ding0 Documentation

open$_{e}Go - Team$

Sep 06, 2023

Contents

DIstribution Network GeneratOr – A tool to generate synthetic medium and low voltage power distribution grids based on open (or at least accessible) data.

CHAPTER 1

What is ding0 about?

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. This software project is part of the research project open_eGo.

The theoretical background is detailed in section *Theoretical background*. Install the software package as explained *Installation on Linux*. Take up on the *How to use ding0?* to understand how to use the software.

A standardized presentation of ding0 can be found in the factsheet on the OEP.

CHAPTER 2

Getting started

2.1 Installation on Linux

Note: Installation is only tested on (Debian-like) Linux OS.

Ding0 is provided through PyPi package management and, thus, installable from sources of pip3. You may need to additionally install some specific system packages for a successful installation of Ding0 and its dependencies.

The script *ding0_system_dependencies.sh* installs required system package dependencies.

```
cd <your-ding0-install-path>

chmod +x ding0_system_dependencies.sh

sudo ./ding0_system_dependencies.sh
```

We recommend installing Ding0 (and in general third-party) python packages in a virtual environment, encapsulated from the system python distribution. This is optional. If you want to follow our suggestion, install the tool virtualenv by

```
sudo apt-get install virtualenv # since Ubuntu 16.04
```

Afterwards *virtualenv* allows you to create multiple parallel python distributions. Since Ding0 relies on Python 3, we specify this in the virtualenv creation. Create a new one for Ding0 by

```
# Adjust path to your specific needs
virtualenv -p python3.8 ~/virtualenvs/ding0
```

Jump into (aka. activate) this python distribution by

```
# Adjust path to your specific needs
source ~/virtualenvs/ding0/bin/activate
```

From that, the latest release of ding0 is installed by

```
pip install ding0
```

Pip allows to install a developer version of a package that uses currently checked out code. A developer mode installation is achieved by cloning the repository to an arbitrary path (e.g. *~/repos/* in the following example) and installing manually via pip:

```
mkdir ~/repos/
cd ~/repos/
git clone https://github.com/openego/ding0.git # for SSH use: git clone git@github.
↪com:openego/ding0.git
pip install -e ~/repos/ding0/
```

2.1.1 Installation on Windows

To install Ding0 in windows, it is currently recommended to use Anaconda or Miniconda and create an environment with the ding0_env.yml file provided.

Note: Normally both miniconda and Anaconda are packaged with the Anaconda Prompt to be used in Windows. Within typical installations, this restricts the use of the conda command to only within this prompt. Depending on your convenience, it may be a wise choice to add the conda command to your path during the installation by checking the appropriate checkbox. This would allow conda to be used from anywhere in the operating system except for PowerShell

Note: Conda and Powershell don't seem to be working well together at the moment. There seems to be an issue with Powershell spawning a new command prompt for the execution of every command. This makes the environment activate in a different prompt from the one you may be working with after activation. This may eventually get fixed later on but for now, we would recommend using only the standard cmd.exe on windows.

If you're using Git Bash on Windows, you might have to add conda to paths to have the *conda* command available (adjust path to your Conda installation):

```
. /c/ProgramData/Anaconda3/etc/profile.d/conda.sh
```

To create a ding0 environment using the yaml file in conda, use the command:

```
conda env create -f ding0_env.yml
```

By default this environment will be called ding0_env. If you would like to use a custom name for your environment use the following variant of the command:

```
conda env create -n custom_env_name -f ding0_env.yml
```

An to activate this environment, from any folder in the operating system, use the command:

```
conda activate ding0_env
```

Once the environment is activated, you have two options to install ding0. Either install it from the local repository with the command:

```
pip install -e \path\to\ding0\
```

Or install it from the pypi repository with the command:

```
pip install ding0
```

2.2 Use Ding0

Have a look at the *How to use ding0?*.

CHAPTER 3

How to use ding0?

3.1 Examples

We provide two examples of how to use Ding0 along with two example for analysis of resulting data. The `first example` shows how Ding0 is applied to a single medium-voltage grid district. Grid topology for the medium- and low-voltage grid level is generated and saved to a file (.pkl). The `analysis script` takes data generated in the first example and produces exemplary key figures and plots.

The `second example` shows how to generate a larger number of grid topology data sets. As the current data source sometimes produces unuseful data or leads to program execution interruptions, grids that cannot be created are excluded from grid topology generation. This is enable by setting `failsafe=` to *True*. The according `analysis script` provides exemplary plots for data of multiple grid districts.

3.2 High-level functions

3.2.1 Run ding0

Check out `run_ding0()` as high-level function which is also used in the `example`.

3.2.2 For larger calculation (parallelization)

To generate data for a larger area consider to parallelize execution of Ding0 as done in the `parallelization example`.

3.3 Analysis of grid data

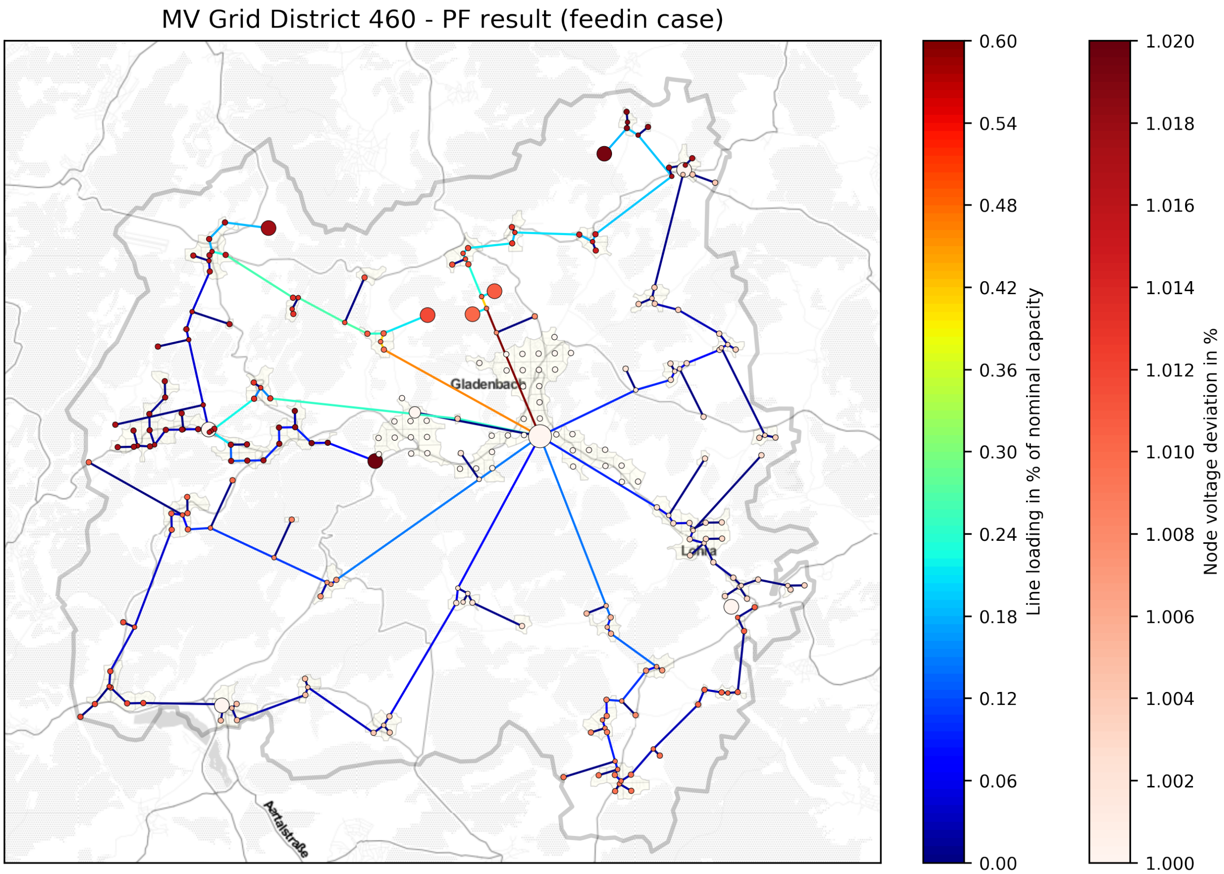
3.3.1 Plot results

The `plot_mv_topology()` allows plots of the MV grid including grid topology with line loadings and node voltages. You can simply fire it using an MVGrid instance or pass argument `export_figures=True` to `run_ding0()` to export some key plots. The plotting allows to draw a background map. For this function, the package *contextily* is needed which is not included in the standard ding0 installation. If you want to use this feature, you can simply install it by

```
pip3 install contextily
```

See plotting function for a detailed description of possible modes and parameters.

Example plot:



3.3.2 Export key figures

We provide a set of functions to export key figures of the generated data. The following assumes a Ding0 network is generated as follows:

```
from egoio.tools import db
from ding0.core import NetworkDing0
```

(continues on next page)

(continued from previous page)

```
engine = db.connection(readonly=True)
session = sessionmaker(bind=engine)()

network = NetworkDing0(name='network')
network.run_ding0(
    session=session,
    mv_grid_districts_no=[3040])
```

Extract key information about medium and low voltage grid topology.

```
from ding0.tools.results import calculate_mvgd_stats

# statistical key figures of medium voltage grid
mv_stats = calculate_mvgd_stats(network)

# statistical key figures of medium voltage grid
lv_stats = calculate_lvgd_stats(network)
```

Information about power flows and voltage levels from final approving power flow analysis can be obtained from `calculate_mvgd_voltage_current_stats()` and `calculate_lvgd_voltage_current_stats()`.

If a large number of grid districts is involved consider to parallelize the execution by

```
mv_stats,
 lvgd_stat
 mv_nodes,
 mv_edges,
 lv_nodes,
 lv_edges = parallel_running_stats(
    districts_list = mv_grid_districts,
    n_of_processes = n_of_processes,
    n_of_districts = n_of_districts,
    source = 'pkl',
    mode = '')
```

Data is read from file and returned in six tables.

Furthermore, the function `to_dataframe()` allows to get tabular information about nodes and edges of the grid topology representing graph.

```
nodes, edges = network.to_dataframe()
```

3.3.3 Compare data versions

Data generated by different versions of Ding0 or different input data can be easily compared. Load datasets designated for comparison and pass to `dataframe_equal()`.

```
network_a = load_nd_from_pickle(filename='filename_a.pkl')
network_b = load_nd_from_pickle(filename='filename_b.pkl')

passed, msg = dataframe_equal(network_a, network_b)
```

3.3.4 Explanation of key figures

Parameter	Description	Unit
km_cable	Cumulative length of underground cables	km

3.4 CSV file export

Ding0 objects are exported in csv files.

3.4.1 Lines

Table 3.1: line.csv

Field	type	Description	Unit
edge_name	str	unambiguous name of edge	n/a
grid_id_db	int	unambiguous id_db of corresponding grid (MVgrid-id if MV-edge, LVgrid-id if LV-edge	n/a
type_kind	str		n/a
type_name	str		n/a
node1	str	id_db of first node	n/a
node2	str	id_db of second node	n/a
length	float	length of line	km
U_n	float	nominal voltage	kV
R	float		Ohm/km
C	float		uF/km
L	float		mH/km
I_max_th	float		A
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.2 LV-Branchtees

Table 3.2: lv_branchtee.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'LV-CableDistributorDing0_LV_#lvgridid#_#ascendingnumber#'	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	None	geometric coordinates	n/a
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.3 LV-Generators

Table 3.3: lv_generator.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'LV-GeneratorDing0_LV_#lvgridid#_#ascendingnumber#'	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
type	str	type of generation	{solar; biomass}
subtype	str	subtype of generation: {solar_roof_mounted, unknown; biomass}	n/a
v_level	int	voltage level of generator	
nominal_capacity	float	nominal capacity	
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.4 LV-Grids

Table 3.4: lv_grid.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'LVGrid-Ding0_LV_#lvgridid#_#lvgridid#'	n/a
LV_grid_id	int	unambiguous number of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 MULTIPOLYGON
population	int	population in LV-Grid	?
voltage_nom	float	voltage level of grid	kV
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.5 LV-Loads

Table 3.5: lv_load.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'LVLoadDing0_LV_#lvgridid#_#ascendingnumber#'	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	None	geometric coordinates	n/a
consumption	{''str'': float}	type of load {residential, agricultural, industrial} and corresponding consumption	n/a
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.6 LV-Stations

Table 3.6: lvmv_station.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'LVStationDing0_MV_#mvgridid#_#lvgridid#'	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.7 LV-Transformers

Table 3.7: lv_transformer.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'TransformerDing0_LV_#mvgridid#_#lvgridid#'	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
voltage_op	float		kV
S_nom	float	nominal apparent power	kVA
X	float		Ohm
R	float		Ohm
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.8 LV-Grids

Table 3.8: mvlv_mapping.csv

Field	type	Description	Unit
LV_grid_id	int	unambiguous number of LV-Grid	n/a
MV_grid_id	int	unambiguous number of MV-Grid	n/a
LV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
MV_grid_id_db	int	unambiguous id_db of MV-Grid	n/a
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.9 MV-Branchtees

Table 3.9: mv_branchtee.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'MV-CableDistributorDing0_MV_#mvgridid#_#ascendingnumber#'	n/a
MV_grid_id_db	int	unambiguous id_db of MV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.10 MV-Generators

Table 3.10: mv_generator.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'MV-GeneratorDing0_MV_#mvgridid#_#ascendingnumber#'	n/a
MV_grid_id_db	int	unambiguous id_db of MV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
type	str	type of generation: {solar; biomass}	n/a
subtype	str	subtype of generation: {solar_ground_mounted, solar_roof_mounted, unknown; biomass, biogas}	n/a
v_level	int	voltage level of generator	
nominal_capacity	float	nominal capacity	
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.11 MV-Grids

Table 3.11: mv_grid.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'MVGrid-Ding0_MV_#mvgridid#_#mvgridid#'	n/a
MV_grid_id	int	unambiguous number of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 MULTIPOLYGON
population	int	population in LV-Grid	?
voltage_nom	float	voltage level of grid	kV
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.12 MV-Loads

Table 3.12: mv_load.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'MVLoadDing0_MV_#mvgridid#_#ascendingnumber#'	n/a
MV_grid_id_db	int	unambiguous id_db of MV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POLYGON
consumption	{''str'': float}	type of load {retail, residential, agricultural, industrial} and corresponding consumption	n/a
is_aggregated	boolean	True if load is aggregated load, else False	n/a
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.13 MV-Stations

Table 3.13: mvhv_station.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'MVStationDing0_MV_#mvgridid#_#mvgridid#'	n/a
MV_grid_id_db	int	unambiguous id_db of MV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
run_id	int	time and date of table generation	yyyyMMddhhmmss

3.4.14 MV-Transformers

Table 3.14: lv_transformer.csv

Field	type	Description	Unit
id_db	str	unambiguous name: 'TransformerDing0_MV_#mvgridid#_#mvgridid#'	n/a
MV_grid_id_db	int	unambiguous id_db of LV-Grid	n/a
geom	wkt	geometric coordinates	WGS84 POINT
voltage_op	float		kV
S_nom	float	nominal apparent power	kVA
X	float		Ohm
R	float		Ohm
run_id	int	time and date of table generation	yyyyMMddhhmmss

CHAPTER 4

Theoretical background

4.1 Data basis

The fundamental data basis is decribed in [Huelk2017] and its extension is detailed by [Amme2017]. Further extensions and additional details are provided in the sections below.

Definition of names introduces terms we stick to in the following text.

4.1.1 MV/LV Substations and LV grid district

Medium-voltage/low-voltage (MV/LV) substations are located on a equidistant grid of points with an interval of 180m within the load areas. Cable length in low-voltage (LV) grids ranges from 100-1.500m (see [Kerber], [Scheffler], [Mohrmann]). According to [Scheffler], a cable length of 200 m to 300 m is most typical. Furthermore, we foud a difference between the cable length and the line over ground is 72% (1.39 Umwegfaktor), see master thesis Jonas Gütter. This seems plausible compared to the value for the MV grid of 77% (1.3). The chosen value concludes in cable lengths of 250m at the shortest distance and 283m at the longest distance between the middle point of the square and its outer line.

- Finding LV-Grid districts (LV-GD): We define Voronoi polygons within the load areas based on a grid of points with an interval of 180m.
- Assign consumption to the LV-GD: This works analogously to the methods for the MV-GD, as described in “Allocation of annual electricity consumption and power generation capacities across multi voltage levels in a high spatial resolution” (Huelk)
- Assign peak load

4.2 Medium-voltage grids

Methodological details and exemplary results are presented in [Amme2017].

