Demand response modeling in oemof.solph

(J. Kochems)

Overview and tutorial on the extended demand response implementation in oemof.solph

Before we get started - Prerequisites

- In order to run the tutorial examples, the following ***prerequisites*** need to be met:
  - You have **Anaconda or Miniconda** installed on your computer or another Python installation with pip package manager
  - You have a **solver** installed that is supported by oemof
    - *Note: I'm using gurobi for the examples. This is a commercial solver which provides free academic licenses.*
    - *You need to change the solver variable to the (open access) solver you are using if you are not a gurobi user.*
    - *oemof recommends to use the CBC solver.*
  - *You have some sort of **git command line tool** available, such as GitBash*
- *Note: We will not have time to do this entire setup. If you are lacking anything of the above, try to follow along and work the tutorial through on your own afterwards if you like. Feel free to contact me afterwards if you are experiencing any trouble.*
- *Note: I'm a Windows user and can provide you with the Windows information only. The setup on Linux or Mac may differ.*

Getting started (with conda)

- To get started clone the repository
  - Open a git command line tool, such as GitBash
  - Navigate to the target folder, e.g.

Change with the (absolute) path you want

cd C:/Users/johannes/DR_mod

  - Type the following command

git clone https://github.com/jokochems/DR_modeling_tutorial.git

- Create a new repo from the environment.yml file
  - Open Anaconda Powershell Prompt
  - Navigate to the local repository folder, e.g.

Change with your (absolute) path + /DR_modeling_tutorial at the end

cd C:/Users/johannes/DR_mod/DR_modeling_tutorial

  - Type the following command

conda env create –f environment.yml

- Activate the environment and open jupyterlab
  - Activate the environment

conda activate dr-modeling

  - Open jupyterlab

jupyter lab

Getting started (with pip)

- To get started clone the repository
  - Open a git command line tool, such as GitBash
  - Navigate to the target folder, e.g.

Change with the (absolute) path you want

cd C:/Users/johannes/DR_mod

  - Type the following command

git clone https://github.com/jokochems/DR_modeling_tutorial.git

- Install packages from the requirements.txt file
  - Open a command line tool
  - Navigate to the local repository folder, e.g.

Change with your (absolute) path + /DR_modeling_tutorial at the end

cd C:/Users/johannes/DR_mod/DR_modeling_tutorial

  - Type the following command

Unix or MacOS: ***python -m pip install -r requirements.txt***
Windows: ***py -m pip install -r requirements.txt***

