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

Simon Hilpert, Martin Söthe, Stephan Günther

June, 2019

Background

Energy system analysis of future energy systems requires tools to increasingly complex systems. In recent years many open source energy (system) models have been developed to cope with challenges of modelling as well as scientific standards in this research field.

For energy system modellers, data handling including input collection, processing and result analysis is one of the most timeconsuming parts. Yet, there is no standardized or custom broadly used model-agnostic data container in the scientific field of energy system modelling to hold energy system related data. However, some formats are widely used and thus tools for analysis and visualization have been designed (IPPC standard by MESSAGE/IIASA).

To improve reproducibility of model results and reusability of existing data, the following data model description has been developed. Energy system related data is stored in the datapackage format. The data model has been implemented in the python package *oemof-tabular* that is based on the Open Energy Modelling Framework (oemof).

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

The facade concept has been applied to allow for oemof solph component has in combination with tabular data sources.

TODO: Short general facade concept (1-3 sentences + one citetation)

Facade classes in the context of `oemof tabular` 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 Implementation

The datapackage holds the topology and parameters of an energy system model instance. At a minimum this includes all values exogenous model variables

and associated meta data. However, it may also include raw data, scripts for processing raw data etc.

For datapackages the frictionless datapackage standard exists.[CITE] On top of this structure we add our own energy system specific logic.

Here 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*.
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.

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.

To facilitate the process of creating datapackages from raw data, compute models and process results the *oemof-tabular* package provides a set of functionalities:

- `oemof.tabular.datapackage.building` contains functions to infer meta data, download raw data, read and write elements, sequences etc.
- `oemof.tabular.datapackage.processing` contains functions to process model results that can be used in the `compute.py` script.
- `oemof.tabular.datapackage.aggregation` allows to aggregate time-series to reduce model complexity.