ding0 Documentation

openGo – Team

Sep 20, 2019
### Contents

1 What is ding0 about? 3

2 Getting started 5
   2.1 Installation ................................................................. 5
      2.1.1 Installation under Windows ....................................... 6
   2.2 Use Ding0 ................................................................. 7

3 How to use ding0? 9
   3.1 Examples .................................................................. 9
   3.2 High-level functions .................................................... 9
      3.2.1 Run ding0 .............................................................. 9
      3.2.2 For larger calculation (parallelization) ......................... 9
   3.3 Analysis of grid data ................................................... 10
      3.3.1 Plot results ........................................................... 10
      3.3.2 Export key figures .................................................. 10
      3.3.3 Compare data versions ............................................. 11
      3.3.4 Explanation of key figures ....................................... 12
   3.4 CSV file export ......................................................... 12
      3.4.1 Lines ................................................................. 12
      3.4.2 LV-Branchtees ...................................................... 13
      3.4.3 LV-Generators ...................................................... 13
      3.4.4 LV-Grids ............................................................ 14
      3.4.5 LV-Loads ............................................................ 14
      3.4.6 LV-Stations .......................................................... 15
      3.4.7 LV-Transformers ................................................... 15
      3.4.8 LV-Grids ............................................................ 16
      3.4.9 MV-Branchtees ...................................................... 16
      3.4.10 MV-Generators .................................................... 17
      3.4.11 MV-Grids ........................................................... 17
      3.4.12 MV-Loads .......................................................... 18
      3.4.13 MV-Stations ....................................................... 19
      3.4.14 MV-Transformers ................................................ 19

4 Theoretical background 21
   4.1 Data basis ................................................................. 21
      4.1.1 MV/LV Substations and LV grid district ....................... 21
   4.2 Medium-voltage grids .................................................. 21
4.3 Low-voltage grids ........................................... 23
  4.3.1 Branches of sector residential .......................... 23
  4.3.2 Branches of sector retail/industrial and agricultural .... 24
      4.3.2.1 Grid stability and equipment ....................... 24
  4.3.3 References ........................................... 25

5 Calculation principles ........................................ 27
  5.1 Reactance .................................................. 27
  5.2 Apparent power ............................................ 27
  5.3 Sign Convention ........................................... 27
      5.3.1 Generator Sign Convention .......................... 28
      5.3.1.1 Load Sign Convention .............................. 28

6 Notes to developers ........................................... 31
  6.1 Test the package installation ................................ 31
  6.2 Test ding0 runs .......................................... 31

7 What's New .................................................... 33
  7.1 Release v0.1.12 September 20, 2019 ........................ 33
      7.1.1 Changes ............................................... 33
  7.2 Release v0.1.10 November 5, 2018 .......................... 34
      7.2.1 Changes ............................................... 34
  7.3 Release v0.1.9 October 22, 2018 ............................ 34
  7.4 Release v0.1.8 September 5, 2018 ........................... 34
  7.5 Release v0.1.7 July 19, 2018 ................................ 34
  7.6 Release v0.1.6 July 6, 2018 ................................ 35
  7.7 Release v0.1.5 (June 6, 2018) ............................... 35
  7.8 Release v0.1.4 (January 17, 2018) ......................... 35
      7.8.1 Added features ........................................ 35
      7.8.2 Bug fixes ............................................. 35
      7.8.3 Other changes ........................................ 35
  7.9 Release v0.1.3 (September 1, 2017) ........................ 36
      7.9.1 Added features ........................................ 36
      7.9.2 Bug fixes ............................................. 36
      7.9.3 Other changes ........................................ 36
  7.10 Release v0.1.2 (July 25, 2017) ............................. 36
  7.11 Release v0.1.0 (July 25, 2017) ............................. 36

8 ding0 .......................................................... 37
  8.1 ding0 package ............................................. 37
      8.1.1 Subpackages .......................................... 37
          8.1.1.1 ding0.config package ......................... 37
              8.1.1.1.1 Submodules ................................ 37
              8.1.1.1.2 ding0.config.config_db_interfaces module ... 37
              8.1.1.1.3 Module contents .............................. 40
          8.1.1.2 ding0.core package ............................... 40
              8.1.1.2.1 Subpackages ................................ 40
                  8.1.1.2.1.1 ding0.core.network package .............. 40
                  8.1.1.2.1.2 Submodules .............................. 40
                  8.1.1.2.1.3 ding0.core.network.cable_distributors module .... 40
                  8.1.1.2.1.4 ding0.core.network.grids module ........... 40
                  8.1.1.2.1.5 ding0.core.network.loads module ........... 46
                  8.1.1.2.1.6 ding0.core.network.stations module ........ 46
                  8.1.1.2.1.7 ding0.core.network.transformers module ...... 47
                  8.1.1.2.1.8 Module contents ............................ 47
| 8.1.1.5.2 | ding0.tools.animation module | 99 |
| 8.1.1.5.3 | ding0.tools.config module | 99 |
| 8.1.1.5.4 | ding0.tools.debug module | 100 |
| 8.1.1.5.5 | ding0.tools.geo module | 101 |
| 8.1.1.5.6 | ding0.tools.logger module | 102 |
| 8.1.1.5.7 | ding0.tools.plots module | 103 |
| 8.1.1.5.8 | ding0.tools.pypsa_io module | 103 |
| 8.1.1.5.9 | ding0.tools.results module | 105 |
| 8.1.1.5.10 | ding0.tools-tests module | 109 |
| 8.1.1.5.11 | ding0.tools-tools module | 110 |
| 8.1.1.5.12 | ding0.tools-validation module | 111 |
| 8.1.1.5.13 | ding0.tools-write_openego_header module | 112 |
| 8.1.1.5.14 | Module contents | 112 |

9 Indices and tables 113

Bibliography 115

Python Module Index 117

Index 119
Dlistribution Network GeneratOr – A tool to generate synthetic medium and low voltage power distribution grids based on open (or at least accessible) data.
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*. 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*. 
2.1 Installation

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.

```bash
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

```bash
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

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

Jump into (aka. activate) this python distribution by

```bash
# Adjust path to your specific needs
source ~/.virtualenvs/ding0/bin/activate
```
From that, the latest release of Ding0 is installed by

```
pip3 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

```
pip3 install -e path/to/cloned/ding0/repository
```

### 2.1.1 Installation under 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.

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 commands:

```
conda activate ding0_env
pip install -U -e \path\to\ding0\n```

Or install it from the pypi repository with the command:

```
conda activate ding0_env
pip install ding0
```

after this, it is possible to install ding0 directly from pip within the conda environment
2.2 Use Ding0

Have a look at the *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:

![MV Grid District 460 - PF result (feedin case)](image)

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:

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

(continues on next page)
Extract key information about medium and low voltage grid topology.

```python
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

```python
mv_stats,
 lv_stats,
 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.

```python
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()`.

```python
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>km_cable</td>
<td>Cumulative length of underground cables</td>
<td>km</td>
</tr>
</tbody>
</table>

3.4 CSV file export

Ding0 objects are exported in csv files.

3.4.1 Lines

Table 3.1: line.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge_name</td>
<td>str</td>
<td>unambiguous name of edge</td>
<td>n/a</td>
</tr>
<tr>
<td>grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of corresponding grid (MVgrid-id if MV-edge, LVgrid-id if LV-edge)</td>
<td>n/a</td>
</tr>
<tr>
<td>type_kind</td>
<td>str</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>type_name</td>
<td>str</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>node1</td>
<td>str</td>
<td>id_db of first node</td>
<td>n/a</td>
</tr>
<tr>
<td>node2</td>
<td>str</td>
<td>id_db of second node</td>
<td>n/a</td>
</tr>
<tr>
<td>length</td>
<td>float</td>
<td>length of line</td>
<td>km</td>
</tr>
<tr>
<td>U_n</td>
<td>float</td>
<td>nominal voltage</td>
<td>kV</td>
</tr>
<tr>
<td>R</td>
<td>float</td>
<td></td>
<td>Ohm/km</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>inductive resistance at 50Hz</td>
<td>uF/km</td>
</tr>
<tr>
<td>L</td>
<td>float</td>
<td></td>
<td>mH/km</td>
</tr>
<tr>
<td>I_max_th</td>
<td>float</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
### 3.4.2 LV-Branchtees

Table 3.2: lv_branchtee.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘LV-CableDistributorD-ing0_LV_#lvgridid#_#ascendingnumber#’</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>None</td>
<td>geometric coordinates</td>
<td>n/a</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>

### 3.4.3 LV-Generators

Table 3.3: lv_generator.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘LV-GeneratorD-ing0_LV_#lvgridid#_#ascendingnumber#’</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>type</td>
<td>str</td>
<td>type of generation</td>
<td>{solar; biomass}</td>
</tr>
<tr>
<td>subtype</td>
<td>str</td>
<td>subtype of generation: {solar_roof_mounted, unknown; biomass}</td>
<td>n/a</td>
</tr>
<tr>
<td>v_level</td>
<td>int</td>
<td>voltage level of generator</td>
<td></td>
</tr>
<tr>
<td>nominal_capacity</td>
<td>float</td>
<td>nominal capacity</td>
<td></td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
### 3.4.4 LV-Grids

Table 3.4: lv_grid.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘LVGridDing0_LV_##lvgridid##_lvgridid#’</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id</td>
<td>int</td>
<td>unambiguous number of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 MULTIPOLYGON</td>
</tr>
<tr>
<td>population</td>
<td>int</td>
<td>population in LV-Grid</td>
<td>?</td>
</tr>
<tr>
<td>voltage_nom</td>
<td>float</td>
<td>voltage level of grid</td>
<td>kV</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>

### 3.4.5 LV-Loads

Table 3.5: lv_load.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘LVLoadDing0_LV_##lvgridid##_ascendingnumber#’</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>None</td>
<td>geometric coordinates</td>
<td>n/a</td>
</tr>
<tr>
<td>consumption</td>
<td>{'str': float}</td>
<td>type of load {residential, agricultural, industrial} and corresponding consumption</td>
<td>n/a</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
### 3.4.6 LV-Stations

Table 3.6: lvmv_station.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ('LVStrationD-ing0_MV_#mvgridid#_#lvgridid#')</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>

### 3.4.7 LV-Transformers

Table 3.7: lv_transformer.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ('TransformerD-ing0_LV_#mvgridid#_#lvgridid#')</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>voltage_op</td>
<td>float</td>
<td></td>
<td>kV</td>
</tr>
<tr>
<td>S_nom</td>
<td>float</td>
<td>nominal apparent power</td>
<td>kVA</td>
</tr>
<tr>
<td>X</td>
<td>float</td>
<td></td>
<td>Ohm</td>
</tr>
<tr>
<td>R</td>
<td>float</td>
<td></td>
<td>Ohm</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
## 3.4.8 LV-Grids

Table 3.8: mvlv_mapping.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LV_grid_id</td>
<td>int</td>
<td>unambiguous number of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id</td>
<td>int</td>
<td>unambiguous number of MV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>LV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of MV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>

## 3.4.9 MV-Branchtees

Table 3.9: mv_branchtee.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘MV-CableDistributorD-#mvgrid#_#ascendingnumber#’</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of MV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
### 3.4.10 MV-Generators

Table 3.10: mv_generator.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘MV-GeneratorD-ing0_MV_#mvgridid#_#ascendingnumber#’</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id_db</td>
<td>int</td>
<td>unambiguous id_db of MV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>type</td>
<td>str</td>
<td>type of generation: {solar; biomass}</td>
<td>n/a</td>
</tr>
<tr>
<td>subtype</td>
<td>str</td>
<td>subtype of generation: {solar_ground_mounted, solar_roof_mounted, unknown; biomass, biogas}</td>
<td>n/a</td>
</tr>
<tr>
<td>v_level</td>
<td>int</td>
<td>voltage level of generator</td>
<td></td>
</tr>
<tr>
<td>nominal_capacity</td>
<td>float</td>
<td>nominal capacity</td>
<td></td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
### 3.4.11 MV-Grids

Table 3.11: mv_grid.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘MVGridDing0_MV_#mvgridid#_#mvgridid#’</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id</td>
<td>int</td>
<td>unambiguous number of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 MULTIPOLYGON</td>
</tr>
<tr>
<td>population</td>
<td>int</td>
<td>population in LV-Grid</td>
<td>?</td>
</tr>
<tr>
<td>voltage_nom</td>
<td>float</td>
<td>voltage level of grid</td>
<td>kV</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmss</td>
</tr>
</tbody>
</table>

### 3.4.12 MV-Loads

Table 3.12: mv_load.csv

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

Table 3.13: mvhv_station.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘MVS-stationD-ing0_MV_#mvgridid_#mvgridid’</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id_db</td>
<td>int</td>
<td>id_db of MV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>

