eGo Documentation

Release 0.3.4

open_eGo-Team

May 27, 2023

Contents

1 Overview **3**
- 1.1 The eGo tool . . . 3
- 1.2 Installation . . . 6
- 1.3 Getting started . . . 8
- 1.4 Theoretical background . . . 9
- 1.5 Developer notes . . . 13
- 1.6 What's new . . . 14
- 1.7 eGo . . . 18

2 Indices and tables **31**

Bibliography **33**

Python Module Index **35**

Index **37**



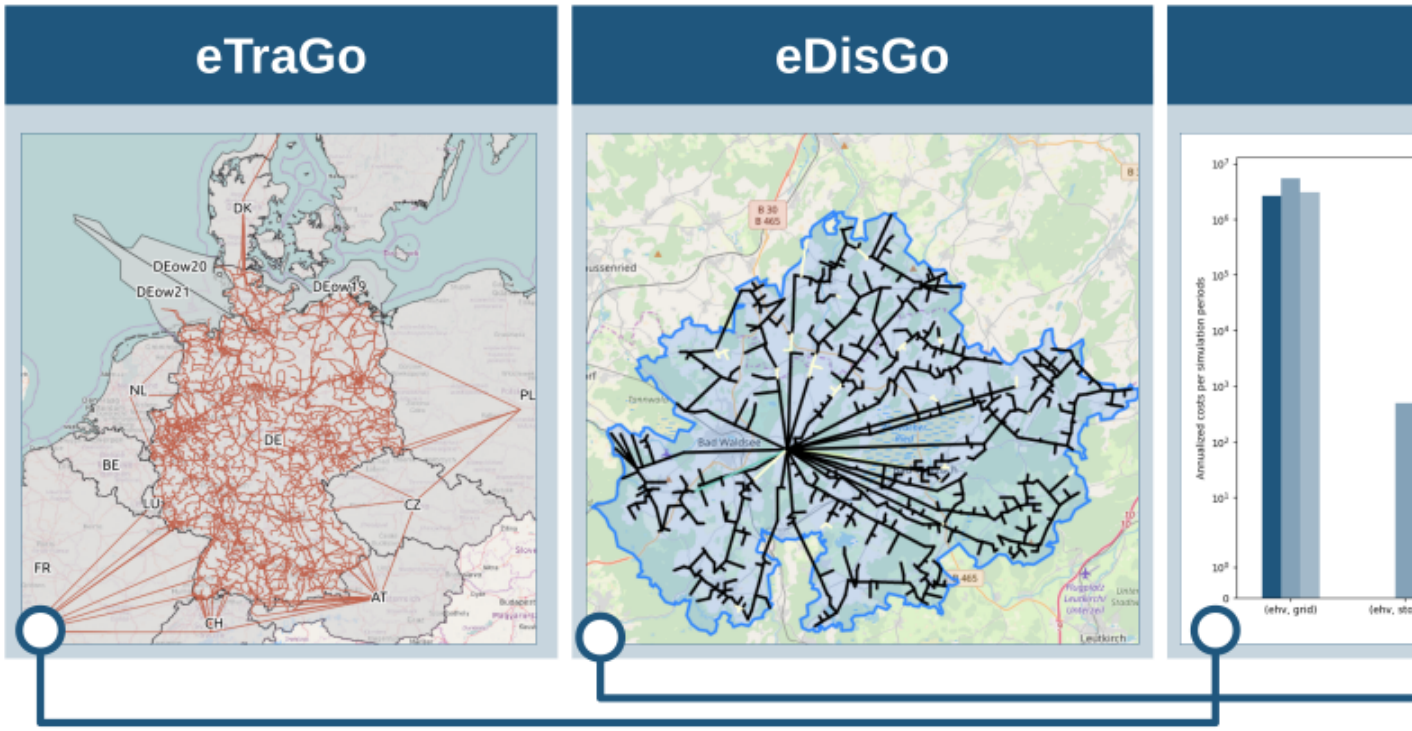
Note: Note, the data source of eGo relies on the Open Energy Database. The registration for the public accessible API can be found on openenergy-platform.org/login.

CHAPTER 1

Overview

1.1 The eGo tool

The python package eGo is a toolbox and also an application which combines **eTraGo** - a tool for optimizing flexibility options for transmission grids based on PyPSA and **eDisGo** - a toolbox in itself capable of analyzing distribution grids for grid issues and evaluating measures responding these.



1.1.1 The open_eGo project

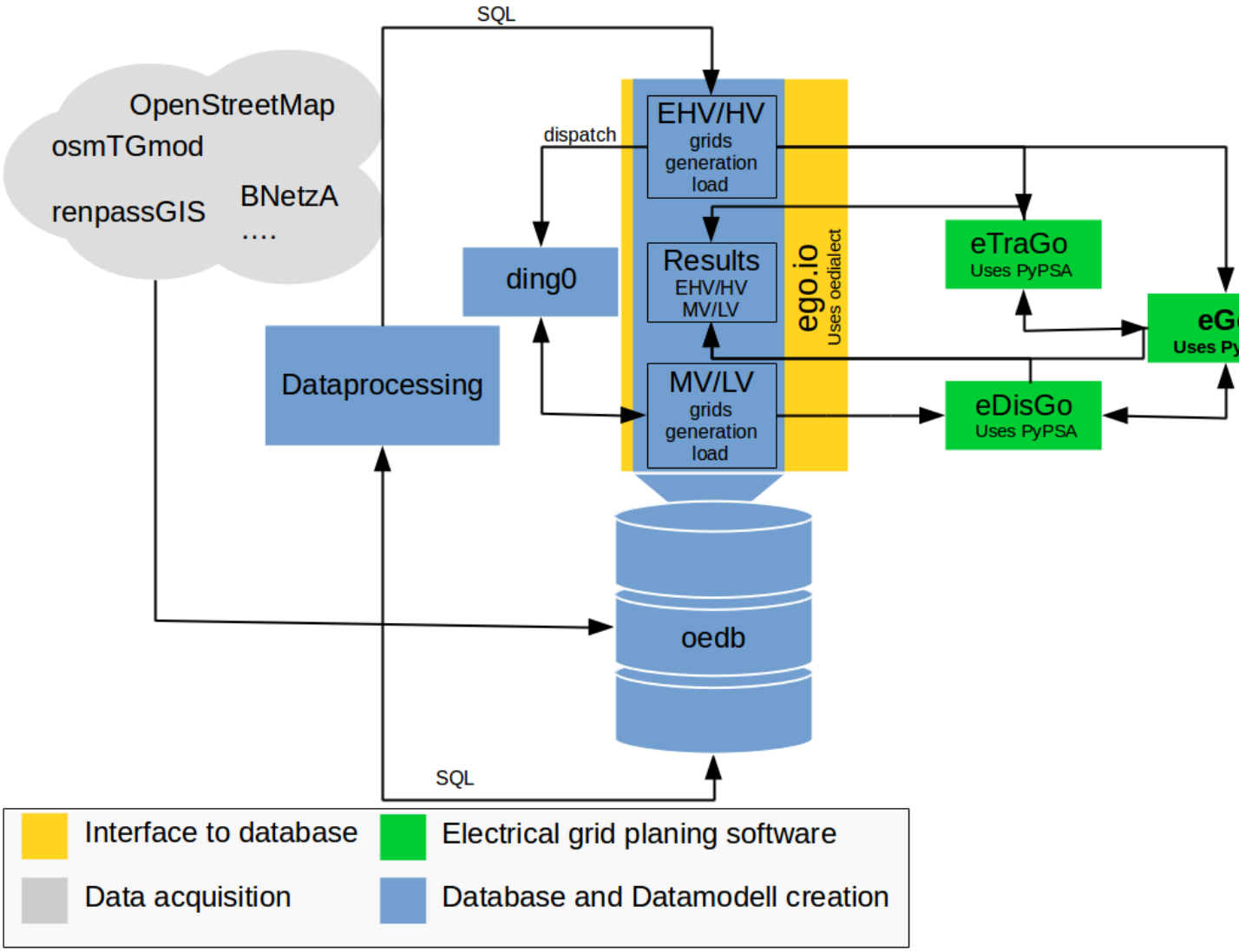
This software project is part of the research project open_eGo.