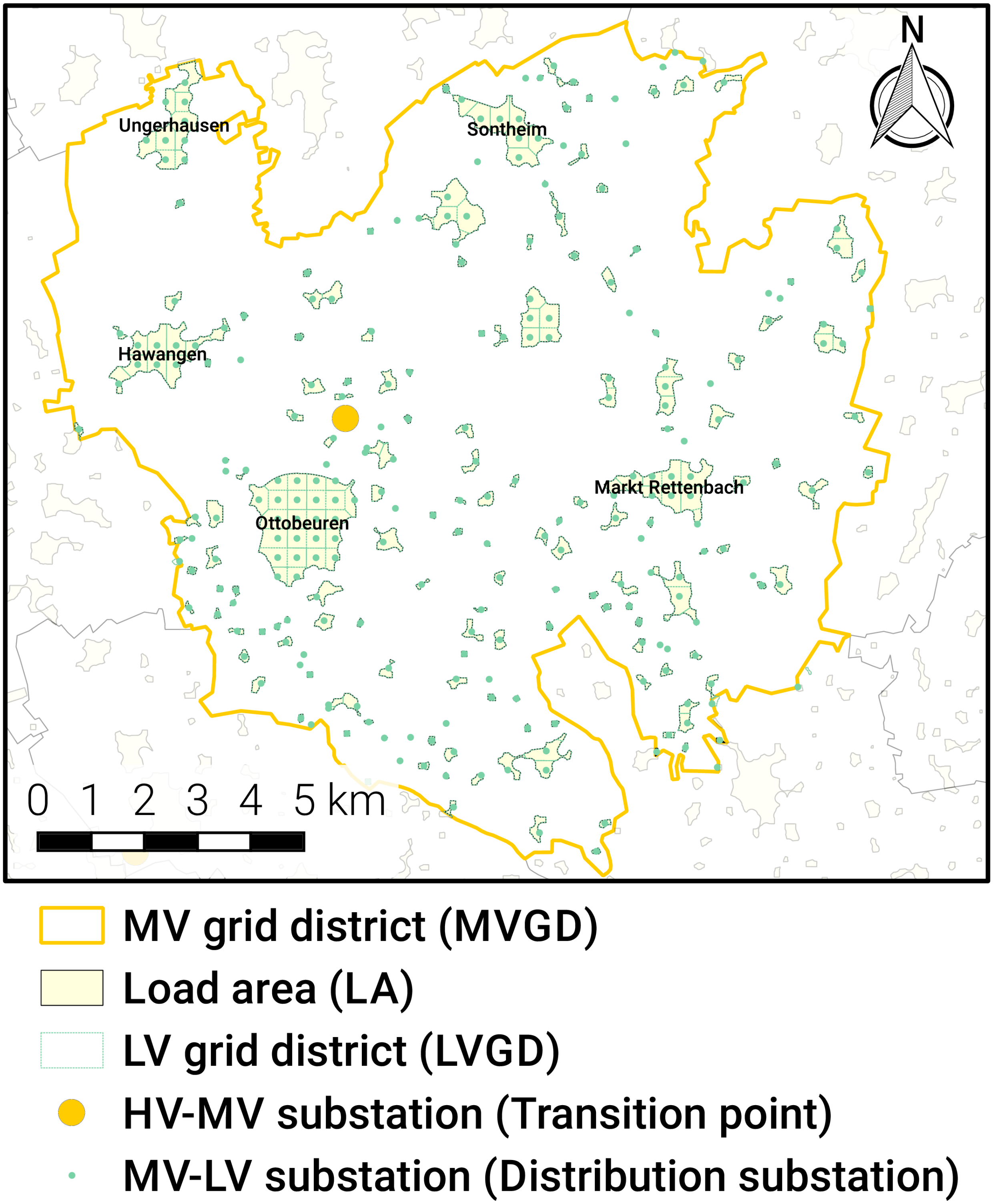


Fig. 4.1: Definition of names

4.3 Low-voltage grids

The topology of low-voltage grids is determined on the basis of typified grid models that are vastly available for the residential sector and partially available for other sector retail, industrial and agricultural. The mentioned sectors are modeled differently: the grid topology of residential sector loads founds on typified grid models from [Kerber]. Retail and industrial sector are treated as a single sector and use same methodology to determine grid topology as applied for the agricultural sector. Loads of each sector are located in separate branches - one for each sector. In the following its creation is described in detail.

However, a method to generate a representative variation of LV-grids, that can be assigned to the modeled LV/MV substations cannot be found. Given data on MV/LV substations:

- land use data divided in industry, commercial, agriculture and residential
- population
- peak load
- Define transformer

4.3.1 Branches of sector residential

1. LV-Branches

   We are using the LV-Branches of Kerber from the grids. They should be assigned to the most plausible types of settlement areas.

2. Define the type of settlement area

   To decide if a LV-grid district is most likely a rural, village or suburban settlement area we are using the population value combined with statistical data. Statisticly, there are 2.3 persons per appartment and 1.5 appartments per house. [see BBR Tabelle B12 http://www.ggr-planung.de/fileadmin/pdf-projekte/SiedEntw_und_InfrastrFolgekosten_Teil_2.pdf] [DEMIREL page 37-41, average has been coosen]. (This is not valid for urban areas.) With this we estimate the amount aus house connections (HC).

   This value can also be found at the explenation of the database of the "Kerber"-grids and is assinged to the type of settlement area:

   - Rural: 622 HC at 43 MV/LV substations results in an average amount of 14.5 HC/substation
   - Village: 2807 HC at 51 MV/LV substations results in an average amount of 55 HC/substation
   - Suburban: 4856 HC at 38 MV/LV substations results in an average amount of 128 HC/substationTher

   With the resulting trendline of this three point, [the Polynomial degree 2 [16.127*(x^2)-7.847*x+6.1848] whereas x is the type of of settlement area], we difine the border values for the typ of settlement area at:

   - Rural <31 HC/substation
   - Village <87 HC/substation
   - Suburban >=87 HC/substation

3. Assinging grid branches to the Substations

   within the "Kerber"-model-grids several grid branches are found.

   - Rural: 5 branches (with l>=78m & l<=676m)

- Village: 7 branches (with l>=102m & l<=588m)
- Suburban: 15 branches (with l>=85 & l<=610m)

Strangzuweisung Zu jeder ONS werden in Abhängigkeit von Netztyp und HA, NS-Stränge zugewiesen Eine Verteilung des Aufkommens der Stränge anhand von der Gesamtstranglänge geschieht mit Hilfe der Scheffler Angaben (Abbildung Länge der Netzstrahlen für ausgewählte Siedlungstypen [44])

1. Categorising grid branches form "Kerber" model grids

Hinzu kommen auf Basis von kerber interpolierte stränge um Lücken in der Vollständigkeit zu schließen

4.3.2 Branches of sector retail/industrial and agricultural

Creating individual LV grid branches for the sectors retail/industrial and agricultural applies the same methodology. The topology of these grid branches determines by the sectoral peak load that is available at high spatial resolution (see [Huelk2017]). Furthermore the number of land-use areas (taken from [OSM]) of each of the sectors determines the number individual loads connected to one or more of these sectoral branches.

The topology of each sectoral branch is affected largely by assumptions on parameters that are provided in the table below.

Parameter	Value
Max. load in each branch	290 kVA
Max. branch length retail/industrial $L_{R/I,max}$	400 m
Max. branch length agricultural $L_{A,max}$	800 m
Length of branch stub	30 m
PV peak power <= 30 kW	residential
PV peak power > 30 kW <= 100 kW	retail/industrial or agricultural
PV peak power > 100 kW	MV/LV station bus bar

In each LV grid district (LVGD) (see *MV/LV Substations and LV grid district*) sectoral peak load of sectors retail+industrial and agricultural are analyzed. The number loads of each sectors determines by dividing sectoral peak load by number of land-use area found in this grid district.

$$N_{loads} = P_{sector} \cdot N_{land-use}$$

In the next step individual loads are allocated to branches considering the limit of max. 290 kVA peak load connected to a single branch. If a single load exceeds the limit of 290 kVA, it is halved until it is smaller than or equal to 290 kVA. Loads are distributed equidistant on the branches while the branch does not necessarily take the maximum length defined in the table above. The distance defines as

$$d_{sector} = \frac{L_{sector,max}}{N_{loads} + 1}$$

Single loads are connected to the branch line by stubs of a length of 30 m.

Photovoltaic (PV) power plants are allocated to different sectoral LV grid branches depending on the nominal power. The allocation by the nominal power is provided in the above table. It follows a simple assumption: smaller PV power plants are allocated to LV grid branches of sector residential, larger power plants are allocated to branches of the other sector, and really large ones are directly connected to the bus bar of the MV-LV substation.

4.3.2.1 Grid stability and equipment

During build of LV grid topology equipment is chosen with respect to max. occurring load and generation according to current grid codes (see [VDEAR]). Nevertheless, some overloading issues may remain. In addition, voltage issues may arise that can't be considered during grid topology creation. Therefore, we adhere to the regulatory framework of [DINEN50160] which is simplified by [VDEAR]. According to [DINEN50160] voltage deviation is limited to +/-10 % of nominal that is for practical use divided into voltage drop/increase for each voltage level and the associated transformers. The allowed voltage increase in the LV grid level is limited to 3 % of nominal voltage. The allowed voltage drop is limited to 5 % as detailed in [Zdrallek].

Following steps do apply during reinforcement of Ding0 LV grids

1. Checks for **overloading** issues at branches and MV-LV transformers first
2. Critical branches (those with line overloading) are extended to appropriate size of cable to transport connected load and generation. Note, if connected load or generation capacity is still exceeding capacity of largest cable type. We keep largest available cable type and the issue most probably will remain
3. Stations are tested for overloading issues for generation and load case as well. If nominal apparent power of transformers of a substation is not sufficient a two-step procedure is applied
    1. Existing transformers are extended (replaced) to comply with load and generation conencted to subsequent grid.
    2. If Step 1 does not resolve all issues additional transformers are build in the substation
4. Subsequently **over-voltage issues** are analyzed for all grid nodes
5. For each node where voltage exceeds 3 % of nominal voltage in feed-in case or 5 % of nominal voltage in load case, branch segments connecting the node with the substation are reinforce until no further issues remain. If a over-voltage issue cannot be solved by installing largest availabe cable (NAYY 4x1x300) this type of cable still remains as well as the overvoltage issue
6. Substations are checked for over-voltage issues at the bus bar individually. Identified issues are resolved by extending nominal apparent power of existing transformer. A ultimately build up to two new transformers in the substation.

4.3.3 References

CHAPTER 5

Calculation principles

5.1 Reactance

We assume all cables and overhead lines to be short lines. Thus, the capacity is not considered in calculation of reactance of overhead lines and cables.

$$x = \omega * L$$

5.2 Apparent power

- Given maximum thermal current *I_th_amx* (*I_L*) is given per conducter (of three cables in a system)/per phase.
- We assume to have delta connection. Thus, nominal voltage per conducted can be applied to calculate apparent power *s_nom* and conductor current *I_L* has to be transformed to *I_delta* respectively to *I* by

$$I = I_{delta} = \sqrt{3} \cdot I_L$$

- Apparent *S* power is calculated to

$$S = U \cdot I = U \cdot I_{th,max}$$

5.3 Sign Convention

Generators and loads in an AC power system can behave either like an inductor or a capacitor. Mathematically, this has two different sign conventions, either from the generator perspective or from the load perspective. This is defined by the direction of power flow from the component.

Both sign conventions are used in Ding0 depending upon the components being defined, similar to pypsa.

5.3.1 Generator Sign Convention

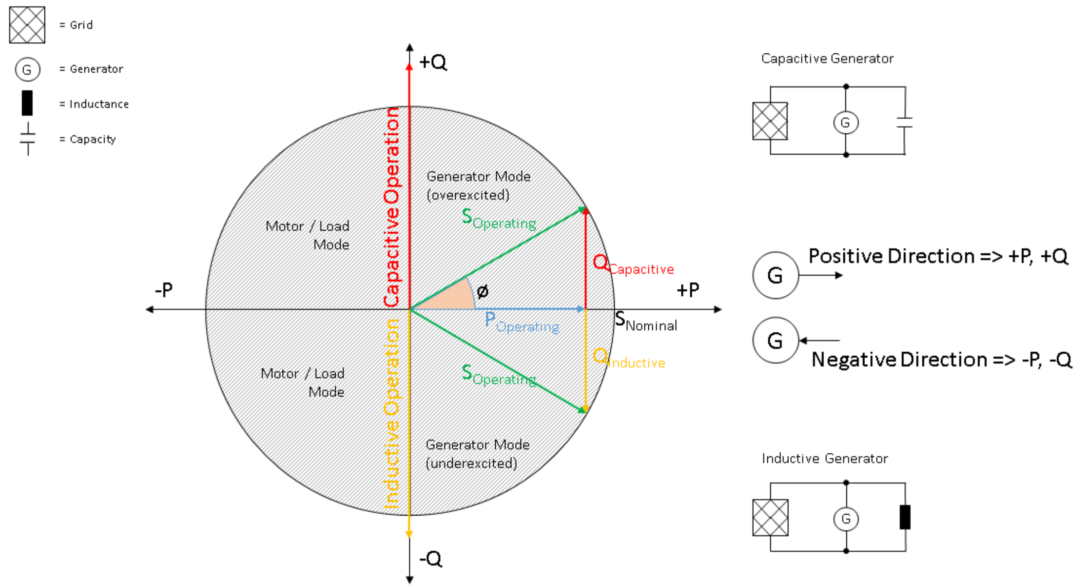


Fig. 5.1: Generator sign convention in detail

5.3.1.1 Load Sign Convention

Ding0 makes the sign convention easier by allowing the user to provide the string values "inductive" or "capacitive" to describe the behaviour of the different assets better. The sign convention for different parts of ding0 are handled internally. By default, generators are assumed to behave capacitively, while loads are assumed to behave inductively.

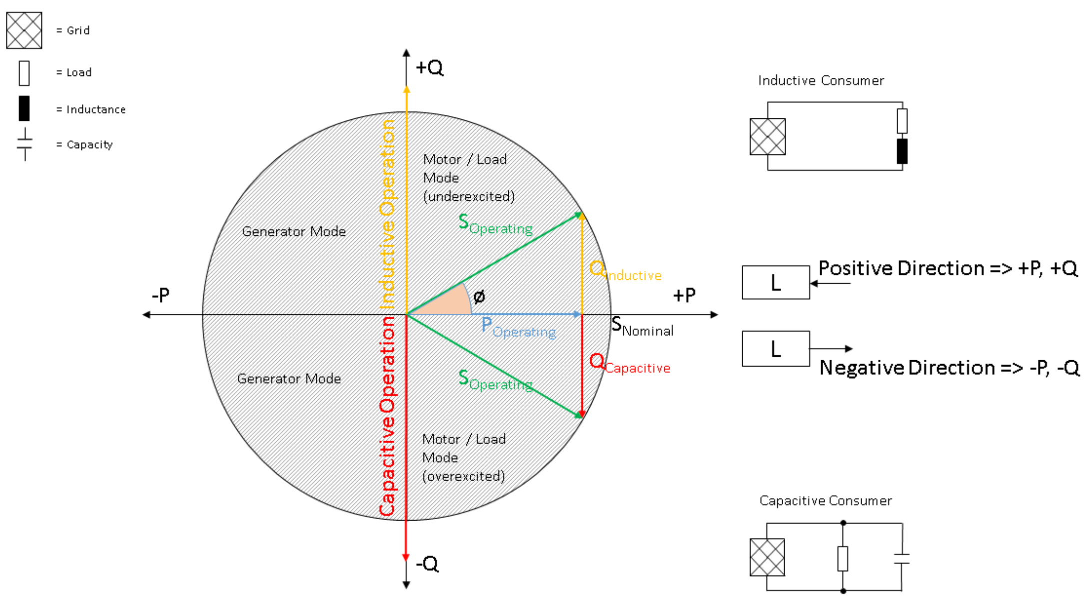


Fig. 5.2: Load sign convention in detail

CHAPTER 6

Notes to developers

If you're interested to contribute and join the project, feel free to submit PR, contact us, or just create an issue if something seems odd.

6.1 Test the package installation

> **Warning:** The scripts for testing the installation might be outdated and will be revised in v0.2.1.

We use Docker to test the build of ding0 on a fresh Ubuntu OS. In order to run such a test make sure docker is installed.

```
cd ./test_installation/
chmod +x install_docker.sh
./install_docker.sh
```

Afterwards you can test if installation of ding0 builts successfully by executing

```
./check_ding0_installation.sh
```

The script `./check_ding0_installation.sh` must be executed in root directory of ding0 repository. Then it installs currently checked out version. The installation process can be observed in the terminal.

6.2 Run unit and integration tests

ding0 comes with a bunch of unit and integration tests on most features. You'll need additional packages listed in *dev_requirements.txt*. To install, use

```
pip install -r /path/to/ding0/dev_requirements.txt
```

To run tests with e.g. 4 workers (you may omit this argument), use

```
cd /path/to/ding0/
pytest --workers 4 -vv
```

6.3 Test ding0 runs

The outcome of different runs of ding0 can be compared with the functions in ~/ding0/tools/tests.py.

To compare the default configuration of a fresh run of ding0 and a saved run use

```
manual_ding0_test()
```

The default behavior is using district [3545] in oedb database and the data in file 'ding0_tests_grids_1.pkl'. For other filenames or districts use, for example:

```
manual_ding0_test([438],'ding0_tests_grids_2.pkl')
```

To create a file with the output of a ding0 run in the default configuration (disctrict [3545] in oedb database and filename 'ding0_tests_grids_1.pkl') use:

```
init_files_for_tests()
```

For other filenames or districts use, for example:

```
init_files_for_tests([438],'ding0_tests_grids_2.pkl')
```

To run the automatic unittest suite use:

```
support.run_unittest(Ding0RunTest)
```

The suite assumes that there are two files allocated in the directory:

- 'ding0_tests_grids_1.pkl'
- 'ding0_tests_grids_2.pkl'

It is assummed that these files store the outcome of different runs of ding0 over different districts.