### 3.4.14 MV-Transformers

Table 3.14: lv_transformer.csv

<table>
<thead>
<tr>
<th>Field</th>
<th>type</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_db</td>
<td>str</td>
<td>unambiguous name: ‘TransformerD-ing0_MV_#mvgridid’</td>
<td>n/a</td>
</tr>
<tr>
<td>MV_grid_id_db</td>
<td>int</td>
<td>id_db of LV-Grid</td>
<td>n/a</td>
</tr>
<tr>
<td>geom</td>
<td>wkt</td>
<td>geometric coordinates</td>
<td>WGS84 POINT</td>
</tr>
<tr>
<td>voltage_op</td>
<td>float</td>
<td></td>
<td>kV</td>
</tr>
<tr>
<td>S_nom</td>
<td>float</td>
<td>nominal apparent power</td>
<td>kVA</td>
</tr>
<tr>
<td>X</td>
<td>float</td>
<td>Ohm</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>float</td>
<td>Ohm</td>
<td></td>
</tr>
<tr>
<td>run_id</td>
<td>int</td>
<td>time and date of table generation</td>
<td>yyyyMMddhhmmss</td>
</tr>
</tbody>
</table>
4.1 Data basis

The fundamental data basis is described 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 an 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 found 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 plausi-
   ble 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 appart-
   ment and 1.5 apartments 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 con-
   nections (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/substation

   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 define 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)


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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. load in each branch</td>
<td>290 kVA</td>
</tr>
<tr>
<td>Max. branch length retail/industrial $L_{R/\text{I, max}}$</td>
<td>400 m</td>
</tr>
<tr>
<td>Max. branch length agricultural $L_{A, \text{max}}$</td>
<td>800 m</td>
</tr>
<tr>
<td>Length of branch stub</td>
<td>30 m</td>
</tr>
<tr>
<td>PV peak power &lt;= 30 kW</td>
<td>residential</td>
</tr>
<tr>
<td>PV peak power &gt; 30 kW &lt;= 100 kW</td>
<td>retail/industrial or agricultural</td>
</tr>
<tr>
<td>PV peak power &gt; 100 kW</td>
<td>MV/LV station bus bar</td>
</tr>
</tbody>
</table>

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.

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

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

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

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 connected 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 available 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
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.

\[ x = \omega \cdot L \]

5.2 Apparent power

- Given maximum thermal current \( I_{th\_amx} \) \((I_L)\) is given per conductor (of three cables in a system)/per phase.
- We assume to have delta connection. Thus, nominal voltage per conductor 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
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

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

```bash
chmod +x install.docker.sh
./install.docker.sh
```

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

```bash
./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 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

```python
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:

```python
manual_ding0_test([438], 'ding0_tests_grids_2.pkl')
```
To create a file with the output of a ding0 run in the default configuration (district [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 assumed 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].
CHAPTER 7

What’s New

See what’s new as per release!

<table>
<thead>
<tr>
<th>Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Release v0.1.12 September 20, 2019</td>
</tr>
<tr>
<td>• Release v0.1.10 November 5, 2018</td>
</tr>
<tr>
<td>• Release v0.1.9 October 22, 2018</td>
</tr>
<tr>
<td>• Release v0.1.8 September 5, 2018</td>
</tr>
<tr>
<td>• Release v0.1.7 July 19, 2018</td>
</tr>
<tr>
<td>• Release v0.1.6 July 6, 2018</td>
</tr>
<tr>
<td>• Release v0.1.5 (June 6, 2018)</td>
</tr>
<tr>
<td>• Release v0.1.4 (January 17, 2018)</td>
</tr>
<tr>
<td>• Release v0.1.3 (September 1, 2017)</td>
</tr>
<tr>
<td>• Release v0.1.2 (July 25, 2017)</td>
</tr>
<tr>
<td>• Release v0.1.0 (July 25, 2017)</td>
</tr>
</tbody>
</table>

7.1 Release v0.1.12 September 20, 2019

7.1.1 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.

### 7.2 Release v0.1.10 November 5, 2018

This release introduces new plotting functionalities.

#### 7.2.1 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

### 7.3 Release v0.1.9 October 22, 2018

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

### 7.4 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

### 7.5 Release v0.1.7 July 19, 2018

A release to update software dependencies.

• Explicit dependencies of Pyomo and Scipy are removed
7.6 Release v0.1.6 July 6, 2018

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

7.7 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

7.8 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.

7.8.1 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

7.8.2 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

7.8.3 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
7.9 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.

7.9.1 Added features

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

7.9.2 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

7.9.3 Other changes

• Update Pandas dependency to 0.20.3
• Update PyPSA dependency to 0.10.0

7.10 Release v0.1.2 (July 25, 2017)

Renaming of package: dingo to ding0

7.11 Release v0.1.0 (July 25, 2017)

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

```python
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?

desc geom_lv_load_area_centres
    Description.
    Type type

desc geom_lv_stations
    Description.
    Type type

desc geom_mv_cable_dists
    Description.
    Type type

desc geom_mv_circuit_breakers
    Description.
```

...
Type `geom_mv_generators`

Description.

Type `geom_mv_lines`

Description.

Type `geom_mv_station`

Description.

Type `grid_id`

Description.

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 `geom`

Description.

Type `grid_id`

Description.

Type `length`

Description.

Type `s_res0`

Description.

Type `s_res1`

Description.

Type `type_kind`

Description.

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**
8.1.1.3 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

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_id
        Returns ...#TODO

        Type str

8.1.1.2.1.4 ding0.core.network.grids module

class ding0.core.network.grids.LVGridDing0(**kwargs)
    Bases: ding0.core.network.GridDing0
    DING0 low voltage grid

    Parameters

    • region(LVLoadAreaDing0) – LV region that is associated with grid

    • default_branch_kind(str) – description #TODO

    • population – description #TODO
Note: It is assumed that LV grid have got cables only (attribute ‘default_branch_kind’)

```python
add_cable_dist(lv_cable_dist)
   Adds a LV cable_dist to _cable_dists and grid graph if not already existing

   Parameters lv_cable_dist – Description #TODO
```

```python
add_load(lv_load)
   Adds a LV load to _loads and grid graph if not already existing

   Parameters lv_load – Description #TODO
```

```python
add_station(lv_station)
   Adds a LV station to _station and grid graph if not already existing
```

```python
build_grid()
   Create LV grid graph
```

```python
connect_generators(debug=False)
   Connects LV generators (graph nodes) to grid (graph)

   Parameters debug (bool, defaults to False) – If True, information is printed during process
```

```python
loads_sector(sector='res')
   Returns a generator for iterating over grid’s sectoral loads

   Parameters sector (str) – possible values:
      • 'res' (residential),
      • 'ria' (retail, industrial, agricultural)

   Yields int – Generator for iterating over loads of the type specified in sector.
```

```python
reinforce_grid()
   Performs grid reinforcement measures for current LV grid.
```

```python
station()
   Returns grid’s station
```

```python
class ding0.core.network.grids.MVGridDing0(**kwargs)
   Bases: ding0.core.network.GridDing0

   DING0 medium voltage grid

   Parameters
      • region (MVGridDistrictDing0) – MV region (instance of MVGridDistrictDing0 class) that is associated with grid
      • default_branch_kind (str) – kind of branch (possible values: ‘cable’ or ‘line’)
      • default_branch_type (pandas.Series) – type of branch (pandas Series object with cable/line parameters)
      • default_branch_type (pandas.Series) – type of branch (pandas Series object with cable/line parameters)
```

```python
add_cable_distributor(cable_dist)
   Adds a cable distributor to _cable_distributors if not already existing

   Parameters cable_dist (float) – Description #TODO
```

```python
add_circuit_breaker(circ_breaker)
   Creates circuit breaker object and . . .
```

8.1. ding0 package
Parameters `circ_breaker` (**CircuitBreakerDing0**): Description #TODO

**add_load** (**lv_load**)
Adds a MV load to _loads and grid graph if not already existing

Parameters `lv_load` (**float**): Description #TODO

**add_ring** (**ring**)
Adds a ring to _rings if not already existing

**add_station** (**mv_station**, **force=False**)
Adds MV station if not already existing

Parameters

• `mv_station` (**MVStationDing0**): Description #TODO

• `force` (**bool**): If True, MV Station is set even though it’s not empty (override)

**circuit_breakers** ()
Returns a generator for iterating over circuit breakers

**circuit_breakers_count** ()
Returns the count of circuit breakers in MV grid

**close_circuit_breakers** ()
Closes all circuit breakers in MV grid

**connect_generators** (**debug=False**)
Connects MV generators (graph nodes) to grid (graph)

Parameters `debug` (**bool**, defaults to False)
If True, information is printed during process

**export_to_pypsa** (**session**, **method='onthefly'**)
Exports MVGridDing0 grid to PyPSA database tables

Peculiarities of MV grids are implemented here. Derive general export method from this and adapt to needs of LVGridDing0

Parameters

• `session` (**SQLAlchemy session object**): Database session

• `method` (**str**): Specify export method:
  + 'db': grid data will be exported to database
  + 'onthefly': grid data will be passed to PyPSA directly (default)

**Note:** It has to be proven that this method works for LV grids as well!

Ding0 treats two stationary case of powerflow:

1) Full load: We assume no generation and loads to be set to peak load

2) Generation worst case:

**get_ring_from_node** (**node**)
Determines the ring (**RingDing0** object) which node is member of. :param node: Ding0 object (member of graph) :type node: GridDing0

**Returns** **RingDing0** – Ringo of which node is member.
**graph_nodes_from_subtree** *(node_source, include_root_node=False)*

Finds all nodes of a tree that is connected to `node_source` and are (except `node_source`) not part of the ring of `node_source` (traversal of graph from `node_source` excluding nodes along ring).

**Example**

A given graph with ring (edges) 0-1-2-3-4-5-0 and a tree starting at node (`node_source`) 3 with edges 3-6-7, 3-6-8-9 will return [6,7,8,9]

**Parameters**

- `node_source` *(GridDing0)* – source node (Ding0 object), member of `_graph`
- `include_root_node` *(bool, defaults to False)* – If True, the root node is included in the list of ring nodes.

**Returns** list of GridDing0 – List of nodes (Ding0 objects)

**import_powerflow_results** *(session)*

Assign results from power flow analysis to edges and nodes

**Parameters**

- `session` *(SQLAlchemy session object)* – Description

**open_circuit_breakers** *

Opens all circuit breakers in MV grid

**parametrize_grid** *(debug=False)*

Performs Parametrization of grid equipment:

i) Sets voltage level of MV grid,

ii) Operation voltage level and transformer of HV/MV station,

iii) Default branch types (normal, aggregated, settlement)

**Parameters**

- `debug` *(bool, defaults to False)* – If True, information is printed during process.

**Note:** It is assumed that only cables are used within settlements.

**reinforce_grid** *

Performs grid reinforcement measures for current MV grid

**remove_cable_distributor** *(cable_dist)*

Removes a cable distributor from `_cable_distributors` if existing

**rings_count** *

Returns the count of rings in MV grid

**Yields** For each ring, tuple composed by ring ID, list of edges, list of nodes

**Note:** Circuit breakers must be closed to find rings, this is done automatically.
**rings_nodes** *(include_root_node=False, include_satellites=False)*

Returns a generator for iterating over rings (=routes of MVGrid’s graph)

**Parameters**

- **include_root_node** *(bool, defaults to False)* – If True, the root node is included in the list of ring nodes.

- **include_satellites** *(bool, defaults to False)* – If True, the satellite nodes (nodes that diverge from ring nodes) is included in the list of ring nodes.

**Yields** list of GridDing0 – List with nodes of each ring of _graph in- or excluding root node (HV/MV station) (arg include_root_node), format:

```
[ ring_m_node_1, ..., ring_m_node_n ]
```

**Note:** Circuit breakers must be closed to find rings, this is done automatically.

**routings** *(debug=False, anim=None)*

Performs routing on Load Area centres to build MV grid with ring topology.

**Parameters**

- **debug** *(bool, defaults to False)* – If True, information is printed while routing

- **anim** *(type, defaults to None)* – Descr #TODO

**run_powerflow** *(session, export_pypsa_dir=None, method='onthefly', debug=False)*

Performs power flow calculation for all MV grids

**Parameters**

- **session** *(SQLAlchemy session object)* – Database session

- **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.

- **method** *(str)* – Specify export method:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'db'</td>
<td>grid data will be exported to database</td>
</tr>
<tr>
<td>'onthefly'</td>
<td>grid data will be passed to PyPSA directly (default)</td>
</tr>
</tbody>
</table>

- **debug** *(bool, defaults to False)* – If True, information is printed during process

**Note:** It has to be proven that this method works for LV grids as well!

Ding0 treats two stationary case of powerflow: 1) Full load: We assume no generation and loads to be set to peak load 2) Generation worst case:

**set_branch_ids**()

Generates and sets ids of branches for MV and underlying LV grids.

While IDs of imported objects can be derived from dataset’s ID, branches are created within DING0 and need unique IDs (e.g. for PF calculation).