  - Open jupyterlab

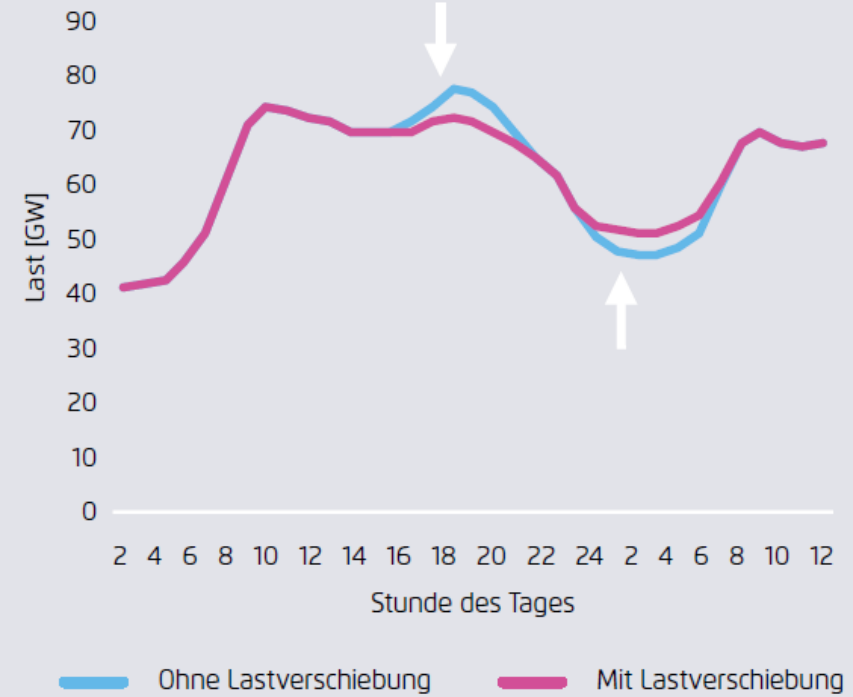
jupyter lab

Outline

1. **Introduction**
2. Changelog
3. Examples & Tutorial

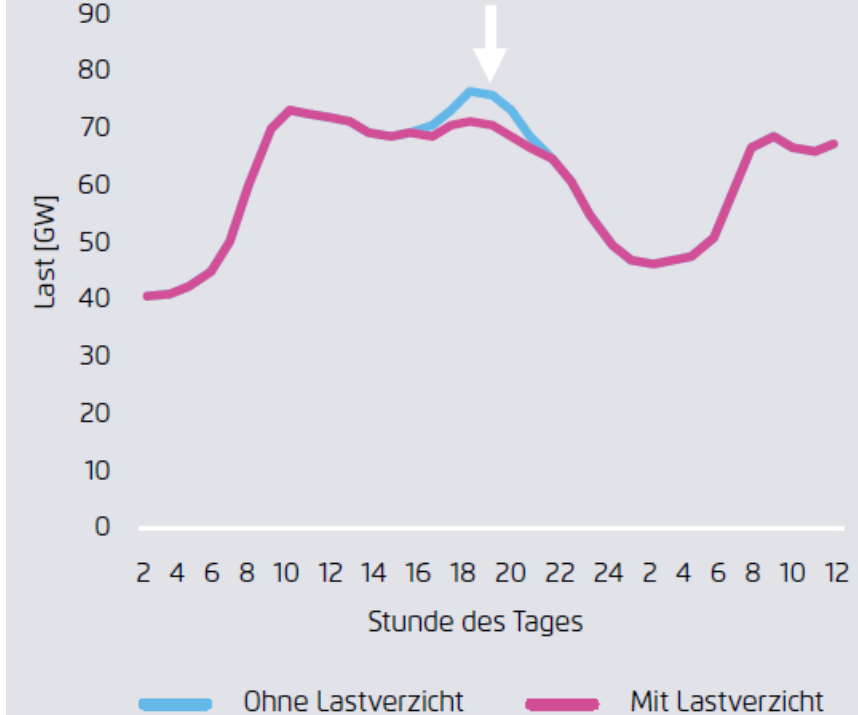
Demand response – load shifting and load shedding

Load shifting



- Forced balancing of loads within given time window
- Usual prerequisite: some sort of (functional) storage