1.1.2 The OpenEnergy Platform

Within this project we developed the OpenEnergy Platform which the eGo toolbox relies upon to get and store in- and output data. Because of this dependency in order to use eGo a registration on the OpenEnergy Platform is required. For more information see openenergy-platform and login.

The OpenEnergy platform mainly addresses students, researchers and scientists in the field of energy modelling and analytics, but also welcomes all other interested parties. The platform provides great tools to make your energy system modelling process transparent. Data of the open_eGo project are stored on this platform. Learn more about the database access.

1.1.3 Model overview



eTraGo

The python package eTraGo provides an optimization of flexibility options for transmission grids based on PyPSA. In particular transmission grids of different voltage levels , that is 380, 220 and 110 kV in Germany, can be handled. Conventionally the 110kV grid is part of the distribution grid. The integration of the transmission and 'upper' distribution grid is part of eTraGo.

The focus of optimization are flexibility options with a special focus on energy storages and grid expansion measures. Learn more here.

eDisGo

The python package eDisGo provides a toolbox for analysis and optimization of distribution grids. It is closely related to the python project Ding0 as this project is currently the single data source for eDisGo providing synthetic grid data for whole Germany. Learn more here.

Dataprocessing

For the open_eGo project several python packages are developed which are feeded by the input data of the data processing. The dataprocessing is written in SQL and Python. Learn more here.

ego.io

The `ego.io` is a SQLAlchemy interface to the OpenEnergy database (oedb). The module provides ORM objects mirroring oedb tables and additionally contains helper functions for I/O operations. Learn more here.

Dingo

The DIstribution Network GeneratOr (Ding0) is a tool to generate synthetic medium and low voltage power distribution grids based on open (or at least accessible) data. Learn more here.

1.1.4 Supported by

This project is supported by the German Federal Ministry for Economic Affairs and Energy (BMWI).

Supported by:
Federal Ministry for Economic Affairs and Energy
on the basis of a decision by the German Bundestag

1.1.5 License

© Copyright 2015-2018

Flensburg University of Applied Sciences, Europa-Universität Flensburg, Centre for Sustainable Energy Systems

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see www.gnu.org/licenses.

1.1.6 Partner







1.2 Installation

eGo is designed as a Python package therefore it is mandatory to have Python 3 installed. If you have a working Python3 environment, use PyPI to install the latest eGo version. We highly recommend to use a virtual environment. Use the following pip command in order to install eGo:

```
$ pip3 install eGo --process-dependency-links
```

Please ensure, that you are using the pip version 18.1. Use `pip install --upgrade pip==18.1` to get the right pip version. In Case of problems with the Installation and the `dependency_links` of the PyPSA fork, please istall PyPSA from the github.com/openego Repository.

```
$ pip3 install -e git+https://github.com/openego/PyPSA@master#egg=0.11.0fork
```

1.2.1 Using virtual environment

At first create a virtual environment and activate it:

```
$ virtualenv venv --clear -p python3.5
$ source venv/bin/activate
$ cd venv
```

Inside your virtual environment you can install eGo with the pip command.

1.2.2 Linux and Ubuntu

The package eGo is tested with Ubuntu 16.04 and 18.04 inside a virtual environment of virtualenv. The installation is shown above.

1.2.3 Windows or Mac OSX users

For Windows and/or Mac OSX user we highly recommend to install and use Anaconda for your Python3 installation. First install anaconda including python version 3.5 or higher from https://www.anaconda.com/download/ and open an anaconda prompt as administrator and run:

```
$ conda install pip
$ conda config --add channels conda-forge
$ conda install shapely
$ pip3 install eGo --process-dependency-links
```

The full documentation can be found on this page. We use Anaconda with an own environment in order to reduce problems with packages and different versions on our system. Learn more about Anacona environments.

1.2.4 Setup database connection

The package `ego.io` gives you a python SQL-Alchemy representation of the **OpenEnergy-Database** (oedb) and access to it by using the oedialect - a SQL-Alchemy binding Python package for the REST-API used by the OpenEnergy Platform (OEP). Your API access / login data will be saved in the folder `.egoio` in the file `config.ini`. You can create a new account on openenergy-platform.org/login.

oedialect connection

```
[oedb]
dialect  = oedialect
username = <username>
database = oedb
host     = openenergy-platform.org
port     = 80
password = <token>
```

Local database connection

```
[local]
username = YourOEDBUserName
database = YourLocalDatabaseName
host = localhost or 127.0.0.1
port = 5433
pw = YourLocalPassword
```

Old developer connection

```
[oedb]
username = YourOEDBUserName
database = oedb
host = oe2.iws.cs.ovgu.de
port = 5432
pw = YourOEDBPassword
```

Please find more information on *Developer notes*.

1.3 Getting started

In order to start and run the eGo-tool a few steps needs to be done.

1.3.1 Steps to run eGo

1. Are you registered on the OpenEnergy Platform? The registration for the public accessible API can be found on openenergy-platform.org/login.
2. You have Python 3 installed? Install for example the Python distribution of https://www.anaconda.com/download.
3. Install and use a virtual environment for your installation (optional).
4. Install the eGo tool `pip3 install eGo --process-dependency-links`.
5. Create mid and low voltage distribution grids with ding0. Learn more about Ding0 on https://dingo.readthedocs.io/en/dev/index.html.
6. Check and prepare your eGo setting in `ego/scenario_setting.json`. Add your local paths and prepare your parameters.
7. Start your calculation and run the tool for example under `eGo/ego` and `>>> python3 appl.py` . You can also use any other Python Terminal, Jupyter Notebook or Editor.

1.3.2 How to use eGo?

Start and use eGo from the terminal.

```
>>> python3 appl.py
>>> ...
>>> INFO:ego:Start calculation
>>> ...
```

Examples

Inside the appl.py

```
# import the eGo tool
from ego.tools.io import eGo

# Run your scenario
ego = eGo(jsonpath='scenario_setting.json')

# Analyse your results on extra high voltage level (etrago)
ego.etrago_line_loading()
```

1.3.3 Tutorials as Jupyter Notebook

Learn more about Jupyter Notebook and how to install and use it on jupyter.org.

Workshop open_eGo Session eGo (in German)

Workshop open_eGo Session eTraGo (in German)

Workshop open_eGo Session DinGo (in German)

Workshop open_eGo Session eDisGo (in German)

OpenMod eTraGo Tutorial (in English)

1.3.4 eGo Result Example of Germany

A small example of the eGo results is displayed below. The full page can be found here