This suite will run three tests:

- Compare the results stored in the files, testing for equality between the data in 'ding0_tests_grids_1.pkl' and itself; and for difference between both files.
- Compare the results of a fresh ding0 run over district [3545] and the data in 'ding0_tests_grids_1.pkl'.
- Compare the results of two fresh runs of ding0 in district [3545].

6.4 Database integration

Different databases can be used. All communication between ding0 and the used database happens in *ding0/tools/egon_data_integration.py*. The function to start a session is in *ding0/tools/database.py*. Start the session with a with statement:

```
from ding0.tools import database
from ding0.core import NetworkDing0
with database.session_scope() as session:
    nd = NetworkDing0(name="network", session=session)
```

The Object-relational mapping is done in *core.NetworkDing0.import_orm*. The ORM uses sqlalchemy and saio. The mapping is configured in *ding0/config/config_db_tables.cfg*. You can select there the database type (model_draft/versioned/local) and submit your local database credentials.

CHAPTER 7

What's New

See what's new as per release!

Releases

7.1 Release v0.2.1 (May 29, 2021)

7.1.1 Changes

Release to fix ego.io dependency in v0.2.0 (PyPI did not accept git link)

7.2 Release v0.2.0 (May 28, 2021)

7.2.1 Changes

- Added Python 3.8 support #325
- Fix installation with Conda #339
- CSV export in PyPSA format #307 , this required further changes and fixing of tests, cf. #312
- Switched from pyproj1 to pyproj2 for CRS transformations #343
- Reproducible stats by fixing #315, for details see #324
- In the CSV export, (in-building) household loads and generators are no more contained as extra nodes but directly connected to the house's grid connection point to reduce the number of nodes. #322
- Fix sum capacity of grid generators #326
- Fix PyPI description #311

7.3 Release v0.1.12 September 20, 2019

7.3.1 Changes

- Connection of generators in lv_connect_generators was made deterministic. Before, this happened randomly leading to different lv_grids using the same input data. The network creation is now reproducible while lv_branches were reinforced differently before. Should solve #245 and at least parts of #40.
- A proper sign convention (see *Sign Convention*) for P,Q is introduced #266, see also PR #271.
- Identification of critical nodes by VDE norm AR 4105 fixed. All power flows behind node are taken into account now. Solves #300.
- Tests for MV and LV grids are introduced. Additionally, synthetically created grids are introduced, that can be used for testing. These tests verify the functionality of most of the functions in `grids` including the creation and modification of MV and LV grids (e.g. adding generators/transformators..). Focus lies on the appropriate creation of the graphs and it's corresponding routings. Tests are done in grids created with oedb-extracted data and/or synthetic grids, depending on the feature being tested.
- Equipment table data is cleaned so that only necessary literature values are used. Should solve #296
- Labels of all components were made unique.
- ding0 now works without an OpenEnergy DataBase account thanks to changes in the ego.io package that allow readonly queries without a token.

7.4 Release v0.1.10 November 5, 2018

This release introduces new plotting functionalities.

7.4.1 Changes

- New plotting function `plot_mv_topology()` allows plots of the MV grid including grid topology with line loadings and node voltages. You can simply fire it using an MVGrid instance or pass argument `export_figures=True` to `run_ding0()` to export some key figures.
- Find a new Jupyter notebook example here (sorry, currently only in German).
- Fix animation feature in `mv_routing()` to allow image export of routing process.
- Minor bugfixes

7.5 Release v0.1.9 October 22, 2018

This release fixes the API documentation at readthedocs and the examples.

7.6 Release v0.1.8 September 5, 2018

A release to update software dependencies.

- Data processing and ego.io versions are updated to 0.4.5

7.7 Release v0.1.7 July 19, 2018

A release to update software dependencies.

- Explicit dependencies of Pyomo and Scipy are removed

7.8 Release v0.1.6 July 6, 2018)

- Update of underlying data version to v0.4.2 of open_eGo data processing

7.9 Release v0.1.5 (June 6, 2018)

This release provides an update of API docs.

- Update docs: API docs now build properly from a technical perspective #45. The content is still not complete
- Added new generator object GeneratorFluctuating that includes a weather_cell_id #254
- Include oedialect

7.10 Release v0.1.4 (January 17, 2018)

This release provides some fixes, a largely extended export function for statistical information about the grid data and an update of input data.

7.10.1 Added features

- Use data of data processing v0.3.0 and egoio v0.3.0
- Python 3.4 compatible (removed some Python3.5+ introduced feature calls)
- Export of statistical key figures in addition to *to_dataframe()* added
- Now uses PyPSA v0.11.0

7.10.2 Bug fixes

- Remove cable distributor from MV grid's cable distributor list when disconnecting a node eDisGo#48
- Workaround for #155 added
- Package data is now correctly included

7.10.3 Other changes

- Generators with unknown subtype have subtype 'unknown' now
- Circuit breakers are closed now #224
- Version upgrade of Pandas eDisGo #22
- Documentation about usage is updated and extended
- Upgrade of versions of dependencies
- oemof.db is now replace by egoio's connection provider

7.11 Release v0.1.3 (September 1, 2017)

This release fixes bugs reported by first users of Ding0 (data). Furthermore, some features related to the use of Ding0 are added.

7.11.1 Added features

- Run ding0 in parallel #222
- Calculate statistical key figures for MV and LV level #189 and #190

7.11.2 Bug fixes

- Changed constraint on MV grid rings to limiting length of each half ring #224
- Bug related to control of circuit breaker status #226
- Consistently use cos(phi) in Ding0 #197

7.11.3 Other changes

- Update Pandas dependency to 0.20.3
- Update PyPSA dependency to 0.10.0

7.12 Release v0.1.2 (July 25, 2017)

Renaming of package: dingo to ding0

7.13 Release v0.1.0 (July 25, 2017)

As this is the first release of ding0, we built everything from scratch.

CHAPTER 8

---

ding0

8.1 ding0 package

8.1.1 Subpackages

8.1.1.1 ding0.config package

8.1.1.1.1 Submodules

8.1.1.1.2 ding0.config.config_db_interfaces module

class ding0.config.config_db_interfaces.**sqla_mv_grid_viz**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

SQLAlchemy table definition for the export of MV grids for visualization purposes

#TODO: Check docstrings *before* definitions! is that ok?

geom_lv_load_area_centres

Description.

Type type

geom_lv_stations

Description.

Type type

geom_mv_cable_dists

Description.

Type type

geom_mv_circuit_breakers

Description.

Type type

geom_mv_generators
Description.

Type type

geom_mv_lines
Description.

Type type

geom_mv_station
Description.

Type type

grid_id
Description.

Type type

class ding0.config.config_db_interfaces.**sqla_mv_grid_viz_branches**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

SQLAlchemy table definition for the export of MV grids' branches for visualization purposes

#TODO: Check docstrings *after* definitions! is that ok?

branch_id
Description.

Type type

geom
Description.

Type type

grid_id
Description.

Type type

length
Description.

Type type

s_res0
Description.

Type type

s_res1
Description.

Type type

type_kind
Description.

Type type

type_name
Description.

Type `type`

type_s_nom
Description.

Type `type`

type_v_nom
Description.

Type `type`

class `ding0.config.config_db_interfaces.`**sqla_mv_grid_viz_nodes**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

SQLAlchemy table definition for the export of MV grids' branches for visualization purposes

#TODO: Check docstrings *before* definitions! is that ok?

geom
Description.

Type `type`

grid_id
Description.

Type `type`

node_id
Description.

Type `type`

v_nom
Description.

Type `type`

v_res0
Description.

Type `type`

v_res1
Description.

Type `type`

8.1.1.1.3 Module contents

8.1.1.2 ding0.core package

8.1.1.2.1 Subpackages

8.1.1.2.1.1 ding0.core.network package

8.1.1.2.1.2 Submodules

8.1.1.2.1.3 ding0.core.network.cable_distributors module

class ding0.core.network.cable_distributors.**LVCableDistributorDing0**(***kwargs*)

Bases: *ding0.core.network.CableDistributorDing0*

LV Cable distributor (connection point)

string_id

Description #TODO

branch_no

Description #TODO

load_no

Description #TODO

in_building

Description #TODO

pypsa_bus_id

Returns specific ID for representing bus in pypsa network.

Returns str – Representative of pypsa bus

class ding0.core.network.cable_distributors.**MVCableDistributorDing0**(***kwargs*)

Bases: *ding0.core.network.CableDistributorDing0*

MV Cable distributor (connection point)

lv_load_area_group

Description #TODO

pypsa_bus_id

Returns specific ID for representing bus in pypsa network.

Returns str – Representative of pypsa bus

8.1.1.2.1.4 ding0.core.network.grids module

8.1.1.2.1.5 ding0.core.network.loads module

class ding0.core.network.loads.**LVLoadDing0**(***kwargs*)

Bases: *ding0.core.network.LoadDing0*

Load in LV grids

Note: Current attributes to fulfill requirements of typified model grids.

pypsa_bus_id
Creates a unique identification for the generator to export to pypsa using the id_db of the mv_grid and the current object

Returns `str`

class `ding0.core.network.loads.`**MVLoadDing0**(***kwargs*)
Bases: *ding0.core.network.LoadDing0*

Load in MV grids

Note: Currently not used, check later if still required

pypsa_bus_id
Creates a unique identification for the generator to export to pypsa using the id_db of the mv_grid and the current object

Returns `str`

8.1.1.2.1.6 ding0.core.network.stations module

class `ding0.core.network.stations.`**LVStationDing0**(***kwargs*)
Bases: *ding0.core.network.StationDing0*

Defines a LV station in DINGO

peak_generation
Calculates cumulative peak generation of generators connected to underlying LV grid.

This is done instantaneously using bottom-up approach.

Returns *float* – Cumulative peak generation

pypsa_bus0_id
Returns specific ID for representing bus in pypsa network. Representative node at medium voltage side (also used for transformer)

Returns `str` – Representative of pypsa bus

pypsa_bus_id
Returns specific ID for representing bus in pypsa network.

Returns `str` – Representative of pypsa bus

class `ding0.core.network.stations.`**MVStationDing0**(***kwargs*)
Bases: *ding0.core.network.StationDing0*

Defines a MV station in DINGO

peak_generation(*mode*)
Calculates cumulative peak generation of generators connected to underlying grids

This is done instantaneously using bottom-up approach.

Parameters **mode** (`str`) – determines which generators are included:

```
'MV':   Only generation capacities of MV level are considered.

'MVLV': Generation capacities of MV and LV are considered
        (= cumulative generation capacities in entire MVGD).
```

Returns *float* – Cumulative peak generation

pypsa_bus0_id
Returns specific ID for representing bus in pypsa network. Representative node at high voltage side (also used for transformer)

Returns `str` – Representative of pypsa bus

pypsa_bus_id
Returns specific ID for representing bus in pypsa network.

Returns `str` – Representative of pypsa bus

select_transformers()
Selects appropriate transformers for the HV-MV substation.

The transformers are chosen according to max. of load case and feedin-case considering load factors. The HV-MV transformer with the next higher available nominal apparent power is chosen. If one trafo is not sufficient, multiple trafos are used. Additionally, in a second step an redundant trafo is installed with max. capacity of the selected trafos of the first step according to general planning principles for MV distribution grids (n-1).

Parameters

- **transformers** (*dict*) – Contains technical information of p hv/mv transformers
- ****kwargs** (*dict*) – Should contain a value behind the key 'peak_load'

Note: Parametrization of transformers bases on[1].

Potential hv-mv-transformers are chosen according to[2].

References

set_operation_voltage_level()
Set operation voltage level

8.1.1.2.1.7 ding0.core.network.transformers module

8.1.1.2.1.8 Module contents

class ding0.core.network.**BranchDing0**(***kwargs*)
Bases: `object`

When a network has a set of connections that don't form into rings but remain as open stubs, these are designated as branches. Typically Branches at the MV level branch out of Rings.

[1] Deutsche Energie-Agentur GmbH (dena), "dena-Verteilnetzstudie. Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030.", 2012

[2] X. Tao, "Automatisierte Grundsatzplanung von Mittelspannungsnetzen", Dissertation, 2006

length
Length of line given in m

Type float

type
Association to pandas Series. DataFrame with attributes of line/cable.

Type pandas.DataFrame

id_db
id according to database table

Type int

ring
The associated *RingDing0* object

Type *RingDing0*

kind
'line' or 'cable'

Type str

connects_aggregated
A boolean True or False to mark if branch is connecting an aggregated Load Area or not. Defaults to False.

Type :obj'bool'

circuit_breaker
The circuit breaker that opens or closes this Branch.

Type :class:`~.ding0.core.network.CircuitBreakerDing0

critical
This a designation of if the branch is critical or not, defaults to False.

Type bool

Note: Important: id_db is not set until whole grid is finished (setting at the end).

ADDED FOR NEW LV GRID APPROACH: geometry : shapely.LineString

due to for lv grids the coordinates of nodes and edges are known coordinates are stored as LineString to enable visualisation of the right course of the road.

network
Getter for the overarching `NetworkDing0` object.

Returns `NetworkDing0`

class ding0.core.network.**CableDistributorDing0**(***kwargs*)
Bases: object

Cable distributor (connection point)

id_db
id according to database table

Type int

geo_data
The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.

Type Shapely Point object

grid
The MV grid that this ring is to be a part of.

Type `MVGridDing0`

network
Getter for the overarching `NetworkDing0` object.

Returns `NetworkDing0`

class `ding0.core.network.`**`CircuitBreakerDing0`**`(**kwargs)`
Bases: `object`

Class for modelling a circuit breaker

id_db
id according to database table

Type `int`

geo_data
The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.

Type Shapely Point object

grid
The MV or LV grid that this Load is to be a part of.

Type *`GridDing0`*

branch
The branch to which the Cable Distributor belongs to

Type *`BranchDing0`*

branch_nodes
A tuple containing a pair of ding0 node objects i.e. *`GeneratorDing0`* or *`GeneratorFluctuatingDing0`* or *`LoadDing0`* or *`StationDing0`* or *`CircuitBreakerDing0`* or *`CableDistributorDing0`* or *`MVLoadDing0`*.

Type `tuple`

status
The `open` or `closed` state of the Circuit Breaker.

Type `str`, default 'closed'

Note: Circuit breakers are nodes of a graph, but are NOT connected via an edge. They are associated to a specific *branch* of a graph (and the branch refers to the circuit breaker via the attribute *circuit_breaker*) and its two *branch_nodes*. Via open() and close() the associated branch can be removed from or added to graph.

---

close()
Close a Circuit Breaker

network
Getter for the overarching `NetworkDing0` object.

Returns NetworkDing0

open()
Open a Circuit Breaker

class ding0.core.network.**GeneratorDing0**(***kwargs*)
Bases: object

Generators (power plants of any kind)

id_db
id according to database table

Type int

name
This is a name that can be given by the user. This defaults to a name automatically generated.

Type str

v_level
voltage level

Type float

geo_data
The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.

Type Shapely Point object

mv_grid
The MV grid that this ring is to be a part of.

Type MVGridDing0

lv_load_area
The LV Load Area the the generator is a part of.

Type *LVLoadAreaDing0*

lv_grid
The LV Grid that the Generator is a part of.

Type LVGridDing0

capacity
The generator's rated power output in kilowatts.

Type float

capacity_factor
The generators capacity factor i.e. the ratio of the average power generated by the generator versus the generator capacity.

Type float

type
The generator's type, an option amongst:

- solar
- wind
- geothermal
- reservoir

- pumped_storage
- run_of_river
- gas
- biomass
- coal
- lignite
- gas
- gas_mine
- oil
- waste
- uranium
- other_non_renewable

  Type `str`

subtype
The generator's subtype, an option amongst:

- solar_roof_mounted
- solar_ground_mounted
- wind_onshore
- wind_offshore
- hydro
- geothermal
- biogas_from_grid
- biomass
- biogas
- biofuel
- biogas_dry_fermentation
- gas_mine
- gas_sewage
- gas_landfill
- gas
- waste_wood
- wood

  Type `str`

network
Getter for the overarching `NetworkDing0` object.

Returns NetworkDing0

pypsa_bus_id
Creates a unique identification for the generator to export to pypsa using the id_db of the mv_grid and the current object

Returns str

class ding0.core.network.**GeneratorFluctuatingDing0**(***kwargs*)
Bases: *ding0.core.network.GeneratorDing0*

Generator object for fluctuating renewable energy sources

_weather_cell_id
ID of the weather cell used to generate feed-in time series

Type str

weather_cell_id
Get weather cell ID :returns: str – See class definition for details.

class ding0.core.network.**GridDing0**(***kwargs*)
Bases: object

The fundamental abstract class used to encapsulate the networkx graph and the relevant attributes of a power grid irrespective of voltage level. By design, this class is not expected to be instantiated directly. This class was designed to be inherited by MVGridDing0 or by LVGridDing0.

Parameters

- **network** (NetworkDing0) – The overarching *CableDistributorDing0* object that this object is connected to.
- **id_db** (str) – id according to database table
- **grid_district** (Shapely Polygon object) – class, area that is covered by the lv grid
- **v_level** (int) – The integer value of the voltage level of the Grid in kV. Typically, either 10 or 20.

cable_distributors
List of *CableDistributorDing0* Objects

Type list

loads
List of of *LoadDing0* Objects. These are objects meant to be considered as MV-Level loads

Type list

generators
list of *GeneratorDing0* or *GeneratorFluctuatingDing0* Objects. These are objects meant to be considered as MV-Level Generators.

Type list

graph
The networkx graph of the network. Initially this is an empty graph which gets populated differently depending upon which child class inherits this class, either LVGridDing0 or MVGridDing0.

