A Short Description of the Frigg - openENTRANCE Model Linkage

Input Data

- `Settings.yaml` specifies the time window for the model and the number of cores to be used as well as the data sources and technologies to use.
  - The entry `openEntrance` specifies for demand, production costs and production capacities data, the scenario, year, model, region and variable to download from the openENTRANCE Scenario Explorer.
  - The entry `sources` specifies the energy sources to consider, which can be set to be either dispatchable or non-dispatchable.
- `~Input/InitialDemand.csv` contains hourly demand data for electricity demand for district heating. That is normalised and used to disaggregate yearly demand data.
- `~Input/Solar.csv`, `~Input/Wind|Offshore.csv`, `~Input/Wind|Onshore.csv` contain normalised production patterns used for the disaggregation of yearly aggregates.

Modules `[~/Modules]`

- `DataReader.py` contains helper functions to read and download data:
  - `get_settings()` reads a `.yaml` file containing model settings and returns it as a dictionary.
  - `get_production_capacities(source='offline', verbose=True)` reads production capacities either from offline or online data, depending on the value of `source`. If `source='online'` is set, installed production capacities, as specified in `Settings.yaml` are downloaded. Then, if possible, these capacities are matched to the technology data specified in `Settings.yaml`. If this fails, mock-up data is used instead. Finally, hourly available production capacities are computed. For dispatchable sources, this corresponds to repeating the installed capacity 8760 times; for non-dispatchable sources, normalised generation patterns are loaded and multiplied with the installed capacities.
  - `get_production_costs(source='offline', verbose=True)` works equivalently. At the moment, we consider all production costs to be flat, i.e. time-constant.
  - `get_initial_demand(source='offline', verbose=True)` works in a similar fashion. The exception is, that this data need not be matched with energy sources specified in `Settings.yaml`.
- `SupplyCosts.py` constructs an object from hourly production cost and capacity data, that contains a function `get_cumulative(y, t)` which returns the cumulative production costs corresponding to a demand level `y` in hour `t`. This is done by sorting production costs in each time step and matching them to production capacities, as well as by applying some simple data transformations.
- `DataWriter.py` prints results, corrently mock-up data:
  - `print_new_demand_to_csv(initial_demand, new_demand, val_col_name)` takes a data frame with the initial demand data as an input and replaces the data in the column `val_col_name` with new demand data computed by Frigg.
  - `validate_df(df)` runs the validation function from the `nomenclature` package
  - `gen_demand_output(demand_array, year, other_columns={'Model': 'Frigg 0.1.0', 'scenario': 'DKFlex', 'unit': 'MWh', 'Region': 'Denmark', 'Variable': 'Final Energy|Electricity|Heat'})` constructs a `pyam.IamDataFrame` from `demand_array`, validates and returns it.

Main File `Frigg_model_linkage.ipynb`

- loads the necessary modules (first cell)
- opens a connection to the openENTRANCE Scenario Explorer (second cell)
- reads production capacities, costs and initial demand data through the `DataReader` module (second cell)
- constructs a `SupplyCosts` object from that data (second cell)
- creates mock-up data as an output (third cell)
- and validates it (fourth cell)

Upload of Mock-up Data

Random numbers showcasing the data structure of our output have been produced through Excel's random number generation and uploaded to the openENTRANCE scenario explorer manually through the browser.
The data also refers to `Frigg 0.01`; scenario `DKFlex`; region `Denmark`; variable `Final Energy|Electricity|Heat`; unit `MWh`; year `2050`; timestamps of that year; `exclude=False`.

Note

Since largely, the necessary data are not available yet in the scenario explorer, our data processing cannot be fully tested. As can be seen in the code, this is catched as an exception and handled accordingly. We process and produce random mock-up data (November 2020).