oemof tabular - python package for reproducible workflows in energy systm modelling

Simon Hilpert, Martin Söthe, Stephan Günther

June, 2019

Background

Energy systems modelling requires versatile tools to model systems with different levels of accuracy and detail. In this regard a major part of is the data handling including input collection, processing and result analysis. There is yet no standardized or custom broadly used model-agnostic data container in the scientific field of energy system modelling to hold energy system related data. To enable transparency and reproducibility as well as reusability of existing data, the following data model description has been developed to store energy system related data in the datapackage format.

The Open Energy Modelling Framework (oemof) is a powefull tool for modelling energy systems. The functionalties range from large linear programming market models to detailed MILP heating system or battery models to assess profitablilty of plants in current and future market environments. The undelying concept and its generic implementation allows for a very versatile modelling. Most oemof components are rather of an abstract type like for example 'LinearTransformer' which can be used to model different energy system components.

For building large energy system models, data is of often stored in tabular data format (for example csv, databases, xlsx). Despite the powerful package, instantiating oemof solph models from tabular data sources requires major knowledge of the package and resources to build in/out data processing.

Facade concept

To enbale for a standard input format so called *facade* concept for oemof solph component has been applied. Facade classes in this context come with multiple advantages:

- Facades allow to instantiate models from two dimenisonal data sources as they provide a simplified interface to complex underlying structures.
- This simplified and thus restricted, less generic mathematical representation leading to more transparent modelling.
- In addtion it allows to build an interface for composed components that are constructed on the basis of multiple oemof solph objects.

As they are subclasses of oemof solph components, facades can also be mixed with all their more generic parent class objects in a model.

Data model

The datamodel is extendable and could be applied for various frameworks (PyPSA, calliope, etc.). However, currenty the implementation for reading datapackages is limited to oemof-tabular classes.

Facades require specific attributes. For all facades the attribute carrier, 'tech' and 'type' need to be set. The type of the attribute is string, therefore you can choose string for these. However, if you want to leverage full postprocessing functionality we recommend using one of the types listed below

Carrier types

- solar, wind, biomass, coal, lignite, uranium, oil, gas, hydro, waste, electricity, heat, other

Tech types

- st, ocgt, ccgt, ce, pv, onshore, offshore, ror, rsv, phs, ext, bp, battery

We recommend use the following naming convention for your facade names bus-carrier-tech-number, for example: DE-gas-ocgt-1.

Datapackage

To construct a model based on the datapackage the following 2 steps are required:

1. **Add the topology of the energy system based on the components and their exogenous model variables** to csv-files in the datapackage format.
2. **Create a python script to construct the energy system and the model from** that data.

We recommend a specific workflow to allow to publish your scenario (input data, assumptions, model and results) altogether in one consistent block based on the datapackage standard (see: Reproducible Workflows).

How to create a Datapackage

We adhere to the frictionless (tabular) datapackage standard. On top of that structure we add our own logic. We require at least two things:

1. A directory named *data* containing at least one sub-folder called *elements* (optionally it may contain a directory *sequences* and *geometries*. Of course you may add any other directory, data or other information.)
2. A valid meta-data .json file for the datapackage

The resulting tree of the datapackage could for example look like this:

```
|-- datapackage
    |-- data
        |-- elements
            |-- demand.csv
            |-- generator.csv
            |-- storage.csv
            |-- bus.csv
        |-- sequences
    |-- scripts
    |-- datapackage.json
```

Inside the datapackage, data is stored in so called resources. For a tabular-datapackage, these resources are CSV files. Columns of such resources are referred to as *fields*. In this sense field names of the resources are equivalent to parameters of the energy system elements and sequences.

To distinguish elements and sequences these two are stored in sub-directories of the data directory. In addition geometrical information can be stored under data/geometries in a .geojson format. To simplifiy the process of creating and processing a datapackage the package also comes with several funtionalities for building datapackages from raw data sources.

Components and mathematical description

In the following a mathematical description for the implemented components is given. We only provide the formulations for fixed installed capacities, i.e. dispatch models. As the package is continounsly developed, the most up-to-date mathematical representation will be found in the documentation. Therefore the full set of equations can be obtained by careful inspection of the oemof tabular documentation and in addtion the oemof solph documentation.

However, the following description gives a compact overview about basic functionalities and the notation used inside the documentation.

Reservoir

Volatile

Dispatchable

The mathematical representations for this components are dependent on the user defined attributes. If the capacity is fixed before (**dispatch mode**) the following equation holds:

$$x_{dispatchable}^{flow}(t) \leq c_{dispatchable}^{capacity} \cdot \ c_{dispatchable}^{profile} \qquad \forall t \in T$$

Where $x_{dispatchable}^{flow}$ denotes the production (endogenous variable) of the dispatchable object to the bus.

Conversion

Conversion components have one input and one output and can thus be used to model power plants as well as with all other conversion processes with a constant efficiencies.

$$x_{conversion}^{flow,input}(t) = c_{conversion}^{efficiency}(t) \cdot \ x_{conversion}^{flow,output}(t) \qquad \forall t \in T$$

$$c_{dispatchable)}^{flow,output}(t) \leq c_{conversion}^{capacity} quad \forall t \in T$$

Link

Backpressure Turbine

Extraction Turbine

Addtional functionalities

Temporal aggregation

Writing results

Building datapackages

Reproducible Workflows

Reproduciblility is a recurring point of discussions in the energy system modelling community. Based on the presented software package we propose the following workflow to build reproducible models.

The starting point of this workflow is the folder strucutre:

```
|-- model
    |-- environment
        |--requirements.txt
    |-- raw-data
    |-- scenarios
        |--scenario1.toml
        |--scenatio2.toml
        |-- ...
    |-- scripts
        |--create_input_data.py
        |--compute.py
        |-- ...
    |-- results
        |--scenario1
            |--input
            |--output
         |-- scenario2
            |--input
            |--ouput
```

The raw-data directory contains all input data files required to build the input datapckages for your modelling. The scenatios directory allows you to specify different scenarios and describe them in a basic way. The scripts inside the scripts directory will build input data for your scenarios from the .toml files and the raw-data. In addition the script to compute the models can be stored there.

Of course the structure may be adapted to your needs. However you should provide all this data when publishing results.

Conclusion