The plot is created by the eGo function:

```
ego.iplot
```

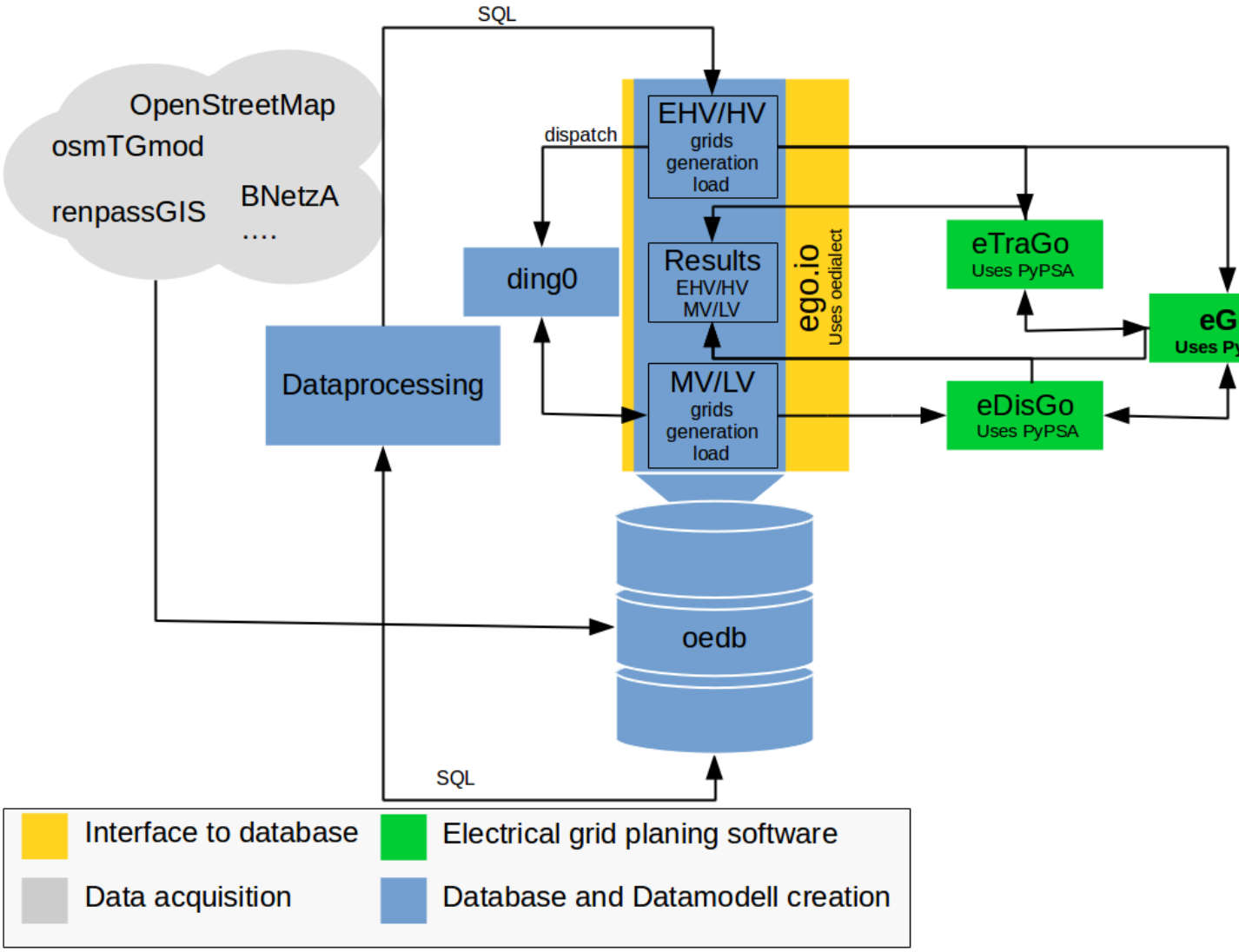
1.4 Theoretical background

Contents

- *Theoretical background*
  - *Models overview*
  - *eTraGo's theoretical Background*
  - *eDisGo's theoretical Background*
  - *eDisGo Cluster Method*
  - *Economic calculation*
    - *Overnight costs*
    - *Annuity costs*
    - *Investment costs ehv/hv*

        * *Investment costs mv/lv*
    - *References*

1.4.1 Models overview



1.4.2 eTraGo's theoretical Background

Learn more about eTraGo's theoretical background of methods and assumptions here.

1.4.3 eDisGo's theoretical Background

Learn more about eTraGo's theoretical background of methods and assumptions here.

1.4.4 eDisGo Cluster Method

In order to achieve acceptable computation times, the problem's complexity can be reduced by applying a k-means cluster-algorithm to MV grids. The algorithm identifies a specified number of representative MV grids and assigns a weighting to each grid. As described here, the available clustering attributes are:

- cumulative installed **wind capacity**,
- cumulative installed **solar capacity**,
- distance between transition point and **farthest node** of the MV grid
- installed **battery capacity** (as a result of eTraGo's investment optimization)

Subsequent to the MV grid simulations with the reduced number of representative grids, the cluster weighting is used to extrapolate the costs back to the original number of MV grids.

1.4.5 Economic calculation

The tool *eGo* unites the extra high (ehv) and high voltage (hv) models with the medium (mv) and low voltage (lv) models to ascertain the costs per selected measure and scenario. This results in a cross-grid-level economic result of the electrical grid and storage optimisation.

Overnight costs

The *overnight costs* represents the investment costs of the components or construction project without any interest, as if the project was completed "overnight". The overnight costs ($C_{\text{Overnight}}$) of the grid measures (lines and transformers) are calculated as:

$$C_{Line\ extension} = S_{Extension}\ [MVA] * C_{assumtion}\ [\frac{EUR}{MVA}] * L_{Line\ length}\ [km]$$

$$C_{Transformer\ extension} = S_{Extension}\ [MVA] * C_{assumtion}\ [\frac{EUR}{MVA}]$$

The total overnight grid extension costs are given by:

$$C_{overnight} = \sum C_{Line\ extension} + \sum C_{Transformer\ extension}$$

The conversion of the given annuity costs of *eTraGo* is done in `etrago_convert_overnight_cost()`.

Annuity costs

The *annuity costs* represents project investment costs with an interest as present value of an annuity. The investment years T and the interest rate p are defined as default in *eGo* with an interest rate (p) of `0.05` and a number of investment years (T) of `40 years`. The values are based on the [StromNEV_A1] for the grid investment regulation in Germany.

The present value of an annuity (PVA) is calculated as:

$$PVA = \frac{1}{p} - \frac{1}{\left(p * (1+p)^T\right)}$$

In order to calculate the $C_{annuity}$ of a given period less than a year the annuity costs are factorized by the hours of the $t_{year} = 8760$ and the defined calculation period.

$$t_{period} = t_{\text{end_snapshot}} - t_{\text{start_snapshot}}\ [h]$$

The annuity costs ($C_{annuity}$) is calculated as:

$$C_{annuity} = C_{\text{overnight}} * PVA * \left(\frac{t_{year}}{(t_{\text{period}} + 1)} \right)$$

Investment costs ehv/hv

The investment costs of the grid and storage expansion are taken from the studies [NEP2015a] for the extra and high voltage components and the [Dena]. The given costs are transformed in respect to PyPSA *[€/MVA]* format [PyPSA] components for the optimisation.

Overview of grid cost assumtions:

The table displays the transformer and line costs which are used for the calculation with *eTraGo*.

Table 1.1: Overview of grid cost assumtions

voltage level	component	capital costs	unit	source
110	AC overhead transmission lines	230	EUR/MVA	Dena 2012
220	AC overhead transmission lines	290	EUR/MVA	NEP 2015
380	AC overhead transmission lines	85	EUR/MVA	NEP 2015
DC	DC overhead transmission lines	375	EUR/MVA	NEP 2015
110/220	transformer	7500	EUR/MVA	Dena 2012
110/380	transformer	17333	EUR/MVA	NEP 2015
220/380	transformer	14166	EUR/MVA	NEP 2015

The *eTraGo* calculation of the annuity costs per simulation period is defined in `set_line_costs()` and `set_trafo_costs()`.