**set_circuit_breakers** *(debug=False)*

Calculates the optimal position of the existing circuit breakers and relocates them within the graph.
Parameters **debug** (*bool*, *defaults to False*) – If True, information is printed during process

See also:

ding0.grid.mv_grid.tools.set_circuit_breakers()

**set_default_branch_type** (*debug=False*)
Determines default branch type according to grid district’s peak load and standard equipment.

Parameters **debug** (*bool*, *defaults to False*) – If True, information is printed during process

Returns

- **pandas.Series** – default branch type: pandas Series object. If no appropriate type is found, return largest possible one.
- **pandas.Series** – default branch type max: pandas Series object. Largest available line/cable type

Note: Parameter values for cables and lines are taken from\(^1\),\(^2\) and\(^3\). Lines are chosen to have 60 % load relative to their nominal capacity according to\(^4\).

Decision on usage of overhead lines vs. cables is determined by load density of the considered region. Urban areas usually are equipped with underground cables whereas rural areas often have overhead lines as MV distribution system\(^5\).

References

**set_nodes_aggregation_flag** (*peak_current_branch_max*)
Set Load Areas with too high demand to aggregated type.

Parameters **peak_current_branch_max** (*float*) – Max. allowed current for line/cable

**set_voltage_level** (*mode='distance'*)
Sets voltage level of MV grid according to load density of MV Grid District or max. distance between station and Load Area.

Parameters **mode** (*str*) – method to determine voltage level

- 'load_density': Decision on voltage level is determined by load density of the considered region. Urban areas (load density of \(\geq 1\) MW/km\(^2\) according to\(^6\)) usually got a voltage of 10 kV whereas rural areas mostly use 20 kV.
- 'distance' (default): Decision on voltage level is determined by the max. distance between Grid District’s HV-MV station and Load Areas (LA’s centre is used). According to\(^7\) a value of 1kV/kV can be assumed. The \(voltage\_per\_km\_threshold\) defines the distance threshold for distinction. (default in config = \((20\text{km}+10\text{km})/2 = 15\text{km}\))

---

2. René Flosdorff et al., “Elektrische Energieverteilung”, Vieweg+Teubner, 2005
4. Deutsche Energie-Agentur GmbH (dena), "dena-Verteilnetzstudie. Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030!", 2012
8.1.1.2.1.5 ding0.core.network.loads module

```python
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.
```

```python
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
```

8.1.1.2.1.6 ding0.core.network.stations module

```python
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
```

```python
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_id
Description

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 8.
Potential hv-mv-transformers are chosen according to 9.

References

set_operation_voltage_level()
Set operation voltage level

8.1.2.1.7 ding0.core.network.transformers module

8.1.2.1.8 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.

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

8 Deutsche Energie-Agentur GmbH (dena), “dena-Verteilnetzstudie. Ausbau- und Innovationsbedarf der Stromverteilnetze in Deutschland bis 2030.”, 2012
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).

**See also:**
- `set_branch_ids()`

**network**
- Getter for the overarching `NetworkDing0` object.
  - Returns `NetworkDing0`

```python
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`

```python
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.
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 generator’s 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
• gasMine
• gas_sewage
• gas_landfill
• gas
• waste_wood
• wood

Type `str`

`network`

Getter for the overarching NetworkDing object.

Returns `NetworkDing`

`pypsa_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.GeneratorFluctuatingDing(**kwargs)`

Bases: `ding0.core.network.GeneratorDing`

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 Obj

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) – source node, member of _graph, ding0 node object
- node_target (GeneratorDing0 or GeneratorFluctuatingDing0 or LoadDing0 or StationDing0 or CircuitBreakerDing0 or CableDistributorDing0) – 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) – source node, member of _graph, ding0 node object
- node_target (GeneratorDing0 or GeneratorFluctuatingDing0 or LoadDing0 or StationDing0 or CircuitBreakerDing0 or CableDistributorDing0) – 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
- 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 one. 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_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)* – 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)* – 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

**graph_draw** *(node)*

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
  - `branch` paired 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.

```python
def 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

def 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

def 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

def 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 CircuitBreakerDing0 or CableDistributorDing0
    node_target: GeneratorDing0 or GeneratorFluctuatingDing0 or LoadDing0 or StationDing0 or CircuitBreakerDing0 or CableDistributorDing0
    Returns float – path length in meters

def loads():
    """Returns a generator for iterating over grid’s loads"
    Returns list generator – List Generator of LoadDing0 objects

def 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

def 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."
```

8.1. ding0 package
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

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 MVGridDistrictD-
ing0.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 identifier used by PyPSA's plots #TODO

  Note: This class can be called as follows:

  i) With scenarios and timeranges:

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

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

  ```python
  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 multiply to Q time series

**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**
Desc
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.region.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

network

class ding0.core.structure.region.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_id
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

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

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.

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 DINGO

mv_grid
   Descr
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.

peak_load
Descr

peak_load_satellites
Descr

peak_load_aggregated
Descr

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

8.1.1.2.1.15 Module contents

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

Defines a region in DING0
8.1.2.2 Module contents

class ding0.core.NetworkDing0(**kwargs)
    Bases: object

Defines the DING0 Network - not a real grid but a container for the MV-grids. Contains the NetworkX graph and associated attributes.

This object behaves like a location to store all the constituent objects required to estimate the grid topology of a given set of shapes that need to be connected.

The most important function that defines ding0’s use case is initiated from this class i.e. run_ding0().

Parameters

- **name** (str) – A name given to the network. This defaults to Network.

- **run_id** (str) – A unique identification number to identify different runs of Ding0. This is usually the date and the time in some compressed format. e.g. 201901010900.

mv_grid_districts

Contains the MV Grid Districts where the topology has to be estimated. A list of MVGridDistrictDing0 objects whose data is stored in the current instance of the NetworkDing0 Object. By default the list is empty. MV grid districts can be added by using the function add_mv_grid_district(). This is done within the function build_mv_grid_district() in the normal course upon calling run_ding0().

Type list iterator

config

These are the configurations that are required for the construction of the network topology given the areas to be connected together. The configuration is imported by calling import_config(). The configurations are stored in text files within the ding0 package in the config folder. These get imported into a python dictionary-like configuration object.

Type dict

pf_config

These are the configuration of the power flows that are run to ensure that the generated network is plausible and is capable of a reasonable amount of loading without causing any grid issues. This object cannot be set at initiation, it gets set by the function import_pf_config() which takes the configurations from :attr: config and sets up the configurations for running power flow calculations.

Type PFConfigDing0

static_data

Data such as electrical and mechanical properties of typical assets in the energy system are stored in ding0. These are used in many parts of ding0’s calculations. Data values:

- **Typical cable types, and typical line types’ electrical impedences**, thermal ratings, operating voltage level.

- **Typical transformers types’ electrical impedences, voltage drops**, thermal ratings, winding voltages

- **Typical LV grid topologies’ line types, line lengths and** distribution

Type dict

orm

The connection parameters to the OpenEnergy Platform and the tables and datasets required for the functioning of ding0.
**Type**: `dict`

**add_mv_grid_district** *(mv_grid_district)*
A method to add `mv_grid_districts` to the *NetworkDing0* Object by adding it to the `mv_grid_districts`.

**build_lv_grid_district** *(lv_load_area, lv_grid_districts, lv_stations)*
Instantiates and associates `lv_grid_district` incl grid and station.
The instantiation creates more or less empty objects including relevant data for transformer choice and grid creation

**Parameters**
- `lv_load_area` *(Shapely Polygon object)* – load_area object
- `lv_grid_districts` *(pandas.DataFrame)* – Table containing `lv_grid_districts` of according load_area
- `lv_stations` *(pandas.DataFrame)* – Table containing `lv_stations` of according load_area

**build_lv_grids()**
Builds LV grids for every non-aggregated LA in every MV grid district using model grids.

**build_mv_grid_district** *(poly_id, subst_id, grid_district_geo_data, station_geo_data)*
Initiates single MV grid_district including station and grid

**Parameters**
- `poly_id` *(int)* – ID of grid_district according to database table. Also used as ID for created grid #TODO: check type
- `subst_id` *(int)* – ID of station according to database table #TODO: check type
- `grid_district_geo_data` *(Shapely Polygon object)* – Polygon of grid district. 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.
- `station_geo_data` *(Shapely Point object)* – Point of station. 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.

**Returns**: `MVGridDistrictDing0`

**config**
Getter for the configuration dictionary.

**Returns**: `dict`

**connect_generators** *(debug=False)*
Connects generators (graph nodes) to grid (graph) for every MV and LV Grid District

**Parameters**
- `debug` *(bool, defaults to False)* – If True, information is printed during process.

**control_circuit_breakers** *(mode=None)*
Opens or closes all circuit breakers of all MV grids.

**Parameters**
- `mode` *(str)* – Set mode='open' to open, mode='close' to close

**export_mv_grid** *(session, mv_grid_districts)*
Exports MV grids to database for visualization purposes

**Parameters**
- `session` *(SQLAlchemy session object)* – Database session
• **mv_grid_districts** (list of) – `MVGridDistrictDing0` objects whose MV
grids are exported.

**export_mv_grid_new** *(session, mv_grid_districts)*
Exports MV grids to database for visualization purposes

**Parameters**

• **session** (SQLAlchemy session object) – Database session

• **mv_grid_districts** (list of) – `MVGridDistrictDing0` objects whose MV
  grids are exported.

**get_mvgd_lvla_lvgd_obj_from_id()**
Build dict with mapping from:

• `LVLoadAreaDing0` id to `LVLoadAreaDing0` object,
• `MVGridDistrictDing0` id to `MVGridDistrictDing0` object,
• `LVGridDistrictDing0` id to `LVGridDistrictDing0` object
• `LVStationDing0` id to `LVStationDing0` object

**Returns**

• **dict** –
  mv_grid_districts_dict:

```python
{
    mv_grid_district_id_1: mv_grid_district_obj_1,
    ..., 
    mv_grid_district_id_n: mv_grid_district_obj_n
}
```

• **dict** –
  lv_load_areas_dict:

```python
{
    lv_load_area_id_1: lv_load_area_obj_1,
    ..., 
    lv_load_area_id_n: lv_load_area_obj_n
}
```

• **dict** –
  lv_grid_districts_dict:

```python
{
    lv_grid_district_id_1: lv_grid_district_obj_1,
    ..., 
    lv_grid_district_id_n: lv_grid_district_obj_n
}
```

• **dict** –
  lv_stations_dict:
import_config()

Loads parameters from config files

Returns `dict` – configuration key value pair dictionary

import_generators (session, debug=False)

Imports renewable (res) and conventional (conv) generators

Parameters

- `session` (SQLAlchemy session object) – Database session
- `debug` (bool, defaults to False) – If True, information is printed during process

Note: Connection of generators is done later on in `NetworkDing0’s method connect_generators()`

import_lv_grid_districts (session, lv_stations)

Imports all lv grid districts within given load area

Parameters

- `session` (SQLAlchemy session object) – Database session

Returns `lv_grid_districts` (pandas.DataFrame) – Table of `lv_grid_districts`

import_lv_load_areas (session, mv_grid_district, lv_grid_districts, lv_stations)

Imports load_areas (load areas) from database for a single MV grid_district

Parameters

- `session` (SQLAlchemy session object) – Database session
- `mv_grid_district` (MV grid_district/station (instance of MVGridDistrictDing0 class) for) – which the import of load areas is performed
- `lv_grid_districts` (pandas.DataFrame) – LV grid districts within this `mv_grid_district`
- `lv_stations` (pandas.DataFrame) – LV stations within this `mv_grid_district`

import_lv_stations (session)

Import lv_stations within the given load_area

Parameters

- `session` (SQLAlchemy session object) – Database session

Returns `lv_stations` (pandas.DataFrame) – Table of `lv_stations`

import_mv_grid_districts (session, mv_grid_districts_no=None)

Imports MV Grid Districts, HV-MV stations, Load Areas, LV Grid Districts and MV-LV stations, instantiates and initiates objects.

Parameters

- `session` (sqlalchemy.orm.session.Session) – Database session
• **mv_grid_districts** *(list of int)* – List of MV grid_districts/stations (int) to be imported (if empty, all grid_districts & stations are imported)

See also:

- `build_mv_grid_district()` used to instantiate MV grid_district objects
- `import_lv_load_areas()` used to import load_areas for every single MV grid_district
- `ding0.core.structure.regions.MVGridDistrictDing0.add_peak_demand()` used to summarize peak loads of underlying load_areas

**import_orm()**

Import ORM classes names for the correct connection to open energy platform and access tables depending on input in config in self.config which is loaded from ‘config_db_tables.cfg’

**Returns** *(obj: dict)* – key value pairs of names of datasets versus sqlalchmey maps to acess various tables where the datasets used to build grids are stored on the open energy platform.

**import_pf_config()**

Creates power flow config class and imports config from file

**Returns** *(PFConfigDing0)*

**import_static_data()**

Imports static data into NetworkDing0 such as equipment.