Load shedding

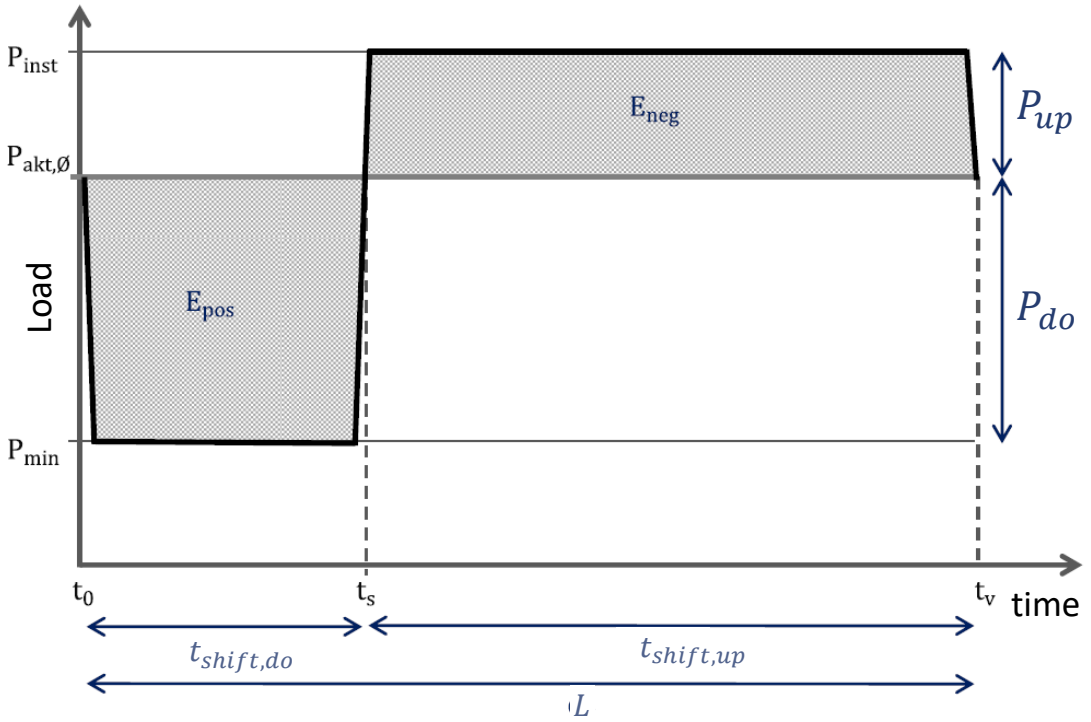


- No balancing
- Usually high costs

Source: Connect (2015), S. 19

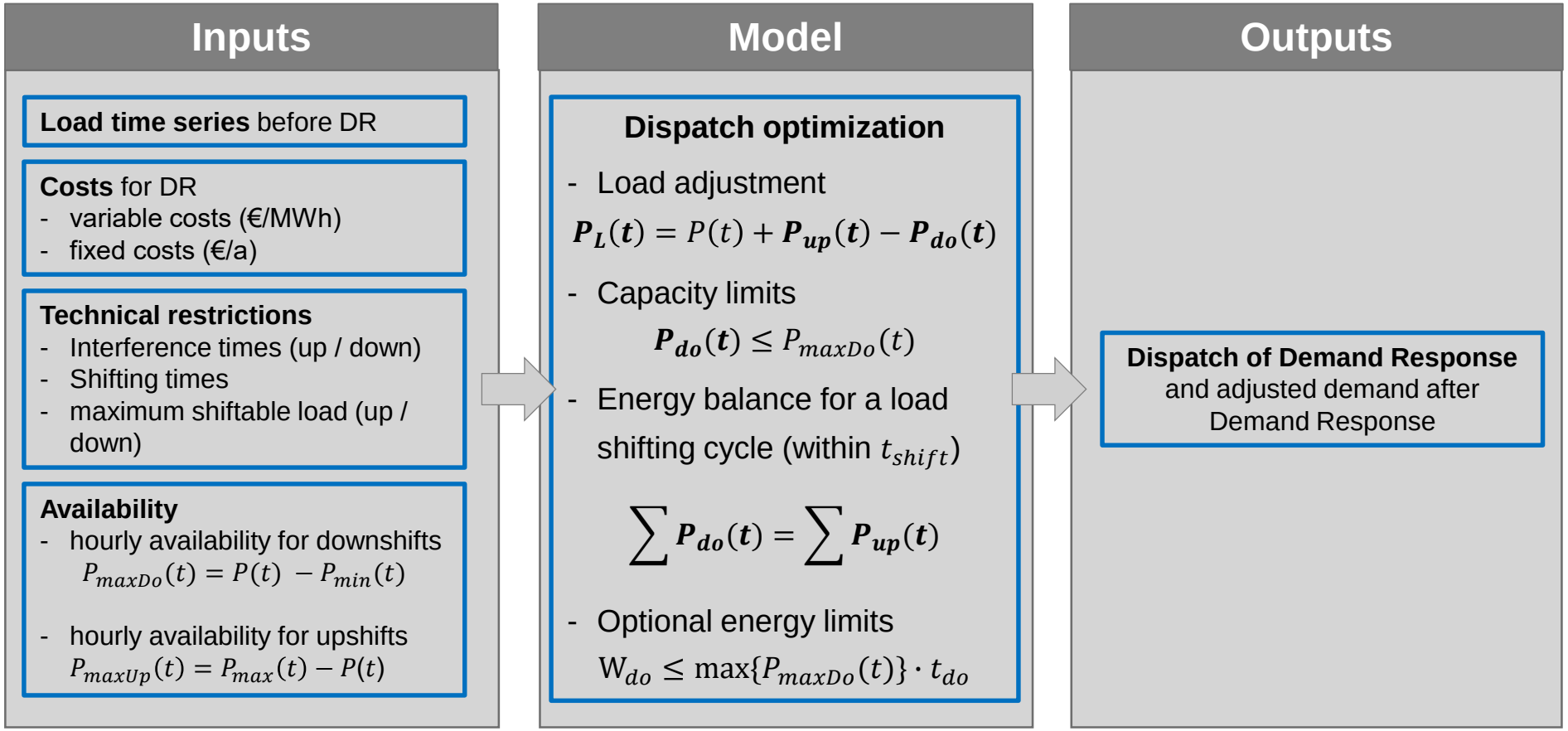
Small terminology for load shifting

- Definitions of **temporal terms for load shifting**
  - **delay time** L or t_{shift}:
    Duration of time until the amount of energy must be completely balanced (parameter)
  - Interference time **/ shift time** d_S or $t_{interference}$:
    Interference time of the load shifting in <u>one direction</u> (parameter)



sources: Steurer (2017), p. 56; modifications from Gils (2015), pp. 13-14; Zerrahn and Schill (2015a), p. 845

