
>>> bel = solph.Bus(label='electricity')
>>> collector = SolarThermalCollector(
...     label='solar_collector',
...     heat_out_bus=bth,
...     electricity_in_bus=bel,
...     electrical_consumption=0.02,
...     peripheral_losses=0.05,
...     aperture_area=1000,
...     latitude=52.2443,
...     longitude=10.5594,
...     collector_tilt=10,
...     collector_azimuth=20,
...     eta_0=0.73,
...     a_1=1.7,
...     a_2=0.016,
...     temp_collector_inlet=20,
...     delta_temp_n=10,
...     irradiance_global=input_data['global_horizontal_W_m2'],
...     irradiance_diffuse=input_data['diffuse_horizontal_W_m2'],
...     temp_amb=input_data['temp_amb'],
)
```

build_solph_components()

class oemof.thermal.facades.**StratifiedThermalStorage**(*label=None*, *inputs=None*, *outputs=None*, *nominal_storage_capacity=None*, *initial_storage_level=None*, *investment=None*, *invest_relation_input_output=None*, *invest_relation_input_capacity=None*, *invest_relation_output_capacity=None*, *min_storage_level=0.0*, *max_storage_level=1.0*, *balanced=True*, *loss_rate=0*, *fixed_losses_relative=0*, *fixed_losses_absolute=0*, *inflow_conversion_factor=1*, *outflow_conversion_factor=1*, *custom_attributes=None*, ***kwargs*)

Bases: oemof.solph.components.generic_storage.GenericStorage, *oemof.thermal.facades.Facade*

Stratified thermal storage unit.

Parameters

- **bus** (*oemof.solph.Bus*) – An oemof bus instance where the storage unit is connected to.
- **diameter** (*numeric*) – Diameter of the storage [m]
- **height** (*numeric*) – Height of the storage [m]
- **temp_h** (*numeric*) – Temperature of the hot (upper) part of the water body.

- **temp_c** (*numeric*) – Temperature of the cold (upper) part of the water body.
- **temp_env** (*numeric*) – Temperature of the environment.
- **heat_capacity** (*numeric*) – Assumed constant for heat capacity of the water.
- **density** (*numeric*) – Assumed constant for density of the water.
- **u_value** (*numeric*) – Thermal transmittance [W/(m2*K)]
- **capacity** (*numeric*) – Maximum production capacity [MW]
- **efficiency** (*numeric*) – Efficiency of charging and discharging process: Default: 1
- **marginal_cost** (*numeric*) – Marginal cost for one unit of output.
- **expandable** (*boolean*) – True, if capacity can be expanded within optimization. Default: False.
- **storage_capacity_cost** (*numeric*) – Investment costs for the storage unit [Eur/MWh].
- **capacity_cost** (*numeric*) – Investment costs for charging/dischargin [Eur/MW]
- **storage_capacity_potential** (*numeric*) – Potential of the investment for storage capacity [MWh]
- **capacity_potential** (*numeric*) – Potential of the investment for capacity [MW]
- **input_parameters** (*dict (optional)*) – Set parameters on the input edge of the storage (see oemof.solph for more information on possible parameters)
- **output_parameters** (*dict (optional)*) – Set parameters on the output edge of the storage (see oemof.solph for more information on possible parameters)

The attribute `nominal_storage_capacity` of the base class `GenericStorage` should not be passed because it is determined internally from `height` and `parameter`.

Examples

```
>>> from oemof import solph
>>> from oemof.thermal.facades import StratifiedThermalStorage
>>> heat_bus = solph.Bus(label='heat_bus')
>>> thermal_storage = StratifiedThermalStorage(
...     label='thermal_storage',
...     bus=heat_bus,
...     diameter=10,
...     height=10,
...     temp_h=95,
...     temp_c=60,
...     temp_env=10,
...     u_value=0.3,
...     initial_storage_level=0.5,
...     min_storage_level=0.05,
...     max_storage_level=0.95
...     capacity=1)
```

build_solph_components()

oemof.thermal.facades.**add_subnodes**(*n*, ***kwargs*)

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

S

U