**Returns** *(obj: dict)* – Dictionary with equipment data

**list_generators(session)**

List renewable (res) and conventional (conv) generators

**Parameters** *(session)* *(SQLAlchemy session object)* – Database session

**Returns** *(pandas.DataFrame)* – A table containing the generator data, the columns being: - subst_id, - la_id (load area id), - mv_lv_subst_id (id of the mv lv substation), - electrical_capacity - generation_type - generation_subtype - voltage_level - geospatial coordinates as Shapely Point object

**list_load_areas(session, mv_districts)**

list load_areas (load areas) peak load from database for a single MV grid_district

**Parameters**

- **session** *(SQLAlchemy session object)* – Database session
- **mv_districts** *(list)* – MVGridDistrictDing0 objects

**list_lv_grid_districts(session, lv_stations)**

Imports all lv grid districts within given load area

**Parameters**

- **session** *(SQLAlchemy session object)* – Database session
- **lv_stations** *(list)* – List required LV_stations==LV districts.

**Returns** *(pandas.DataFrame)* – Pandas Data Frame Table of lv_grid_districts

**metadata**

Provide metadata on a Ding0 run

**Parameters** *(run_id)* *(str, (defaults to current date))* – Distinguish multiple versions of Ding0 data by a run_id. If not set it defaults to current date in the format YYYYMMDDhhmms

---

8.1. ding0 package 67
Returns dict – Metadata

mv_grid_districts()
A generator for iterating over MV grid_districts
Returns list iterator – A list iterator containing the MVGridDistrictDing0 objects.

mv_parametrize_grid(debug=False)
Performs Parametrization of grid equipment of all MV grids.
Parameters debug (:obj:bool, defaults to False) – If True, information is printed during process.

See also:
ding0.core.network.grids.MVGridDing0.parametrize_grid()

mv_routing(debug=False, animation=False)
Performs routing on all MV grids.
Parameters
- debug (bool, default to False) – If True, information is printed while routing
- animation (bool, default to False) – If True, images of route modification steps are exported during routing process. A new animation object is created.

See also:
ding0.core.network.grids.MVGridDing0.routing() for details on MVGridDing0 objects routing
ding0.tools.animation.AnimationDing0() for details on animation function.

orm
Getter for the stored ORM configurations.
Returns obj: dict

pf_config
Getter for the power flow calculation configurations.
Returns PFConfigDing0

reinforce_grid()
Performs grid reinforcement measures for all MV and LV grids

run_ding0(session, mv_grid_districts_no=None, debug=False, export_figures=False)
Let DING0 run by shouting at this method (or just call it from NetworkDing0 instance). This method is a wrapper for the main functionality of DING0.
Parameters
- session (sqlalchemy.orm.session.Session) – Database session
- mv_grid_districts_no (list of int objects.) – List of MV grid_districts/stations to be imported (if empty, all grid_districts & stations are imported)
- debug (obj:bool, defaults to False) – If True, information is printed during process
- export_figures (bool, defaults to False) – If True, figures are shown or exported (default path: ~/.ding0/) during run.
Returns msg (obj:str) – Message of invalidity of a grid district
Note: The steps performed in this method are to be kept in the given order since there are hard dependencies between them. Short description of all steps performed:

- **STEP 1:** Import MV Grid Districts and subjacent objects
  - Imports MV Grid Districts, HV-MV stations, Load Areas, LV Grid Districts and MV-LV stations, instantiates and initiates objects.

- **STEP 2:** Import generators
  - Conventional and renewable generators of voltage levels 4..7 are imported and added to corresponding grid.

- **STEP 3:** Parametrize grid
  - Parameters of MV grid are set such as voltage level and cable/line types according to MV Grid District’s characteristics.

- **STEP 4:** Validate MV Grid Districts
  - Tests MV grid districts for validity concerning imported data such as count of Load Areas.

- **STEP 5:** Build LV grids
  - Builds LV grids for every non-aggregated LA in every MV Grid District using model grids.

- **STEP 6:** Build MV grids
  - Builds MV grid by performing a routing on Load Area centres to build ring topology.

- **STEP 7:** Connect MV and LV generators
  - Generators are connected to grids, used approach depends on voltage level.

- **STEP 8:** Set IDs for all branches in MV and LV grids
  - While IDs of imported objects can be derived from dataset’s ID, branches are created in steps 5+6 and need unique IDs (e.g. for PF calculation).

- **STEP 9:** Relocate switch disconnectors in MV grid
  - Switch disconnectors are set during routing process (step 6) according to the load distribution within a ring. After further modifications of the grid within step 6+7 they have to be relocated (note: switch disconnectors are called circuit breakers in DING0 for historical reasons).

- **STEP 10:** Open all switch disconnectors in MV grid
  - Under normal conditions, rings are operated in open state (half-rings). Furthermore, this is required to allow powerflow for MV grid.

- **STEP 11:** Do power flow analysis of MV grid
  - The technically working MV grid created in step 6 was extended by satellite loads and generators. It is finally tested again using powerflow calculation.

- **STEP 12:** Reinforce MV grid
  - MV grid is eventually reinforced pursuant to results from step 11.

**STEP 13:** Close all switch disconnectors in MV grid
The rings are finally closed to hold a complete graph (if the SDs are open, the edges adjacent to a SD will not be exported!)

```python
run_powerflow(session, method='onthefly', export_pypsa=False, debug=False)
```
Performs power flow calculation for all MV grids
Parameters

- **session** (SQLAlchemy session object) – Database session
- **method** (str) – Specify export method If method='db' grid data will be exported to database
  If method='onthefly' grid data will be passed to PyPSA directly (default)
- **export_pypsa** (bool) – If True PyPSA networks will be exported as csv to output/debug/grid/<MV-GRID_NAME>/
- **debug** (bool, defaults to False) – If True, information is printed during process

**set_branch_ids()**

Performs generation and setting of ids of branches for all MV and underlying LV grids.

See also:

`ding0.core.network.grids.MVGridDing0.set_branch_ids()`

**set_circuit_breakers**(debug=False)

Calculates the optimal position of the existing circuit breakers and relocates them within the graph for all MV grids.

Parameters **debug** (bool, defaults to False) – If True, information is printed during process

See also:

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

**static_data**

Getter for the static data

Returns **obj** : dict

**to_dataframe()**

Export grid data to dataframes for statistical analysis.

The export to dataframe is similar to db tables exported by `export_mv_grid_new`.

Returns **pandas.DataFrame** – Pandas Data Frame

See also:

`ding0.core.NetworkDing0.export_mv_grid_new()`

**validate_grid_districts()**

Method to check the validity of the grid districts. MV grid districts are considered valid if:

1. The number of nodes of the graph should be greater than 1
2. All the load areas in the grid district are NOT tagged as aggregated load areas.

Invalid MV grid districts are subsequently deleted from Network.

8.1.3 ding0.flexopt package

8.1.3.1 Submodules

8.1.3.2 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 
  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\(^2\):

---

\(^1\) dena VNS
\(^2\) dena VNS
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

\[
\Delta \tau = \frac{S_{A_{max}}}{U_{nom}} \cdot (R_{kV} \cdot \cos(\phi) + X_{kV} \cdot \sin(\phi))
\]

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.

\[
\Delta \tau = \frac{S_{A_{max}}}{U_{nom}} \cdot (R_{kV} \cdot \cos(\phi) - X_{kV} \cdot \sin(\phi))
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \tau$</td>
<td>Voltage drop/increase at node</td>
</tr>
<tr>
<td>$S_{A_{max}}$</td>
<td>Apparent power</td>
</tr>
<tr>
<td>$R_{kV}$</td>
<td>Short-circuit resistance</td>
</tr>
<tr>
<td>$X_{kV}$</td>
<td>Short-circuit reactance</td>
</tr>
<tr>
<td>$\cos(\phi)$</td>
<td>Power factor</td>
</tr>
<tr>
<td>$U_{nom}$</td>
<td>Nominal voltage</td>
</tr>
</tbody>
</table>

\(^3\) VDE Anwenderrichtlinie: Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz, 2011
**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) – Ding0 grid object
- `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

```python
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 preceding branch
- `x` *(float)* – Reactance of preceding branch

**Returns** `float` – Delta voltage for branch

```python
ding0.flexopt.check_tech_constraints.peak_load_generation_at_node(nodes)
```

Get maximum occurring load 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

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

Estimate voltage 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

**Returns** `float` – Voltage drop or increase

**References**

8.1.1.3.3 ding0.flexopt.reinforce_grid module

```python
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].

\(^4\) VDE Anwenderrichtlinie: Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz, 2011
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:** Currently 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) – Specify 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')

- 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)

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

Choose appropriate cable type for branches with line overloading

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

\(^5\) Ackermann et al. (RP VNS)
• **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 branches 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

```python
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

AI 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.

```python
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 separate 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

```python
>>> 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 branch with particular structure, e.g.:

```python
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

```python
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

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

or

```python
{
    '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.

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

Choose transformer and add to grid’s station

**Parameters** **grid** *(LVGridDing0)* – LV grid data
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 occurring current

8.1.1.4.1.5 ding0.grid.lv_grid.lv_connect module

 ding0.grid.lv_grid.lv_connect.lv_connect_generators(lv_grid_district, 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

8.1. ding0 package
```python
defeat()
    Returns the depot node.

    Returns type – Depot node
distance(i, j)
    Returns the distance between node i and node j

    Parameters
    • i (type) – Desc
    • 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
close()
    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_solution)
    Bases: object

    CVRP route, consists of consecutive nodes
```
Parameters `cvrp_problem` *(type)* – Desc

**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 adjacent 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

technconstraints_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. $2\times10$ 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 Erzeugungsseinheiten, 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) – Descr 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 allocated.
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 search algorithm founds on the following publications\textsuperscript{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 graph 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

\textsuperscript{1} W. Wenger, “Multikriterielle Tourenplanung”, Dissertation, 2009

\textsuperscript{2} M. Kämpf, “Probleme der Tourenbildung”, Chemnitzer Informatik-Berichte, 2006


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 detailed 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). Performs best move (max. saving) and starts over again with new route until no improvement is found.

Parameters
- **graph** (NetworkX Graph Obj) – A NetworkX graph 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 graph 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

```python
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 graph 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 graph 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
```

8.1. ding0 package
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
```

**Note:** It is assummed that the first row of matrix contains all needed headers.
ding0 documentation

**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

**ding0.grid.mv_grid.mv_connect**.connect_node(node, node_shp, mv_grid, target_obj, proj, graph, conn_dist_ring_mod, debug)

Connects node to target_obj.

**Parameters**

- **node (LVLoadAreaCentreDing0, i.e.)** – Origin node - Ding0 graph object (e.g. LVLoadAreaCentreDing0)
- **node_shp (Shapely Point object)** – Shapely Point object of origin node
- **target_obj (type)** – object that node shall be connected to
- **proj (pyproj Proj object)** – equidistant CRS to conformal CRS (e.g. ETRS -> WGS84)
- **graph (NetworkX Graph Obj)** – NetworkX graph object with nodes and newly created branches
- **conn_dist_ring_mod (float)** – Max. distance when nodes are included into route instead of creating a new line.
- **debug (bool)** – If True, information is printed during process.

**Returns**

- **LVLoadAreaCentreDing0** – object that node was connected to.

  (instance of LVLoadAreaCentreDing0 or MVCableDistributorDing0.)

If node is included into line instead of creating a new line (see arg conn_dist_ring_mod), target_obj_result is None.

See also:

**ding0.grid.mv_grid.mv_connect()** for details on the conn_dist_ring_mod parameter.

**ding0.grid.mv_grid.mv_connect**.disconnect_node(node, target_obj_result, graph, debug)

Disconnects node from target_obj

**Parameters**

- **node (LVLoadAreaCentreDing0, i.e.)** – Origin node - Ding0 graph object (e.g. LVLoadAreaCentreDing0)
- **target_obj_result (LVLoadAreaCentreDing0, i.e.)** – Origin node - Ding0 graph object (e.g. LVLoadAreaCentreDing0)
- **graph (NetworkX Graph Obj)** – NetworkX graph object with nodes and newly created branches
• **debug** *(bool)* – If True, information is printed during process

```python
ding0.grid.mv_grid.mv_connect.find_connection_point(node, node_shp, graph, proj, conn_objects_min_stack, conn_dist_ring_mod, debug)
```

Goes through the possible target connection objects in `conn_objects_min_stack` (from nearest to most far object) and tries to connect `node` to one of them.

Function searches from nearest to most far object.

**Parameters**

- **node** *(LVLoadAreaCentreDing0, i.e.)* – Origin node - Ding0 graph object (e.g. LVLoadAreaCentreDing0)
- **node_shp** *(Shapely Point object)* – Shapely Point object of node
- **graph** *(NetworkX Graph Obj)* – NetworkX graph object with nodes
- **proj** *(pyproj Proj object)* – equidistant CRS to conformal CRS (e.g. ETRS -> WGS84)
- **conn_objects_min_stack** *(list)* – List of connection objects. Each object is represented by dict with Ding0 object, shapely object, and distance to node, sorted ascending by distance.
- **conn_dist_ring_mod** *(type)* – Max. distance when nodes are included into route instead of creating a new line.
- **debug** *(bool)* – If True, information is printed during process

See also:

```python
ding0.grid.mv_grid.mv_connect()```

for details on the `conn_dist_ring_mod` parameter.

```python
ding0.grid.mv_grid.mv_connect.find_nearest_conn_objects(node_shp, branches, proj, conn_dist_weight, debug, branches_only=False)
```

Searches all `branches` for the nearest possible connection object per branch.