Type NetworkX Graph Objnetworkx.Graph

add_generator(*generator*)
Adds a generator to _generators and grid graph if not already existing

Parameters **generator** (*GeneratorDing0* or *GeneratorFluctuatingDing0*) – Ding0's generator object

cable_distributors()
Provides access to the cable distributors in the grid.

Returns list – List generator of *CableDistributorDing0* objects

cable_distributors_count()
Returns the count of cable distributors in grid

Returns int – Count of the *CableDistributorDing0* objects

control_generators(*capacity_factor*)
Sets capacity factor of all generators of a grid.

A capacity factor of 0.6 means that all generators are to provide a capacity of 60% of their nominal power.

Parameters **capacity_factor** (float) – Value between 0 and 1.

find_and_union_paths(*node_source*, *nodes_target*)
Determines shortest paths from *node_source* to all nodes in *node_target* in _graph using find_path().

The branches of all paths are stored in a set - the result is a list of unique branches.

Parameters

- **node_source** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – source node, member of _graph, ding0 node object
- **node_target** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – target node, member of _graph, ding0 node object

Returns list – List of *BranchDing0* objects

find_path(*node_source*, *node_target*, *type='nodes'*)
Determines shortest path

Determines the shortest path from *node_source* to *node_target* in _graph using networkx' shortest path algorithm.

Parameters

- **node_source** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – source node, member of _graph, ding0 node object
- **node_target** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – target node, member of _graph, ding0 node object
- **type** (str) – Specify if nodes or edges should be returned. Default is *nodes*

Returns

- list – List of ding0 node object i.e. *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*

- **path** (shortest path from *node_source* to) – *node_target* (list of nodes in _graph)

Note: WARNING: The shortest path is calculated using the count of hops, not the actual line lengths! As long as the circuit breakers are open, this works fine since there's only one path. But if they are closed, there are 2 possible paths. The result is a path which have min. count of hops but might have a longer total path length than the second sone. See networkx' function shortest_path() function for details on how the path is calculated.

generators()
Returns a generator for iterating over grid's generators

Returns `list` generator – List of *GeneratorDing0* and *GeneratorFluctuatingDing0* objects

graph
Provide access to the graph

graph_add_node(*node_object*)
Adds a station or cable distributor object to grid graph if not already existing

Parameters **node_object** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – The ding0 node object to be added to the graph

graph_branches_from_node(*node*)
Returns branches that are connected to *node*

Parameters **node** (*GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*) – Ding0 node object (member of graph)

Returns

`list` – List of `tuple` objects i.e. List of tuples (node, branch in *BranchDing0*)

```
(node , branch_0 ),
...,
(node , branch_N ),
```

node in ding0 is either *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*

graph_draw(*mode*)
Draws grid graph using networkx

This method is for debugging purposes only. Use `plot_mv_topology()` for advanced plotting.

Parameters **mode** (`str`) – Mode selection 'MV' or 'LV'.

Note: The geo coords (for used crs see database import in class *NetworkDing0*) are used as positions for drawing but networkx uses cartesian crs. Since no coordinate transformation is performed, the drawn graph representation is falsified!

graph_edges()
Returns a generator for iterating over graph edges

The edge of a graph is described by the two adjacent node and the branch object itself. Whereas the branch object is used to hold all relevant power system parameters.

Returns

`dict` generator –

Dictionary generator with the keys:

- *adj_nodes* paired to the Ding0 node object i.e. *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*
- *branch* paried with the Ding0 branch object *BranchDing0*

Note: There are generator functions for nodes (*Graph.nodes()*) and edges (*Graph.edges()*) in NetworkX but unlike graph nodes, which can be represented by objects, branch objects can only be accessed by using an edge attribute ('branch' is used here)

To make access to attributes of the branch objects simpler and more intuitive for the user, this generator yields a dictionary for each edge that contains information about adjacent nodes and the branch object.

Note, the construction of the dictionary highly depends on the structure of the in-going tuple (which is defined by the needs of networkX). If this changes, the code will break.

---

graph_isolated_nodes()

Finds isolated nodes = nodes with no neighbors (degree zero)

Returns `list` – List of ding0 node objects i.e. *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*

graph_nodes_from_branch(*branch*)

Returns nodes that are connected by *branch* i.e. a *BranchDing0* object.

Parameters **branch** (*BranchDing0*) –

Returns `tuple` – Tuple of node objects in ding0. 2-tuple of Ding0 node objects i.e. *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*

graph_nodes_sorted()

Returns an sorted list of graph's nodes. The nodes are arranged based on the name in ascending order.

Returns `list` – List of *GeneratorDing0* or *GeneratorFluctuatingDing0* or *LoadDing0* or *StationDing0* or *CircuitBreakerDing0* or *CableDistributorDing0* or *MVLoadDing0*

graph_path_length(*node_source*, *node_target*)

Calculates the absolute distance between *node_source* and *node_target* in meters using find_path() and branches' length attribute.

node_source: ***GeneratorDing0*** **or** ***GeneratorFluctuatingDing0*** **or** ***LoadDing0*** **or** ***StationDing0*** **or** ***Cir***

source node, member of _graph, ding0 node object

node_target: ***GeneratorDing0*** **or** ***GeneratorFluctuatingDing0*** **or** ***LoadDing0*** **or** ***StationDing0*** **or** ***Cir***

target node, member of _graph, ding0 node object

Returns `float` – path length in meters

loads()
Returns a generator for iterating over grid's loads

Returns list generator – List Generator of *LoadDing0* objects

loads_count()
Returns the count of loads in grid

Returns int – Count of the *LoadDing0* objects

class ding0.core.network.**LoadDing0**(***kwargs*)
Bases: object

Class for modelling a load

id_db
id according to database table

Type int

geo_data
The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.

Type Shapely Point object

grid
The MV or LV grid that this Load is to be a part of.

Type *GridDing0*

peak_load
Peak load of the current object

Type float

building_id
refers to OSM oder eGo^n ID, depending on chosen database

Type int

network
Getter for the overarching NetworkDing0 object.

Returns NetworkDing0

class ding0.core.network.**RingDing0**(***kwargs*)
Bases: object

Represents a medium voltage Ring.

Parameters **grid** (MVGridDing0) – The MV grid that this ring is to be a part of.

branches()
Getter for the branches in the *RingDing0* object.

Returns list generator – List generator of *BranchDing0* objects

lv_load_areas()
Getter for the LV Load Areas that this Ring covers.

Returns list generator – List generator of *LVLoadAreaDing0* objects

network
Getter for the overarching NetworkDing0 object.

Returns NetworkDing0

class ding0.core.network.**StationDing0**(***kwargs*)
Bases: object

The abstract definition of a substation irrespective of voltage level. This object encapsulates the attributes that can appropriately represent a station in a networkx graph as a node. By design, this class is not expected to be instantiated directly. This class was designed to be inherited by *MVStationDing0* or by *LVStationDing0*.

Parameters

- **id_db** (str) – id according to database table
- **v_level_operation** (float) – operation voltage level in kilovolts (kV) at station (the station's voltage level differs from the nominal voltage level of the grid due to grid losses). It is usually set to a slightly higher value than the nominal voltage, e.g. 104% in MV grids.
- **geo_data** (Shapely Point object) – The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.
- **grid** (*GridDing0*) – Either a MVGridDing0 or MVGridDing0 object
- **_transformers** (list of) – *TransformerDing0* objects

add_transformer(*transformer*)
Adds a transformer to _transformers if not already existing

Parameters **transformer** (*TransformerDing0*) – The *TransformerDing0* object to be added to the current *StationDing0*

network
Getter for the overarching NetworkDing0 object

Returns NetworkDing0

peak_load
Cumulative peak load of loads connected to underlying MV or LV grid

(taken from MV or LV Grid District -> top-down)

Returns float – Peak load of the current *StationDing0* object

---

Note: This peak load includes all loads which are located within Grid District: When called from MV station, all loads of all Load Areas are considered (peak load was calculated in MVGridDistrictDing0.add_peak_demand()). When called from LV station, all loads of the LVGridDistrict are considered.

---

transformers()
Returns a generator for iterating over transformers

Returns list generator – List generator of *TransformerDing0* objects

class ding0.core.network.**TransformerDing0**(***kwargs*)
Bases: object

Transformers are essentially voltage converters, which enable to change between voltage levels based on the usage.

id_db
id according to database table

Type int

grid
The MV grid that this ring is to be a part of.

Type `MVGridDing0`

v_level
voltage level [kV]

Type `float`

s_max_a
rated power (long term) [kVA]

Type `float`

s_max_b
rated power (short term)

Type `float`

s_max_c
rated power (emergency)

Type `float`

phase_angle
phase shift angle

Type `float`

tap_ratio
off nominal turns ratio

Type `float`

network
Getter for the overarching `NetworkDing0` object.

Returns `NetworkDing0`

z (*voltage_level=None*)
Calculates the complex impedance in Ohm related to voltage_level. If voltage_level is not inserted, the secondary voltage of the transformer is chosen as a default. :param voltage_level: voltage in [kV] :return: Z_tr in [Ohm]

8.1.1.2.1.9 ding0.core.powerflow package

8.1.1.2.1.10 Module contents

class `ding0.core.powerflow.`**PFConfigDing0**(***kwargs*)
Bases: `object`

Defines the PF scenario configuration

Parameters

- **scenarios** (`list` of `str`) – List of strings describing the scenarios
- **timerange** (`list` of pandas.DatetimeIndex) – List of Pandas DatetimeIndex objects
- **timesteps_count** (`int`) – count of timesteps the timesteps to be created
- **timestep_start** (pandas.DatetimeIndex) – Description #TODO

- **resolution** (str) – String or pandas offset object, e.g. 'H'=hourly resolution,

  to learn more see http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases

- **srid** (*type*) – partial reference system indentifier used by PyPSA's plots #TODO

Note: This class can be called as follows:

i) With scenarios and timeranges:

```
scenarios = ['scn_1', ...,  'scn_n'],
timeranges= [timerange_1, ..., timerange_n]
```

ii) With scenarios, start time and count of timesteps:

```
scenarios = ['scn_1', ...,  'scn_n'],
timesteps_count = m,
timestep_start = datetime()
```

(in this case, n timeranges with m timesteps starting from datetime will be created)

---

resolution
Returns resolution

scenarios
Returns a generator for iterating over PF scenarios

srid
Returns SRID

timesteps
Returns a generator for iterating over PF timesteps

ding0.core.powerflow.**q_sign**(*reactive_power_mode_string*, *sign_convention*)
Gets the correct sign for Q time series given 'inductive' and 'capacitive' and the 'generator' or 'load' convention.

Parameters

- **reactive_power_mode_string** (str) – Either 'inductive' or 'capacitive'
- **sign_convention** (str) – Either 'load' or 'generator'

Returns obj: *int* : +1 or -1 – A sign to mulitply to Q time sereis

8.1.1.2.1.11 ding0.core.structure package

8.1.1.2.1.12 Submodules

8.1.1.2.1.13 ding0.core.structure.groups module

class ding0.core.structure.groups.**LoadAreaGroupDing0**(***kwargs*)
Bases: object

Container for small load_areas / load areas (satellites).

A group of stations which are within the same satellite string. It is required to check whether a satellite string has got more load or string length than allowed, hence new nodes cannot be added to it.

id_db
Descr

Type int

mv_grid_district
Desc

Type Shapely Polygon object

add_lv_load_area(*lv_load_area*)
Adds a LV load_area to _lv_load_areas if not already existing

Parameters **lv_load_area** (Shapely Polygon object) – Descr

can_add_lv_load_area(*node*)
Sums up peak load of LV stations

That is, total peak load for satellite string

Parameters **node** (GridDing0) – Descr

Returns obj: *bool* – True if ????

lv_load_areas()
Returns a generator for iterating over load_areas

Yields *int* – generator for iterating over load_areas

network

8.1.1.2.1.14 ding0.core.structure.regions module

class ding0.core.structure.regions.**LVGridDistrictDing0**(***kwargs*)
Bases: *ding0.core.structure.RegionDing0*

Describes region that is covered by a single LV grid

Parameters

- **geo_data** (Shapely Polygon object) – The geo-spatial polygon in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.
- **lv_load_area** (Shapely Polygon object) – Descr
- **lv_grid** (Shapely Polygon object) – Descr
- **population** (float) – Descr
- **peak_load_residential** (float) – Descr
- **peak_load_retail** (float) – Descr
- **peak_load_industrial** (float) – Descr
- **peak_load_agricultural** (float) – Descr
- **peak_load** (float) – Descr
- **sector_count_residential** (int) – Descr
- **sector_count_retail** (int) – Descr
- **sector_count_industrial** (int) – Descr
- **sector_count_agricultural** (int) – Descr

TODO UPDATE DESCR FOR GRAPH AND BUILDINGS

network

class ding0.core.structure.regions.**LVLoadAreaCentreDing0**(***kwargs*)

Bases: object

Defines a region centre in Ding0.

The centres are used in the MV routing as nodes.

Note: Centre is a point within a region's polygon that is located most central (e.g. in a simple region shape like a circle it's the geometric center).

Parameters

- **id_db** (int) – unique ID in database (=id of associated load area)
- **grid** (int) – Descr
- **geo_data** (Shapely Point object) – The geo-spatial point in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.
- **lv_load_area** (LVLoadAreaDing0) – Descr

network

pypsa_bus_id

Remove Returns specific ID for representing bus in pypsa network.

Returns str – Representative of pypsa bus

Type Todo

class ding0.core.structure.regions.**LVLoadAreaDing0**(***kwargs*)

Bases: *ding0.core.structure.RegionDing0*

Defines a LV-load_area in DINGO

ring

Descr

Type int

mv_grid_district

Descr

Type Shapely Polygon object

lv_load_area_centre

Descr

Type Shapely Point object

lv_load_area_group

Descr

Type Shapely Polygon object

is_satellite

Descr

Type bool

is_aggregated
Descr

Type bool

db_data
Descr

Type pandas.DatetimeIndex

 # new osm approach

load_area_graph
contains all streets in load_area

Type networkx.MultiDiGraph

MVLoads
list containing ding0.MVLoads in lvla

Type list

add_lv_grid_district(*lv_grid_district*)
Adds a LV grid district to _lv_grid_districts if not already existing

Parameters **lv_grid_district** (Shapely Polygon object) – Descr

add_mv_load(*mv_load*)
Adds a MVLoad to _lv_grid_districts if not already existing

Parameters **mv_load** (*ding0.core.network.MVLoadDing0*) –

lv_grid_districts()
Returns a generator for iterating over LV grid districts

Yields *int* – generator for iterating over LV grid districts

lv_grid_districts_count()
Returns the count of LV grid districts

Returns *int* – Number of LV grid districts.

mv_loads_count()
Returns the count of MV loads

Returns *int* – Number of MV loads.

network

peak_generation
Cumulative peak generation of generators connected to LV grids of underlying LVGDs

class ding0.core.structure.regions.**MVGridDistrictDing0**(***kwargs*)
Bases: *ding0.core.structure.RegionDing0*

Defines a MV-grid_district in DfINGO

mv_grid
Descr

Type int

geo_data
The geo-spatial Polygon in the coordinate reference system with the SRID:4326 or epsg:4326, this is the project used by the ellipsoid WGS 84.

Type Shapely Polygon object

peak_load
Descr

Type float

peak_load_satellites
Descr

Type float

peak_load_aggregated
Descr

Type float

add_aggregated_peak_demand()
Summarizes peak loads of underlying aggregated load_areas

add_lv_load_area(*lv_load_area*)
Adds a Load Area *lv_load_area* to _lv_load_areas if not already existing

Additionally, adds the associated centre object to MV grid's _graph as node.

Parameters **lv_load_area** (LVLoadAreaDing0) – instance of class LVLoadAreaDing0

add_lv_load_area_group(*lv_load_area_group*)
Adds a LV load_area to _lv_load_areas if not already existing.

add_peak_demand()
Summarizes peak loads of underlying load_areas in kVA.

(peak load sum and peak load of satellites)

lv_load_area_groups()
Returns a generator for iterating over LV load_area groups.

Yields *int* – generator for iterating over LV load_areas

lv_load_area_groups_count()
Returns the count of LV load_area groups in MV region

Returns *int* – Number of LV load_area groups in MV region.

lv_load_areas()
Returns a generator for iterating over load_areas

Yields *int* – generator for iterating over load_areas

network

8.1.1.2.1.15 Module contents

class ding0.core.structure.**RegionDing0**(***kwargs*)
Bases: object

Defines a region in DING0

8.1.1.2.2 Module contents

8.1.1.3 ding0.flexopt package

8.1.1.3.1 Submodules

8.1.1.3.2 ding0.flexopt.check_tech_constraints module

ding0.flexopt.check_tech_constraints.**check_load**(*grid*, *mode*)

Checks for over-loading of branches and transformers for MV or LV grid.

Parameters

- **grid** (GridDing0) – Grid identifier.
- **mode** (str) – Kind of grid ('MV' or 'LV').

Returns

- dict – Dict of critical branches with max. relative overloading, and the following format:

```
{
branch_1: rel_overloading_1,
...,
branch_n: rel_overloading_n
}
```

- list of *TransformerDing0* objects –

  List of critical transformers with the following format:

```
[trafo_1, ..., trafo_m]
```

Note: Lines'/cables' max. capacity (load case and feed-in case) are taken from[1].