General approach for modeling demand response in a linear (power) system model



- **Variables: bold print**
- Parameters: normal print
- Parameters / variables for upshifts are analogous to downshift ones

t :	timestep
$P(t)$:	demand before DR
$P_{min}(t)$:	minimum demand
$P_{maxDo}(t)$:	maximum availability for downshift

$\boldsymbol{P_L(t)}$	demand after DR
$\boldsymbol{P_{do}(t)}$:	downshift
W_{do}:	energy limit for downshift
t_{shift} :	shifting time (syn.: delay time)
t_{do}:	interference duration for downshift

How to access? (If not using the tutorial setup anyways)

- Feature will be available from v0.4.2 on (oemof.solph PR #740)

- For now, you have to install the developer version of oemof.solph
  - Set up a clean conda environment (See this conda blog post if you are new to this)
  - Open Anaconda (Powershell) Prompt activate your environment and type

pip install https://github.com/oemof/oemof-solph/archive/dev.zip

- Now you have the developer version and the feature available. Have fun! ☺

Outline

1. Introduction

2. **Changelog**

3. Examples & Tutorial

Starting point: Prior demand response implementation

- There already existed an implementation for demand response from v0.3.2 on (written by Julian Endres and Guido Pleßmann and reviewed by Uwe Krien and Patrick Schönfeldt; see this PR)

- This implementation supported two implementation for load shifting:
  - An approach from Zerrahn & Schill (2015), both working at DIW which was contained in the method `delay`.
  - An own simple approach which only demands for the load to be balanced within a given interval contained in the method `interval`.

- The implementation – while being a really great, well-documented and tested contribution (!) – did fall short on:
  - Considering load shedding,
  - Allowing for investments in demand response and
  - Parameters which characterize other details of load shifting processes, such as the shift time and a potential recovery time.

Changes made

- **Enhanced custom SinkDSM**

  - Renamed keyword argument `method` to `approach`
  - Renamed approaches `interval` to `oemof` and `delay` to `DIW`