Picks out 1 object out of 3 possible objects:

- 2 branch-adjacent stations and
- 1 potentially created cable distributor on the line (perpendicular projection)).

The resulting stack (list) is sorted ascending by distance from node.

**Parameters**

- **node_shp** *(Shapely Point object)* – Shapely Point object of node
- **branches** *(BranchDing0)* – BranchDing0 objects of MV region
- **proj** *(pyproj Proj object)* – nodes’ CRS to equidistant CRS (e.g. WGS84 -> ETRS)
- **conn_dist_weight** *(float)* – length weighting to prefer stations instead of direct line connection.
- **debug** *(bool)* – If True, information is printed during process
- **branches_only** *(bool, defaults to False)* – If True, only branch objects are considered as connection objects

**Returns** *list* – List of connection objects. Each object is represented by dict with Ding0 object, shapely object, and distance to node.
See also:

```python
mv_connect_satellites() for details on conn_dist_weight param
```

ding0.grid.mv_grid.mv_connect.
```python
get_lv_load_area_group_from_node_pair(node1, node2)
```

ding0.grid.mv_grid.mv_connect.
```python
mv_connect_generators(mv_grid_district, graph, debug=False)
```

Connect MV generators to MV grid

**Parameters**

- `mv_grid_district` ([MVGridDistrictDing0](#)) – MVGridDistrictDing0 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

```python
ding0.grid.mv_grid.mv_connect.
```
```python
mv_connect_satellites(mv_grid, graph, mode='normal', debug=False)
```

Connect satellites (small Load Areas) to MV grid

**Parameters**

- `mv_grid` ([MVGridDing0](#)) – MV grid instance
- `graph` ([NetworkX Graph Obj](#)) – NetworkX graph object with nodes
- `mode` (str, defaults to ‘normal’) – Specify mode how satellite [LVLoadAreaCentreDing0](#) are connected to the grid. Mode normal (default) considers for restrictions like max. string length, max peak load per string. The mode ‘isolated’ disregards any connection restrictions and connects the node [LVLoadAreaCentreDing0](#) to the next connection point.
- `debug` (bool, defaults to False) – If True, information is printed during process.

**Note:** conn_dist_weight: The satellites can be connected to line (new terminal is created) or to one station where the line ends, depending on the distance from satellite to the objects. This threshold is a length weighting to prefer stations instead of direct line connection to respect grid planning principles.

Example: The distance from satellite to line is 1km, to station1 1.2km, to station2 2km. With conn_dist_threshold=0.75, the ‘virtual’ distance to station1 would be 1.2km * 0.75 = 0.9km, so this conn. point would be preferred.

**Returns** [NetworkX Graph Obj](#) – NetworkX graph object with nodes and newly created branches

```python
ding0.grid.mv_grid.mv_connect.
```
```python
mv_connect_stations(mv_grid_district, graph, debug=False)
```

Connect LV stations to MV grid

**Parameters**

- `mv_grid_district` ([MVGridDistrictDing0](#)) – MVGridDistrictDing0 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

ding0.grid.mv_grid.mv_connect.parametrize_lines(mv_grid)
Set unparametrized branches to default branch type

**Parameters**  

mv_grid (MVGreedingO) – MV grid instance  

---

**Note:** During the connection process of satellites, new branches are created - these have to be parametrized.

### 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. LocalSearch-Solution) (=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 (AnimationDingO) – AnimationDingO object

**Returns**  

NetworkX Graph Obj – NetworkX graph object with nodes and edges

**See also:**

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

### 8.1.1.4.1.31 ding0.grid.mv_grid.tools module

**ding0.grid.mv_grid.tools.set_circuit_breakers(mv_grid, mode='load', debug=False)**  
Calculates the optimal position of a circuit breaker on all routes of mv_grid, adds and connects them to graph.

**Parameters**  

• `mv_grid` (MVGridDing0) – Description#TODO
• `debug` (bool, defaults to False) – If True, information is printed during process

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\(^1\).\(^2\). 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).

If a ring is dominated by loads (peak load > peak capacity of generators), only loads are used for determining the location of circuit breaker. If generators are prevailing (peak load < peak capacity of generators), only generator capacities are considered for relocation.

The core of this function (calculation of the optimal circuit breaker position) is the same as in ding0.grid.mv_grid.models.Route.calc_circuit_breaker_position but here it is 1. applied to a different data type (NetworkX Graph) and it 2. adds circuit breakers to all rings.

The re-location of circuit breakers is necessary because the original position (calculated during routing with method mentioned above) shifts during the connection of satellites and therefore it is no longer valid.

References

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


8.1.4.4 Module contents

8.1.5 ding0.tools package

8.1.5.1 Submodules

8.1.5.2 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

8.1.5.3 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

(continues on next page)
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 loaded from disk with hard-coded file name.
- mode ('write' or 'compare') –

Returns –
8.1.1.5.5 ding0.tools.geo module

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

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)

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)

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_matrix_vincenty**(nodes_pos)

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 vincenty function (formula devised by Thaddeus Vincenty, with an accurate ellipsoidal model of the earth). As default ellipsoidal model of the earth WGS-84 is used. For more options see

https://geopy.readthedocs.org/en/1.10.0/index.html?highlight=vincenty#geopy.distance.vincenty
Parameters **nodes_pos** (*dict*) – dictionary of nodes with positions, with x=longitude, y=latitude, and the following format:

```python
{
    '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:

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

ding0.tools.geo.calc_geo_dist_vincenty(*node_source*, *node_target*)

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

Returns float – Distance in m

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*, *loglevel=10*)

Instantiate logger

Parameters

• **log_dir** (*str*) – Directory to save log, default: ~/.ding0/logging/

• **loglevel** – Level of logger.
8.1.1.5.7 ding0.tools.plots module

ding0.tools.plots.plot_mv_topology(grid, subtitle=", subtitle=None, filename=None, testcase='load', line_color=None, node_color='type', limits_cb_lines=None, limits_cb_nodes=None, background_map=True)

Draws MV grid graph using networkx

Parameters

• grid (MVGridDing0) – MV grid to plot.

• subtitle (str) – Extend plot’s title by this string.

• filename (str) – If provided, the figure will be saved and not displayed (default path: ~/.ding0/). A prefix is added to the file name.

• testcase (str) – Defines which case is to be used. Refer to config_calc.cfg to see further assumptions for the cases. Possible options are:
  – 'load' (default) Heavy-load-flow case
  – 'feedin' Feedin-case

• line_color (str) – Defines whereby to choose line colors. Possible options are:
  – 'loading' Line color is set according to loading of the line in heavy load case. You can use parameter limits_cb_lines to adjust the color range.
  – None (default) Lines are plotted in black. Is also the fallback option in case of wrong input.

• node_color (str) – Defines whereby to choose node colors. Possible options are:
  – 'type' (default) Node color as well as size is set according to type of node (generator, MV station, etc.). Is also the fallback option in case of wrong input.
  – 'voltage' Node color is set according to voltage deviation from 1 p.u.. You can use parameter limits_cb_nodes to adjust the color range.

• limits_cb_lines (tuple) – Tuple with limits for colorbar of line color. First entry is the minimum and second entry the maximum value. E.g. pass (0, 1) to adjust the colorbar to 0..100% loading. Default: None (min and max loading are used).

• limits_cb_nodes (tuple) – Tuple with limits for colorbar of nodes. First entry is the minimum and second entry the maximum value. E.g. pass (0.9, 1) to adjust the colorbar to 90%..100% voltage. Default: None (min and max loading are used).

• background_map (bool, optional) – If True, a background map is plotted (default: stamen toner light). The additional package contextily is needed for this functionality. Default: True

Note: WGS84 pseudo mercator (epsg:3857) is used as coordinate reference system (CRS). Therefore, the drawn graph representation may be falsified!

8.1.1.5.8 ding0.tools.pypsa_io module

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

Write results obtained from PF to graph

8.1. ding0 package
Parameters

- `grid (ding0.network)` –
- `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 (ding0.network)` –
- `line_data (pandas.DataFrame)` – DataFrame containing active/reactive at nodes obtained from PF analysis

`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(grid, edges)`

Export edges to DataFrame

Parameters

- `grid (ding0.Network)` –
- `edges (list)` – Edges of Ding0.Network graph

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.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 powerflow calculation

`ding0.tools.pypsa_io.nodes_to_dict_of_dataframes(grid, nodes, lv_transformer=True)`

Creates dictionary of dataframes containing grid

Parameters

- `grid (ding0.Network)` –
• **nodes** *(list of ding0 grid components objects)* – Nodes of the grid graph

• **lv_transformer** *(bool, True)* – Toggle transformer representation in power flow analysis

• **Returns** –

• **components** *(dict of pandas.DataFrame)* – DataFrames contain components attributes. Dict is keyed by components type

• **components_data** *(dict of pandas.DataFrame)* – DataFrame containing components time-varying data

```python
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

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

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)* –

• **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.

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

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

series: Pandas DataFrame

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

### 8.1.1.5.9 ding0.tools.results module

```python
ding0.tools.results.calculate_lvgd_stats(nw)
```

LV Statistics for an arbitrary network

**Parameters**

nw *(list of NetworkDing0)* – The MV grid(s) to be studied

**Returns** lvgd_stats *(pandas.DataFrame)* – Dataframe containing several statistical numbers about the LVGD

```python
ding0.tools.results.calculate_lvgd_voltage_current_stats(nw)
```

LV Voltage and Current Statistics for an arbitrary network

**Note:** Aggregated Load Areas are excluded.
ding0 Documentation

Parameters **nw** *(list of NetworkDing0)* – The MV grid(s) to be studied

Returns

- **pandas.DataFrame** – `nodes_df` : Dataframe containing voltage, respectively current, statistics for every critical node, resp. every critical station, in every LV grid in `nw`.
- **pandas.DataFrame** – `lines_df` : Dataframe containing current statistics for every critical line, in every LV grid in `nw`.

**ding0.tools.results.calculate_mvgd_stats**(nw)

MV Statistics for an arbitrary network

Parameters **nw** *(list of NetworkDing0)* – The MV grid(s) to be studied

Returns **mvgd_stats** *(pandas.DataFrame)* – Dataframe containing several statistical numbers about the MVGD

**ding0.tools.results.calculate_mvgd_voltage_current_stats**(nw)

MV Voltage and Current Statistics for an arbitrary network

Parameters **nw** *(list of NetworkDing0)* – The MV grid(s) to be studied

Returns

- **pandas.DataFrame** – `nodes_df` : Dataframe containing voltage statistics for every node in the MVGD
- **pandas.DataFrame** – `lines_df` : Dataframe containing voltage statistics for every edge in the MVGD

**ding0.tools.results.concat_nd_pickles**(self, mv_grid_districts)

Read multiple pickles, join nd objects and save to file

Parameters **mv_grid_districts** *(list)* – Ints describing MV grid districts

**ding0.tools.results.create_ding0_db_tables**(engine)

**ding0.tools.results.drop_ding0_db_tables**(engine)

**ding0.tools.results.export_data_to_oedb**(session, srid, lv_grid, lv_gen, lv_cd, mvlv_stations, mvlv_trafos, lv_loads, mv_grid, mv_gen, mv_cd, hvmv_stations, hvmv_trafos, mv_loads, lines, mvlv_mapping)

**ding0.tools.results.export_data_tocsv**(path, run_id, lv_grid, lv_gen, lv_cd, lv_stations, mvlv_trafos, lv_loads, mv_grid, mv_gen, mv_cb, mv_cd, mv_stations, hvmv_trafos, mv_loads, lines, mapping)

**ding0.tools.results.export_network**(nw, mode="")

Export all nodes and lines of the network nw as DataFrames

Parameters

- **nw** *(list of NetworkDing0)* – The MV grid(s) to be studied
- **mode** *(str)* – If ‘MV’ export only medium voltage nodes and lines If ‘LV’ export only low voltage nodes and lines else, exports MV and LV nodes and lines

Returns

- **pandas.DataFrame** – `nodes_df` : Dataframe containing nodes and its attributes
- **pandas.DataFrame** – `lines_df` : Dataframe containing lines and its attributes
**ding0 tools.results.export_network_to_oedb** *(session, table, tabletype, srid)*

**ding0 tools.results.init_mv_grid** *(mv_grid_districts=[3545], file_name='ding0_tests_grids_1.pkl')*

Runs ding0 over the districts selected in `mv_grid_districts`

It also writes the result in `filename`. If `filename = False`, then the network is not saved.