References

See also:

ding0.flexopt.reinforce_measures.reinforce_branches_current(), *ding0.flexopt.reinforce_measures.reinforce_branches_voltage()*

ding0.flexopt.check_tech_constraints.**check_voltage**(*grid*, *mode*)

Checks for voltage stability issues at all nodes for MV or LV grid

Parameters

- **grid** (GridDing0) – Grid identifier.
- **mode** (str) – Kind of grid ('MV' or 'LV').

Returns

list of Ding0 node object (member of graph) either –

- *GeneratorDing0* or

[1] dena VNS

- *GeneratorFluctuatingDing0* or
- *LoadDing0* or
- *StationDing0* or
- *CircuitBreakerDing0* or
- *CableDistributorDing0*

List of critical nodes, sorted descending by voltage difference.

Note: The examination is done in two steps, according to[2] :

1. It is checked #TODO: what?
2. #TODO: what's next?

References

`ding0.flexopt.check_tech_constraints.`**`get_critical_line_loading`**(*grid*)

Assign line loading to each branch determined by peak load and peak generation of descendant branches

The attribute *s_res* is a list of two elements 1. apparent power in load case 2. apparent power in feed-in case

Parameters **grid** (`LVGridDing0`) – Ding0 LV grid object

Returns

- `list` – List of critical branches incl. its line loading
- `list` – List of critical stations incl. its transformer loading

`ding0.flexopt.check_tech_constraints.`**`get_critical_voltage_at_nodes`**(*grid*)

Estimate voltage drop/increase induced by loads/generators connected to the grid.

Based on voltage level at each node of the grid critical nodes in terms of exceed tolerable voltage drop/increase are determined. The tolerable voltage drop/increase is defined by[3] a adds up to 3 % of nominal voltage. The longitudinal voltage drop at each line segment is estimated by a simplified approach (neglecting the transverse voltage drop) described in[3].

Two equations are available for assessing voltage drop/ voltage increase.

The first is used to assess a voltage drop in the load case

$$Deltau = fracS_{Amax} \cdot (R_{kV} \cdot cos(\phi) + X_{kV} \cdot sin(\phi))U_{nom}$$

The second equation can be used to assess the voltage increase in case of feedin. The only difference is the negative sign before X. This is related to consider a voltage drop due to inductive operation of generators.

$$Deltau = fracS_{Amax} \cdot (R_{kV} \cdot cos(\phi) - X_{kV} \cdot sin(\phi))U_{nom}$$

[2] dena VNS

[3] VDE Anwenderrichtlinie: Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz, 2011

Symbol	Description
Δu	Voltage drop/increase at node
S_{Amax}	Apparent power
R_{kV}	Short-circuit resistance
X_{kV}	Short-circuit reactance
$cos(\phi)$	Power factor
U_{nom}	Nominal voltage

Parameters **grid** (LVGridDing0) – Ding0 LV grid object

Note: The implementation highly depends on topology of LV grid. This must not change its topology from radial grid with stubs branching from radial branches. In general, the approach of[3] is only applicable to grids of radial topology.

We consider the transverse voltage drop/increase by applying the same methodology successively on results of main branch. The voltage drop/increase at each house connection branch (aka. stub branch or grid connection point) is estimated by superposition based on voltage level in the main branch cable distributor.

References

`ding0.flexopt.check_tech_constraints.`**`get_cumulated_conn_gen_load`**(*graph*, *node*)

Get generation capacity/ peak load of all descending nodes

Parameters

- **graph** (NetworkX Graph Obj) – Directed graph
- **node** (*graph node*) – Node of the main branch of LV grid

Returns

`list` – A list containing two items

cumulated peak load of connected loads at descending nodes of node # cumulated generation capacity of connected generators at descending nodes of node

`ding0.flexopt.check_tech_constraints.`**`get_delta_voltage_preceding_line`**(*grid*, *tree*, *node*)

Parameters

- **grid** (LVGridDing0) – Ding0 grid object
- **tree** (NetworkX Graph Obj) – Tree of grid topology
- **node** (*graph node*) – Node at end of line

Returns `float` – Voltage drop over preceding line of node

`ding0.flexopt.check_tech_constraints.`**`get_mv_impedance_at_voltage_level`**(*grid*, *voltage_level*)

Determine MV grid impedance (resistance and reactance separately)

Parameters

- **grid** (LVGridDing0) –

- **voltage_level** (*float*) – voltage level to which impedance is rescaled (normally 0.4 kV for LV)

Returns list – List containing resistance and reactance of MV grid

ding0.flexopt.check_tech_constraints.**get_voltage_at_bus_bar**(*grid*, *tree*)

Determine voltage level at bus bar of MV-LV substation

Parameters

- **grid** (LVGridDing0) – Ding0 grid object
- **tree** (NetworkX Graph Obj) – Tree of grid topology:

Returns list – Voltage at bus bar. First item refers to load case, second item refers to voltage in feedin (generation) case

ding0.flexopt.check_tech_constraints.**get_voltage_delta_branch**(*tree*, *node*, *r*, *x*)

Determine voltage for a branch with impedance r + jx

Parameters

- **tree** (NetworkX Graph Obj) – Tree of grid topology
- **node** (*graph node*) – Node to determine voltage level at
- **r** (*float*) – Resistance of preceeding branch
- **x** (*float*) – Reactance of preceeding branch

Returns float – Delta voltage for branch

ding0.flexopt.check_tech_constraints.**peak_load_generation_at_node**(*nodes*)

Get maximum occuring load and generation at a certain node

Summarizes peak loads and nominal generation power of descendant nodes of a branch

Parameters **nodes** (list) – Any LV grid Ding0 node object that is part of the grid topology

Returns

- float – peak_load : Sum of peak loads of descendant nodes
- float – peak_generation : Sum of nominal power of generation at descendant nodes

ding0.flexopt.check_tech_constraints.**voltage_delta_vde**(*v_nom*, *s_max*, *r*, *x*, *cos_phi*)

Estimate voltrage drop/increase

The VDE[4] proposes a simplified method to estimate voltage drop or increase in radial grids.

Parameters

- **v_nom** (int) – Nominal voltage
- **s_max** (float) – Apparent power
- **r** (float) – Short-circuit resistance from node to HV/MV substation (in ohm)
- **x** (float) – Short-circuit reactance from node to HV/MV substation (in ohm). Must be a signed number indicating (+) inductive reactive consumer (load case) or (-) inductive reactive supplier (generation case)
- **cos_phi** (float) – The cosine phi of the connected generator or load that induces the voltage change

[4] VDE Anwenderrichtlinie: Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz, 2011

Returns `float` – Voltage drop or increase

References

8.1.1.3.3 ding0.flexopt.reinforce_grid module

ding0.flexopt.reinforce_grid.**reinforce_grid**(*grid*, *mode*)

Evaluates grid reinforcement needs and performs measures

Grid reinforcement according to methods described in [VNSRP] supplemented by [DENA].

Parameters

- **grid** (GridDing0) – Grid instance
- **mode** (`str`) – Choose of: 'MV' or 'LV'

Note: Currently only MV branch reinforcement is implemented. HV-MV stations are not reinforced since not required for status-quo scenario.

References

8.1.1.3.4 ding0.flexopt.reinforce_measures module

ding0.flexopt.reinforce_measures.**extend_substation**(*grid*, *critical_stations*, *grid_level*)

Reinforce MV or LV substation by exchanging the existing trafo and installing a parallel one if necessary.

First, all available transformers in a *critical_stations* are extended to maximum power. If this does not solve all present issues, additional transformers are build.

Parameters

- **grid** (GridDing0) – Ding0 grid container
- **critical_stations** (`list`) – List of stations with overloading
- **grid_level** (`str`) – Either "LV" or "MV". Basis to select right equipment.

Note: Curently straight forward implemented for LV stations

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

ding0.flexopt.reinforce_measures.**extend_substation_voltage**(*crit_stations*, *grid_level='LV'*)

Extend substation if voltage issues at the substation occur

Follows a two-step procedure:

i) Existing transformers are extended by replacement with large nominal apparent power
ii) New additional transformers added to substation (see 'Note')

Parameters

- **crit_stations** (`list`) – List of stations with overloading or voltage issues.

- **grid_level** (str) – Specifiy grid level: 'MV' or 'LV'

Note: At maximum 2 new of largest (currently 630 kVA) transformer are additionally built to resolve voltage issues at MV-LV substation bus bar.

ding0.flexopt.reinforce_measures.**extend_trafo_power**(*extendable_trafos*, *trafo_params*)

Extend power of first trafo in list of extendable trafos

Parameters

- **extendable_trafos** (list) – Trafos with rated power below maximum size available trafo
- **trafo_params** (pandas.DataFrame) – Transformer parameters

ding0.flexopt.reinforce_measures.**new_substation**(*grid*)

Reinforce MV grid by installing a new primary substation opposite to the existing one

Parameters **grid** (MVGridDing0) – MV Grid identifier.

ding0.flexopt.reinforce_measures.**reinforce_branches_current**(*grid*, *crit_branches*)

Reinforce MV or LV grid by installing a new branch/line type

Parameters

- **grid** (GridDing0) – Grid identifier.
- **crit_branches** (*dict*) – Dict of critical branches with max. relative overloading.

Note: The branch type to be installed is determined per branch using the rel. overloading. According to[5] only cables are installed.

References

See also:

ding0.flexopt.check_tech_constraints.check_load(), *ding0.flexopt.reinforce_measures.reinforce_branches_voltage()*

ding0.flexopt.reinforce_measures.**reinforce_branches_voltage**(*grid*, *crit_branches*, *grid_level='MV'*)

Reinforce MV or LV grid by installing a new branch/line type

Parameters

- **grid** (GridDing0) – Grid identifier.
- **crit_branches** (list of int) – List of critical branches. #TODO: check if a list or a dictionary
- **grid_level** (str) – Specifying either 'MV' for medium-voltage grid or 'LV' for low-voltage grid level.

[5] Ackermann et al. (RP VNS)

Note: The branch type to be installed is determined per branch - the next larger cable available is used. According to Ackermann only cables are installed.

See also:

ding0.flexopt.check_tech_constraints.check_load(), *ding0.flexopt.reinforce_measures.reinforce_branches_voltage()*

`ding0.flexopt.reinforce_measures.`**`reinforce_lv_branches_overloading`**(*grid*, *crit_branches*)

Choose appropriate cable type for branches with line overloading

Parameters

- **grid** (`LVGridDing0`) – Ding0 LV grid object
- **crit_branches** (`list`) – List of critical branches incl. its line loading

Note: If maximum size cable is not capable to resolve issue due to line overloading largest available cable type is assigned to branch.

Returns `list` – unsolved_branches : List of braches no suitable cable could be found

8.1.1.3.5 ding0.flexopt.reinforce_measures_dena module

`ding0.flexopt.reinforce_measures_dena.`**`extend_substation`**(*grid*)

Reinforce MV or LV substation by exchanging the existing trafo and installing a parallel one if necessary with according to dena

Parameters **grid** (`GridDing0`) – Grid identifier.

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

`ding0.flexopt.reinforce_measures_dena.`**`new_substation`**(*grid*)

Reinforce MV grid by installing a new primary substation opposite to the existing one according to dena

Parameters **grid** (`MVGridDing0`) – Grid identifier.

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

`ding0.flexopt.reinforce_measures_dena.`**`parallel_branch`**(*grid*, *node_target*)

Reinforce MV or LV grid by installing a new parallel branch according to dena

Parameters

- **grid** (`GridDing0`) – Grid identifier.
- **node_target** (`int`) – node where the parallel cable (starting from HV/MV substation) is connected to (used when grid is a MV grid)

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

`ding0.flexopt.reinforce_measures_dena.`**`split_ring`**(*grid*)

Reinforce MV grid by splitting a critical ring into two new rings according to dena

Parameters **grid** (`MVGridDing0`) – Grid identifier.

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

8.1.1.3.6 Module contents

8.1.1.4 ding0.grid package

8.1.1.4.1 Subpackages

8.1.1.4.1.1 ding0.grid.lv_grid package

8.1.1.4.1.2 Submodules

8.1.1.4.1.3 ding0.grid.lv_grid.build_grid module

ding0.grid.lv_grid.build_grid.**build_lv_graph_residential**(*lvgd*, *selected_string_df*)

Builds nxGraph based on the LV grid model

Parameters

- **lvgd** (`LVGridDistrictDing0`) – Low-voltage grid district object
- **selected_string_df** (pandas.DataFrame) – Table of strings of the selected grid model

Note: To understand what is happening in this method a few data table columns are explained here

- *count house branch*: number of houses connected to a string
- *distance house branch*: distance on a string between two house branches
- *string length*: total length of a string
- *length house branch A|B*: cable from string to connection point of a house

A|B in general brings some variation in to the typified model grid and refer to different length of house branches and different cable types respectively different cable widths.

ding0.grid.lv_grid.build_grid.**build_lv_graph_ria**(*lvgd*, *grid_model_params*)

Build graph for LV grid of sectors retail/industrial and agricultural

Based on structural description of LV grid topology for sectors retail/industrial and agricultural (RIA) branches for these sectors are created and attached to the LV grid's MV-LV substation bus bar.

LV loads of the sectors retail/industrial and agricultural are located in separat branches for each sector (in case of large load multiple of these). These loads are distributed across the branches by an equidistant distribution.

This function accepts the dict *grid_model_params* with particular structure

```
>>> grid_model_params = {
>>> ... 'agricultural': {
>>> ...     'max_loads_per_branch': 2
>>> ...     'single_peak_load': 140,
>>> ...     'full_branches': 2,
>>> ...     'remaining_loads': 1,
>>> ...     'load_distance': 800/3,
>>> ...     'load_distance_remaining': 400}}
```

Parameters

- **lvgd** (`LVGridDistrictDing0`) – Low-voltage grid district object

- **grid_model_params** (*dict*) – Dict of structural information of sectoral LV grid branchwith particular structure, e.g.:

```
grid_model_params = {
    'agricultural': {
        'max_loads_per_branch': 2
        'single_peak_load': 140,
        'full_branches': 2,
        'remaining_loads': 1,
        'load_distance': 800/3,
        'load_distance_remaining': 400
    }
}
```

Note: We assume a distance from the load to the branch it is connected to of 30 m. This assumption is defined in the config files.

ding0.grid.lv_grid.build_grid.**build_residential_branches**(*lvgd*)
Based on population and identified peak load data, the according grid topology for residential sector is determined and attached to the grid graph

Parameters **lvgd** (LVGridDistrictDing0) – Low-voltage grid district object

ding0.grid.lv_grid.build_grid.**build_ret_ind_agr_branches**(*lvgd*)
Determine topology of LV grid for retail/industrial and agricultural sector and create representative graph of the grid

Parameters **lvgd** (LVGridDistrictDing0) – Low-voltage grid district object

ding0.grid.lv_grid.build_grid.**grid_model_params_ria**(*lvgd*)
Determine grid model parameters for LV grids of sectors retail/industrial and agricultural

Parameters **lvgd** (LVGridDistrictDing0) – Low-voltage grid district object

Returns dict – Structural description of (parts of) LV grid topology

ding0.grid.lv_grid.build_grid.**select_grid_model_residential**(*lvgd*)
Selects typified model grid based on population

Parameters **lvgd** (LVGridDistrictDing0) – Low-voltage grid district object

Returns

- pandas.DataFrame – Selected string of typified model grid
- pandas.DataFrame – Parameters of chosen Transformer

Note: In total 196 distinct LV grid topologies are available that are chosen by population in the LV grid district. Population is translated to number of house branches. Each grid model fits a number of house branches. If this number exceeds 196, still the grid topology of 196 house branches is used. The peak load of the LV grid district is uniformly distributed across house branches.

ding0.grid.lv_grid.build_grid.**select_grid_model_ria**(*lvgd, sector*)
Select a typified grid for retail/industrial and agricultural

Parameters

- **lvgd** (`ding0.core.structure.regions.LVGridDistrictDing0`) – Low-voltage grid district object
- **sector** (`str`) – Either 'retail/industrial' or 'agricultural'. Depending on choice different parameters to grid topology apply

Returns `dict` – Parameters that describe branch lines of a sector

`ding0.grid.lv_grid.build_grid.`**`select_transformers`**(*grid*, *s_max=None*)

Selects LV transformer according to peak load of LV grid district.

The transformers are chosen according to max. of load case and feedin-case considering load factors and power factor. The MV-LV transformer with the next higher available nominal apparent power is chosen. Therefore, a max. allowed transformer loading of 100% is implicitly assumed. If the peak load exceeds the max. power of a single available transformer, multiple transformer are build.

By default *peak_load* and *peak_generation* are taken from *grid* instance. The behavior can be overridden providing *s_max* as explained in `Arguments`.

Parameters

- **grid** (*LVGridDing0*) – LV grid data
- **s_max** (*dict*) – dict containing maximum apparent power of load or generation case and str describing the case. For example

```
{
    's_max': 480,
    'case': 'load'
}
```

or

```
{
    's_max': 120,
    'case': 'gen'
}
```

s_max passed overrides *grid.grid_district.peak_load* respectively *grid.station().peak_generation*.

Returns

- pandas.DataFrame – Parameters of chosen Transformer
- `int` – Count of transformers

Note: The LV transformer with the next higher available nominal apparent power is chosen. Therefore, a max. allowed transformer loading of 100% is implicitly assumed. If the peak load exceeds the max. power of a single available transformer, use multiple trafos.