Overview of storage cost assumtions:

Technology		**Batterien (Li-Ion)**			**Hydrogen Caverns**		
Scenario		**Status Quo**	**NEP 2035**	**eGo 2050**	**Status Quo**	**NEP 2035**	**eGo 2050**
		AcatechLilon	AcatechLilon	AcatechLilon	Acatech	Acatech	Acatech
a	Jahr	2013	2023	2050	2013	2023	2050
C_p	Preis pro MW	160000	72500	45000	1215000	815000	575000
C_e	Preis pro MWh	445000	141000	105500	450	450	450
h	max_hours	6	6	6	168	168	168
T	Jahre	20	25	30	25	25	25
i	Zinssatz	0.05	0.05	0.05	0.05	0.05	0.05
K	Investition (EUR/MW)	2830000	918500	678000	1290600	890600	650600
RBF	Rentenbarwertfaktor	12	14	15	14	14	14
C_inv	Capital_cost (EUR/MW/a)	227087	65170	44105	91571	63190	46162
O&M	Betriebskosten (EUR/MW/a	2271	652	441	3205	2212	1616
Cap_cost	Finale Kosten f. PyPSA	**229357**	**65822**	**44546**	**94776**	**65402**	**47777**
Efficiency	round trip	0.855	0.87	0.9	0.2531	0.3081	0.4475

Investment costs mv/lv

The tool *eDisGO* is calculating all grid expansion measures as capital or *overnight* costs. In order to get the annuity costs of eDisGo's optimisation results the function `edisgo_convert_capital_costs()` is used. The cost assumption of [eDisGo] are taken from the [Dena] and [CONSENTEC] study. Based on the component the costs including earthwork costs can depend on population density according to [Dena].

1.4.6 References

1.5 Developer notes

1.5.1 Installation

Note: Installation is only tested on (Ubuntu 16.04) linux OS.

Please read the Installation Guideline ego.doc.installation.

1. Use virtual environment

Create a virtual environment and activate it:

```
$ virtualenv --clear -p python3.5  ego_dev``
$ cd ego_dev/
$ source bin/activate
```

2. Get eGo

Clone eGo from github.com by running the following command in your terminal:

```
$ git clone https://github.com/openego/eGo
```

With your activated environment *cd* to the cloned directory and run `pip3 install -e eGo --process-dependency-links` . This will install all needed packages into your environment.

3. Get your Database login data

Learn more here.

4. Create Dingo grids

Install ding0 from github.com and run the `example_parallel_multiple_grid_districts.py` script, which can be found under `ding0/ding0/examples/`.

```
$ git clone https://github.com/openego/ding0.git
$ pip3 install -e ding0
$ python3 ding0/ding0/examples/example_parallel_multiple_grid_districts.py
```

Learn more about Dingo. Before you run the script check also the configs of Dingo and eDisGo in order to use the right database version. You find this files under `ding0/ding0/config/config_db_tables.cfg` and `~.edisgo/config/config_db_tables.cfg`. Your created ding0 grids are stored in `~.ding0/...`

eDisGo and eTraGo

Please read the Developer notes of eDisGo and eTraGo.

Error handling

1. Installation Error use pip-18.1 for your installation. `pip install --upgrade pip==18.1`
2. Installation Error of eTraGo, eDisGo, Pypsa fork or ding0. If you have problems with one of those packages please clone it from *github.com* and install it from the master or dev branch. For example `pip3 install -e git+https://github.com/openego//PyPSA.git@master#egg=pypsafork`
3. Matplotlib error on server and few other systems. Please change your settings in `matplotlibrc` from `backend : TkAgg` to `backend : PDF`. You can find the file for example in a virtual environment under `~/env/lib/python3.5/site-packages/matplotlib/mpl-data$ vim matplotlibrc`. Learn more here..
4. Geopandas error caused by Rtree `Could not find libspatialindex_c library` Please reinstall Rtree with `sudo pip3 install Rtree` or install `libspatialindex_c` via `sudo apt install python3-rtree`. On Windows or macOS you maybe install `libspatialindex_c` straight from source.

1.6 What's new

Releases

- *Release v0.3.4 (December 10, 2018)*
- *Release v0.3.3 (December 7, 2018)*
- *Release v0.3.2 (October 27, 2018)*
- *Release v0.3.1 (October 27, 2018)*
- *Release v0.3.0 (September 07, 2018)*
- *Release v0.2.0 (July 18, 2018)*
- *Release v0.1.0 (March 29, 2018)*
- *Release v0.0.1 (February 02, 2018)*

1.6.1 Release v0.3.4 (December 10, 2018)

Update eDisGo release.

Added features

- Update eDisGo version from 0.0.8 to 0.0.9

1.6.2 Release v0.3.3 (December 7, 2018)

Add interactive map for documentation.

Added features

- Create and add interactive result map

- Add workshop jupyter notebook to docs

Fixes

- Fix bug of period calculation
- removed duplicate matplotlib from setup.py
- fixed csv import

1.6.3 Release v0.3.2 (October 27, 2018)

Making eGo quotable with zenodo.

Added features

- Registration at zenodo.org

1.6.4 Release v0.3.1 (October 27, 2018)

This release contains documentation and bug fixes for the new features introduced in 0.3.0.

Added features

- Update of interactiv plot (iplot)
- Update of Documentation
- Update of eTraGo functionalities
- Update of eDisGo functionalities
- Change and update of API file scenario_setting.json
- Improved cluster plot of `ego.plot_edisgo_cluster()`
- Improved cost differentiation
- Add jupyter notebook eGo tutorials

Fixes

- Fix installation problems of the pypsa 0.11.0 fork (use pip 18.1)
- Fix parallel calculation of mv results

1.6.5 Release v0.3.0 (September 07, 2018)

Power Flow and Clustering. eGo is now using eTraGo non-linear power flows based on optimization results and its disaggregation of clustered results to an original spatial complexities. With the release of eDisGo speed-up options, a new storage integration methodology and more are now available.

Added features

- Update of dependencies
- Implementing of Ding0 grid parallelization
- Redesign of scenario settings and API simplifications
- Adding and using the Power Flow of eTraGo in eGo
- Testing and using new dataprocessing Version v0.4.3, v0.4.4 and v0.4.5
- make eGo installable from pip via `pip3 install eGo -- process-dependency-links`
- Implementing eDisGo's storage distribution for MV and LV grids
- Improved logging and the creation of status files
- Maximal calculation time for ding0 grids can be set by user
- eDisGo results import and export (all eGo-relevant data from eDisGo can be re-imported after a run)
- Storage-related investment costs are also allocated to MV grids
- Update of cluster plots
- Plot of investment costs per line and bus
- Update of `ego.iplot` for an interactive visualization

1.6.6 Release v0.2.0 (July 18, 2018)

Fundamental structural changes of the eGo tool are included in this release. A new feature is the integration of the MV grid power flow simulations, performed by the tool eDisGo.. Thereby, eGo can be used to perform power flow simulations and optimizations for EHV, HV (*eTraGo*) and MV (*eDisGo*) grids.

Moreover, the use of the Dataprocessing versions `''v0.4.1''` and `''v0.4.2''` is supported. Please note, that this release is still under construction and only recommended for developers of the *open_eGo* project.

Furthermore, overall cost aggregation functions are available.

Added features

- **Cleaned and restructured eGo classes and functions**
  - Move classes of eGo from results.py to io.py
  - Move serveral function
- **Introduce new files for *eDisGo* handling**
  - edisgo_integration.py
  - mv_cluster.py
- Introduce new file storages.py for eTraGo
- Updated eTraGo 0.6 and integrated eTraGo's new functions and features to eGo
- Updated eDisGo 0.0.3 version which includes the features of a parallelization for custom function and other important API changes.
- Started to implemented pep8 style to eGo Code
- Implemented logging function for the whole model

- Using the Restfull-API for the OpenEnergy Database connection, buy using ego.io v0.4.2. A registration is needed and can be done on openenergy-platform.org/login
- Remove functionalities from `ego_main.py` to the eGo class
- Fixed eTraGo scenario import of `etrago_from_oedb()`

Notes

- As an external user you need to have an account on the openenergy-platform.org/login
- In future versions, all MV grids (*ding0* grids) will be queried from your database. However, in this version all MV grids have to be generated with the tool ding0 and stored in *eGo*'s *data* folder.
- Total operational costs are missing in this release

1.6.7 Release v0.1.0 (March 29, 2018)

As this is the second release of eGo. This Release introduce the results class and is still under construction and not ready for a normal use.