**Parameters**

- **mv_grid_districts** *(list of int)* – Districts IDs: Defaults to [3545]
- **filename** *(str)* – Defaults to ‘ding0_tests_grids_1.pkl’ If `filename=False`, then the network is not saved

**Returns** *NetworkDing0* – The created MV network.

**ding0 tools.results.load_nd_from_pickle** *(filename=None, path=“”)*

Use pickle to save the whole nd-object to disc

**Parameters**

- **filename** *(str)* – Filename of nd pickle
- **path** *(str)* – Absolute or relative path where pickle should be saved. Default is ‘’ which means pickle is save to PWD

**Returns** *nd* *(NetworkDing0)* – Ding0 grid container object

**ding0 tools.results.lv_grid_generators_bus_bar** *(nd)*

Calculate statistics about generators at bus bar in LV grids

**Parameters** *nd* *(ding0.NetworkDing0)* – Network container object

**Returns** *lv_stats* *(dict)* – Dict with keys of LV grid repr() on first level. Each of the grids has a set of statistical information about its topology

**ding0 tools.results.parallel_running_stats** *(districts_list, n_of_processes, n_of_districts=1, source=’pkl’, mode=”, critical=False, save_csv=False, save_path=”)*

Organize parallel runs of ding0 to calculate stats

The function take all districts in a list and divide them into `n_of_processes` parallel processes. For each process, the assigned districts are given to the function `process_runs()` with arguments `n_of_districts`, `source`, `mode`, and `critical`

**Parameters**

- **districts_list** *(list of int)* – List with all districts to be run.
- **n_of_processes** *(int)* – Number of processes to run in parallel
- **n_of_districts** *(int)* – Number of districts to be run in each cluster given as argument to `process_stats()`
- **source** *(str)* – If ‘pkl’, pickle files are read. Otherwise, ding0 is run over the districts.
- **mode** *(str)* – If ‘MV’, medium voltage stats are calculated. If ‘LV’, low voltage stats are calculated. If empty, medium and low voltage stats are calculated.
- **critical** *(bool)* – If True, critical nodes and branches are returned
- **path** *(str)* – path to save the pkl and csv files

**Returns**
• **DataFrame** – mv_stats: MV stats in a DataFrame. If mode=='LV', then DataFrame is empty.
• **DataFrame** – lv_stats: LV stats in a DataFrame. If mode=='MV', then DataFrame is empty.
• **DataFrame** – mv_crit_nodes: MV critical nodes stats in a DataFrame. If mode=='LV', then DataFrame is empty. If critical==False, then DataFrame is empty.
• **DataFrame** – mv_crit_edges: MV critical edges stats in a DataFrame. If mode=='LV', then DataFrame is empty. If critical==False, then DataFrame is empty.
• **DataFrame** – lv_crit_nodes: LV critical nodes stats in a DataFrame. If mode=='MV', then DataFrame is empty. If critical==False, then DataFrame is empty.
• **DataFrame** – lv_crit_edges: LV critical edges stats in a DataFrame. If mode=='MV', then DataFrame is empty. If critical==False, then DataFrame is empty.

See also:

`process_stats()`

ding0.tools.results.plot_cable_length(stats, plotpath)
Cable length per MV grid district

ding0.tools.results.plot_generation_over_load(stats, plotpath)
Plot of generation over load

ding0.tools.results.plot_km_cable_vs_line(stats, plotpath)

Parameters

• `stats` –
• `plotpath` –

Returns

`process_stats()`

Generates stats dataframes for districts in mv_districts.

If source=='ding0', then runned districts are saved to a pickle named filename+str(n_of_districts[0])+'_to_'+str(n_of_districts[-1])+'.pkl'

Parameters

• `districts_list` (list of int) – List with all districts to be run.
• `n_of_districts` (int) – Number of districts to be run in each cluster
• `source` (str) – If ‘pkl’, pickle files are read. If ‘ding0’, ding0 is run over the districts.
• `mode` (str) – If ‘MV’, medium voltage stats are calculated. If ‘LV’, low voltage stats are calculated. If empty, medium and low voltage stats are calculated.
• `critical` (bool) – If True, critical nodes and branches are returned
• `filename` (str) – filename prefix for saving pickles
• `output` – outer variable where the output is stored as a tuple of 6 lists:

* mv_stats: MV stats DataFrames.
  If mode=='LV', then DataFrame is empty.

* lv_stats: LV stats DataFrames.
  If mode=='MV', then DataFrame is empty.
* **mv_crit_nodes**: MV critical nodes stats DataFrames.
  If `mode=='LV'`, then DataFrame is empty.
  If `critical==False`, then DataFrame is empty.

* **mv_crit_edges**: MV critical edges stats DataFrames.
  If `mode=='LV'`, then DataFrame is empty.
  If `critical==False`, then DataFrame is empty.

* **lv_crit_nodes**: LV critical nodes stats DataFrames.
  If `mode=='MV'`, then DataFrame is empty.
  If `critical==False`, then DataFrame is empty.

* **lv_crit_edges**: LV critical edges stats DataFrames.
  If `mode=='MV'`, then DataFrame is empty.
  If `critical==False`, then DataFrame is empty.

```python
ding0.tools.results.save_nd_to_pickle(nd, path=' ', filename=None)
```

Use pickle to save the whole nd-object to disc

The network instance is entirely pickled to a file.

**Parameters**

- `nd (NetworkDing0)`: Ding0 grid container object
- `path (str)`: Absolute or relative path where pickle should be saved. Default is `'` which means pickle is save to PWD

### 8.1.1.5.10 ding0.tools.tests module

```python
class ding0.tools.tests.Ding0RunTest (methodName='runTest')
    Bases: unittest2.case.TestCase

    setUp()
        Hook method for setting up the test fixture before exercising it.

    test_ding0()
    test_ding0_file()
    test_files()
```

```python
ding0.tools.tests.dataframe_equal (network_one, network_two)
```

Compare two networks and returns True if they are identical

**Parameters**

- `network_one (GridDing0)`
- `network_two (GridDing0)`

**Returns**

- `bool` – True if both networks are identical, False otherwise.
- `str` – A message explaining the result.

```python
ding0.tools.tests.init_files_for_tests (mv_grid_districts=[3545],
    filename='ding0_tests_grids_1.pkl')
```

Runs ding0 over the districtis selected in `mv_grid_districts` and writes the result in `filename`.

### 8.1. ding0 package
Parameters

- **mv_grid_districts** (list of int) – Districts IDs: Defaults to [3545]
- **filename** (str) – Defaults to ‘ding0_tests_grids_1.pkl’

**ding0.tools.tests**.**manual_ding0_test**(mv_grid_districts=[3545], file-name=’ding0_tests_grids_1.pkl’)

Compares a new run of ding0 over districts and an old one saved in filename.

Parameters

- **mv_grid_districts** (list of int) – Districts IDs: Defaults to [3545]
- **filename** (str) – Defaults to ‘ding0_tests_grids_1.pkl’

### 8.1.5.11 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](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

### 8.1.1.5.12 ding0.tools.validation module

ding0.tools.validation.compare_grid_impedances *(nw1, nw2)*

Compare if two grids have the same impedances.

**Parameters**

- **nw1** – Network 1
- **nw2** – Network 2

**Returns** *Bool* – True if network elements have same impedances.

ding0.tools.validation.get_line_and_trafo_dict *(nw)*

Get dictionaries of line and transformer data (in order to compare two networks)

**Parameters** **nw** – Network

**Returns**

- *Dictionary* – mv_branches_dict
- *Dictionary* – lv_branches_dict
- *Dictionary* – lv_transformer_dict

ding0.tools.validation.validate_generation *(session, nw)*

Validate if total generation of a grid in a pkl file is what expected.

**Parameters**

- **session** *(sqlalchemy.orm.session.Session)* – Database session
- **nw** – The network

**Returns**

- *DataFrame* – compare_by_level
- *DataFrame* – compare_by_type
ding0 Documentation

```
ding0.tools.validation.validate_load_areas(session, nw)
Validate if total load of a grid in a pkl file is what expected from load areas

Parameters
    • session (sqlalchemy.orm.session.Session) – Database session
    • nw – The network

Returns
    • DataFrame – compare_by_la
    • Bool – True if database IDs of LAs are the same as the IDs in the grid

ding0.tools.validation.validate_lv_districts(session, nw)
Validate if total load of a grid in a pkl file is what expected from LV districts

Parameters
    • session (sqlalchemy.orm.session.Session) – Database session
    • nw – The network

Returns
    • DataFrame – compare_by_district
    • DataFrame – compare_by_loads
```

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) – NumPy 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