  Renaming / minor changes

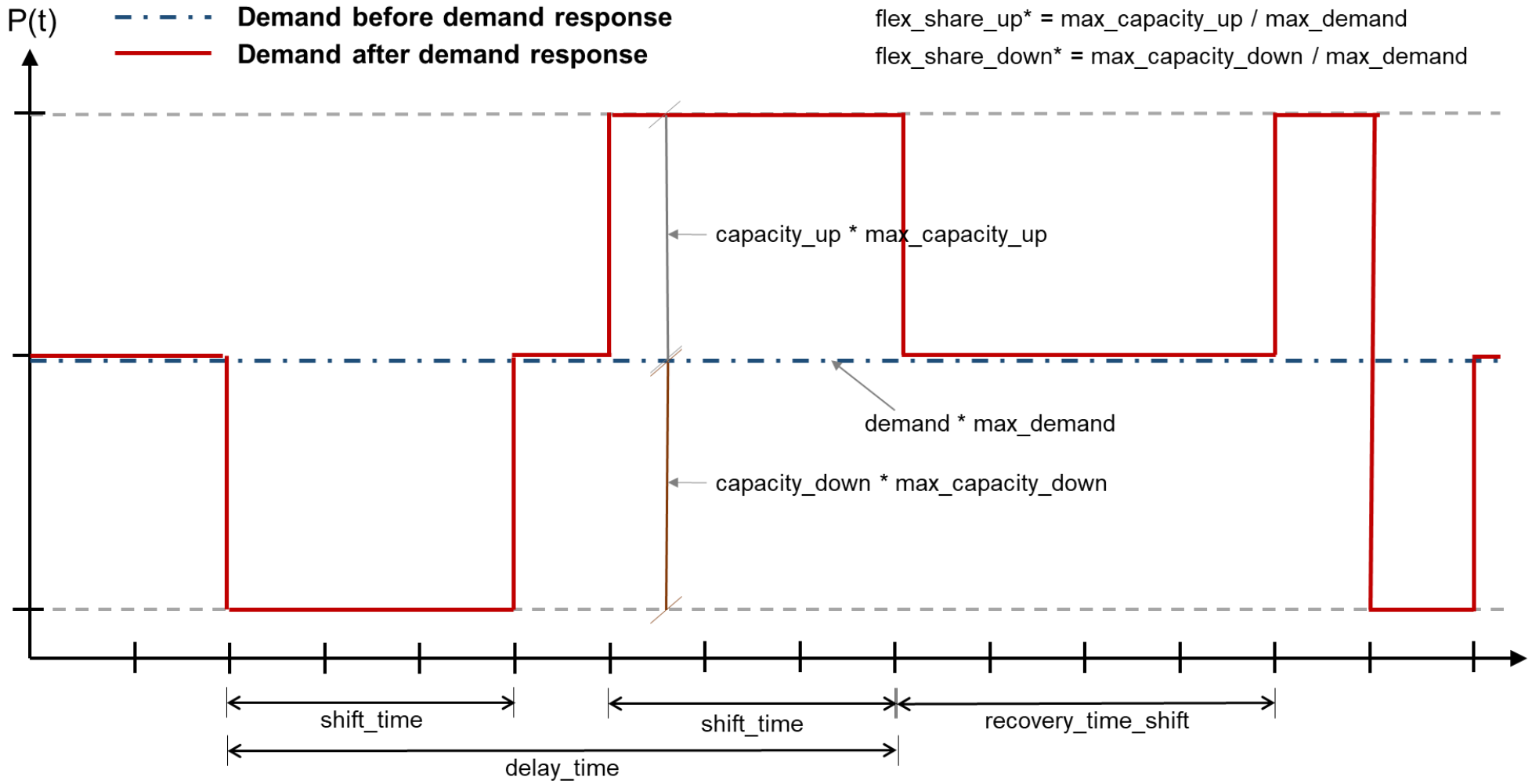
  - Added modeling approach `DLR` (PhD thesis of Hans Christian Gils 2015)
  - Included load shedding
  - Introduced `recovery_time` in `DIW` approach
  - Introduced `shift_time` and other parameters for `DLR` approach

  New approach + load shedding
  & enhancements for existing approaches

  - Included investments in DSM
  - Normalized keyword arguments `demand`, `capapcity_up` and `capacity_down`

  Investment possibility and
  followng adjustments

Parameters



* Either explicitly set or calculated. But specifying flex shares and max capacities at the same time leads to overdetermination.

Further information

- Detailed information on the modeling approaches used can be found in the docstring documentation of the component.
  - https://github.com/oemof/oemof-solph/blob/dev/src/oemof/solph/custom/sink_dsm.py

- In addition to that, two of the three approaches have been compared in a presentation given at the INREC 2020
  - Presentation: https://github.com/jokochems/DR_modeling_oemof/blob/master/Kochems_Demand_Response_INREC.pdf
  - Repository & Jupyter Notebook: https://github.com/jokochems/DR_modeling_oemof/tree/master/INREC_examples