Added features

- Update of interface between eTraGo and eDisGo (specs)
- New structure of eGo module / resulte class
- Restructuring of functions
- Add import function of eTraGo results form oedb

Notes

- The 'direct_specs' function is not working and needs to be set to `false`

1.6.8 Release v0.0.1 (February 02, 2018)

As this is the first release of eGo. The tool eGo use the Python3 Packages eTraGo (Optimization of flexibility options for transmission grids based on PyPSA) and eDisGo (Optimization of flexibility options and grid expansion for distribution grids based on PyPSA) for an electrical power calculation from extra high voltage to selected low voltage level.

Added features

- Interface between eTraGo and eDisGo
- Plots with folium
- First result structure

1.7 eGo

1.7.1 Overview of modules

ego.tools package

ego.tools.economics

This module collects useful functions for economic calculation of eGo which can mainly distinguished in operational and investment costs.

ego.tools.economics.**annuity_per_period**(*capex*, *n*, *wacc*, *t*, *p*)
Calculate per given period

Parameters

- **capex** (*float*) – Capital expenditure (NPV of investment)
- **n** (*int*) – Number of years that the investment is used (economic lifetime)
- **wacc** (*float*) – Weighted average cost of capital
- **t** (*int*) – Timesteps in hours
- **p** (*float*) – interest rate

ego.tools.economics.**edisgo_convert_capital_costs**(*overnight_cost*, *t*, *p*, *json_file*)
Get scenario and calculation specific annuity cost by given capital costs and lifetime.

Parameters

- **json_file** (*:obj:dict*) – Dictionary of the scenario_setting.json file
- **_start_snapshot** (*int*) – Start point of calculation from scenario_setting.json file
- **_end_snapshot** (*int*) – End point of calculation from scenario_setting.json file
- **_p** (*numeric*) – interest rate of investment
- **_t** (*int*) – lifetime of investment

Returns **annuity_cost** – Scenario and calculation specific annuity cost by given capital costs and lifetime

Return type numeric

Examples

$$PVA = (1/p) - (1/(p * (1+p)^t))$$

ego.tools.economics.**etrago_convert_overnight_cost**(*annuity_cost*, *json_file*, *t=40*, *p=0.05*)
Get annuity cost of simulation and calculation total overnight_costs by given capital costs and lifetime.

Parameters

- **json_file** (*:obj:dict*) – Dictionary of the scenario_setting.json file

- **_start_snapshot** (*int*) – Start point of calculation from `scenario_setting.json` file
- **_end_snapshot** (*int*) – End point of calculation from `scenario_setting.json` file
- **_p** (*numeric*) – interest rate of investment
- **_T** (*int*) – lifetime of investment

Returns **overnight_cost** – Scenario and calculation total `overnight_costs` by given annuity capital costs and lifetime.

Return type numeric

Examples

$$PVA = (1/p) - (1/(p * (1+p)^t))$$
$$K_{ON} = K_a * PVA * ((t/(period+1))$$

`ego.tools.economics.`**`etrago_operating_costs`**(*network*)

Function to get all operating costs of eTraGo.

Parameters **network_etrago** (`etrago.tools.io.NetworkScenario`) – eTraGo network object compiled by `etrago.appl.etrago()`

Returns **operating_costs** – DataFrame with aggregate operational costs per component and voltage level in [EUR] per calculated time steps.

Return type pandas.Dataframe

Example

```
>>> from ego.tools.io import eGo
>>> ego = eGo(jsonpath='scenario_setting.json')
>>> ego.etrago.operating_costs
```

component	operation_costs	voltage_level
biomass	27.0	ehv
line losses	0.0	ehv
wind_onshore	0.0	ehv

`ego.tools.economics.`**`etrago_grid_investment`**(*network*, *json_file*, *session*)

Function to get grid expantion costs from eTraGo

Parameters

- **network_etrago** (`etrago.tools.io.NetworkScenario`) – eTraGo network object compiled by `etrago.appl.etrago()`
- **json_file** (*:obj:dict*) – Dictionary of the `scenario_setting.json` file

Returns **grid_investment_costs** – Dataframe with `voltage_level`, `number_of_expansion` and `capital_cost` per calculated time steps

Return type pandas.Dataframe

Example

```
>>> from ego.tools.io import eGo
>>> ego = eGo(jsonpath='scenario_setting.json')
>>> ego.etrago.grid_investment_costs
```

differentiation	voltage_level	number_of_expansion	capital_cost
cross-border	ehv	27.0	31514.1305
domestic	hv	0.0	0.0

ego.tools.economics.**edisgo_grid_investment**(*edisgo*, *json_file*)
Function aggregates all costs, based on all calculated eDisGo grids and their weightings :param edisgo: Contains multiple eDisGo networks :type edisgo: *ego.tools.edisgo_integration.EDisGoNetworks*

Returns Dataframe containing annuity costs per voltage level

Return type None or pandas.DataFrame

ego.tools.economics.**get_generator_investment**(*network*, *scn_name*)
Get investment costs per carrier/ generator.

ego.tools.edisgo_integration

This file is part of the eGo toolbox. It contains the class definition for multiple eDisGo networks.

class ego.tools.edisgo_integration.**EDisGoNetworks**(*json_file*, *etrago_network*)
Bases: object

Performs multiple eDisGo runs and stores the resulting edisgo_grids

Parameters

- **json_file** (*:obj:dict*) – Dictionary of the scenario_setting.json file
- **etrago_network** (etrago.tools.io.NetworkScenario) – eTraGo network object compiled by etrago.appl.etrago()

network
Container for EDisGo objects, including all results

Returns Dictionary of EDisGo objects, keyed by MV grid ID

Return type dict[int, edisgo.EDisGo]

grid_choice
Container for the choice of MV grids, including their weighting

Returns Dataframe containing the chosen grids and their weightings 'no_of_points_per_cluster', 'the_selected_network_id', 'represented_grids'

Return type pandas.DataFrame

successful_grids
Relative number of successfully calculated MV grids (Includes clustering weighting)

Returns Relative number of grids

Return type int

grid_investment_costs
Grid investment costs

Returns Dataframe containing annuity costs per voltage level

Return type None or pandas.DataFrame

plot_storage_integration(*mv_grid_id*, ***kwargs*)
Plots storage position in MV grid of integrated storages. For more information see edisgo.tools.plots.mv_grid_topology().

plot_grid_expansion_costs(*mv_grid_id*, ***kwargs*)
Plots costs per MV line. For more information see edisgo.tools.plots.mv_grid_topology().

plot_line_loading(*mv_grid_id*, ***kwargs*)
Plots relative line loading (current from power flow analysis to allowed current) of MV lines. For more information see edisgo.tools.plots.mv_grid_topology().

plot_mv_grid_topology(*mv_grid_id*, ***kwargs*)
Plots plain MV grid topology. For more information see edisgo.tools.plots.mv_grid_topology().

run_edisgo(*mv_grid_id*)
Performs a single eDisGo run

Parameters **mv_grid_id** (*int*) – MV grid ID of the ding0 grid

Returns Returns the complete eDisGo container, also including results

Return type edisgo.EDisGo

ego.tools.edisgo_integration.**parallelizer**(*ding0_id_list*, *func*, *func_arguments*, *workers=2*, *max_calc_time*, *worker_lifetime=1*)
Use python multiprocessing toolbox for parallelization

Several grids are analyzed in parallel based on your custom function that defines the specific application of eDisGo.

Parameters

- **ding0_id_list** (*list of int*) – List of ding0 grid data IDs (also known as HV/MV substation IDs)
- **func** (*any function*) – Your custom function that shall be parallelized
- **func_arguments** (*tuple*) – Arguments to custom function func
- **workers** (*int*) – Number of parallel process
- **worker_lifetime** (*int*) – Bunch of grids sequentially analyzed by a worker

Notes

Please note, the following requirements for the custom function which is to be executed in parallel

1. It must return an instance of the type EDisGo.
2. The first positional argument is the MV grid district id (as int). It is prepended to the tuple of arguments func_arguments

Returns **containers** – Dict of EDisGo instances keyed by its ID

Return type dict of EDisGo

ego.tools.io

This file contains the eGo main class as well as input & output functions of eGo in order to build the eGo application container.

class ego.tools.io.**egoBasic**(**args*, ***kwargs*)
Bases: object

The eGo basic class select and creates based on your scenario_setting.json file your definded eTraGo and eDisGo results container. And contains the session for the database connection.