[OSM] OpenStreetMap contributors: *Open street map*, 2017


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


Python Module Index

d
- ding0, 112
- ding0.config, 40
- ding0.config.config_db_interfaces, 37
- ding0.core, 63
- ding0.core.network, 47
- ding0.core.network.cable_distributors, 40
- ding0.core.network.grids, 40
- ding0.core.network.loads, 46
- ding0.core.network.stations, 46
- ding0.core.network.transformers, 47
- ding0.core.powerflow, 58
- ding0.core.structure, 62
- ding0.core.structure.groups, 59
- ding0.core.structure.regions, 60
- ding0.flexopt, 78
- ding0.flexopt.check_tech_constraints, 70
- ding0.flexopt.reinforce_grid, 74
- ding0.flexopt.reinforce_measures, 75
- ding0.flexopt.reinforce_measures_dena, 77
- ding0.grid, 99
- ding0.grid.lv_grid, 81
- ding0.grid.lv_grid.build_grid, 78
- ding0.grid.lv_grid.check, 81
- ding0.grid.lv_grid.lv_connect, 81
- ding0.grid.mv_grid, 98
- ding0.grid.mv_grid.models, 85
- ding0.grid.mv_grid.models.models, 81
- ding0.grid.mv_grid.mv_connect, 94
- ding0.grid.mv_grid.mv_routing, 97
- ding0.grid.mv_grid.solvers, 92
- ding0.grid.mv_grid.solvers.base, 85
- ding0.grid.mv_grid.solvers.savings, 91
- ding0.grid.mv_grid.tools, 112
- ding0.grid.mv_grid.tests.run_test_case, 92
- ding0.grid.mv_grid.tools, 97
- ding0.grid.mv_grid.util, 94
- ding0.grid.mv_grid.util.data_input, 92
- ding0.grid.mv_grid.util.util, 93
- ding0.grid.tools, 98
- ding0.tools, 112
- ding0.tools.animation, 99
- ding0.tools.config, 99
- ding0.tools.debug, 100
- ding0.tools.geo, 101
- ding0.tools.logger, 102
- ding0.tools.plots, 103
- ding0.tools.pypsa_io, 103
- ding0.tools.results, 105
- ding0.tools.tests, 109
- ding0.tools.tools, 110
- ding0.tools.validation, 111
- ding0.tools.write_openego_header, 112
### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_weather_cell_id</td>
<td>(ding0.core.network.GeneratorFluctuatingDing0 attribute), 51</td>
<td></td>
</tr>
<tr>
<td>absolute_file_paths()</td>
<td>(in module ding0.tools.write_openego_header), 112</td>
<td></td>
</tr>
<tr>
<td>adapt_numpy_int64()</td>
<td>(in module ding0), 112</td>
<td></td>
</tr>
<tr>
<td>add_aggregated_peak_demand()</td>
<td>(ding0.core.structure.regions.MVGridDistrictDing0 method), 62</td>
<td></td>
</tr>
<tr>
<td>add_cable_dist()</td>
<td>(ding0.core.network.grids.LVGridDing0 method), 41</td>
<td></td>
</tr>
<tr>
<td>add_cable_distributor()</td>
<td>(ding0.core.network.grids.MVGridDing0 method), 41</td>
<td></td>
</tr>
<tr>
<td>add_circuit_breaker()</td>
<td>(ding0.core.network.grids.MVGridDing0 method), 41</td>
<td></td>
</tr>
<tr>
<td>add_generator()</td>
<td>(ding0.core.network.GridDing0 method), 52</td>
<td></td>
</tr>
<tr>
<td>add_load()</td>
<td>(ding0.core.network.grids.LVGridDing0 method), 41</td>
<td></td>
</tr>
<tr>
<td>add_load()</td>
<td>(ding0.core.network.grids.MVGridDing0 method), 42</td>
<td></td>
</tr>
<tr>
<td>add_lv_grid_district()</td>
<td>(ding0.core.structure.regions.LVLoadAreaDing0 method), 61</td>
<td></td>
</tr>
<tr>
<td>add_lv_load_area()</td>
<td>(ding0.core.structure.groups.LoadAreaGroupDing0 method), 59</td>
<td></td>
</tr>
<tr>
<td>add_lv_load_area()</td>
<td>(ding0.core.structure.regions.MVGridDistrictDing0 method), 62</td>
<td></td>
</tr>
<tr>
<td>add_lv_load_area_group()</td>
<td>(ding0.core.structure.regions.MVGridDistrictDing0 method), 62</td>
<td></td>
</tr>
<tr>
<td>add_mv_grid_district()</td>
<td>(ding0.core.NetworkDing0 method), 64</td>
<td></td>
</tr>
<tr>
<td>add_peak_demand()</td>
<td>(ding0.core.structure.regions.MVGridDistrictDing0 method), 62</td>
<td></td>
</tr>
<tr>
<td>add_ring()</td>
<td>(ding0.core.network.grids.MVGridDing0 method), 42</td>
<td></td>
</tr>
<tr>
<td>add_station()</td>
<td>(ding0.core.network.grids.LVGridDing0 method), 41</td>
<td></td>
</tr>
<tr>
<td>add_station()</td>
<td>(ding0.core.network.grids.MVGridDing0 method), 42</td>
<td></td>
</tr>
<tr>
<td>add_transformer()</td>
<td>(ding0.core.network.StationDing0 method), 56</td>
<td></td>
</tr>
<tr>
<td>allocate()</td>
<td>(ding0.grid.mv_grid.models.models.Route method), 83</td>
<td></td>
</tr>
<tr>
<td>AnimationDing0</td>
<td>(class in ding0.tools.animation), 99</td>
<td></td>
</tr>
<tr>
<td>assign_bus_results()</td>
<td>(in module ding0.tools.pypsa_io), 103</td>
<td></td>
</tr>
<tr>
<td>assign_line_results()</td>
<td>(in module ding0.tools.pypsa_io), 104</td>
<td></td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Function</th>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute_file_paths()</td>
<td>(in module ding0.tools.write_openego_header), 112</td>
<td></td>
</tr>
<tr>
<td>adapt_numpy_int64()</td>
<td>(in module ding0), 112</td>
<td></td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Class</th>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseSolution</td>
<td>(class in ding0.grid.mv_grid.solvers.base), 85</td>
<td></td>
</tr>
<tr>
<td>BaseSolver</td>
<td>(class in ding0.grid.mv_grid.solvers.base), 86</td>
<td></td>
</tr>
<tr>
<td>benchmark_operator_order()</td>
<td>(ding0.grid.mv_grid.solvers.local_search.LocalSearchSolver method), 87</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>(ding0.core.network.CircuitBreakerDing0 attribute), 49</td>
<td></td>
</tr>
<tr>
<td>branch_id</td>
<td>(ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38</td>
<td></td>
</tr>
<tr>
<td>branch_no</td>
<td>(ding0.core.network.cable_distributors.LVCableDistributorDing0 attribute), 40</td>
<td></td>
</tr>
<tr>
<td>branch_nodes</td>
<td>(ding0.core.network.CircuitBreakerDing0 attribute), 49</td>
<td></td>
</tr>
<tr>
<td>BranchDing0</td>
<td>(class in ding0.core.network), 47</td>
<td></td>
</tr>
<tr>
<td>branches()</td>
<td>(ding0.core.network.RingDing0 method), 56</td>
<td></td>
</tr>
<tr>
<td>build_grid()</td>
<td>(ding0.core.network.grids.LVGridDing0 method), 41</td>
<td></td>
</tr>
</tbody>
</table>
build_lv_graph_residential() (in module ding0.grid.lv_grid.build_grid), 78
build_lv_graph_ria() (in module ding0.grid.lv_grid.build_grid), 78
build_lv_grid_district() (ding0.core.NetworkDing0 method), 64
build_lv_grids() (ding0.core.NetworkDing0 method), 64
build_mv_grid_district() (ding0.core.NetworkDing0 method), 64
build_residential_branches() (in module ding0.grid.lv_grid.build_grid), 79
build_ret_ind_agr_branches() (in module ding0.grid.lv_grid.build_grid), 79
cable_distributors (ding0.core.network.GridDing0 attribute), 52
cable_distributors() (ding0.core.network.GridDing0 method), 52
cable_distributors_count() (ding0.core.network.GridDing0 method), 52
cable_type() (in module ding0.grid.tools), 98
CableDistributorDing0 (class in ding0.core.network), 48
calc_circuit_breaker_position() (ding0.grid.mv_grid.models.models.Route method), 83
calc_geo_branches_in_buffer() (in module ding0.tools.geo), 101
calc_geo_branches_in_polygon() (in module ding0.tools.geo), 101
calc_geo_centre_point() (in module ding0.tools.geo), 101
calc_geo_dist_matrix_vincenty() (in module ding0.tools.geo), 101
calc_geo_dist_vincenty() (in module ding0.tools.geo), 101
calculate_euc_distance() (in module ding0.grid.mv_grid.util.data_input), 102
calculate_lvgd_stats() (in module ding0.tools.results), 105
calculate_lvgd_voltage_current_stats() (in module ding0.tools.results), 105
calculate_mvvdg_stats() (in module ding0.tools.results), 106
calculate_mvvdg_voltage_current_stats() (in module ding0.tools.results), 106
can_add_lv_load_area() (ding0.core.structure.groups.LoadAreaGroupDing0 method), 59
can_allocate() (ding0.grid.mv_grid.models.models.Route method), 83
can_process() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 85
can_process() (ding0.grid.mv_grid.solvers.savings.SavingsSolution method), 91
capacity (ding0.core.network.GeneratorDing0 attribute), 50
capacity_factor (ding0.core.network.GeneratorDing0 attribute), 50
check_load() (in module ding0.flexopt.check_tech_constraints), 70
check_voltage() (in module ding0.flexopt.check_tech_constraints), 71
circuit_breaker (ding0.core.network.BranchDing0 attribute), 48
circuit_breakers() (ding0.core.network.grids.MVGridDing0 method), 42
circuit_breakers_count() (ding0.core.network.grids.MVGridDing0 method), 42
CircuitBreakerDing0 (class in ding0.core.network), 48
ClarkeWrightSolver (class in ding0.grid.mv_grid.solvers.savings), 91
close() (ding0.grid.mv_grid.models.models.Node method), 82
close() (ding0.grid.mv_grid.models.models.Route method), 83
close() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 85
close() (ding0.grid.mv_grid.solvers.local_search.LocalSearchSolution method), 86
close() (ding0.grid.mv_grid.solvers.savings.SavingsSolution method), 91
close() (ding0.core.network.CircuitBreakerDing0 method), 49
close_circuit_breakers() (ding0.core.network.grids.MVGridDing0 method), 42
calculate_grid_impedances() (in module ding0.tools.validation), 111
compute_savings_list() (ding0.grid.mv_grid.solvers.savings.ClarkeWrightSolver method), 91
concat_nd_pickles() (in module ding0.tools.results), 106
compare_graphs() (in module ding0.tools.debug), 100
compare_grid_impedances() (in module ding0.tools.validation), 111
concat_nd_pickles() (in module ding0.tools.results), 106
can_add_lv_load_area() (ding0.core.structure.groups.LoadAreaGroupDing0 method), 59
can_allocate() (ding0.grid.mv_grid.models.models.Route method), 83
can_process() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 85
can_process() (ding0.grid.mv_grid.solvers.savings.SavingsSolution method), 91
capacity (ding0.core.network.GeneratorDing0 attribute), 50
capacity_factor (ding0.core.network.GeneratorDing0 attribute), 50
check_load() (in module ding0.flexopt.check_tech_constraints), 70
check_voltage() (in module ding0.flexopt.check_tech_constraints), 71
circuit_breaker (ding0.core.network.BranchDing0 attribute), 48
circuit_breakers() (ding0.core.network.grids.MVGridDing0 method), 42
circuit_breakers_count() (ding0.core.network.grids.MVGridDing0 method), 42
CircuitBreakerDing0 (class in ding0.core.network), 48
ClarkeWrightSolver (class in ding0.grid.mv_grid.solvers.savings), 91
close() (ding0.grid.mv_grid.models.models.Node method), 82
close() (ding0.grid.mv_grid.models.models.Route method), 83
close() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 85
close() (ding0.grid.mv_grid.solvers.local_search.LocalSearchSolution method), 86
close() (ding0.grid.mv_grid.solvers.savings.SavingsSolution method), 91
close() (ding0.core.network.CircuitBreakerDing0 method), 49
close_circuit_breakers() (ding0.core.network.grids.MVGridDing0 method), 42
calculate_grid_impedances() (in module ding0.tools.validation), 111
compute_savings_list() (ding0.grid.mv_grid.solvers.savings.ClarkeWrightSolver method), 91
concat_nd_pickles() (in module ding0.tools.results), 106
can_add_lv_load_area() (ding0.core.structure.groups.LoadAreaGroupDing0 method), 59
index
connect_generators()
(ding0.core.network.grids.MVGridDing0
(method), 42)
connect_generators() (ding0.core.NetworkDing0
(method), 64)
connect_node() (in module
ding0.grid.mv_grid.mv_connect), 94
connects_aggregated
(ding0.core.network.BranchDing0
attribute), 48
control_circuit_breakers()
(ding0.core.NetworkDing0
(method), 64)
control_generators()
(ding0.core.network.GridDing0
method), 52
create_ding0_db_tables() (in module
ding0.tools.results), 106
create_dir() (in module ding0.tools.logger), 102
create_home_dir() (in module ding0.tools.logger), 102
create_poly_from_source() (in module
ding0.tools.pypsa_io), 104
critical (ding0.core.network.BranchDing0
attribute), 48

data_integrity() (in module
ding0.tools.pypsa_io), 104
dataframe_equal() (in module ding0.tools.tests), 109
db_data (ding0.core.structure.regions.LVLoadAreaDing0
attribute), 61
deallocate() (ding0.grid.mv_grid.models.models.Route
method), 83
demand() (ding0.grid.mv_grid.models.models.Node
method), 82
demand() (ding0.grid.mv_grid.models.models.Route
method), 83
depot() (ding0.grid.mv_grid.models.models.Graph
method), 81
ding0 (module), 112
ding0.config (module), 40
ding0.config.config_db_interfaces (mod-ule), 37
ding0.core (module), 63
ding0.core.network (module), 47
ding0.core.network.cable_distributors
(module), 40
ding0.core.network.grids (module), 40
ding0.core.network.loads (module), 46
ding0.core.network.stations (module), 46
ding0.core.network.transformers (module), 47
ding0.core.powerflow (module), 58
ding0.core.structure (module), 62
ding0.core.structure.groups (module), 59
ding0.core.structure.regions (module), 60
ding0.flexopt (module), 78
ding0.flexopt.check_tech_constraints
(module), 70
ding0.flexopt.reinforce_grid (module), 74
ding0.flexopt.reinforce_measures (mod-ule), 75
ding0.flexopt.reinforce_measures_dena
(module), 77
ding0.grid (module), 99
ding0.grid.lv_grid (module), 81
ding0.grid.lv_grid.build_grid (module), 78
ding0.grid.lv_grid.check (module), 81
ding0.grid.lv_grid.lv_connect (module), 81
ding0.grid.mv_grid (module), 98
ding0.grid.mv_grid.models (module), 85
ding0.grid.mv_grid.models.models (mod-ule), 81
ding0.grid.mv_grid.mv_connect (module), 94
ding0.grid.mv_grid.mv_routing (module), 97
ding0.grid.mv_grid.solvers (module), 92
ding0.grid.mv_grid.solvers.base (module), 85
ding0.grid.mv_grid.solvers.local_search
(module), 86
ding0.grid.mv_grid.solvers.savings (mod-ule), 91
ding0.grid.mv_grid_tests (module), 92
ding0.grid.mv_grid_tests.run_test_case
(module), 92
ding0.grid.mv_grid.tools (module), 97
ding0.grid.mv_grid.util (module), 94
ding0.grid.mv_grid.util.data_input (mod-ule), 92
ding0.grid.mv_grid.util.util (module), 93
ding0.grid.tools (module), 98
ding0.tools (module), 112
ding0.tools.animation (module), 99
ding0.tools.config (module), 99
ding0.tools.debug (module), 100
ding0.tools.geo (module), 101
ding0.tools.logger (module), 102
ding0.tools.plots (module), 103
ding0.tools.pypsa_io (module), 103
ding0.tools.results (module), 105
ding0.tools.tests (module), 109
ding0.tools.tools (module), 110
ding0.tools.validation (module), 111
ding0.tools.write_openego_header (module), 112

Ding0RunTest (class in ding0.tools.tests), 109
disconnect_node() (in module ding0.grid.mv_grid.mv_connect), 94
distance() (ding0.grid.mv_grid.models.models.Graph method), 82
draw_network() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 85
drop_ding0_db_tables() (in module ding0.tools.results), 106