`ding0.grid.lv_grid.build_grid.`**`transformer`**(*grid*)

Choose transformer and add to grid's station

Parameters

- **grid** (*LVGridDing0*) – LV grid data
- **s_max_from_buildings** (*boolean*) – new approach if s_max_from_buildings

8.1.1.4.1.4 ding0.grid.lv_grid.check module

ding0.grid.lv_grid.check.**get_branches**(*grid*)
Individual graphs of sectoral loads

Parameters **geid** –

Returns

ding0.grid.lv_grid.check.**overloading**(*graph*)
Check a grid for line overloading due to current exceeding I_th_max

Parameters **graph** (*networkx.Graph*) – Graph structure as container for a grid topology including its equipment

Returns **overloaded** (*tuple*) – Pairwise edges of graph a maximum occuring current

8.1.1.4.1.5 ding0.grid.lv_grid.lv_connect module

ding0.grid.lv_grid.lv_connect.**lv_connect_generators**(*mv_grid*, *graph*, *debug=False*)
Connect LV generators to LV grid

Parameters

- **lv_grid_district** (LVGridDistrictDing0) – LVGridDistrictDing0 object for which the connection process has to be done
- **graph** (NetworkX Graph Obj) – NetworkX graph object with nodes
- **debug** (*bool, defaults to False*) – If True, information is printed during process

Returns NetworkX Graph Obj – NetworkX graph object with nodes and newly created branches

8.1.1.4.1.6 Module contents

8.1.1.4.1.7 ding0.grid.mv_grid package

8.1.1.4.1.8 Subpackages

8.1.1.4.1.9 ding0.grid.mv_grid.models package

8.1.1.4.1.10 Submodules

8.1.1.4.1.11 ding0.grid.mv_grid.models.models module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

class ding0.grid.mv_grid.models.models.**Graph**(*data*)
Bases: object

Class for modelling a CVRP problem data

Parameters **data** (*type*) – TSPLIB parsed data

depot()
Returns the depot node.

Returns *type* – Depot node

distance(*i*, *j*)
Returns the distance between node i and node j

Parameters

- **i** (*type*) – Descr
- **j** (*type*) – Desc

Returns *float* – Distance between node i and node j.

edges()
Returns a generator for iterating over edges

Yields *type* – Generator for iterating over edges.

nodes()
Returns a generator for iterating over nodes.

Yields *type* – Generator for iterating over nodes.

class ding0.grid.mv_grid.models.models.**Node**(*name*, *demand*)
Bases: object

CVRP node (MV transformer/customer)

Parameters

- **name** – Node name
- **demand** – Node demand

clone()
Returns a deep copy of self

Function clones:

- allocation
- nodes

Returns *type* – Deep copy of self

demand()
Returns the node demand

Returns *float* – Node's demand

name()
Returns node name

Returns *str* – Node's name

route_allocation()
Returns the route which node is allocated

Returns *type* – Node's route

class ding0.grid.mv_grid.models.models.**Route**(*cvrp_problem*)
Bases: object

CVRP route, consists of consecutive nodes

Parameters **cvrp_problem** (*type*) – Descr

allocate (*nodes*, *append=True*)
Allocates all nodes from *nodes* list in this route

Parameters

- **nodes** (*type*) – Desc
- **append** (*bool, defaults to True*) – Desc

calc_circuit_breaker_position (*debug=False*)
Calculates the optimal position of a circuit breaker on route.

Parameters **debug** (*bool, defaults to False*) – If True, prints process information.

Returns *int* – position of circuit breaker on route (index of last node on 1st half-ring preceding the circuit breaker)

Note: According to planning principles of MV grids, a MV ring is run as two strings (half-rings) separated by a circuit breaker which is open at normal operation. Assuming a ring (route which is connected to the root node at either sides), the optimal position of a circuit breaker is defined as the position (virtual cable) between two nodes where the conveyed current is minimal on the route. Instead of the peak current, the peak load is used here (assuming a constant voltage).

The circuit breakers are used here for checking tech. constraints only and will be re-located after connection of satellites and stations in ding0.grid.mv_grid.tools.set_circuit_breakers

References

See also:

`ding0.grid.mv_grid.tools.set_circuit_breakers()`

can_allocate (*nodes*, *pos=None*)
Returns True if this route can allocate nodes in *nodes* list

Parameters

- **nodes** (*type*) – Desc
- **pos** (*type, defaults to None*) – Desc

Returns *bool* – True if this route can allocate nodes in *nodes* list

clone ()
Returns a deep copy of self

Function clones:

- allocation
- nodes

Returns *type* – Deep copy of self

deallocate (*nodes*)
Deallocates all nodes from *nodes* list from this route

Parameters **nodes** (*type*) – Desc

demand()

Returns the current route demand

Returns *type* – Current route demand.

insert(*nodes*, *pos*)

Inserts all nodes from *nodes* list into this route at position *pos*

Parameters

- **nodes** (*type*) – Desc
- **pos** (*type*) – Desc

is_interior(*node*)

Returns True if node is interior to the route, i.e., not adjascent to depot

Parameters **nodes** (*type*) – Desc

Returns *bool* – True if node is interior to the route

last(*node*)

Returns True if node is the last node in the route

Parameters **nodes** (*type*) – Desc

Returns *bool* – True if node is the last node in the route

length()

Returns the route length (cost)

Returns *int* – Route length (cost).

length_from_nodelist(*nodelist*)

Returns the route length (cost) from the first to the last node in nodelist

nodes()

Returns a generator for iterating over nodes

Yields *type* – Generator for iterating over nodes

tech_constraints_satisfied()

Check route validity according to technical constraints (voltage and current rating)

It considers constraints as

- current rating of cable/line
- voltage stability at all nodes

Note: The validation is done for every tested MV grid configuration during CVRP algorithm. The current rating is checked using load factors from[1]. Due to the high amount of steps the voltage rating cannot be checked using load flow calculation. Therefore we use a simple method which determines the voltage change between two consecutive nodes according to[2]. Furthermore it is checked if new route has got more nodes than allowed (typ. 2*10 according to[3]).

[1] Deutsche Energie-Agentur GmbH (dena), "dena-Verteilnetzstudie. Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030.", 2012

[2] M. Sakulin, W. Hipp, "Netzaspekte von dezentralen Erzeugungseinheiten, Studie im Auftrag der E-Control GmbH", TU Graz, 2004

[3] Klaus Heuck et al., "Elektrische Energieversorgung", Vieweg+Teubner, Wiesbaden, 2007

References

8.1.1.4.1.12 Module contents

8.1.1.4.1.13 ding0.grid.mv_grid.solvers package

8.1.1.4.1.14 Submodules

8.1.1.4.1.15 ding0.grid.mv_grid.solvers.base module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

class ding0.grid.mv_grid.solvers.base.**BaseSolution**(*cvrp_problem*)
Bases: `object`

Base abstract class for a CVRP solution

Parameters **cvrp_problem** (`type`) – Desc Graph instance?

can_process(*pairs*)
Returns True if this solution can process *pairs*

Parameters **pairs** (`list` of pairs) – List of pairs

Returns *bool* – True if this solution can process *pairs*

Todo: Not yet implemented

clone()
Returns a deep copy of self

Function clones:

- route
- allocation
- nodes

Returns *type* – Deep copy of self

draw_network(*anim*)
Draws solution's graph using networkx

Parameters **AnimationDing0** – AnimationDing0 object

get_pair(*pair*)
get pair description

Parameters **pair** (`list` of nodes) – Descr

Returns *type* – Descr

is_complete()
Returns True if this is a complete solution, i.e, all nodes are allocated

Returns *bool* – True if all nodes are llocated.

length()
Returns the solution length (or cost)

Returns *float* – Solution length (or cost).

process(*node_or_pair*)
Processes a node or a pair of nodes into the current solution

MUST CREATE A NEW INSTANCE, NOT CHANGE ANY INSTANCE ATTRIBUTES

Parameters **node_or_pair** (*type*) – Desc

Returns *type* – A new instance (deep copy) of self object

Todo: Not yet implemented

routes()
Returns a generator for iterating over solution routes

Yields *type* – Generator for iterating over solution routes.

class ding0.grid.mv_grid.solvers.base.**BaseSolver**
Bases: object

Base algorithm solver class

solve(*data*, *vehicles*, *timeout*)
Must solves the CVRP problem

Must return BEFORE timeout

Must returns a solution (BaseSolution class derived)

Parameters

- **data** (*type*) – Graph instance
- **vehicles** (int) – Vehicles number
- **timeout** (int) – max processing time in seconds

Todo: Not yet implemented

8.1.1.4.1.16 ding0.grid.mv_grid.solvers.local_search module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

class ding0.grid.mv_grid.solvers.local_search.**LocalSearchSolution**(*cvrp_problem*, *solution*)
Bases: *ding0.grid.mv_grid.solvers.base.BaseSolution*

Solution class for Local Search metaheuristic

Parameters

- **cvrp_problem** (*type*) – Descr
- **solution** (BaseSolution) – Descr

clone()
Returns a deep copy of self

Function clones:

- route
- allocation
- nodes

Returns *LocalSearchSolution* – Deep copy of self

class ding0.grid.mv_grid.solvers.local_search.**LocalSearchSolver**
Bases: *ding0.grid.mv_grid.solvers.base.BaseSolver*

Improve initial savings solution using local search

The implementation of the local searach algorithm founds on the following publications[1],[2],[3],[4]

Graph operators:

```
Or-Opt (intra-route)
Relocate (inter-route)
Exchange (inter-route)
```

Todo:

- Cross (inter-route) - to remove crossing edges between two routes

References

benchmark_operator_order(*graph*, *solution*, *op_diff_round_digits*)
performs all possible permutations of route improvement and prints graph length

Parameters

- **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.
- **solution** (BaseSolution) – BaseSolution instance
- **op_diff_round_digits** (*float*) – Precision (floating point digits) for rounding route length differences.

  Details: In some cases when an exchange is performed on two routes with one node each, the difference between the both solutions (before and after the exchange) is not zero. This is due to internal rounding errors of float type. So the loop won't break (alternating between these two solutions), we need an additional criterion to avoid this behaviour: A threshold to handle values very close to zero as if they were zero (for a more detailed

[1] W. Wenger, "Multikriterielle Tourenplanung", Dissertation, 2009

[2] M. Kämpf, "Probleme der Tourenbildung", Chemnitzer Informatik-Berichte, 2006

[3] O. Bräysy, M. Gendreau, "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms", Transportation Science, vol. 39, Issue 1, pp. 104-118, 2005

[4] C. Boomgaarden, "Dynamische Tourenplanung und -steuerung", Dissertation, 2007

description of the matter see http://floating-point-gui.de or https://docs.python.org/3.5/tutorial/floatingpoint.html)

operator_cross (*graph*, *solution*, *op_diff_round_digits*)
applies Cross inter-route operator to solution

Takes every node from every route and calculates savings when inserted into all possible positions in other routes. Insertion is done at position with max. saving and procedure starts over again with newly created graph as input. Stops when no improvement is found.

Parameters

- **graph** (NetworkX Graph Obj) – Descr
- **solution** (`BaseSolution`) – Descr
- **op_diff_round_digits** (*float*) – Precision (floating point digits) for rounding route length differences.

  Details: In some cases when an exchange is performed on two routes with one node each, the difference between the both solutions (before and after the exchange) is not zero. This is due to internal rounding errors of float type. So the loop won't break (alternating between these two solutions), we need an additional criterion to avoid this behaviour: A threshold to handle values very close to zero as if they were zero (for a more detaisled description of the matter see http://floating-point-gui.de or https://docs.python.org/3.5/tutorial/floatingpoint.html)

Returns *LocalSearchSolution* – A solution (LocalSearchSolution class)

Todo:

- allow moves of a 2-node chain
- Remove ugly nested loops, convert to more efficient matrix operations

operator_exchange (*graph*, *solution*, *op_diff_round_digits*, *anim*)
applies Exchange inter-route operator to solution

Takes every node from every route and calculates savings when exchanged with another one of all possible nodes in other routes. Insertion is done at position with max. saving and procedure starts over again with newly created graph as input. Stops when no improvement is found.

Parameters

- **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.
- **solution** (`BaseSolution`) – BaseSolution instance
- **op_diff_round_digits** (*float*) – Precision (floating point digits) for rounding route length differences.

  Details: In some cases when an exchange is performed on two routes with one node each, the difference between the both solutions (before and after the exchange) is not zero. This is due to internal rounding errors of float type. So the loop won't break (alternating between these two solutions), we need an additional criterion to avoid this behaviour: A threshold to handle values very close to zero as if they were zero (for a more detailed description of the matter see http://floating-point-gui.de or https://docs.python.org/3.5/tutorial/floatingpoint.html)
- **anim** (`AnimationDing0`) – AnimationDing0 object

Returns *LocalSearchSolution* – A solution (LocalSearchSolution class)

Note: (Inner) Loop variables:

- i: node that is checked for possible moves (position in the route *tour*, not node name)
- j: node that precedes the insert position in target route (position in the route *target_tour*, not node name)

Todo:

- allow moves of a 2-node chain
- Remove ugly nested loops, convert to more efficient matrix operations

operator_oropt(*graph*, *solution*, *op_diff_round_digits*, *anim=None*)

Applies Or-Opt intra-route operator to solution

Takes chains of nodes (length=3..1 consecutive nodes) from a given route and calculates savings when inserted into another position on the same route (all possible positions). Performes best move (max. saving) and starts over again with new route until no improvement is found.

Parameters

- **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.
- **solution** (`BaseSolution`) – BaseSolution instance
- **op_diff_round_digits** (*`float`*) – Precision (floating point digits) for rounding route length differences.

  Details: In some cases when an exchange is performed on two routes with one node each, the difference between the both solutions (before and after the exchange) is not zero. This is due to internal rounding errors of float type. So the loop won't break (alternating between these two solutions), we need an additional criterion to avoid this behaviour: A threshold to handle values very close to zero as if they were zero (for a more detailed description of the matter see http://floating-point-gui.de or https://docs.python.org/3.5/tutorial/floatingpoint.html)
- **anim** (`AnimationDing0`) – AnimationDing0 object

Returns *LocalSearchSolution* – A solution (LocalSearchSolution class)

Note: Since Or-Opt is an intra-route operator, it has not to be checked if route can allocate (Route's method can_allocate()) nodes during relocation regarding max. peak load/current because the line/cable type is the same along the entire route. However, node order within a route has an impact on the voltage stability so the check would be actually required. Due to large line capacity (load factor of lines/cables ~60 %) the voltage stability issues are neglected.

(Inner) Loop variables:

- s: length (count of consecutive nodes) of the chain that is moved. Values: 3..1
- i: node that precedes the chain before moving (position in the route *tour*, not node name)
- j: node that precedes the chain after moving (position in the route *tour*, not node name)

Todo:

- insert literature reference for Or-algorithm here
- Remove ugly nested loops, convert to more efficient matrix operations

operator_relocate (*graph*, *solution*, *op_diff_round_digits*, *anim*)
applies Relocate inter-route operator to solution

Takes every node from every route and calculates savings when inserted into all possible positions in other routes. Insertion is done at position with max. saving and procedure starts over again with newly created graph as input. Stops when no improvement is found.

Parameters

- **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.
- **solution** (BaseSolution) – BaseSolution instance
- **op_diff_round_digits** (*float*) – Precision (floating point digits) for rounding route length differences.

  Details: In some cases when an exchange is performed on two routes with one node each, the difference between the both solutions (before and after the exchange) is not zero. This is due to internal rounding errors of float type. So the loop won't break (alternating between these two solutions), we need an additional criterion to avoid this behaviour: A threshold to handle values very close to zero as if they were zero (for a more detailed description of the matter see http://floating-point-gui.de or https://docs.python.org/3.5/tutorial/floatingpoint.html)
- **anim** (AnimationDing0) – AnimationDing0 object

Returns *LocalSearchSolution* – A solution (LocalSearchSolution class)

Note: (Inner) Loop variables:

- i: node that is checked for possible moves (position in the route *tour*, not node name)
- j: node that precedes the insert position in target route (position in the route *target_tour*, not node name)

Todo:

- Remove ugly nested loops, convert to more efficient matrix operations

solve (*graph*, *savings_solution*, *timeout*, *debug=False*, *anim=None*)
Improve initial savings solution using local search

Parameters

- **graph** (NetworkX Graph Obj) – Graph instance
- **savings_solution** (SavingsSolution) – initial solution of CVRP problem (instance of *SavingsSolution* class)
- **timeout** (int) – max processing time in seconds
- **debug** (*bool, defaults to False*) – If True, information is printed while routing
- **anim** (AnimationDing0) – AnimationDing0 object

Returns *LocalSearchSolution* – A solution (LocalSearchSolution class)

8.1.1.4.1.17 ding0.grid.mv_grid.solvers.savings module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

class ding0.grid.mv_grid.solvers.savings.**ClarkeWrightSolver**

Bases: *ding0.grid.mv_grid.solvers.base.BaseSolver*

Clark and Wright Savings algorithm solver class

compute_savings_list(*graph*)

Compute Clarke and Wright savings list

A saving list is a matrix containing the saving amount S between i and j

S is calculated by S = d(0,i) + d(0,j) - d(i,j) (CLARKE; WRIGHT, 1964)

Parameters **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.