Parameters **jsonpath** (json) – Path to scenario_setting.json file.

Returns

- **json_file** (*:obj:dict*) – Dictionary of the scenario_setting.json file
- **session** (sqlalchemy.orm.session.Session) – SQLAlchemy session to the OEDB

class ego.tools.io.**eTraGoResults**(**args*, ***kwargs*)
Bases: *ego.tools.io.egoBasic*

The eTraGoResults class creates and contains all results of eTraGo and it's network container for eGo.

Returns

- **network_etrago** (etrago.tools.io.NetworkScenario) – eTraGo network object compiled by etrago.appl.etrago()
- **etrago** (pandas.Dataframe) – DataFrame which collects several eTraGo results

class ego.tools.io.**eDisGoResults**(**args*, ***kwargs*)
Bases: *ego.tools.io.eTraGoResults*

The eDisGoResults class create and contains all results of eDisGo and its network containers.

edisgo
Contains basic informations about eDisGo

Returns

Return type pandas.DataFrame

class ego.tools.io.**eGo**(*jsonpath*, **args*, ***kwargs*)
Bases: *ego.tools.io.eDisGoResults*

Main eGo module which includs all results and main functionalities.

Returns

- **network_etrago** (etrago.tools.io.NetworkScenario) – eTraGo network object compiled by etrago.appl.etrago()
- **edisgo.network** (*ego.tools.edisgo_integration.EDisGoNetworks*) – Contains multiple eDisGo networks
- **edisgo** (pandas.Dataframe) – aggregated results of eDisGo
- **etrago** (pandas.Dataframe) – aggregated results of eTraGo

total_investment_costs
Contains all investment informations about eGo

Returns

Return type pandas.DataFrame

total_operation_costs
Contains all operation costs information about eGo

Returns

Return type pandas.DataFrame

plot_total_investment_costs (*filename=None*, *display=False*, ***kwargs*)
Plot total investment costs

plot_power_price (*filename=None*, *display=False*)
Plot power prices per carrier of calculation

plot_storage_usage (*filename=None*, *display=False*)
Plot storage usage by charge and discharge

plot_edisgo_cluster (*filename=None*, *display=False*, ***kwargs*)
Plot the Clustering of selected Dingo networks

plot_line_expansion (***kwargs*)
Plot line expantion per line

plot_storage_expansion (***kwargs*)
Plot storage expantion per bus

iplot
Get iplot of results as html

ego.tools.io.**results_to_excel** (*ego*)
Wirte results of ego.total_investment_costs to an excel file

ego.tools.io.**etrago_from_oedb** (*session*, *json_file*)
Function which import eTraGo results for the Database by the `result_id` number.

Parameters

- **session** (sqlalchemy.orm.session.Session) – SQLAlchemy session to the OEDB
- **json_file** (`dict`) – Dictionary of the `scenario_setting.json` file

Returns **network_etrago** – eTraGo network object compiled by `etrago.appl.etrago()`

Return type `etrago.tools.io.NetworkScenario`

ego.tools.io.**recover_resultsettings** (*session*, *json_file*, *orm_meta*, *result_id*)
Recover scenario_setting from database

ego.tools.mv_cluster

ego.tools.plots

Module which collects useful functions for plotting eTraGo, eDisGo and eGo results.

ego.tools.plots.**carriers_colore**()
Return matplotlib colore set per carrier (technologies of generators) of eTraGo.

Returns **colors** – List of carriers and matplotlib colores

Return type `dict`

ego.tools.plots.**ego_colore**()
Get the four eGo colores

Returns **colors** – List of eGo matplotlib hex colores

Return type dict

ego.tools.plots.**plot_storage_expansion**(*ego*, *filename=None*, *dpi=300*, *column='overnight_costs'*, *scaling=1*)

Plot line expantion

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo eGo inclueds eTraGo and eDisGo results
- **filename** (*str*) – Filename and/or path of location to store graphic
- **dpi** (*int*) – dpi value of graphic
- **column** (*str*) – column name of eTraGo's line costs. Default: overnight_costs in EURO. Also available s_nom_expansion in MVA or annualized investment_costs in EURO
- **scaling** (*numeric*) – Factor to scale storage size of bus_sizes

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot matplotlib.pyplot.show

ego.tools.plots.**plot_line_expansion**(*ego*, *filename=None*, *dpi=300*, *column='overnight_costs'*)

Plot line expantion

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo eGo inclueds eTraGo and eDisGo results
- **filename** (*str*) – Filename and or path of location to store graphic
- **dpi** (*int*) – dpi value of graphic
- **column** (*str*) – column name of eTraGo's line costs. Default: overnight_costs in EUR. Also available s_nom_expansion in MVA or annualized investment_costs in EUR

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot matplotlib.pyplot.show

ego.tools.plots.**plot_grid_storage_investment**(*costs_df*, *filename*, *display*, *var=None*)

Plot total grid and storage investment.

Parameters

- **costs_df** (pandas.DataFrame) – Dataframe containing total_investment_costs of ego
- **filename** (*str*) – Filename and or path of location to store graphic
- **display** (*bool*) – Display plot
- **var** (*str*) – Cost variable of overnight_cost by default displays annualized costs of timesteps

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot matplotlib.pyplot.show

ego.tools.plots.**power_price_plot**(*ego*, *filename*, *display*)

Plot power price of calculated scenario of timesteps and carrier

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo eGo inclueds eTraGo and eDisGo results

- **filename** (*str*) – Filename and or path of location to store graphic
- **display** (*bool*) – Display plot

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot `matplotlib.pyplot.show`

`ego.tools.plots.`**plot_storage_use**(*ego*, *filename*, *display*)
Plot storage use by charge and discharge values

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo `eGo` inclueds eTraGo and eDisGo results
- **filename** (*str*) – Filename and or path of location to store graphic
- **display** (*bool*) – Display plot

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot `matplotlib.pyplot.show`

`ego.tools.plots.`**get_country**(*session*, *region=None*)
Get Geometries of scenario Countries.

Parameters

- **session** (sqlalchemy.orm.session.Session) – SQLAlchemy session to the OEDB
- **region** (*list*) – List of background countries e.g. ['DE', 'DK']

Returns **country** – GeoDataFrame inclueds MultiPolygon of selected regions or countries

Return type `geopandas.GeoDataFrame`

`ego.tools.plots.`**prepareGD**(*session*, *subst_id=None*, *version=None*)
Get MV grid districts for plotting form oedb.

Parameters

- **session** (sqlalchemy.orm.session.Session) – SQLAlchemy session to the OEDB
- **subst_id** (*list*) – List of integer ids of substation of the pf ehv/hv grid model_draft
- **version** (*str*) – Name of data version saved in the OEDB

Returns **region** – GeoDataFrame inclueds MultiPolygon of selected MV grids

Return type `geopandas.GeoDataFrame`

`ego.tools.plots.`**plot_edisgo_cluster**(*ego, filename, region=['DE'], display=False, dpi=150, add_ehv_storage=False, grid_choice=None, title='', cmap='jet', labelsize=10, fontsize=10*)
Plot the Clustering of selected Dingo networks

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo `eGo` inclueds on eTraGo and eDisGo results
- **filename** (*str*) – file name for plot e.g. `cluster_plot.pdf`
- **region** (*list*) – List of background countries e.g. ['DE', 'DK']
- **display** (*bool*) – True show plot false print plot as `filename`
- **add_ehv_storage** (*bool*) – Display eTraGo ehv/hv storage distribution
- **grid_choice** (*str*) – path to seperate mv/lv grid choice csv file

- **title** (*str*) – Title of Plot
- **cmap** (*str*) – Name of colormap from https://matplotlib.org/gallery/color/colormap_reference.html

Returns https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.show

Return type plot `matplotlib.pyplot.show`

`ego.tools.plots.`**igeoplot**(*ego*, *tiles=None*, *geoloc=None*, *save_image=False*)
Plot function in order to display eGo results on leaflet OSM map. This function will open the results in your main web browser.