E
edges() (ding0.grid.mv_grid.models.models.Graph method), 82
edges_to_dict_of_dataframes() (in module ding0.tools.pypsa_io), 104
export_data_to_oedb() (in module ding0.tools.results), 106
export_data_tocsv() (in module ding0.tools.results), 106
export_mv_grid() (ding0.core.NetworkDing0 method), 64
export_mv_grid_new() (ding0.core.NetworkDing0 method), 65
export_network() (in module ding0.tools.results), 106
export_network_to_oedb() (in module ding0.tools.results), 106
export_to_dir() (in module ding0.tools.pypsa_io), 104
export_to_pypsa() (ding0.core.network.grids.MVGridDing0 method), 42
extend_substation() (in module ding0.flexopt.reinforce_measures), 75
extend_substation() (in module ding0.flexopt.reinforce_measures_dena), 77
extend_substation_voltage() (in module ding0.flexopt.reinforce_measures), 75
extend_trafo_power() (in module ding0.flexopt.reinforce_measures), 75

F
find_and_union_paths() (ding0.core.network.GridDing0 method), 53
find_connection_point() (in module ding0.grid.mv_grid.mv_connect), 95
find_nearest_conn_objects() (in module ding0.grid.mv_grid.mv_connect), 95
find_path() (ding0.core.network.GridDing0 method), 53

G
GeneratorDing0 (class in ding0.core.network), 49
GeneratorFluctuatingDing0 (class in ding0.core.network), 51
generators (ding0.core.network.GridDing0 attribute), 52
generators() (ding0.core.network.GridDing0 method), 53
geo_data (ding0.core.network.CableDistributorDing0 attribute), 48
geo_data (ding0.core.network.CircuitBreakerDing0 attribute), 49
geo_data (ding0.core.network.GeneratorDing0 attribute), 50
geo_data (ding0.core.network.LoadDing0 attribute), 55
geo_data (ding0.core.structure.regions.MVGridDistrictDing0 attribute), 62
geom (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38
geom (ding0.config.config_db_interfaces.sqla_mv_grid_viz_nodes attribute), 39
geom_lv_load_area_centres (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 37
geom_lv_stations (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 37
geom_mv_cable_dists (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 37
geom_mv_circuit_breakers (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 37
geom_mv_generators (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 38
geom_mv_lines (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 38
geom_mv_station (ding0.config.config_db_interfaces.sqla_mv_grid_viz attribute), 38
get() (in module ding0.tools.config), 100
get_branches() (in module ding0.grid.lv_grid.check), 81
get_cart_dest_point() (in module ding0.tools.tools), 110
get_critical_line_loading() (in module ding0.flexopt.check_tech_constraints), 72
get_critical_voltage_at_nodes() (in module ding0.flexopt.check_tech_constraints), 72
get_cumulated_conn_gen_load() (in module ding0.flexopt.check_tech_constraints), 73
parallel_running_stats() (in module ding0.tools.results), 107
parametrize_grid() (ding0.core.network.grids.MVGridDing0 method), 43
parametrize_lines() (in module ding0.grid.mv_grid.mv_connect), 97
ParseException, 92
peak_generation (ding0.core.network.stations.LVStationDing0 attribute), 46
peak_generation (ding0.core.structure.regions.LVLoadAreaDing0 attribute), 61
peak_generation() (ding0.core.network.stations.MVStationDing0 method), 46
peak_load (ding0.core.network.LoadDing0 attribute), 56
peak_load (ding0.core.network.StationDing0 attribute), 57
peak_load (ding0.core.structure.regions.MVGridDistrictDing0 attribute), 62
peak_load_aggregated (ding0.core.structure.regions.MVGridDistrictDing0 attribute), 62
peak_load_generation_at_node() (in module ding0.flexopt.check_tech_constraints), 74
peak_load_satellites (ding0.core.structure.regions.MVGridDistrictDing0 attribute), 62
pf_config (ding0.core.NetworkDing0 attribute), 63, 68
PFConfigDing0 (class in ding0.core.powerflow), 58
phase_angle (ding0.core.network.TransformerDing0 attribute), 57
plot_cable_length() (in module ding0.tools.results), 108
plot_generation_over_load() (in module ding0.tools.results), 108
plot_km_cable_vs_line() (in module ding0.tools.results), 108
plot_mv_topology() (in module ding0.tools.plots), 103
print_solution() (in module ding0.grid.mv_grid.util.util), 93
print_upper_triangular_matrix() (in module ding0.grid.mv_grid.util.util), 93
print_upper_triangular_matrix_as_complete() (in module ding0.grid.mv_grid.util.util), 93
process() (ding0.grid.mv_grid.solvers.base.BaseSolution method), 86
process() (ding0.grid.mv_grid.solvers.savings.SavingsSolution method), 92
process_pf_results() (in module ding0.tools.pypsa_io), 105
process_stats() (in module ding0.tools.results), 108
pypsa_id (ding0.core.network.cable_distributors.MVCableDistributorDing0 attribute), 40
pypsa_id (ding0.core.network.GeneratorDing0 attribute), 51
pypsa_id (ding0.core.network.stations.LVStationDing0 attribute), 46
pypsa_id (ding0.core.network.stations.MVStationDing0 attribute), 46
reinforce_branches_current() (in module ding0.flexopt.reinforce_measures), 76
reinforce_branches_voltage() (in module ding0.flexopt.reinforce_measures), 76
reinforce_grid() (ding0.core.network.grids.LVGridDing0 method), 41
reinforce_grid() (ding0.core.network.grids.MVGridDing0 method), 43
reinforce_grid() (ding0.core.NetworkDing0 method), 68
reinforce_grid() (in module ding0.flexopt.reinforce_grid), 74
reinforce_grid() (in module ding0.flexopt.reinforce_measures), 76
remove_cable_distributor() (ding0.core.network.grids.MVGridDing0 method), 43
resolution (ding0.core.powerflow.PFConfigDing0 attribute), 58
ring (ding0.core.network.BranchDing0 attribute), 47
ring (ding0.core.structure.regions.LVLoadAreaDing0 attribute), 61
RingDing0 (class in ding0.core.network), 56
rings_count() (ding0.core.network.grids.MVGridDing0 method), 43
rings_full_data() (ding0.core.network.grids.MVGridDing0 method), 43
rings_nodes() (ding0.core.network.grids.MVGridDing0 method), 43
Route (class in ding0.grid.mv_grid.models.models), 82
route_allocation()  
  (ding0.grid.mv_grid.models.models.Node method), 82
routes()  (ding0.grid.mv_grid.solvers.base.BaseSolution method), 86
routing()  (ding0.core.network.grids.MVGridDing0 method), 44
routing_solution_to_ding0_graph()  (in module ding0.grid.mv_grid.mv_routing), 97
run_ding0()  (ding0.core.NetworkDing0 method), 68
run_powerflow()  (ding0.core.network.grids.MVGridDing0 method), 44
run_powerflow()  (ding0.core.NetworkDing0 method), 44
run_powerflow_onthefly()  (in module ding0.tools.pypsa_io), 105
routing_solution_to_ding0_graph()  (in module ding0.grid.mv_grid.mv_routing), 97
run_ding0()  (ding0.core.NetworkDing0 method), 68
run_powerflow()  (ding0.core.network.grids.MVGridDing0 method), 44
run_powerflow()  (ding0.core.NetworkDing0 method), 69
run_powerflow_onthefly()  (in module ding0.tools.pypsa_io), 105

S

s_max_a  (ding0.core.network.TransformerDing0 attribute), 57
s_max_b  (ding0.core.network.TransformerDing0 attribute), 57
s_max_c  (ding0.core.network.TransformerDing0 attribute), 57
s_res0  (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38
s_real  (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38
sanitize()  (in module ding0.grid.mv_grid.util.data_input), 93
save_nd_to_pickle()  (in module ding0.tools.results), 109
SavingsSolution  (class in ding0.config.config_db_interfaces), 37
scenarios  (ding0.core.powerflow.PFConfigDing0 attribute), 58
select_grid_model_residential()  (in module ding0.grid.lv_grid.build_grid), 79
select_grid_model_ria()  (in module ding0.grid.lv_grid.build_grid), 79
select_transformers()  (ding0.core.network.stations.MVStationDing0 method), 47
select_transformers()  (in module ding0.grid.mv_grid.build_grid), 80
set()  (in module ding0.tools.config), 100
set_branch_ids()  (ding0.core.network.grids.MVGridDing0 method), 44
set_branch_ids()  (ding0.core.NetworkDing0 method), 70
set_circuit_breakers()  (ding0.core.NetworkDing0 method), 70
set_circuit_breakers()  (in module ding0.grid.mv_grid.tools), 97
set_default_branch_type()  (ding0.core.network.grids.MVGridDing0 method), 45
set_nodes_aggregation_flag()  (ding0.core.network.grids.MVGridDing0 method), 45
set_operation_voltage_level()  (ding0.core.network.stations.MVStationDing0 method), 47
set_voltage_level()  (ding0.core.network.grids.MVGridDing0 method), 45
setUp()  (ding0.tools-tests.Ding0RunTest method), 109
setup_logger()  (in module ding0.tools.logger), 102
solve()  (ding0.grid.mv_grid.solvers.base.BaseSolver method), 86
solve()  (ding0.grid.mv_grid.solvers.local_search.LocalSearchSolver method), 90
solve()  (ding0.grid.mv_grid.solvers.savings.ClarkeWrightSolver method), 91
solve()  (in module ding0.grid.mv_grid.mv_routing), 97
split_ding()  (in module ding0.flexopt.reinforce_measures_dena), 77
sqla_mv_grid_viz  (class in ding0.config.config_db_interfaces), 37
sqla_mv_grid_viz_branches  (class in ding0.config.config_db_interfaces), 38
sqla_mv_grid_viz_nodes  (class in ding0.config.config_db_interfaces), 38
srid  (ding0.core.powerflow.PFConfigDing0 attribute), 58
static_data  (ding0.core.NetworkDing0 attribute), 63, 70
station()  (ding0.core.network.grids.LVGridDing0 method), 41
station()  (ding0.core.network.grids.MVGridDing0 method), 46
StationDing0  (class in ding0.core.network), 56
status  (ding0.core.network.CircuitBreakerDing0 attribute), 49
string_id  (ding0.core.network.cable_distributors.LVCableDistributorDing0 attribute), 40
strip()  (in module ding0.grid.mv_grid.util.data_input), 93
subtype  (ding0.core.network.GeneratorDing0 attribute), 51
tap_ratio (ding0.core.network.TransformerDing0 attribute), 57

tech_constraints_satisfied()
(ding0.grid.mv_grid.models.models.Route method), 84

test_ding0() (ding0.tools.tests.Ding0RunTest method), 109

test_ding0_file() (ding0.tools.tests.Ding0RunTest method), 109

test_files() (ding0.tools.tests.Ding0RunTest method), 109

timesteps (ding0.core.powerflow.PFConfigDing0 attribute), 59

to_dataframe() (ding0.core.NetworkDing0 method), 70

transform_timeseries4pypsa() (in module ding0.tools.pypsa_io), 105

transformer() (in module ding0.grid.lv_grid.build_grid), 80

TransformerDing0 (class in ding0.core.network), 57

transformers() (ding0.core.network.StationDing0 method), 57

type (ding0.core.network.BranchDing0 attribute), 47

type (ding0.core.network.GeneratorDing0 attribute), 50

type_kind (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38

type_name (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 38

type_s_nom (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 39

type_v_nom (ding0.config.config_db_interfaces.sqla_mv_grid_viz_branches attribute), 39

validate_generation() (in module ding0.tools.validation), 111

validate_grid_districts() (ding0.core.NetworkDing0 method), 70

validate_load_areas() (in module ding0.tools.validation), 111

validate_lv_districts() (in module ding0.tools.validation), 112

value (ding0.grid.mv_grid.util.data_input.ParseException attribute), 92

to_dataframe() (in module ding0.grid.mv_grid.models.models.Route method), 84

to_dataframe() (in module ding0.core.powerflow.PFConfigDing0 attribute), 59

voltage_delta_vde() (in module ding0.flexopt.check_tech_constraints), 74

weather_cell_id (ding0.core.network.GeneratorFluctuatingDing0 attribute), 51

z() (ding0.core.network.TransformerDing0 method), 58

v_level (ding0.core.network.GeneratorDing0 attribute), 49

v_level (ding0.core.network.TransformerDing0 attribute), 57

v_nom (ding0.config.config_db_interfaces.sqla_mv_grid_viz_nodes attribute), 39

v_res0 (ding0.config.config_db_interfaces.sqla_mv_grid_viz_nodes attribute), 39

v_res1 (ding0.config.config_db_interfaces.sqla_mv_grid_viz_nodes attribute), 39

validate_generation() (in module ding0.tools.validation), 111

validate_grid_districts() (ding0.core.NetworkDing0 method), 70

validate_load_areas() (in module ding0.tools.validation), 111

validate_lv_districts() (in module ding0.tools.validation), 112