Returns `list` of *Node* – List of nodes sorted by its savings

solve(*graph*, *timeout*, *debug=False*, *anim=None*)

Solves the CVRP problem using Clarke and Wright Savings methods

Parameters

- **graph** (NetworkX Graph Obj) – A NetworkX graaph is used.
- **timeout** (`int`) – max processing time in seconds
- **debug** (*bool, defaults to False*) – If True, information is printed while routing
- **anim** (`AnimationDing0`) –

Returns *SavingsSolution* – A solution

class ding0.grid.mv_grid.solvers.savings.**SavingsSolution**(*cvrp_problem*)

Bases: *ding0.grid.mv_grid.solvers.base.BaseSolution*

Solution class for a Clarke and Wright Savings parallel algorithm

can_process(*pairs*)

Returns True if this solution can process *pairs*

Parameters **pairs** (`list` of pairs of Route) – List of pairs

Returns *bool* – True if this solution can process *pairs*.

clone()

Returns a deep copy of self

Function clones:

- routes
- allocation
- nodes

Returns *SavingsSolution* – A clone (deepcopy) of the instance itself

is_complete()

Returns True if this is a complete solution, i.e, all nodes are allocated

Todo: TO BE REVIEWED

Returns *bool* – True if this is a complete solution.

process (*pair*)
Processes a pair of nodes into the current solution

MUST CREATE A NEW INSTANCE, NOT CHANGE ANY INSTANCE ATTRIBUTES

Returns a new instance (deep copy) of self object

Parameters **pair** (*type*) – description

Returns *type* – Description (Copy of self?)

8.1.1.4.1.18 Module contents

8.1.1.4.1.19 ding0.grid.mv_grid.tests package

8.1.1.4.1.20 Submodules

8.1.1.4.1.21 ding0.grid.mv_grid.tests.run_test_case module

ding0.grid.mv_grid.tests.run_test_case.**main**()
Description of Test Case

8.1.1.4.1.22 Module contents

8.1.1.4.1.23 ding0.grid.mv_grid.util package

8.1.1.4.1.24 Submodules

8.1.1.4.1.25 ding0.grid.mv_grid.util.data_input module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

exception ding0.grid.mv_grid.util.data_input.**ParseException**(*value*)
Bases: Exception

Exception raised when something unexpected occurs in a TSPLIB file parsing

value
Description

Type type

Parameters **value** (*type*) – Description

ding0.grid.mv_grid.util.data_input.**calculate_euc_distance**(*a*, *b*)
Calculates Eclidian distances from two points a and b

Parameters

- **a** ((float, float)) – Two-dimension tuple (x1,y1)
- **b** ((float, float)) – Two-dimension tuple (x2,y2)

Returns *float* – the distance.

`ding0.grid.mv_grid.util.data_input.`**`read_file`**(*filename*)

Reads a TSPLIB file and returns the problem data.

Parameters **filename** (str) –

Returns *type* – Problem specs.

`ding0.grid.mv_grid.util.data_input.`**`sanitize`**(*filename*)

Returns a sanitized file name with absolut path

Example

~/input.txt -> /home/<your_home/input.txt

Returns *str* – The sanitized file name with absolut path.

`ding0.grid.mv_grid.util.data_input.`**`strip`**(*line*)

Removes any *\r* or *\n* from line and remove trailing whitespaces

Parameters **line** (str) –

Returns *str* – the stripped line.

8.1.1.4.1.26 ding0.grid.mv_grid.util.util module

Based on code by Romulo Oliveira copyright (C) 2015, https://github.com/RomuloOliveira/monte-carlo-cvrp Originally licensed under the Apache License, Version 2.0. You may obtain a copy of the license at http://www.apache.org/licenses/LICENSE-2.0

`ding0.grid.mv_grid.util.util.`**`print_solution`**(*solution*)

Prints a solution

Parameters **solution** (BaseSolution) –

Example

```
[8, 9, 10, 7]: 160
[5, 6]: 131
[3, 4, 2]: 154
Total cost: 445
```

`ding0.grid.mv_grid.util.util.`**`print_upper_triangular_matrix`**(*matrix*)

Prints a CVRP data dict matrix

Parameters **matrix** (*dict*) – Description

Note: It is assummed that the first row of matrix contains all needed headers.

ding0.grid.mv_grid.util.util.**print_upper_triangular_matrix_as_complete**(*matrix*)
Prints a CVRP data dict upper triangular matrix as a normal matrix

Doesn't print headers.

Parameters **matrix** (*dict*) – Description

8.1.1.4.1.27 Module contents

8.1.1.4.1.28 Submodules

8.1.1.4.1.29 ding0.grid.mv_grid.mv_connect module

8.1.1.4.1.30 ding0.grid.mv_grid.mv_routing module

ding0.grid.mv_grid.mv_routing.**ding0_graph_to_routing_specs**(*graph*)
Build data dictionary from graph nodes for routing (translation)

Parameters **graph** (NetworkX Graph Obj) – NetworkX graph object with nodes

Returns dict – Data dictionary for routing.

See also:

ding0.grid.mv_grid.models.models.Graph() for keys of return dict

ding0.grid.mv_grid.mv_routing.**routing_solution_to_ding0_graph**(*graph*, *solution*)
Insert *solution* from routing into *graph*

Parameters

- **graph** (NetworkX Graph Obj) – NetworkX graph object with nodes
- **solution** (BaseSolution) – Instance of *BaseSolution* or child class (e.g. *LocalSearchSolution*) (=solution from routing)

Returns NetworkX Graph Obj – NetworkX graph object with nodes and edges

ding0.grid.mv_grid.mv_routing.**solve**(*graph*, *debug=False*, *anim=None*)
Do MV routing for given nodes in *graph*.

Translate data from node objects to appropriate format before.

Parameters

- **graph** (NetworkX Graph Obj) – NetworkX graph object with nodes
- **debug** (*bool, defaults to False*) – If True, information is printed while routing
- **anim** (AnimationDing0) – AnimationDing0 object

Returns NetworkX Graph Obj – NetworkX graph object with nodes and edges

See also:

ding0.tools.animation.AnimationDing0() for a more detailed description on anim parameter.

8.1.1.4.1.31 ding0.grid.mv_grid.tools module

8.1.1.4.1.32 Module contents

8.1.1.4.2 Submodules

8.1.1.4.3 ding0.grid.tools module

`ding0.grid.tools.`**`cable_type`**`(`*nom_power*, *nom_voltage*, *avail_cables*`)`

Determine suitable type of cable for given nominal power

Based on maximum occurring current which is derived from nominal power (either peak load or max. generation capacity) a suitable cable type is chosen. Thus, no line overloading issues should occur.

Parameters

- **nom_power** (*float*) – Nominal power of generators or loads connected via a cable
- **nom_voltage** (*float*) – Nominal voltage in kV
- **avail_cables** (pandas.DataFrame) – Available cable types including it's electrical parameters

Returns pandas.DataFrame – Parameters of cable type

8.1.1.4.4 Module contents

8.1.1.5 ding0.tools package

8.1.1.5.1 Submodules

8.1.1.5.2 ding0.tools.animation module

class `ding0.tools.animation.`**`AnimationDing0`**`(`***kwargs*`)`

Bases: `object`

Class for visual animation of the routing process (solving CVRP).

(basically a central place to store information about output file and count of saved images). Use argument 'animation=True' of method 'NetworkDing0.mv_routing()' to enable image export. The images are exported to ding0's home dir which is usually ~/.ding0/ .

Subsequently, FFMPEG can be used to convert images to animation, e.g.

> ffmpeg -r 5 -i mv-routing_ani_%04d.png -vframes 200 -r 15 -vcodec libx264 -y -an mv-routing_ani.mp4 -s 640x480

See also:

`ding0.core.NetworkDing0.mv_routing`

8.1.1.5.3 ding0.tools.config module

Based on code by oemof development team

This module provides a highlevel layer for reading and writing config files. The config file has to be of the following structure to be imported correctly.

```
[netCDF]

     RootFolder = c://netCDF

     FilePrefix = cd2_


[mySQL]

    host = localhost

    user = guest

    password = root

    database = znes


[SectionName]

    OptionName = value

    Option2 = value2
```

Based on code by oemof development team

`ding0.tools.config.`**`get`**(*section*, *key*)
Returns the value of a given key of a given section of the main config file.

Parameters

- **section** (str) – the section.
- **key** (str) – the key

Returns float – the value which will be casted to float, int or boolean. if no cast is successful, the raw string will be returned.

See also:

set()

`ding0.tools.config.`**`load_config`**(*filename*)
Read config file specified by *filename*

Parameters **filename** (str) – Description of filename

`ding0.tools.config.`**`set`**(*section*, *key*, *value*)
Sets a value to a [section] key - pair.

if the section doesn't exist yet, it will be created.

Parameters

- **section** (str) – the section.
- **key** (str) – the key.
- **value** (*float*, *int*, *str*) – the value.

See also:

get()

8.1.1.5.4 ding0.tools.debug module

ding0.tools.debug.**compare_graphs**(*graph1*, *mode*, *graph2=None*)

Compares graph with saved one which is loaded via networkx' gpickle

Parameters

- **graph1** (*networkx.graph*) – First Ding0 MV graph for comparison
- **graph2** (*networkx.graph*) – Second Ding0 MV graph for comparison. If a second graph is not provided it will be laoded from disk with hard-coded file name.
- **mode** (*'write' or 'compare'*) –
- **Returns** –

ding0.tools.debug.**log_errors**(*f*)

Decorator object that logs every exception into the defined logger object.

8.1.1.5.5 ding0.tools.geo module

ding0.tools.geo.**calc_edge_geometry**(*node_source*, *node_target*, *srid=3035*)

returns straight edge geometry and related length between two nodes as LineString

Parameters

- **node_source** (*LVStationDing0, GeneratorDing0, or CableDistributorDing0*) – source node, member of GridDing0.graph
- **node_target** (*LVStationDing0, GeneratorDing0, or CableDistributorDing0*) – target node, member of GridDing0.graph
- **srid** (*defines crs. 4326 by default check config_misc.cfg*) –

Returns

- **geometry** (*LineString,*)
- float – Distance in m

ding0.tools.geo.**calc_geo_branches_in_buffer**(*node*, *mv_grid*, *radius*, *radius_inc*, *proj*, *srid=3035*)

NEW PARAM srid=3035 to calculate calc_geo_branches_in_buffer. Before srid=4326 was assumed.

Determines branches in nodes' associated graph that are at least partly within buffer of *radius* from *node*.

If there are no nodes, the buffer is successively extended by *radius_inc* until nodes are found.

Parameters

- **node** (*LVStationDing0, GeneratorDing0, or CableDistributorDing0*) – origin node (e.g. LVStationDing0 object) with associated shapely object (attribute *geo_data*) in any CRS (e.g. WGS84)
- **radius** (*float*) – buffer radius in m
- **radius_inc** (*float*) – radius increment in m

- **proj** (int) – pyproj projection object: nodes' CRS to equidistant CRS (e.g. WGS84 -> ETRS)

Returns list of NetworkX Graph Obj – List of branches (NetworkX branch objects)

ding0.tools.geo.**calc_geo_branches_in_polygon**(*mv_grid*, *polygon*, *mode*, *proj*, *srid=3035*)
NEW PARAM srid=3035 to calculate calc_geo_branches_in_buffer. Before srid=4326 was assumed.

Calculate geographical branches in polygon.

For a given *mv_grid* all branches (edges in the graph of the grid) are tested if they are in the given *polygon*. You can choose different modes and projections for this operation.

Parameters

- **mv_grid** (MVGridDing0) – MV Grid object. Edges contained in *mv_grid.graph_edges()* are taken for the test.
- **polygon** (Shapely Point object) – Polygon that contains edges.
- **mode** (str) – Choose between 'intersects' or 'contains'.
- **proj** (int) – EPSG code to specify projection

Returns list of *BranchDing0* objects – List of branches

ding0.tools.geo.**calc_geo_centre_point**(*node_source*, *node_target*, *srid=3035*)
Calculates the geodesic distance between *node_source* and *node_target* incorporating the detour factor specified in config_calc.cfg.

Parameters

- **node_source** (LVStationDing0, GeneratorDing0, *or* CableDistributorDing0) – source node, member of GridDing0.graph
- **node_target** (LVStationDing0, GeneratorDing0, *or* CableDistributorDing0) – target node, member of GridDing0.graph

Returns float – Distance in m.

ding0.tools.geo.**calc_geo_dist**(*node_source*, *node_target*, *srid=3035*)
Calculates the geodesic distance between *node_source* and *node_target* incorporating the detour factor specified in ding0/ding0/config/config_calc.cfg.

Parameters

- **node_source** (LVStationDing0, GeneratorDing0, *or* CableDistributorDing0) – source node, member of GridDing0.graph
- **node_target** (LVStationDing0, GeneratorDing0, *or* CableDistributorDing0) – target node, member of GridDing0.graph
- **srid** (*defines crs. 4326 by default check config_misc.cfg*) –

Returns float – Distance in m

ding0.tools.geo.**calc_geo_dist_matrix**(*nodes_pos*, *srid=3035*)
Calculates the geodesic distance between all nodes in *nodes_pos* incorporating the detour factor in config_calc.cfg.

For every two points/coord it uses geopy's geodesic function. As default ellipsoidal model of the earth WGS-84 is used. For more options see

https://geopy.readthedocs.io/en/stable/index.html?highlight=geodesic#geopy.distance.geodesic

Parameters **nodes_pos** (*dict*) – dictionary of nodes with positions, with x=longitude, y=latitude, and the following format:

```
{
    'node_1': (x_1, y_1),
    ...,
    'node_n': (x_n, y_n)
}
```

Returns

`dict` –

dictionary with distances between all nodes (in km), with the following format:

```
{
    'node_1': {'node_1': dist_11, ..., 'node_n': dist_1n},
    ...,
    'node_n': {'node_1': dist_n1, ..., 'node_n': dist_nn}
}
```

8.1.1.5.6 ding0.tools.logger module

`ding0.tools.logger.`**`create_dir`**(*dirpath*)
Create directory and report about it

Parameters **dirpath** (`str`) – Directory including path

`ding0.tools.logger.`**`create_home_dir`**(*ding0_path=None*)
Check if ~/.ding0 exists, otherwise create it

Parameters **ding0_path** (`str`) – Path to store Ding0 related data (logging, etc)

`ding0.tools.logger.`**`get_default_home_dir`**()
Return default home directory of Ding0

Returns `str` – Default home directory including its path

`ding0.tools.logger.`**`setup_logger`**(*log_dir=None, filename='ding0.log', loglevel=10*)
Instantiate logger

Parameters

- **log_dir** (`str`) – Directory to save log, default: ~/.ding0/logging/
- **filename** (`str`) – Name of log file, default: ding0.log
- **loglevel** – Level of logger.

8.1.1.5.7 ding0.tools.plots module

8.1.1.5.8 ding0.tools.pypsa_io module

`ding0.tools.pypsa_io.`**`append_bus_v_mag_set_df`**(*bus_v_mag_set_df, node, node_name=None*)
Fills bus v_mag_set data needed for power flow calculation

Parameters

- **bus_v_mag_set_df** (pandas.DataFrame) – Dataframe of buses with entries name, temp_id, v_mag_pu_set
- **node** (*obj:node object of generator*) –
- **node_name** (str) – Optional parameter for name of bus

Returns **bus_v_mag_set_df** (pandas.DataFrame) – Dataframe of buses with entries name, temp_id, v_mag_pu_set

ding0.tools.pypsa_io.**append_buses_df**(*buses_df*, *grid*, *node*, *node_name=''*)

Appends buses to dataframe of buses in pypsa format.

Parameters

- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building
- **grid** (*GridDing0*) –
- **node** –
- **node_name** (str) – name of node, per default is set to node.pypsa_bus_id

Returns **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building

ding0.tools.pypsa_io.**append_generator_pq_set_df**(*conf*, *generator_pq_set_df*, *node*)

Fills generator pq_set data needed for power flow calculation

Parameters

- **conf** (dict) – dictionary with technical constants
- **generator_pq_set_df** (pandas.DataFrame) – Dataframe of generators with entries name, temp_id, p_set and q_set
- **node** (*obj:node object of generator*) –

Returns **generator_pq_set_df** (pandas.DataFrame) – Dataframe of generators with entries name, temp_id, p_set and q_set

ding0.tools.pypsa_io.**append_generators_df**(*generators_df*, *node*, *name_bus=None*)

Appends generator to dataframe of generators in pypsa format.

Parameters

- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **node** – GeneratorDing0
- **name_bus** (str) – Optional parameter for name of bus

Returns **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype

ding0.tools.pypsa_io.**append_lines_df**(*edge*, *lines_df*, *buses_df*)

Append edge to lines_df

Parameters

- **edge** – Edge of Ding0.Network graph
- **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type, geometry

- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building

Returns **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type

ding0.tools.pypsa_io.**append_load_area_to_load_df**(*sector*, *load_area*, *loads_df*, *name_bus*, *name_load*, *return_time_varying_data=False*, ***kwargs*)

Appends LVLoadArea or LVGridDistrict to dataframe of loads in pypsa format.

Parameters

- **sector** (*str*) – load sector: 'agricultural', 'industrial', 'residential' or 'retail'
- **load_area** – LVGridDistrictDing0 or LVLoadAreaDing0, load area of which load is to be aggregated and added
- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption and sector
- **name_bus** (str) – name of bus to which load is connected
- **name_load** (str) – name of load
- **return_time_varying_data** (bool) – Determines whether data for power flow calculation is exported as well
- **kwargs** (*list of conf, load_pq_set_df*) – Both arguments have to be inserted if return_time_varying_data is True.