Parameters

- **ego** (*ego.tools.io.eGo*) – eGo `eGo` inclueds eTraGo and eDisGo results
- **tiles** (*str*) – Folium background map style *None* as OSM or *Nasa*
- **geoloc** (*list*) – List which define center of map as (lon, lat)
- **save_image** (*bool*) – save iplot map as image

Returns **plot** – HTML file with .js plot

Return type html

`ego.tools.plots.`**colormapper_lines**(*colormap*, *lines*, *line*, *column='s_nom'*)
Make Colore Map for lines.

`ego.tools.plots.`**iplot_griddistrict_legend**(*mp*, *repre_grids*, *start=False*)
Add legend to iplot function of mv grids.

`ego.tools.plots.`**iplot_totalresults_legend**(*mp*, *ego*, *start=False*)
Add total results as legend to iplot function.

ego.tools.results

This module include the results functions for analyze and creating results based on eTraGo or eDisGo for eGo.

`ego.tools.results.`**create_etrago_results**(*network*, *scn_name*)
Create eTraGo results

Parameters

- **network** (`NetworkScenario`) – eTraGo `NetworkScenario` based on PyPSA Network. See also pypsa.network
- **scn_name** (*str*) – Name of used scenario

Returns **generator** – Result of generator as DataFrame in `ego.etrago.generator`

Return type pandas.DataFrame

ego.tools.specs

ego.tools.storages

This module contains functions for storage units.

`ego.tools.storages.`**etrago_storages**(*network*)
Sum up the pysical storage values of the total scenario based on eTraGo results.

Parameters **network** (etrago.tools.io.NetworkScenario) – eTraGo NetworkScenario based on PyPSA Network. See also pypsa.network

Returns **results** – Summarize and returns a DataFrame of the storage optimaziation.

Return type pandas.DataFrame

Notes

The results dataframe incluedes following parameters:

charge [numeric] Quantity of charged energy in MWh over scenario time steps

discharge [numeric] Quantity of discharged energy in MWh over scenario time steps

count [int] Number of storage units

p_nom_o_sum: numeric Sum of optimal installed power capacity

ego.tools.storages.**etrago_storages_investment**(*network*, *json_file*, *session*)
Calculate storage investment costs of eTraGo

Parameters **network** (etrago.tools.io.NetworkScenario) – eTraGo NetworkScenario based on PyPSA Network. See also pypsa.network

Returns **storage_costs** – Storage costs of selected snapshots in [EUR]

Return type numeric

ego.tools.utilities

This module contains utility functions for the eGo application.

ego.tools.utilities.**define_logging**(*name*)
Helps to log your modeling process with eGo and defines all settings.

Parameters **log_name** (*str*) – Name of log file. Default: ego.log.

Returns **logger** – Set up logger object of package logging

Return type logging.basicConfig.

ego.tools.utilities.**get_scenario_setting**(*jsonpath=None*)
Get and open json file with scenaio settings of eGo. The settings incluede eGo, eTraGo and eDisGo specific settings of arguments and parameters for a reproducible calculation.

Parameters **json_file** (*str*) – Default: scenario_setting.json Name of scenario setting json file

Returns **json_file** – Dictionary of json file

Return type dict

ego.tools.utilities.**fix_leading_separator**(*csv_file*, ***kwargs*)
Takes the path to a csv-file. If the first line of this file has a leading separator in its header, this field is deleted. If this is done the second field of every row is removed, too.

ego.tools.utilities.**get_time_steps**(*json_file*)
Get time step of calculation by scenario settings.

Parameters **json_file** (dict) – Dictionary of the scenario_setting.json file

Returns **time_step** – Number of timesteps of the calculation.

Return type int

ego.tools.utilities.**open_oedb_session**(*ego*)

1.7.2 scenario_settings.json

With the `scenario_settings.json` file you set up your calcualtion. The file can be found on github.

scenario_setting.json
This file contains all input settings for the eGo tool.

Object Properties

- **global** (*global*) – Global (superordinate) settings that are valid for both, eTraGo and eDisGo.
- **eTraGo** (*eTraGo*) – eTraGo settings, only valid for eTraGo runs.
- **eDisGo** (*eDisGo*) – eDisGo settings, only valid for eDisGo runs.

global

Object Properties

- **eTraGo** (*bool*) – Decide if you want to run the eTraGo tool (HV/EHV grid optimization).
- **eDisGo** (*bool*) – Decide if you want to run the eDisGo tool (MV grid optimiztaion). Please note: eDisGo requires eTraGo= `true`.
- **csv_import_eTraGo** (*string*) – `false` or path to previously calculated eTraGo results (in order to reload the results instead of performing a new run).
- **csv_import_eDisGo** (*string*) – `false` or path to previously calculated eDisGo results (in order to reload the results instead of performing a new run).

eTraGo
This section of *scenario_setting.json* contains all input parameters for the eTraGo tool. A description of the parameters can be found here.

eDisGo
This section of *scenario_setting.json* contains all input parameters for the eDisGo tool and the clustering of MV grids.

Object Properties

- **gridversion** (*string*) – This parameter is currently not used.
- **grid_path** (*string*) – Path to the MV grid files (created by ding0) (e.g. `''data/MV_grids/20180713110719''`)
- **choice_mode** (*string*) – Mode that eGo uses to chose MV grids out of the files in **grid_path** (e.g. `''manual''`, `''cluster''` or `''all''`). If `''manual''` is chosen, the parameter **manual_grids** must contain a list of the desired grids. If `''cluster''` is chosen, **no_grids** must specify the desired number of clusters and **cluster_attributes** must specify the applied cluster attributes. If `''all''` is chosen, all MV grids from **grid_path** are calculated.
- **cluster_attributes** (*list*) – List of strings containing the desired cluster attributes. Available attributes are all attributes returned from :py:func:‘~ego.mv_clustering.mv_clustering.get_cluster_attributes.
- **only_cluster** (*bool*) – If `true`, eGo only identifies cluster results, but performs no eDisGo run. Please note that for **only_cluster** an eTraGo run or dataset must be provided.

- **manual_grids** (*list*) – List of MV grid ID's in case of **choice_mode** = ''manual'' (e.g. [1718,1719]). Ohterwise this parameter is ignored.
- **n_clusters** (*int*) – Number of MV grid clusters (from all grids in **grid_path**, a specified number of representative clusters is calculated) in case of **choice_mode** = ''cluster''. Otherwise this parameter is ignored.
- **parallelization** (*bool*) – If false, eDisgo is used in a consecutive way (this may take very long time). In order to increase the performance of MV grid simulations, true allows the parallel calculation of MV grids. If **parallelization** = true, **max_calc_time** and **max_workers** must be specified.
- **max_calc_time** (*float*) – Maximum calculation time in hours for eDisGo simulations. The calculation is terminated after this time and all costs are extrapolated based on the unfinished simulation. Please note that this parameter is only used if **parallelization** = true.
- **max_workers** (*ing*) – Number of workers (cpus) that are allocated to the simulation. If the given value exceeds the number of available workers, it is reduced to the number of available workers. Please note that this parameter is only used if **parallelization** = true.
- **max_cos_phi_renewable** (*float*) – Maximum power factor for wind and solar generators in MV grids (e.g. 0.9). If the reactive power (as calculated by eTraGo) exceeds this power factor, the reactive power is reduced in order to reach the power factor conditions.
- **solver** (*string*) – Solver eDisGo uses to optimize the curtailment and storage integration (e.g. ''gurobi'').
- **results** (*string*) – Path to folder where eDisGo's results will be saved.
- **tasks** (*list*) – List of string defining the tasks to run. The eDisGo calculation for each MV grid can be devided into separate tasks which is helpful in case one tasks fails and calculations do not need to started in the beginning. The following tasks exist: ''1_setup_grid'', ''2_specs_overlying_grid'', ''3_temporal_complexity_reduction'', ''4_optimisation'', ''5_grid_reinforcement''.