Outline

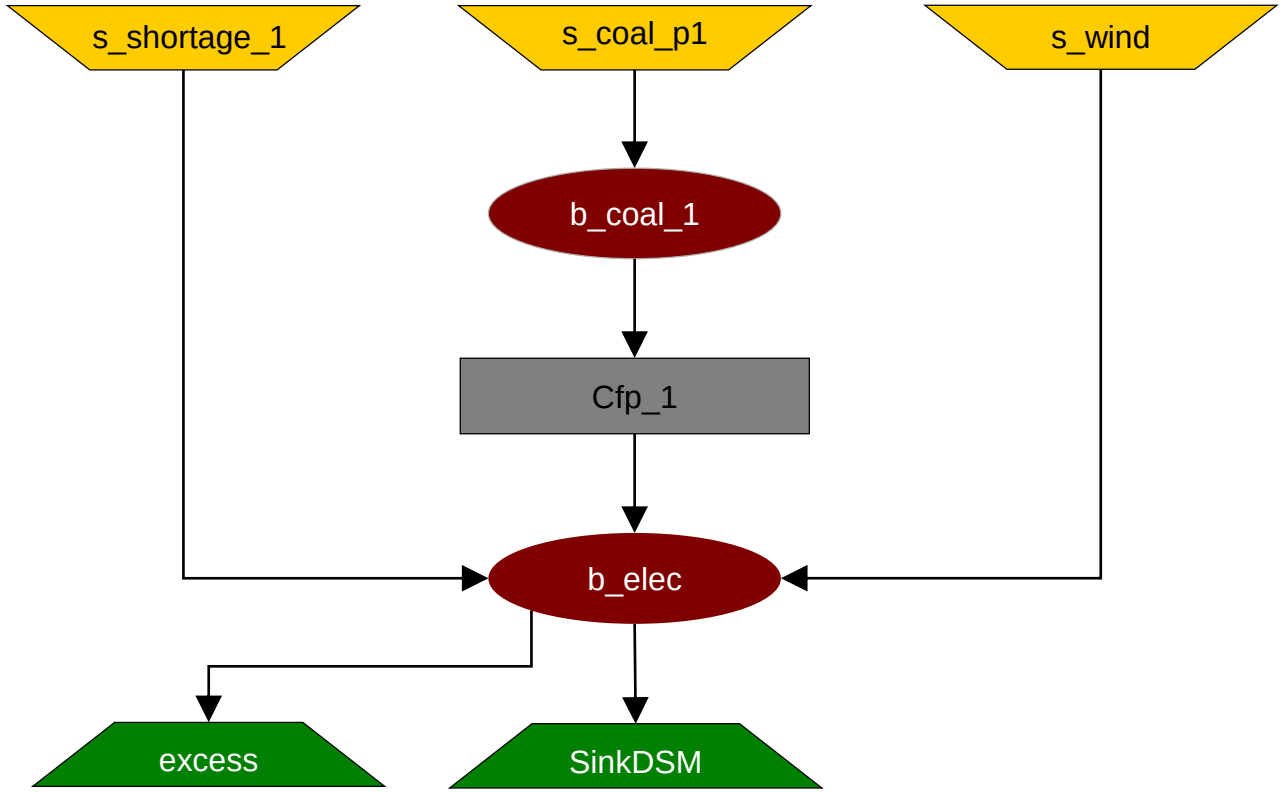
1. Introduction
2. Changelog
3. **Examples & Tutorial**

Examples and tutorial

- Open jupyterlab and navigate to the local folder where you have cloned the repository to.
- Open the jupyter notebook file: ***dsm_modeling_tutorial.ipynb***
- If you haven‘t been succesful to set it up, you can find it here: https://github.com/jokochems/DR_modeling_tutorial/blob/main/dsm_modeling_tutorial.ipynb

- Now you have the examples in front of you. There are three examples:
    - Example 1: Build a simple energy system and use **one approach** (the `DLR` approach) for DSM modeling
    - Example 2: Build a simple energy system and use **all approaches in a for-loop** (can be used for a comparison)
    - Example 3: Build a simple energy system and determe **investments** in DSM.

The toy energy system



Outlook – What to expect?

- **Other approaches**
  - Based on my INREC presentation, I came to the conclusing that the other implementations I considered there won‘t bring along significant improvement to users:
    - The IER approach is quite hard to parameterize without doing some upfront residual load analysis (which requires you to have perfect foresight).
    - The TUD approach basically is some kind of extension of the `oemof` approach implemented. While being computationally efficient, it brings along some drawbacks

- **Code reduction & revision**
  - I‘m having a look on whether it would be possible to reduce the code and pretty it up a little bit.
  - There is some parameterization difficulty when it comes to investment modeling. Maybe there is a way to make this easier.

- **Model & multi-period integration**
  - The implementation is used in POMMES for both, investment and dispatch modeling. The dispatch variant of POMMES will be published somewhat soon (I‘m on it ;-)).
  - There is a pending PR of mine on enabling oemof.solph to run multi-period models (i.e. allow for investments in every period instead of once and track lifetimes). The SinkDSM will allow to do so.

Thank you!

- A special thank you goes to Julian Endres and Guido Pleßmann who provided the basic implementation which has been build upon. Parts of the tutorial are based on their code basis.

- A warm thank you goes out to the oemof-dev community for their fruitful advice on this feature, esp. Patrick Schönfeldt for the extensive code review and comments.

- Another special thank you goes to Yannick Werner for his advice on certain implementation issues.

Contact

Johannes Kochems

research associate at DLR, Institute of Networked Energy Systems, Stuttgart |
PhD candidate at Technical University of Berlin

E-Mail:
johannes.kochems[at]dlr.de
kochems[at]campus.tu-berlin.de

GitHub: jokochems