Returns

- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption and sector
- **load_pq_set_df** (pandas.DataFrame) – Dataframe of loads with entries name, temp_id, p_set and q_set, only exported if return_time_varying_data is True

ding0.tools.pypsa_io.**append_load_areas_to_df**(*loads_df*, *generators_df*, *node*, *return_time_varying_data=False*, ***kwargs*)

Appends lv load area (or single lv grid district) to dataframe of loads and generators. Also returns power flow time varying data if return_time_varying_data is True. Each sector (agricultural, industrial, residential, retail) is represented by own entry of load. Each generator in underlying grid districts is added as own entry. Generators and load are connected to BusBar of the respective grid (LVGridDing0 for LVStationDing0 and MVGridDing0 for LVLoadAreaCentreDing0)

Parameters

- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption, sector
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **node** – Node, which is either LVStationDing0 or LVLoadAreaCentreDing0
- **return_time_varying_data** (bool) – Determines whether data for power flow calculation is exported as well
- **kwargs** (*list of conf, load_pq_set_df, generator_pq_set_df*) – All three arguments have to be inserted if return_time_varying_data is True.

Returns

- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption, sector
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **load_pq_set_df** (pandas.DataFrame) – Dataframe of loads with entries name, temp_id, p_set and q_set, only exported if return_time_varying_data is True
- **generator_pq_set_df** (pandas.DataFrame) – Dataframe of generators with entries name, temp_id, p_set and q_set, only exported if return_time_varying_data is True

`ding0.tools.pypsa_io.`**`append_load_pq_set_df`**(*conf*, *load_pq_set_df*, *node*, *node_name=None*, *peak_load=None*)

Fills load pq_set data needed for power flow calculation

Parameters

- **`conf`** (`dict`) – dictionary with technical constants
- **`load_pq_set_df`** (pandas.DataFrame) – Dataframe of loads with entries name, temp_id, p_set and q_set
- **`node`** (*`obj:node object of generator`*) –
- **`node_name`** (`str`) – Optional parameter for name of load
- **`p_set`** (`float`) – Optional parameter for peak_load

Returns **load_pq_set_df** (pandas.DataFrame) – Dataframe of loads with entries name, temp_id, p_set and q_set

`ding0.tools.pypsa_io.`**`append_transformers_df`**(*transformers_df*, *trafo*, *type=nan*, *bus0=None*, *bus1=None*)

Appends transformer to dataframe of buses in pypsa format.

Parameters

- **`transformers_df`** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type
- **`trafo`** (*`:obj:TransformerDing0`*) – Transformer to be added
- **`type`** (`str`) – Optional parameter for type of transformer
- **`bus0`** (`str`) – Name of primary side bus. Defaults to None and is set to primary side of transformer station by default.
- **`bus1`** (`str`) – Name of secondary side bus. Defaults to None and is set to secondary side of transformer station by default.

Returns **transformers_df** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type

`ding0.tools.pypsa_io.`**`assign_bus_results`**(*grid*, *bus_data*)

Write results obtained from PF to graph

Parameters

- **`grid`** (*`GridDing0`*) –
- **`bus_data`** (pandas.DataFrame) – DataFrame containing voltage levels obtained from PF analysis

`ding0.tools.pypsa_io.`**`assign_line_results`**(*grid*, *line_data*)

Write results obtained from PF to graph

Parameters

- **grid** (*GridDing0*) –
- **line_data** (pandas.DataFrame) – DataFrame containing active/reactive at nodes obtained from PF analysis

`ding0.tools.pypsa_io.`**`circuit_breakers_to_df`**(*grid*, *components*, *component_data*, *open_circuit_breakers*, *return_time_varying_data=False*)

Appends circuit breakers to component dicts. If circuit breakers are open a virtual bus is added to the respective dataframe and bus1 of the line attached to the circuit breaker is set to the new virtual node.

Parameters

- **grid** (*GridDing0*) –
- **components** (components: dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Line', 'Load', 'Transformer'
- **component_data** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Load', needed for power flow calculations
- **open_circuit_breakers** (dict) – Dictionary containing names of open circuit breakers
- **return_time_varying_data** (bool) – States whether time varying data needed for power flow calculations are constructed as well. Set to True to run power flow, set to False to export network to csv.

Returns

- **components** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Line', 'Load', 'Transformer', 'Switch'
- **component_data** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Load', needed for power flow calculations

`ding0.tools.pypsa_io.`**`create_powerflow_problem`**(*timerange*, *components*)

Create PyPSA network object and fill with data :param timerange: Time range to be analyzed by PF :type timerange: Pandas DatetimeIndex :param components: :type components: dict

Returns **network** (*PyPSA powerflow problem object*)

`ding0.tools.pypsa_io.`**`data_integrity`**(*components*, *components_data*)

Check grid data for integrity

Parameters

- **components** (*dict*) – Grid components
- **components_data** (*dict*) – Grid component data (such as p,q and v set points)

`ding0.tools.pypsa_io.`**`edges_to_dict_of_dataframes`**(*edges*, *lines_df*, *buses_df*)

Export edges to DataFrame

Parameters

- **edges** (list) – Edges of Ding0.Network graph
- **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type
- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building

Returns **edges_dict** (*dict*)

`ding0.tools.pypsa_io.`**`export_to_dir`**(*network*, *export_dir*)

Exports PyPSA network as CSV files to directory

Parameters

- **network** (*:pypsa:pypsa.Network*) –
- **export_dir** (str) – Sub-directory in output/debug/grid/ where csv Files of PyPSA network are exported to.

`ding0.tools.pypsa_io.`**`fill_component_dataframes`**(*grid*, *buses_df*, *lines_df*, *transformer_df*, *generators_df*, *loads_df*, *only_export_mv=False*, *return_time_varying_data=False*)

Returns component and if necessary time varying data for power flow or csv export of inserted mv or lv grid

Parameters

- **grid** (*GridDing0*) – Grid that is exported
- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building
- **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type_info
- **transformer_df** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption, sector
- **only_export_mv** (bool) –
- **return_time_varying_data** (bool) – States whether time varying data needed for power flow calculations are constructed as well. Set to True to run power flow, set to False to export network to csv.

Returns

- **components** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Line', 'Load', 'Transformer', 'Switch'
- **component_data** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Load', needed for power flow calculations

`ding0.tools.pypsa_io.`**`fill_mvgd_component_dataframes`**(*mv_grid_district*, *buses_df*, *generators_df*, *lines_df*, *loads_df*, *transformer_df*, *only_export_mv=False*, *return_time_varying_data=False*)

Returns component and if necessary time varying data for power flow or csv export of inserted mv grid district

Parameters

- **mv_grid_district** (*MVGridDistrictDing0*) –
- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building

- **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type
- **transformer_df** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, building_id, annual_consumption, sector
- **only_export_mv** (`bool`) – Bool that determines export modes for grid district, if True only mv grids are exported with lv grids aggregated at respective station, if False lv grids are fully exported
- **return_time_varying_data** (`bool`) – States whether time varying data needed for power flow calculations are constructed as well. Set to True to run power flow, set to False to export network to csv.

Returns

- **mv_components** (`dict`) – Dictionary of component Dataframes 'Bus', 'Generator', 'Line', 'Load', 'Transformer', 'Switch'
- **network_df** (pandas.DataFrame) – Dataframe of network containing name, srid, geom and population
- **mv_component_data** (`dict`) – Dictionary of component Dataframes 'Bus', 'Generator', 'Load', needed for power flow calculations

`ding0.tools.pypsa_io.`**`init_pypsa_network`**(*time_range_lim*)

Instantiate PyPSA network :param time_range_lim:

Returns

- **network** (*PyPSA network object*) – Contains powerflow problem
- **snapshots** (*iterable*) – Contains snapshots to be analyzed by powerplow calculation

`ding0.tools.pypsa_io.`**`initialize_component_dataframes`**()

Initializes and returns empty component dataframes

Returns

- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building
- **lines_df** (pandas.DataFrame) – Dataframe of lines with entries name, bus0, bus1, length, x, r, s_nom, num_parallel, type, geometry (LineString)
- **transformer_df** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name, bus, p_set, annual_consumption, sector

`ding0.tools.pypsa_io.`**`nodes_to_dict_of_dataframes`**(*grid, nodes, buses_df, generators_df, loads_df, transformer_df, only_export_mv=False, return_time_varying_data=False*)

Creates dictionary of dataframes containing grid nodes and transformers

Parameters

- **grid** (*GridDing0*) –
- **nodes** (list of ding0 grid components objects) – Nodes of the grid graph
- **buses_df** (pandas.DataFrame) – Dataframe of buses with entries name, v_nom, geom, mv_grid_id, lv_grid_id, in_building
- **generators_df** (pandas.DataFrame) – Dataframe of generators with entries name, bus, control, p_nom, type, weather_cell_id, subtype
- **loads_df** (pandas.DataFrame) – Dataframe of loads with entries name,bus,p_set, building_id, annual_consumption,sector
- **transformer_df** (pandas.DataFrame) – Dataframe of trafos with entries name, bus0, bus1, x, r, s_nom, type
- **only_export_mv** (bool) – Bool that indicates whether only mv grid should be exported, per default lv grids are exported too
- **return_time_varying_data** (bool) – Set to True when running power flow. Then time varying data are returned as well.

Returns

- **components** (dict of pandas.DataFrame) – DataFrames contain components attributes. Dict is keyed by components type
- **component_data** (dict) – Dictionary of component Dataframes 'Bus', 'Generator', 'Load', needed for power flow calculations, only exported when return_time_varying_data is True empty dict otherwise.

`ding0.tools.pypsa_io.`**`process_pf_results`**`(`*network*`)`

Parameters **network** (*pypsa.Network*) –

Returns

- **bus_data** (pandas.DataFrame) – Voltage level results at buses
- **line_data** (pandas.DataFrame) – Resulting apparent power at lines

`ding0.tools.pypsa_io.`**`run_powerflow_onthefly`**`(`*components, components_data, grid, export_pypsa_dir=None, debug=False, export_result_dir=None*`)`

Run powerflow to test grid stability

Two cases are defined to be tested here:

i) load case
ii) feed-in case

Parameters

- **components** (dict of pandas.DataFrame) –
- **components_data** (dict of pandas.DataFrame) –
- **grid** (*GridDing0*) –
- **export_pypsa_dir** (str) – Sub-directory in output/debug/grid/ where csv Files of PyPSA network are exported to. Export is omitted if argument is empty.
- **debug** (bool) –

- **export_result_dir** (str) – Directory where csv Files of power flow results are exported to. Export is omitted if argument is empty.

ding0.tools.pypsa_io.**select_and_append_load_area_trafos**(*aggregated_load_area*, *node_name*, *transformer_df*)

Selects the right trafos for aggregrated load areas and appends them to the transformer dataframe.

Parameters

- **aggregated_load_area** (LVLoadAreaDing0) – Aggregated load area to be appended
- **node_name** (*str*) – Name of LV side bus for appending LV load area
- **transformer_df** (pandas.DataFrame) – Transformer dataframe of network

Returns pandas.DataFrame – Transformer dataframe of network with appended transformers

ding0.tools.pypsa_io.**transform_timeseries4pypsa**(*timeseries*, *timerange*, *column=None*)

Transform pq-set timeseries to PyPSA compatible format :param timeseries: Containing timeseries :type timeseries: Pandas DataFrame

Returns **pypsa_timeseries** (*Pandas DataFrame*) – Reformated pq-set timeseries

8.1.1.5.9 ding0.tools.results module

8.1.1.5.10 ding0.tools.tests module

8.1.1.5.11 ding0.tools.tools module

ding0.tools.tools.**create_poly_from_source**(*source_point*, *left_m*, *right_m*, *up_m*, *down_m*)

Create a rectangular polygon given a source point and the number of meters away from the source point the edges have to be.

Parameters

- **source_point** (Shapely Point object) – The start point in WGS84 or epsg 4326 coordinates
- **left_m** (float) – The distance from the source at which the left edge should be.
- **right_m** (float) – The distance from the source at which the right edge should be.
- **up_m** (float) – The distance from the source at which the upper edge should be.
- **down_m** (float) – The distance from the source at which the lower edge should be.

ding0.tools.tools.**get_cart_dest_point**(*source_point*, *east_meters*, *north_meters*)

Get the WGS84 point in the coordinate reference system epsg 4326 at in given a cartesian form of input i.e. providing the position of the destination point in relative meters east and meters north from the source point. If the source point is (0, 0) and you would like the coordinates of a point that lies 5 meters north and 3 meters west of the source, the bearing in degrees is hard to find on the fly. This function allows the input as follows: >>> get_cart_dest_point(source_point, -3, 5) # west is negative east

Parameters

- **source_point** (Shapely Point object) – The start point in WGS84 or epsg 4326 coordinates
- **east_meters** (float) – Meters to the east of source, negative number means west

- **north_meters** (float) – Meters to the north of source, negative number means south

Returns Shapely Point object – The point in WGS84 or epsg 4326 coordinates at the destination which is north_meters north of the source and east_meters east of source.

ding0.tools.tools.**get_dest_point**(*source_point*, *distance_m*, *bearing_deg*)
Get the WGS84 point in the coordinate reference system epsg 4326 at a distance (in meters) from a source point in a given bearing (in degrees) (0 degrees being North and clockwise is positive).

Parameters

- **source_point** (Shapely Point object) – The start point in WGS84 or epsg 4326 coordinates
- **distance_m** (float) – Distance of destination point from source in meters
- **bearing_deg** (float) – Bearing of destination point from source in degrees, 0 degrees being North and clockwise is positive.

Returns Shapely Point object – The point in WGS84 or epsg 4326 coordinates at the destination which is distance meters away from the source_point in the bearing provided

ding0.tools.tools.**merge_two_dicts**(*x*, *y*)
Given two dicts, merge them into a new dict as a shallow copy.

Parameters

- **x** (*dict*) –
- **y** (*dict*) –

Notes

This function was originally proposed by http://stackoverflow.com/questions/38987/how-to-merge-two-python-dictionaries-in-a-single-expression

Credits to Thomas Vander Stichele. Thanks for sharing ideas!

Returns dict – Merged dictionary keyed by top-level keys of both dicts

ding0.tools.tools.**merge_two_dicts_of_dataframes**(*dict1*, *dict2*)
Merge two dicts of pandas.DataFrame with the same keys

Parameters

- **dict1** (*dict of dataframes*) –
- **dict2** (*dict of dataframes*) –

8.1.1.5.12 ding0.tools.validation module

8.1.1.5.13 ding0.tools.write_openego_header module

ding0.tools.write_openego_header.**absolute_file_paths**(*directory*)

ding0.tools.write_openego_header.**line_prepender**(*filename*, *line*)

ding0.tools.write_openego_header.**openego_header**()
openego header in files

Returns str – openego group py-file header

8.1.1.5.14 Module contents

8.1.2 Module contents

ding0.**adapt_numpy_int64**(*numpy_int64*)

Adapting numpy.int64 type to SQL-conform int type using psycopg extension, see[1] for more info.

Parameters **numpy_int64** (int) – numpty 64bits integer.

Returns *type* – #TODO: Description of return. Change type in the previous line accordingly

References

[1] http://initd.org/psycopg/docs/advanced.html#adapting-new-python-types-to-sql-syntax

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Bibliography

[Amme2017] J. Amme, G. Pleßmann, J. Bühler, L. Hülk, E. Kötter, P. Schwaegerl: *The eGo grid model: An open-source and open-data based synthetic medium-voltage grid model for distribution power supply systems*. Journal of Physics Conference Series 977(1):012007, 2018, doi:10.1088/1742-6596/977/1/012007

[Huelk2017] L. Hülk, L. Wienholt, I. Cussmann, U. Mueller, C. Matke and E. Kötter: *Allocation of annual electricity consumption and power generation capacities across multi voltage levels in a high spatial resolution*. International Journal of Sustainable Energy Planning and Management Vol. 13 2017 79–92, doi:10.5278/ijsepm.2017.13.6

[Kerber] G. Kerber: Aufnahmefähigkeit von Niederspannungsverteilnetzen für die Einspeisung aus Photovoltaikkleinanlagen, Dissertation, TU München, 2011

[Scheffler] J. Scheffler: Bestimmung der maximal zulässigen Netzanschlussleistung photovoltaischer Energiewandlungsanlagen in Wohnsiedlungsgebieten, Dissertation, TU Chemnitz, 2002

[Mohrmann] M. Mohrmann, C. Reese, L. Hofmann, J. Schmiesing: Untersuchung von Niederspannungsverteilnetzen anhand synthetische Netzstrukturen. In: Proceedings of VDE ETG Kongress, 2013

[OSM] OpenStreetMap contributors: Open street map, 2017

[VDEAR] VDE Anwenderrichtlinie: Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz, 2011

[DINEN50160] DIN EN 50160 Merkmale der Spannung in öffentlichen Elektrizitätsversorgungsnetzen, 2011

[Zdrallek] Planungs und Betriebsgrundsätze für ländliche Verteilungsnetze – Leitfaden zur Ausrichtung der Netze an ihren zukünftigen Anforderungen, 2016

[DENA] Deutsche Energie-Agentur GmbH (dena), "dena-Verteilnetzstudie. Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030.", 2012

[VNSRP] Ackermann, T., Untsch, S., Koch, M., & Rothfuchs, H. (2014). Verteilnetzstudie Rheinland-Pfalz. Hg. v. Ministerium für Wirtschaft, Klimaschutz, Energie und Landesplanung Rheinland-Pfalz (MWKEL). energynautics GmbH.

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

I