1.7.3 appl.py

This is the application file for the tool eGo. The application eGo calculates the distribution and transmission grids of eTraGo and eDisGo.

Note: Note, the data source of eGo relies on the Open Energy Database. - The registration for the public accessible API can be found on openenergy-platform.org/login.

Run the appl.py file with:

```
>>> python3 -i appl.py
>>> ...
>>> INFO:ego:Start calculation
>>> ...
```

The eGo application works like:

```
>>> from ego.tools.io import eGo
>>> ego = eGo(jsonpath='scenario_setting.json')
```

(continues on next page)

(continued from previous page)

```
>>> ego.etrago_line_loading()
>>> print(ego.etrago.storage_costs)
>>> ...
>>> INFO:ego:Start calculation
>>> ...
```

Take also a look into the documentation of eTraGo and eDisGo which are part of eGo.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[NEP2015a] Übertragungsnetzbetreiber Deutschland. (2015). *Netzentwicklungsplan Strom 2025 - Kostenschaetzungen*, Version 2015, 1. Entwurf, 2015. (https://www.netzentwicklungsplan.de/sites/default/files/paragraphs-files/kostenschaetzungen_nep_2025_1_entwurf.pdf)

[Dena] dena Verteilnetzstudie. (2012). *Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030.* , Version 2015. (https://shop.dena.de/sortiment/detail/produkt/dena-verteilnetzstudie-ausbau-und-innovationsbedarf-der-stromverteilnetze-in-deutschland-bis-2030/)

[PyPSA] PyPSA's documentation (2018). *Documentation of components.* , Version v0.11.0. (https://pypsa.org/doc/components.html)

[StromNEV_A1] Stromnetzentgeltverordnung - StromNEV Anlage 1 (2018). *Verordnung über die Entgelte für den Zugang zu Elektrizitätsversorgungsnetzen (Stromnetzentgeltverordnung - StromNEV) Anlage 1 (zu § 6 Abs. 5 Satz 1) Betriebsgewöhnliche Nutzungsdauern*. (https://www.gesetze-im-internet.de/stromnev/anlage_1.html)

[eDisGo] eDisGo - grid expantion costs (2018). *Cost assumption on mv and lv grid components*. (https://github.com/openego/eDisGo/blob/dev/edisgo/config/config_grid_expansion_default.cfg#L85-L107)

[CONSENTEC] CONSENTEC et.al (2006). *Untersuchung der Voraussetzungen und möglicher Anwendung analytischer* *Kostenmodelle in der deutschen Energiewirtschaft *. (https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Energie/Unternehmen_Institutionen/Netzentgelte/Anreizregulierung/GA_AnalytischeKostenmodelle.pdf?__blob=publicationFile&v=1)

Python Module Index

e

ego.tools.economics, 18
ego.tools.edisgo_integration, 20
ego.tools.io, 22
ego.tools.plots, 23
ego.tools.results, 26
ego.tools.storages, 26
ego.tools.utilities, 27

Index

A

annuity_per_period() (*in module ego.tools.economics*), 18

C

carriers_colore() (*in module ego.tools.plots*), 23
colormapper_lines() (*in module ego.tools.plots*), 26
create_etrago_results() (*in module ego.tools.results*), 26

D

define_logging() (*in module ego.tools.utilities*), 27

E

edisgo (*ego.tools.io.eDisGoResults attribute*), 22
edisgo_convert_capital_costs() (*in module ego.tools.economics*), 18
edisgo_grid_investment() (*in module ego.tools.economics*), 20
EDisGoNetworks (*class in ego.tools.edisgo_integration*), 20
eDisGoResults (*class in ego.tools.io*), 22
eGo (*class in ego.tools.io*), 22
ego.tools.economics (*module*), 18
ego.tools.edisgo_integration (*module*), 20
ego.tools.io (*module*), 22
ego.tools.plots (*module*), 23
ego.tools.results (*module*), 26
ego.tools.storages (*module*), 26
ego.tools.utilities (*module*), 27
ego_colore() (*in module ego.tools.plots*), 23
egoBasic (*class in ego.tools.io*), 22
etrago_convert_overnight_cost() (*in module ego.tools.economics*), 18
etrago_from_oedb() (*in module ego.tools.io*), 23
etrago_grid_investment() (*in module ego.tools.economics*), 19
etrago_operating_costs() (*in module ego.tools.economics*), 19
etrago_storages() (*in module ego.tools.storages*), 26
etrago_storages_investment() (*in module ego.tools.storages*), 27
eTraGoResults (*class in ego.tools.io*), 22

F

fix_leading_separator() (*in module ego.tools.utilities*), 27

G

get_country() (*in module ego.tools.plots*), 25
get_generator_investment() (*in module ego.tools.economics*), 20
get_scenario_setting() (*in module ego.tools.utilities*), 27
get_time_steps() (*in module ego.tools.utilities*), 27
grid_choice (*ego.tools.edisgo_integration.EDisGoNetworks attribute*), 20
grid_investment_costs (*ego.tools.edisgo_integration.EDisGoNetworks attribute*), 20

I

igeoplot() (*in module ego.tools.plots*), 26
iplot (*ego.tools.io.eGo attribute*), 23
iplot_griddistrict_legend() (*in module ego.tools.plots*), 26
iplot_totalresults_legend() (*in module ego.tools.plots*), 26

J

JSON Objects
    eDisGo, 28
    eTraGo, 28
    global, 28
    scenario_setting.json, 28

N

network (*ego.tools.edisgo_integration.EDisGoNetworks attribute*), 20

O

open_oedb_session() (*in module ego.tools.utilities*), 28

P

parallelizer() (*in module ego.tools.edisgo_integration*), 21
plot_edisgo_cluster() (*ego.tools.io.eGo method*), 23
plot_edisgo_cluster() (*in module ego.tools.plots*), 25
plot_grid_expansion_costs() (*ego.tools.edisgo_integration.EDisGoNetworks method*), 21
plot_grid_storage_investment() (*in module ego.tools.plots*), 24
plot_line_expansion() (*ego.tools.io.eGo method*), 23
plot_line_expansion() (*in module ego.tools.plots*), 24
plot_line_loading() (*ego.tools.edisgo_integration.EDisGoNetworks method*), 21
plot_mv_grid_topology() (*ego.tools.edisgo_integration.EDisGoNetworks method*), 21
plot_power_price() (*ego.tools.io.eGo method*), 23
plot_storage_expansion() (*ego.tools.io.eGo method*), 23
plot_storage_expansion() (*in module ego.tools.plots*), 24
plot_storage_integration() (*ego.tools.edisgo_integration.EDisGoNetworks method*), 21
plot_storage_usage() (*ego.tools.io.eGo method*), 23
plot_storage_use() (*in module ego.tools.plots*), 25
plot_total_investment_costs() (*ego.tools.io.eGo method*), 23
power_price_plot() (*in module ego.tools.plots*), 24
prepareGD() (*in module ego.tools.plots*), 25

R

recover_resultsettings() (*in module ego.tools.io*), 23
results_to_excel() (*in module ego.tools.io*), 23
run_edisgo() (*ego.tools.edisgo_integration.EDisGoNetworks method*), 21

S

successful_grids (*ego.tools.edisgo_integration.EDisGoNetworks attribute*), 20

T

total_investment_costs (*ego.tools.io.eGo attribute*), 22
total_operation_costs (*ego.tools.io.eGo attribute*), 22